

Web dizajn

Nenad Kojić



HTML, CSS i JavaScript

Beograd, 2024.

UNIVERZITET SINGIDUNUM

Nenad Kojić

WEB DIZAJN: HTML, CSS i JavaScript

Četvrto izdanje

Beograd, 2024.

WEB DIZAJN: HTML, CSS i JavaScript

Autor:

dr Nenad Kojić

Recenzenti:

dr Dragoslav Danilović

dr Goran Zajić

Izdavač:

UNIVERZITET SINGIDUNUM

Beograd, Danijelova 32

www.singidunum.ac.rs

Za izdavača:

dr Milovan Stanišić

Priprema za štampu:

Jelena Petrović

Dizajn korica:

Aleksandar Mihajlović

Godina izdanja:

2024.

Tiraž:

400 primeraka

Štampa:

Birograf, Beograd

ISBN 978-86-7912-661-0

Copyright:

© 2024. Univerzitet Singidunum

Izdavač zadržava sva prava.

Reprodukcija pojedinih delova ili celine ove publikacije nije dozvoljena.

Predgovor

Ovaj udžbenik namenjen je svima koji žele da ovladaju tehnikama i veštinama izrade interaktivnih statičkih web sajtova a prevashodno je namenjena studentima Univerziteta Singidunum koji slušaju predmet Web dizajn. U skladu sa tim, materija ovog udžbenika je usklađena sa silabusom predmeta i izučava oblast klijentskih web orijentisanih jezika i tehnika.

Materija knjige se tako može podeliti u dve celine: materija za izradu statičkih i materija za izradu interaktivnih web sajtova.

Na samom početku korisnici se mogu upoznati sa opštim pojmovima na Internetu, principom rada i tehnologijama za upotrebu i programiranje web sajtova.

Zatim se u prvom delu izučava jezik HTML, u obe svoje aktuelne verzije, HTML 4 i HTML 5, kao i CSS, u verzijama CSS 2 i CSS 3. Verzije HTML 5 i CSS 3 su izdvojene u posebna poglavlja jer predstavljaju nadogradnju verzija HTML 4 i CSS 2. U svim ovim poglavljima akcenat je stavljen na one elemente jezika koji su zadržani u narednim verzijama, imaju veću primenu u realnim web sajtovima i koji su najbitniji za prvi susret korisnika sa ovom materijom. Na taj način, materija ove knjige je prilagođena svima koji se nikada ranije nisu susreli sa izradom web sajtova, ali imaju bazno poznavanje osnova tj. principa programiranja. Ishod koji se očekuje nakon izučavanja prvog dela knjige je da studenti samostalno mogu da kreiraju proizvoljni statički web sajt.

U drugom delu se izučava programski jezik JavaScript sa svojom aktuelnom bibliotekom jQuery. Upotreba skripta omogućava da se sajt napravi interaktivnim i dobije kako na estetskom tako i na kvalitativnom nivou. Pored namene, osnovnih pravila jezika i upotrebe, posebna pažnja je posvećena implementacijama JavaScript-a kroz događaje, regularne izraze i manipulacije DOM-a. Na kraju drugog dela je obrađen jQuery sa akcentom na primere izrade menija, galerija i interaktivnu manipulaciju HTML i CSS elemenata unutar web strane.

Knjiga je podeljena na jedanaest poglavlja čiji je zadatak da čitaoca postepeno, ali kompletno, osposobe za samostalnu izradu interaktivnih statičkih web sajtova.

Ova knjiga predstavlja prvi korak u izučavanju postupka izrade web sajtova. U narednim predmetima će se obrađivati materija baza podataka i serverskih jezika kojima će se materija ovog udžbenika unapređivati u cilju kreiranja dinamičkih web sajtova.

Autor se ovim putem unapred zahvaljuje svim komentarima, predlozima i sugestijama čitalaca u nadi da će ova knjiga biti od koristi mnogima koji se bave web dizajnom i web programiranjem.

Beograd, Septembar 2017.

Autor

Predgovor

Izrada statičkih web sajtova

I Osnove WEB-a	1
1.1. Web strana	2
1.2. Web sajt	3
1.3. Podele programskih jezika za web sajtove	4
1.4. Tipovi web sajtova	5
1.5. Vrste korisnika web sajta	7
1.6. Web pregledač	8
1.7. URL adresa	11
1.8. HTTP	14
1.9. Komunikacija na Internetu	14
1.10. Domain Name System – DNS	17
1.11. Hosting	17
1.12. Domen	18
1.13. Web server	20
1.14. Web pretraživač	21
1.15. Oblast delovanja web dizajnera	24
II HTML - HyperText Markup Language	25
2.1. ML - Markup Language	26
2.2. Istorijski razvoj HTML-a	28
2.3. Princip funkcionisanja web stranice	29
2.4. Početak izrade web sajta	31
2.5. Struktura jezika HTML	33
2.6. HTML tagovi	34
2.7. Rad sa tekstom u HTML-u	37
2.7.1. Prikaz srpskih slova	37
2.7.2. Rad sa pasusima	37
2.7.3. Prelazak u novi red i načini ispisa unutar reda	38
2.7.4. Stilizovanje teksta	39
2.7.5. Entiteti	41
2.7.6. Tagovi H1-H6	42
2.7.7. Efekti nad tekstom	43
2.8. Rad sa grafikom	43
2.8.1. Kreiranje linija	44

2.8.2. Rad sa bojom	44
2.8.3. Postavljanje pozadinske boje	45
2.8.4. Rad sa listama	46
2.8.5. Definicione liste	47
2.8.6. Referenciranje objekata	48
2.8.7. Rad sa slikom	49
2.9. Hyper linkovi	51
2.9.1. Prikaz linkovanih dokumenta	53
2.9.2. Formatiranje teksta linka	53
2.9.3. Realizacija linka za slanje mail-a	54
2.9.4. Dodavanje slike u URL adresu	54
2.10. Rad sa multimedijalnim fajlovima	56
2.10.1. Pozadinski zvuk	56
2.10.2. Flash objekat	57
2.10.3. Video fajl	58
Organizacija sadržaja web stranice	59
2.11. Rad sa frame-ovima	59
2.12. Rad sa tabelama	61
2.13. Organizacija sadržaja pomoću CSS-a	65
2.14. HTML forme	70
2.14.1. Tekstualna polja	71
2.14.2. Radio tasteri	73
2.14.3. Check polja	74
2.14.4. Dropdown liste	74
2.14.5. Tasteri	76
2.14.6. Dizajn forme	78
2.14.7. Organizacija elemenata forme	78
2.15. Meta tagovi	80
2.16. xHTML	82
III CSS - Cascading Style Sheets	85
3.1. Struktura CSS fajla	89
3.2. CSS Selektori	90
3.2.1. Selektor u formi imena taga	90
3.2.2. Selektor u formi atributa id	91
3.2.3. Selektor u formi atributa class	92
3.2.4. Selektor u formi specifičnog atributa	95
3.3. Povezivanje HTML koda sa CSS kodom	96
3.4. Svojstva u CSS-u	99
3.4.1. Svojstva za pozadinu	100
3.4.2. Svojstva za tekst	104
3.4.3. Svojstva za fontove	106

3.4.4. Svojstva za linije	108
3.4.5. Svojstva za liste	109
3.4.6. Svojstva za pozicioniranje i prikaz	110
3.5. Primer organizacije sajta primenom CSS-a	116
3.5.1. Primer 1	117
3.5.2. Primer 2	131
3.6. Pozicioniranje elemenata	134
3.7. Uslovni komentari	142
3.8. Jedinice mere u CSS-u	142
3.9. Primeri parcijalnih primena	144
<i>Primer 1: Učitavanje eksternog fonta</i>	144
<i>Primer 2: Izrada horizontalnog menija</i>	145
<i>Primer 3: Stilizovanje forme</i>	146
IV HTML 5	149
4.1. Ključni novine u HTML 5	150
4.2. HTML 4 u odnosu na HTML 5	151
4.3. Primena HTML 5 na početku web strane	153
4.4. Primena HTML 5 za organizaciju strane	154
4.5. Primena HTML 5 za vizuelne efekte	157
4.6. HTML 5 tagovi za forme	160
4.7. HTML 5 tag <time>	161
4.8. Novi tipovi taga <input>	162
4.9. Novi atributi	165
4.10. Rad sa audio i video fajlovima	169
4.11. Rad Canvas-om	170
4.12. Rad sa geolokacijom	172
4.13. Web storage API	173
V CSS 3	175
5.1. Svojstvo border	176
5.2. Svojstvo background	178
5.3. Svojstva za tekst	183
5.4. Svojstvo @font-face	186
5.5. Rad sa tranzicijama	186
5.6. Rad sa sadržajem u više kolona	187
5.6.1. Svojstvo flex	189
5.7. Svojstvo resize	192
5.8. Primeri primene HTML 5 i CSS 3 koda	193
<i>Primer 1: Primer implementacije svojstva border-radius u realnim primenama</i>	193
<i>Primer 2: Primer upotrebe linearnog gradijenta</i>	194
<i>Primer 3: Primer realizacije neaktivnog tastera</i>	194

Izrada interaktivnih web sajtova

VI Uvod u JavaScript	195
6.1. O JavaScript-u	196
6.2. O Front-End-u i Back-End-u	199
6.3. Čemu služi i šta radi JavaScript	200
6.4. Šta je potrebno za pisanje JavaScript-a	203
6.5. Kako i gde se piše JavaScript	204
VII Osnove JavaScript-a	209
7.1. Identifikatori u JavaScript-u	210
7.2. Komentari u JavaScript-u	211
7.3. Promenljive u JavaScript-u	212
7.4. Operatori u JavaScript-u	213
7.5. Ispis sadržaja van vidljivog dela sajta	215
7.6. Specijalni stringovi	217
7.7. Operator typeof	220
7.8. Konverzija	223
7.9. Rad sa DOM-om	226
7.10. Sistemski dijalozi	228
7.11. Kontrole toka i petlje	231
7.12. Rad sa funkcijama	235
7.13. Rad sa nizovima	238
7.14. Rad sa objektima	244
7.15. Napredna upotreba objekata	247
7.16. Ispisivanje u dokumentu	252
7.17. Princip funkcionisanja JavaScript-a	255
7.18. Metode za rad sa stringovima	260
7.19. Metode za rad sa brojevima	264
7.20. Metode za rad sa datumom i vremenom	265
VIII Upravljanje DOM elementima	269
8.1. Metod getElementById()	270
8.2. Metod getElementsByClassName()	274
8.3. Metod getElementsByTagName()	275
8.4. Metod querySelector()	276
8.5. Metod querySelectorAll()	278
8.6. Svojstvo attributes i druga slična svojstva	278
8.7. Svojstva za rad sa node-ovima	280
8.7.1. Kreiranje novog elementa DOM-a sa tekstualnim sadržajem	282
8.7.2. Kreiranje novog elementa DOM-a sa proizvoljnim sadržajem	284
8.8. Svojstva za rad sa dimenzijama i pozicijama elemenata	287
8.9. HTML DOM objekti, kolekcije i svojstva	289
8.10. Svojstva objekta style	290

8.11. Svojstvo classList i njegove metode	292
IX Rad sa događajima	295
9.1. Načini aktivacije događaja	297
9.2. Vrste događaja	298
9.3. Implementacija događaja	300
9.3.1. Implementacija događaja u HTML-u	300
9.3.2. Implementacija događaja u JavaScript-u	303
9.4. Primeri implementacija događaja	308
9.5. Rad sa tajmerima	311
X Rad sa elementima forme	315
10.1. Rad sa elementima forme	317
10.1.1. Rad sa tekstualnim poljima	318
10.1.2. Rad sa checkbox-ovima	319
10.1.3. Rad sa radio tasterima	321
10.1.4. Rad sa dropdown listama	324
10.1.5. Rad sa tasterima	328
10.2. Regularni izrazi	330
XI jQuery	343
11.1. Princip rada jQuery-ja	344
11.2. Selektori u jQuery-ju	346
11.3. Metode jQuery-ja	347
11.4. Rad sa događajima u jQuery-ju	356
11.4.1. Upotreba događaja kao metoda objekta	356
11.4.2. Upotreba događaja pomoću metoda on()	360
11.5. Izrada menija u jQuery-ju	361
11.5.1. Ekspandirajući meni	362
11.5.2. Dropdown meni	364
11.5.3. Realizacija tabova	366
11.6. Rad sa elementima forme	370
Literatura	375
Zaključak	379

I Osnove WEB-a

U savremenom društvu, elektronski oblik komunikacija postao je vrlo dominantan. Na taj način, uloga web sajtova postaje još bitnija, naročito imajući u vidu da današnji korisnici korišćenje web sajtova doživljavaju kao svakodnevni izvor najšireg spektra informacija. Iz tog razloga, danas skoro da ne postoji firma ili kompanija, pojedinac ili grupa, roba ili usluga, koja se u nekom obliku ne nalazi na Internetu tj. na nekom web sajtu.

Ovakva potreba za web sajtovima dovela je do povećane tražnje za licima koja su stručna za kreiranje web sajtova i njihovo kasnije ažuriranje i održavanje. Pored toga, paralelno se vrlo intenzivno razvijaju nove tehnologije, okruženja i različiti programski i skript jezici kojima se posao kreiranja web sajta ubrzava, uprošćava i podiže na viši nivo.

Cilj ove knjige je da korisnike nauči prvim koracima u kreiranju realnih web sajtova. Međutim, pre nego što se to započne, neophodno je usvojiti pojedine termine, definisati njihova značenja i uvesti određene klasifikacije, da bi dalji rad bio jednostavniji i razumljiviji.

U tom smislu, u ovom poglavlju definisaće se najbitniji termini i dati njihova objašnjenja, na bazi kojih će se dalje usvajati novi nivoi detalja.

1.1. Web strana

Web strana je fajl, najčešće pisan u HTML-u ili nekom sličnom ML-u (*Markup Language*), koji je namenjen za WWW (*World Wide Web*) i prikaz u web browser-u. Ovaj fajl je skup određenog broja linija koda i predstavlja isključivo tekst. Međutim kada se ovaj kod interpretira, dobija se grafički prikaz koda koji je atraktivniji za krajnjeg korisnika. Grafičku vizuelizaciju koda realizuje web pregledač (*web browser*).

Na slici 1.1.a. prikazan je deo inicijalno napisane web stranice jednog sajta, a na slici 1.1.b. njena grafička vizuelizacija u browser-u.

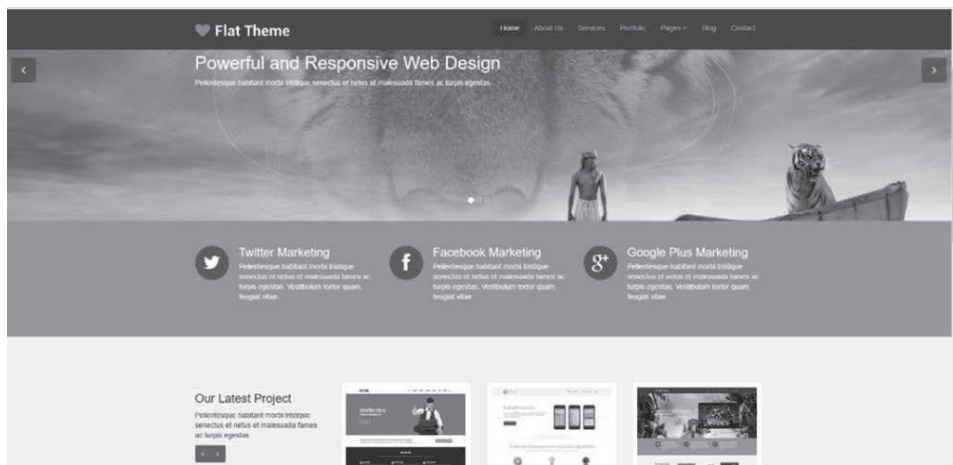
```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta name="description" content="">
7   <meta name="author" content="">
8   <title>Home | Flat Theme</title>
9   <link href="css/bootstrap.min.css" rel="stylesheet">
10  <link href="css/font-awesome.min.css" rel="stylesheet">
11  <link href="css/prettyPhoto.css" rel="stylesheet">
12  <link href="css/animate.css" rel="stylesheet">
13  <link href="css/main.css" rel="stylesheet">
14  <!--[if lt IE 9]>
15  <script src="js/html5shiv.js"></script>
16  <script src="js/respond.min.js"></script>
17  <![endif]-->
18  <link rel="shortcut icon" href="images/ico/favicon.ico">
19  <link rel="apple-touch-icon-precomposed" sizes="144x144" href="images/ico/apple-touch-icon-144-precomposed.png">
20  <link rel="apple-touch-icon-precomposed" sizes="114x114" href="images/ico/apple-touch-icon-114-precomposed.png">
21  <link rel="apple-touch-icon-precomposed" sizes="72x72" href="images/ico/apple-touch-icon-72-precomposed.png">
22  <link rel="apple-touch-icon-precomposed" href="images/ico/apple-touch-icon-57-precomposed.png">
23 </head><!--/head-->
24 <body>
25   <header class="navbar navbar-inverse navbar-fixed-top wet-asphalt" role="banner">
26     <div class="container">
27       <div class="navbar-header">
28         <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
29           <span class="sr-only">Toggle navigation</span>
30           <span class="icon-bar"></span>
31           <span class="icon-bar"></span>
32           <span class="icon-bar"></span>
33         </button>
34         <a class="navbar-brand" href="index.html"></a>
35       </div>
36       <div class="collapse navbar-collapse">
37         <ul class="nav navbar-nav navbar-right">
38           <li class="active"><a href="index.html">Home</a></li>
39           <li><a href="about-us.html">About Us</a></li>
40           <li><a href="services.html">Services</a></li>
41           <li><a href="portfolio.html">Portfolio</a></li>
42           <li class="dropdown">
43             <a href="#" class="dropdown-toggle" data-toggle="dropdown">Pages <i class="icon-angle-down"></i></a>
44             <ul class="dropdown-menu">
45               <li><a href="career.html">Career</a></li>
46               <li><a href="blog-item.html">Blog Single</a></li>
47               <li><a href="pricing.html">Pricing</a></li>
48               <li><a href="404.html">404</a></li>
49               <li><a href="registration.html">Registration</a></li>
50               <li class="divider"></li>
51               <li><a href="privacy.html">Privacy Policy</a></li>
52               <li><a href="terms.html">Terms of Use</a></li>
53             </ul>

```

Slika 1.1. a) Kod web strane

Kao što se vidi na slici 1.1. a) kod web strane je “čist tekst”, u kome nema slika, boje, fonta, nijansi, efekata, pozicije i sl. na način kako bi se možda moglo očekivati za početnike. Ovaj kod je pisan vrlo specifičnim jezikom koji browser jednoznačno razume i na kraju korisniku prikazuje u grafičkom obliku.



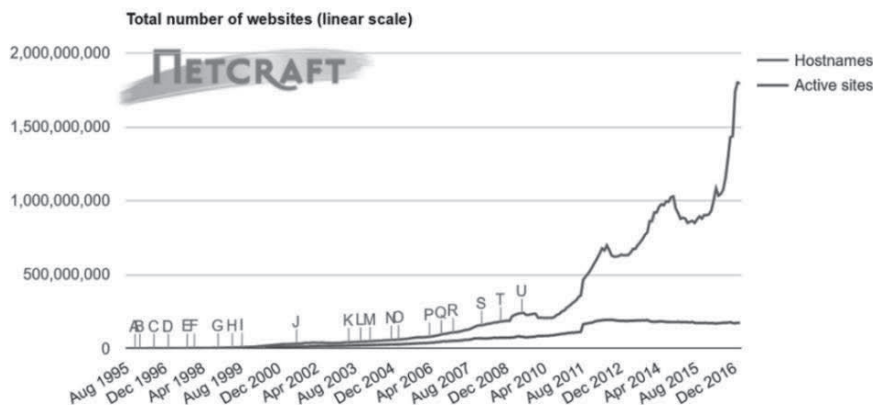
Slika 1.1. b) grafički prikaz strane sa slike 1.1. a) u browser-u.

Naš cilj je da naučimo da pišemo taj kod i da pomoću njega možemo da kreiramo bilo koji sadržaj i dizajn web strane.

1.2. Web sajt

Skup web strana, koje su najčešće međusobno povezane linkovima, predstavljaju jednu logičku celinu i nalaze se na istoj Internet lokaciji naziva se web sajt.

Danas postoji veliki broj web sajtova, a još veći broj web stranica. Prema podacima sa sajta <http://news.netcraft.com> broj sajtova u svetu, po godinama, dat je na slici 1.2. Obzirom da je prvi sajt zvanično pušten u rad 6. avgusta 1991. i da je bio jedini u toj godini, a da danas imamo preko 700 miliona sajtova sa jedinstvenom Internet adresom, jasno je u kakvoj ekspanziji web-a i sami učestvujemo. Ovakvi podaci obično su dostupni nakon kraja kalendarske godine.



Slika 1.2. Ukupan broj web sajtova u celom svetu, po godinama.
(izvor <http://news.netcraft.com>)

Takođe, treba razlikovati broj dostupnih Internet adresa, koje su zakupljene, ali na njima još uvek nema realnih web sajtova od broja aktivnih sajtova.

1.3. Podele programskih jezika za web sajtove

Pravljenje sajtova je složen i kontinualan proces, ali se u zavisnosti od načina rada, primenjenih tehnologija, koncepta organizacije sadržaja, navigacije i slično, može napraviti veći broj različitih podela.

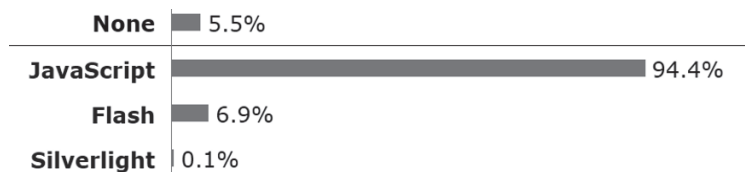
Obzirom da je web sajt produkt programiranja, možda prvo treba definisati programske jezike, i njihove podele, kada je reč o izradi web sajtova. Naime, u web programiranju postoji veliki broj markap, skript i programskih jezika koji se mogu koristiti za izradu celih ili dela sajtova. Svima njima je zajednička podela prema mestu tj. lokaciji, gde se ti jezici izvršavaju (tj. gde se konkretno realizuje ideja koju je programer osmislio). Ova podela je izvršena na klijentske i serverske jezike.

Klijentski jezici se izvršavaju na strani klijenta tj. na korisničkom računaru. Ovi jezici su manje komplikovani i manje tehnički zahtevni pa se lakše osigurava njihov rad na različitim performansama personalnih računara. Ovi jezici treba da budu nezavisni od strukture hardvera, operativnog sistema i korisničkih privilegija na tom računaru. Da bi ovi jezici imali grafičku vizualizaciju i da bi mogli da se realizuju potreban je poseban softver – *browser*. Klijentski jezici su po pravilu u osnovi jednostavniji, sa manje ograničenja i pravila, pa samim tim i prostiji za upotrebu. U ovu grupu jezika spadaju HTML, CSS, ActionScript, JavaScript, jQuery, ECMAScript, Jscript, VBScript, ... Kompletan kod, koji je pisan kao klijentski jezik, korisnik može videti u originalu u browser-u, ili u nekom tekstualnom editoru, na svom računaru.

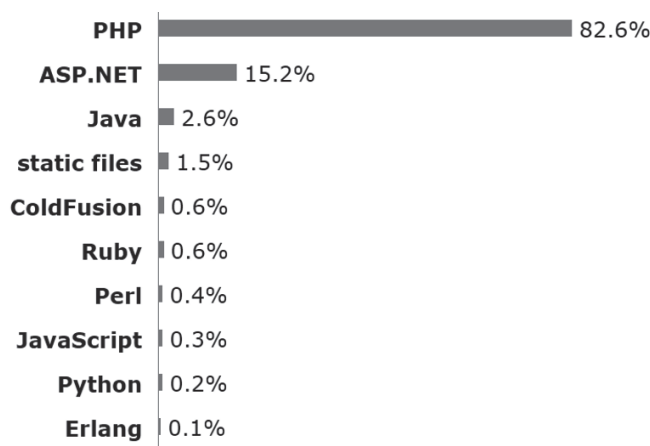
Serverski jezici se izvršavaju na web serveru. Ovi jezici su po pravilu kompleksniji i imaju mnogo strožija pravila korišćenja i sintaksu, nego klijentski. Potreban im je web server da bi mogli da se realizuju. Po pravilu, mogu da ostvare komunikaciju sa bazama podataka ili različitim tipovima tekstualnih i drugih fajlova. Na taj način, pored čitanja podataka, mogu da realizuju upis, promenu i brisanje podataka koji se trajno čuvaju. Serverski jezici najčešće služe za realizaciju pojedinih funkcionalnosti sajta, različite vrste bezbednosnih provera, generisanja dinamičkih sadržaja i slično. Međutim, iako deluju napredniji od klijentskih, serverski jezici ne mogu da prikažu korisniku rezultat svoga rada bez upotrebe klijentskih jezika. Na taj način, rezultat rada serverskih jezika se ugrađuje (*embed*-uje) u klijentske, i tek tada web sajt funkcioniše kao celina. To znači, da se rezultat svakog serverskog jezika na kraju mora predstaviti kao HTML ili CSS kod. Na taj način korisnik na svom računaru nikada ne može da vidi originalni serverski kod, kao što to može sa klijentskim, nego vidi samo konačni rezultat njegovog rada.

Primeri serverskih jezika su PHP, ASP, ASP.NET, Java, ColdFusion, Perl, Ruby, Python, server-side JavaScript, ...

Prema podacima sa sajta <http://w3techs.com/>, zaključno sa martom 2017. godine, zastupljenost klijentskih i serverskih jezika u sajtovima je prikazana na slici 1.3 i slici 1.4, respektivno.



Slika 1.3. Zastupljenost klijentskih jezika u sajtovima. (izvor <http://w3techs.com/>)



Slika 1.4. Zastupljenost serverskih jezika u sajtovima. (izvor <http://w3techs.com/>)

1.4. Tipovi web sajtova

Pored podela na tipove korišćenih programskih jezika, jedna od osnovnih koncepcijskih podela je i prema načinu kako se sadržaj web stranice generiše. Ova podela izvršena je na statičke i dinamičke sajtove.

Statički sajt podrazumeva skup više web stranica, koje programer u nekom trenutku kreira, i nakon toga one postanu dostupne krajnjim korisnicima. Svaki put kada korisnik želi da vidi neku od ovih stranica, njen sadržaj se u originalu prikazuje korisniku, na način kako je to inicijalno, ranije, napravio programer, za svaku konkretnu stranicu kojoj korisnik pristupa. Žargonski, statički sajtovi se porede sa štampanim novinama, koje se jednom kreiraju i ostaju stalno iste. Obzirom da se uvek prikazuje isti sadržaj, i da on ne zavisi od trenutka pristupa, načina rada, nekog karakterističnog događaja, tipa korisnika, prava i privilegija i slično, ovakvi sajtovi se nazivaju statički sajtovi. Najčešće su kreirani primenom

HTML-a i CSS-a. Obzirom na zahteve savremenih korisnika, ovakvi sajtovi su sve više prevaziđeni i koriste se za slučajeve kada se ne očekuje česta promena u sadržaju sajta i kada nema potrebe da se način prikaza na bilo koji način modifikuje.

Dinamički sajt podrazumeva da se ceo sadržaj, ili njegov veći deo, preuzima iz baze podataka (ili nekog tipa tekstualnog fajla). Kod dinamičkih sajtova se sadržaj koji se šalje korisniku, a koji predstavlja sadržaj željene web stranice, formira u trenutku kada to korisnik zahteva od web servera. Na taj način, ukoliko je došlo do promene sadržaja u bazi podataka, i sadržaj stranice može biti drugačiji od prethodnog prikaza, pa se ovakvi sajtovi mogu kategorisati kao dinamički sajtovi. Primer ovakvih sajtova su socijalne mreže, gde se npr. dodavanjem slike ili komentara jednog korisnika, kod svih njegovih prijatelja menja sadržaj početne stranice, na način da se od tog trenutka prikazuje i ta slika tj. komentar njihovog prijatelja. Na ovaj način, kod dinamičkih sajtova, sadržaj iste web stranice sajta može biti drugačiji, u istom trenutku, kod različitih korisnika, a dodatno različit u narednom trenutku vremena.

Da bi sajt bio dinamički, on mora biti delom pisan u nekom od posebnih tzv. serverskih jezika (PHP, ASP, ...) koji imaju mogućnost komunikacije sa bazom podataka. Ovakvi sajtovi moraju imati integrisane i sve one jezike koji se koriste za statičke sajtove (HTML i CSS), pa se smatraju nadogradnjom, tj. višim nivoom statičkog sajta. Kod ovakvih sajtova, sajt može imati svaku web stranicu realizovanu kroz poseban kod/stranu (kao i kod statičkih sajtova), npr. stranicu *kontakt.php* ili *artikli.php*, ali se za razliku od statičkih može realizovati i tako da ceo sajt (sa proizvoljnim brojem strana) bude realizovan kroz samo jednu fizičku stranu *index.php* (*Single page website*) ili kroz neki drugi oblik.

Podela sajtova na statičke i dinamičke, kao što smo videli, zavisi od trenutka kreiranja korisničkog sadržaja, programskih jezika i tehnologije koja to može da omogući. Međutim, dobijanjem sadržaja web strane, od strane web servera, korisnik može aktivno i pasivno učestvovati u daljem korišćenju te web stranice. Pasivno korišćenje podrazumeva da korisnik čita tj. gleda sadržaj date stranice, na način kako je ona inicijalno prikazana, dok mu aktivni pristup može omogućiti da dalje eventualno menja način prikaza podataka ili sadržaj koji se vidi. Ukoliko određeni sajt omogućava ovakav „aktivan“ režim rada, onda se on kategoriše kao *interaktivan*.

Interaktivan sajt omogućava korisnicima da se u zavisnosti od određenog događaja (kliku mišem na neku oblast, prelaska mišem preko neke slike ili linka, pokušajem napuštanja web strane, ...), trenutka (vreme provedeno od učitavanja strane, vreme od nekog događaja, vreme u odnosu na prethodnu fazu rada, ...) ili na bazi prethodnih statistika tj. aktivnosti korisnika na tom sajtu, sadržaj web strane delimično ili potpuno promeni kao i način prikaza podataka. Ovo se najčešće odnosi na prikaz dodatnih informacija o proizvodu ili događaju, dinamičku promenu slika i teksta u centralnom delu sajta, dinamičku promenu

vesti, sakrivanje ili prikaz dela menija, primenu neke od animacija za tekst ili sliku, automatsko učitavanje novih ili dodatnih podataka, prikaz slika u većem formatu unutar iste stranice, aktivacije tajmera i slično. Interaktivan sajt ne pripada istoj grupi podela kao statički i dinamički, i odnosi se na aktivnost korisnika nakon dobijanja sadržaja sajta. Na taj način, i statički i dinamički sajt, mogu ali ne moraju biti interaktivni, i to isključivo zavisi od dodatnog koda koga programer definiše unutar neke web strane. Najčešće se ovakve aktivnosti dešavaju na strani korisnika i za to se koriste klijentski programski jezici (JavaScript, jQuery, Action script, Angular, React...).

1.5. Vrste korisnika web sajta

Životni vek jednog web sajta je definisan ili trenutkom dokle je on fizički prisutan na Internetu ili dok ga bilo koji korisnik posećuje. Sa druge strane, ni svi korisnici sajta nisu isti. Korisnici sajta mogu biti lica sa potpuno različitim privilegijama. Tako se može definisati i podela po tipu korisnika, koji pristupaju određenom web sajtu, na neautorizovane i autorizovane korisnike.

Neautorizovani korisnici su inicijalno najšira populacija krajnjih korisnika web sajta. Ovi korisnici imaju prava da pristupaju i gledaju sve stranice sajta koji su javno dostupne. Neautorizovani korisnici najčešće nemaju prava da menjaju ili utiču na promenu sadržaja i obima sajta, pa se definišu kao korisnici sa najnižim pravima. U praksi, oni najčešće imaju samo prava čitanja pojedinih sadržaja web sajta.

Autorizovani korisnici su korisnici koji su uspešno prošli proces logovanja (autentifikacije i autorizacije).

Autentifikacija podrazumeva proces kojim server utvrđuje identitet korisnika. To podrazumeva da je neko ranije definisao njihove pristupne podatke (korisničko ime i lozinku) koje korisnik svaki put unosi sa ciljem da se serveru jedinstveno predstavi.

Autorizacija podrazumeva definisanje prava i privilegija koje korisnik ima. Najčešće se autorizacija vezuje za prethodni proces autentifikacije, tj. identifikacije korisnika. Ovo se realizuje sa ciljem da se definišu prava upotrebe resursa i pristupa fajlovima u odnosu na korisnika koji želi da im pristupi.

U praksi se najčešće definišu tri tipa autorizovanih korisnika:

Klasični autorizovani korisnici – ovi korisnici su posetioци sajta kojima je dato veće pravo pristupa pojedinim zaštićenim stranama sajta, a ponekad i mogućnost definisanja dela sadržaja web strane, nego što je to slučaj sa neautorizovanim korisnicima. Ovi korisnici mogu da pristupe pojedinim zaštićenim stranama i na taj način imaju dodatne informacije ili funkcionalnosti (mogućnost kupovine,

ostavljanja komentara, dodavanja tema na blogovima i forumima, postavljanje pitanja, ocenjivanje pojedinih sadržaja,...).

Moderatori – stručna lica, koja ne moraju biti programeri, i koja imaju zadatak da se bave strategijom razvoja sajta i definisanjem sadržaja stranica (proširenja broja strana, promene sadržaja pojedinih stranica sajta, sloganima, uređivanje foruma, blogova, promenama elemenata u navigaciji, definisanju banera, editovanjem dnevnih sadržaja, generisanje vesti, slika i slično).

Administratori – stručna lica, po pravilu web programeri, koji se bave fizičkom realizacijom zahteva, od strane moderatora ili vlasnika sajta, u smislu proširivanja ili modifikacije obima sadržaja ili funkcionalnostima sajta. Administratori su po pravilu jedini korisnici koji imaju pristup svim delovima sajta i mogu upravljati svim drugim nalogima autentifikovanih korisnika.

1.6. Web pregledač

Web pregledač (*web browser*) je softver koji je inicijalno definisan kao desktop aplikacija (što najčešće podrazumeva da postoji *exe* verzija softvera, koji je potrebno preuzeti i instalirati na korisničkom računaru). Web browser ima dve osnovne funkcije:

1. Komunikaciju i razmenu podatka sa web serverom i
2. Grafičku vizuelizaciju klijentskih jezika.

Da bi krajnji korisnik video sadržaj neke web strane, prvi korak je da se putem browser-a obrati nekoj udaljenoj web lokaciji, na kojoj se nalazi željena web stranica. Ovo se realizuje unosom Internet adrese nekog web sajta u browser i klikom na *Enter*.

Ova komunikacija podrazumeva nekoliko koraka koji se odvijaju u „pozadini“, i koje korisnik ne vidi, sa ciljem razmene većeg broja informacija o samom računaru, tipu i verziji browser-a, vrsti i verziji protokola, rezoluciji monitora, i sl. Kada se ovi koraci realizuju, i potvrdi da su stečeni svi uslovi za transfer podataka sa web servera ka browser-u, prvo se prosleđuje HTML kod web strane. Učitavanjem HTML koda, počinju da se generišu ponovni zahtevi ka web serveru za svaki pojedinačni eksterni fajl koji je definisan u tom HTML-u kodu (CSS kod, skript kodove, slike, animacije, audio fajlove i sl.). Sve ove aktivnosti browser obavlja automatski, bez intervencije korisnika.

Na taj način browser ima veći broj komunikacija sa web serverom putem kojih zahteva i dobija pojedine fajlove potrebne za prikaz željene web strane.

Druga funkcija browser-a je grafička vizuelizacija kompletnog sadržaja web strane. Ova druga uloga browser-a podrazumeva da sav kod koji predstavlja jednu

web stranicu bude „virtuelno“ pretvoren u grafiku (boju, sliku, poziciju, veličinu teksta, linkove, animacije, zvuk...).

Browser može da vizualizuje samo kod napisan u klijentskim jezicima, dok sav drugi ili uopšte ne analizira i ne prikazuje ili prikazuje u originalu, jer ga “ne razume”.

Obzirom da je browser softver, nakon uspešnog pokretanja istog, njegov rad bi trebalo da bude nezavisan od hardvera i operativnog sistema na kome radi. Na žalost, veliki problem u realnom životu je što postoji veći broj proizvođača browser-a, koji su u obavezi da klijentski kod tumače i prikazuju u standardizovanom obliku, ali u praksi postoje odstupanja. Tako se često dešava da čak i isti proizvođač browser-a, ali u različitim verzijama istog browser-a, istu web stranicu prikazuju sa manje ili više promenjenim detaljima. Ovo se najčešće odnosi na precizan prikaz pozicija pojedinih delova sajta, inicijalnih margina i međusobnih rastojanja, pojedinih boja i fontova itd. Ovaj problem web programeri u najvećem broju slučajeva mogu da reše pisanjem dodatnog koda koji prepoznaje vrstu i verziju browser-a, pa isporučivanjem različitog koda kojim se dobija isti krajnji vizuelni efekat web strane kod korisnika.

Trenutno najpopularniji browser-i su: Chrome, Internet Explorer, Firefox, Opera, Safari, Edge ... Zastupljenost pojedinih browser-a je vrlo različita u odnosu na različite kriterijume. Tako su potpuno različite statistike kada se posmatra zastupljenost browser-a po pojedinim zemljama, kontinentima, kalendarskim godinama, polu korisnika, starosnim strukturama i sl.

Na slikama 1.5.a i 1.5.b prikazane su regionalne zastupljenosti browser-a u odnosu na kontinente u različitim godinama. Cilj ovih slika je da pokaže velike dinamike promena u kratkim vremenskim periodima.



Slika 1.5.a Većinska geografska zastupljenost browser-a u januaru 2009. po redosledu IE, Firefox, Opera i Ostali (izvor <http://gs.statcounter.com/>)



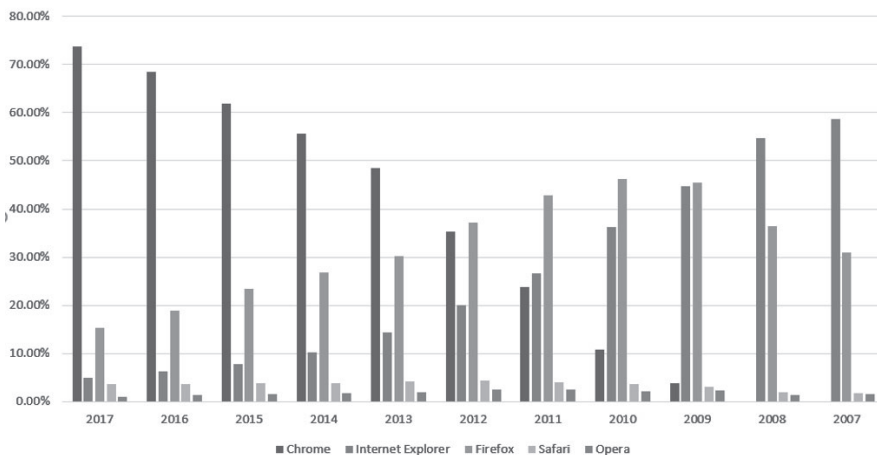
Slika 1.5.b Većinska geografska zastupljenost browser-a u januaru 2017. po redosledu Chrome, UC, Opera, Safari, Firefox, IE,... (izvor <http://gs.statcounter.com/>)

Imajući u vidu da browser-i ne prikazuju uvek na identičan način kod web strane, i da se programer trudi da sajt najbolje izgleda u najzastupljenijem browser-u a da se zastupljenost browser-a menja velikom dinamikom, posao web programera je konstantan. Za razliku od drugih struka gde kada se neki proizvod napravi, i dok traje nema potreba da se dodatno „dorađuje“, sajt se permanentno mora ažurirati, održavati i prilagođavati promenama u web browser-ima, web-pretraživačima, web serverima, ...

U tabeli 1.1 i slici 1.6 su dati uporedni podaci za različite godine na bazi podataka sa <http://www.w3schools.com/>.

Tabela 1.1. Procentualna zastupljenost browser-a, u ukupnoj populaciji, po godinama.

Januar	Chrome	Internet Explorer	Firefox	Safari	Opera
2017.	73.7 %	4.9 %	15.4 %	3.6 %	1.0 %
2016.	68.4 %	6.2 %	18.8 %	3.7 %	1.4 %
2015.	61.9 %	7.8 %	23.4 %	3.8 %	1.6 %
2014.	55.7 %	10.2 %	26.9 %	3.9 %	1.8 %
2013.	48.4 %	14.3 %	30.2 %	4.2 %	1.9 %
2012.	35.3 %	20.1 %	37.2 %	4.3 %	2.4 %
2011.	23.8 %	26.6 %	42.8 %	4.0 %	2.5 %
2010.	10.8 %	36.2 %	46.3 %	3.7 %	2.2 %
2009.	3.9 %	44.8 %	45.5 %	3.0 %	2.3 %
2008.	/	54.7 %	36.4 %	1.9 %	1.4 %
2007.	/	58.6 %	31.0 %	1.7 %	1.5 %



Slika 1.6. Zastupljenost browser-a u ukupnoj populaciji po godinama.

Ovi podaci imaju za cilj da ukažu na veliku fluktuaciju proizvođača browser-a, njihovu zastupljenost, dinamiku u promenama i slično.

Obzirom na činjenicu da se različiti proizvođači browser-a trude da privuku što više korisnika, dizajn browser-a je najčešće kompletno različit ne samo između proizvođača nego i verzija istog browser-a. Međutim, osnovne korisničke funkcionalnosti su u najvećoj meri iste (tasteri: *Back*, *Forward*, *Reload*, *Home*, polje za URL adresu, ...). Međutim, veliki broj *plug-in*-ova za pojedine browser-e mogu ponuditi potpuno drugačije specifične pogodnosti za korisnike.

Intenzivnim razvojem tehnologije, okruženja i programskih jezika za izradu web sajtova, aktuelne programske mogućnosti, a posebno različiti efekti i animacije u web sajtovima, ne mogu da se vide u starijim verzijama browser-a, pa smo prinuđeni da konstantno ažuriramo browser-e i budemo u toku sa aktuelnim promenama.

1.7. URL adresa

Termin URL je skraćenica od *Uniform Resource Locator* i predstavlja mrežnu adresu nekog fajla. URL definiše jedinstvenu adresu pomoću koje se bilo koji dokument koji se nalazi negde u mreži može jednoznačno locirati. Ova jedinstvena mrežna adresa se može uporediti sa jedinstvenom poštanskom adresom u realnom životu (npr. Petar Petrović, Mike Mikića 16, 11000 Beograd).

Iz tog razloga, URL adresa sadrži i ime servera, i kompletnu strukturu foldera na njemu, do fajla, zaključno sa imenom i ekstenzijom željenog fajla. Kako je termin URL inicijalno korišćen za pronalaženje i dohvaćanje fajlova, neophodno je da se definiše i protokol kojim će se željeni dokument preuzeti. Na taj način, URL adresa ima tri dela, koji se automatski detektuju, ukucavanjem URL adrese u browser:

1. Protokol
2. Ime servera
3. Putanja i ime fajla na samom serveru.

Ukoliko se određena HTML stranica, *proba.html*, nalazi na serveru *www.proba.com*, i to unutar dva foldera *sajtovi/proba_sajt*, puna URL adresa u ovom slučaju bi bila

http://www.proba.com/sajtovi/proba_sajt/proba.html

gde je:

1. Protokol (*http*)
2. Ime servera (*www.proba.com*)
3. Putanja i ime fajla na samom serveru (*sajtovi/proba_sajt/proba.html*).

Na sličan način, primer URL adrese za fajl *dokument.txt*, koji se nalazi u folderu *files* na web serveru *server.com*, a kome se pristupa *ftp* protokolom, bila bi:

<ftp://server.com/files/dokument.txt>

U osnovi, ime servera je uvek jedinstveni skup cifara, u standardizovanom formatu, i naziva se IP adresa. Tako na primer IP adresa za <https://www.telenor.rs/> je 217.65.192.17. Kako je u praksi često teško pamtići veliki skup cifara, omogućeno je da se umesto IP adrese, u URL adresu unese tekst, koji je bliži logici pamćenja kod korisnika. Međutim, čak i ako se unese tekst umesto IP adrese, u prvom koraku koji se dešava u pozadini procesa komunikacije browser-a sa web serverom, ovaj tekst se pretvara u IP adresu, i dalje se traženje željenog servera nastavlja po IP adresi. Pretvaranje tekstualnog imena servera u IP adresu jednoznačno je definisano skupom pravila DNS (*Domain name system*). Ovu konverziju fizički realizuje DNS server. Na taj način, URL adrese:

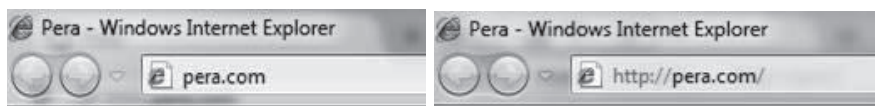
<http://217.65.192.17> i

<http://www.telenor.rs/>

su potpuno identično tumačene od strane browser-a, i jednoznačno je definisano koji će se sadržaj prikazati korisniku.

URL adresa se uvek sastoji od tri definisana dela, ali je dozvoljeno da korisnik ne unese svaki od ovih delova. U tom slučaju, sam browser ima definisane podrazumevane vrednosti, koje upisuje na mesta gde je to korisnik propustio. Na ovaj način moguće je ne uneti protokol kao i putanju i ime fajla na samom serveru, dok ime servera mora da bude definisano. Na primer, ako se kao URL adresa upiše *pera.com*, što je ime servera, browser će pokušati da tom sadržaju

pristupi primenom *http* protokola, i sam će definisati taj nedostajući deo, nakon klika na *Enter*, slika 1.7.



Slika 1.7. Prikaz automatskog dodavanja protokola u browser-u.

Pored toga, browser će u nedostatku definisane stranice pokušati da pristupi stranici *index.html*, *index.htm* ili *index.php*. Ako ta stranica postoji, automatski će se prikazati njen sadržaj, kao da je korisnik upisao npr. *http://pera.com/index.php*. U slučaju da ime stranice nije definisano u URL-u, a pritom ta stranica i ne postoji na serveru, browser će opet pokušati da joj pristupi, ali će se, u ovom slučaju, na kraju korisniku prikazati greška.

Pored inicijalne namene URL adrese, ona se može dodatno koristiti i za prenos podataka od strane korisnika ka web serveru. U ovom slučaju, početni deo URL adrese ostaje isti, ali se proširenje realizuje iza imena stranice kojoj se pristupa. Da bi se odvojilo ime stranice i podaci koji se prosleđuju koristi se simbol *?*. Iza ovog simbola definiše se standardizovani paket informacija u formatu:

```
ime_podatka1=sadržaj_podatka1
```

U slučaju da postoji više ovakvih paketa informacija, oni se međusobno odvajaju sa *&*. Tako je URL adresa, pomoću koje se prenosi podatak o imenu i prezimenu korisnika formata:

```
http://primer.com/index.php?ime=petar&prezime=peric
```

Na ovaj način korisnik koji popuni svoje podatke u nekom web formularu, te podatke prenosi udaljenoj web stranici, definisane URL adresom, koja se fizički nalazi na web serveru, i koja te podatke dalje obrađuje.

Pored ovakvog načina prenosa podataka ka web serveru, putem URL adrese, moguće je da se koristi i drugačija sintaksa u slučaju korišćenja npr. MVC arhitekture. Tada format URL adrese, koja prenosi podatke o imenu i prezimenu korisnika može izgledati kao:

```
http://primer.com/obrada/petar/peric
```

URL adresa je termin koji se ne koristi samo u kontekstu browser-a, nego i u kontekstu delova koda kojim se ukazuje na neki konkretan fajl u mreži. U ovakvim slučajevima treba biti posebno obazriv, jer ukoliko se tada ne upiše npr. protokol, ili se neispravno koriste velika i mala slova, kod neće moći da se realizuje, jer tada nema isprogramirane funkcionalnosti browser-a koji će pokušati da te nedostatke kompenzuje.

1.8. HTTP

HTTP je skraćenica od *Hypertext Transfer Protocol* i predstavlja protokol namenjen prenosu informacija na WWW (*World Wide Web*). Protokol je skup tehnika i pravila koja koriste mrežni uređaji da bi preneli sadržaj iz jedne na drugu mrežnu lokaciju. HTTP je namenjen komunikaciji i prenosu *hypermedia* u koje spada i *Hypertext*. Hypertext je posebno struktuiran tekst koji pored sadržaja teksta ima i dodatna značenja istog (na primer kojom bojom, stilom, poravnanjem, fontom, veličinom i sl. da se neki sadržaj prikaže korisniku).

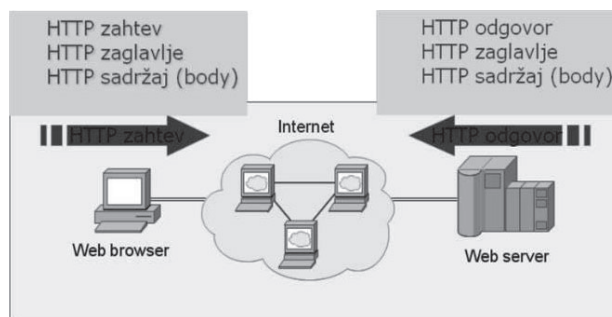
Kako je svaki web sajt inicijalno baziran na *Hypertext*-u (što je i jezik HTML), HTTP predstavlja ključni element u komunikaciji i razmeni podataka između web browser-a i web servera na kome se nalaze svi potrebni elementi web stranice. Kako je ova komunikacija definisana u više koraka, bazični princip rada HTTP je *request/response* karakteristika. To znači da browser svaki put kada ima potrebu da nešto traži od servera, za potrebe prikaza web strane (npr. sliku, CSS kod, način animacije, zvuk...), generiše *http request*. HTTP-om se ovaj zahtev isporučuje web serveru (definisanim unutar URL adrese) i server nakon obavljenog posla svoj odgovor šalje putem HTTP-a u formi *http response*. Ovaj postupak se ponavlja onoliko puta koliko browser ima potrebe da zahteva pojedine fajlove ili podatke od web servera.

Razvojem HTTP-a je između ostalog koordinirao i W3C (*World Wide Web Consortium*), <http://www.w3.org/>, koji je posebno bitan za sve segmente web dizajna i web programiranja, obzirom da definiše veliki broj pravila i standarda web aplikacija, arhitekturu, tehnologiju, servise, browser-e i alate u domenu web-a. Iz tog razloga, toplo preporučujem da se povremeno prate informacije koje se na ovoj adresi objavljuju.

1.9. Komunikacija na Internetu

Komunikacija između browser-a i web servera započinje nakon ispravnog unosa URL adrese i klikom na odgovarajući taster (npr. *Enter*) ili automatski ako je tako isprogramirano (AJAX). Klikom na taster započinje postupak slanja i deljenja URL adrese na njene delove, da bi se znalo kojim protokolom se nastavlja komunikacija, sa kojim serverom (na kojoj IP adresi) i sa kojim fajlom na samom serveru (web strana).

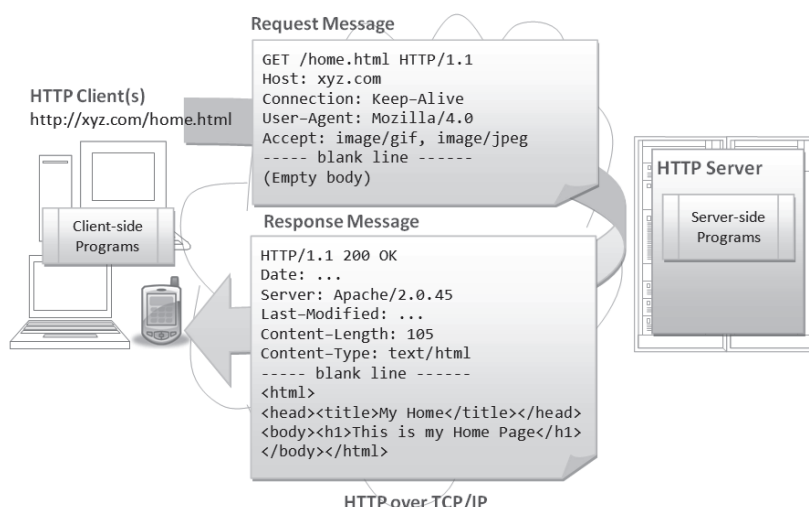
Ako pretpostavimo da je upisana adresa <http://primer.com/>, dalja komunikacija će se nastaviti HTTP protokolom ka serveru *primer.com*, i pokušajem da se pristupi strani *index.html* ili *index.php*. Ako pretpostavimo da takva stranica postoji, i to *index.html*, i da u sebi pored nekog teksta ima i dve slike (*1.jpg* i *2.jpg*) i poseban CSS fajl (*stil.css*), komunikacija se obavlja u nekoliko koraka (*request/response*), slika 1.8.a i slika 1.8.b.



Slika 1.8.a. Grafički prikaz klijent-server komunikacije.

1. Prvo browser generiše *http request* – Ovaj request u sebi sadrži HTTP zahtev, HTTP zaglavlje i HTTP sadržaj.

- ✓ HTTP zahtev sadrži tri tipa informacija (naredbu metod, putanju od web servera do podataka koje je klijent zahtevao i verziju HTTP-a). Primer: *Get/proba.html HTTP/1.1*.
- ✓ U HTTP zaglavlju se nalaze detalji o tipovima dokumenata koje će klijent prihvatiti sa servera, tipu web browser-a, datumu i opštim informacijama o konfiguraciji računara.
- ✓ U HTTP sadržaju nalaze se sve informacije koje se šalju serveru od strane korisnika (ako je metod slanja bio POST).



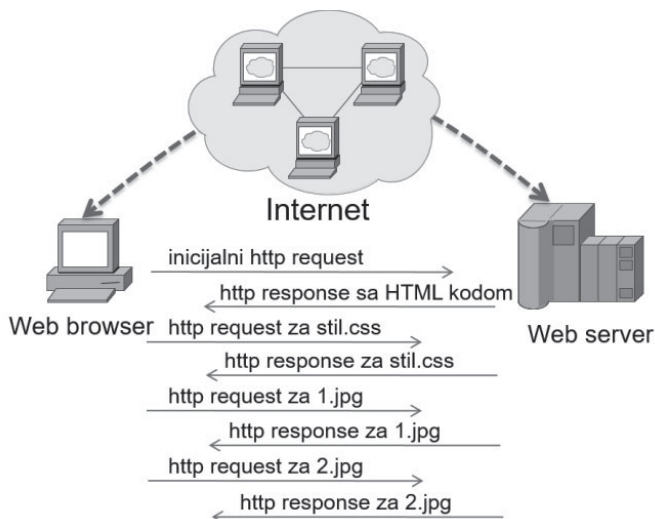
Slika 1.8.b. Prikaz klijent-server komunikacije na nivou http poruka.

2. Server browser-u generiše *http response* – Ovaj response u sebi sadrži HTTP odgovor, HTTP zaglavlje i HTTP sadržaj.

- ✓ HTTP odgovor sadrži verziju HTTP-a i kod HTTP-ovog zahteva koji definiše uspeh ili neuspeh zahteva. Primer: *HTTP/1.1 200 OK*.

- ✓ HTTP zaglavlje ima tri tipa informacija: Opšte - opšte informacije o klijentu i serveru, Entitet - sadrži informacije o podacima koje se razmenjuju između klijenta i servera, Zahtev - sadrži informacije o serveru koji šalje odgovor i uputstvo kako može da se odgovor obradi. Ako je zahtev uspešan onda se u HTTP sadržaju nalazi HTML kod tražene web stranice.
3. Dobijanjem HTML koda tražene stranice, browser počinje da HTML kod pretvara u grafiku. Sav tekst ove stranice je dobijen u ovom koraku sa HTML kodom, ali svi drugi fajlovi nisu (npr. slike i CSS-fajl).

Međutim, u HTML kodu se nalaze putanje do tih fajlova. Kada browser dođe do linije koda, gde se definiše potreba za nekim od tih fajlova (npr. *stil.css*), on ponovo generiše *http request* ka web serveru, zahtevajući taj fajl. Uspešnim pronalaženjem traženog fajla, server generiše *http response* sa sadržajem tog fajla. Ovo se ponavlja za sve fajlove koji se pozivaju u HTML kodu. Ovo je šematski prikazano na slici 1.9.



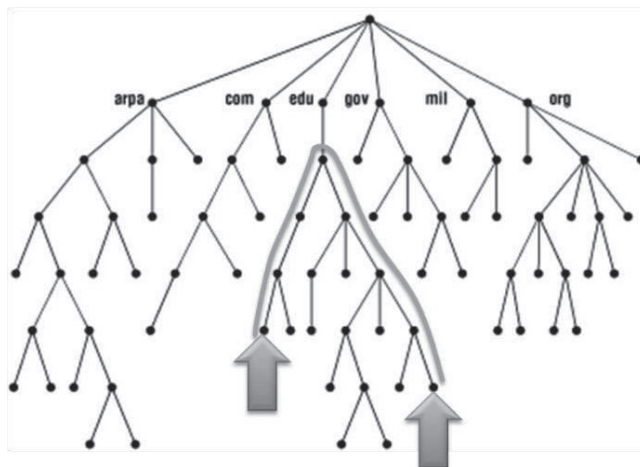
Slika 1.9. Detaljan prikaz generisanja *http request-a* i *http response-a*, na primeru konkretne web stranice.

U slučaju da se u postupku *request/response* komunikacije, bilo koji od dodatno traženih fajlova ne pronađe, browser će nastaviti sa narednim fajlom i neće prekidati rad. Na kraju, svi fajlovi koji nisu dobijeni se neće prikazati, a svi drugi hoće.

Ova celokupna komunikacija se odvija u pozadini rada browser-a i bez intervencije korisnika, nakon unosa URL adrese. Svaki kasniji klik na neki link unutar web stranice, iz početka započinje proces generisanja *request/response* komunikacije sa ciljem prikaza nove stranice, slike, teksta ili sl.

1.10. Domain Name System - DNS

Kao što je već pomenuto, DNS predstavlja servis za jedinstveno prevođenje *domain names* u IP adresu. Nakon što korisnik, u browser, unese tekstualnu URL adresu, browser ovu adresu prosleđuje (najbližem) DNS-u, koji se nalazi kod Internet provajdera (Internet provajder je kompanija koja pruža Internet usluge i omogućava nam da „preko njih“ naš računar bude povezan na globalnu mrežu).



Slika 1.10. Primer putanje unutar DNS hijerarhije.

Tamo se tekstualna adresa prevodi u IP adresu i saznaje se adresa web servera gde se nalazi tražena web stranica. Ukoliko je tražena web strana kod istog Internet provajdera, gde je i DNS, preko njega se “zatvara” putanja do tražene strane. Ukoliko nije, lokalni DNS se obraća nadređenom (u hijerarhijskom nivou) dok ne nađe nekog “ko zna za taj server” i kako da usmeri zahtev traženom serveru. Tako se može stići i do vrha u hijerarhiji (npr. *com* domena), slika 1.10.

1.11. Hosting

Hosting je usluga koju Internet provajderi nude vlasnicima web sajtova i podrazumeva iznajmljivanje prostora na hard disku web servera uz dodatne usluge administracije baze podataka i tehničke podrške. Na taj način, neki web sajt se nalazi na nekom web serveru i postaje dostupan tj. vidljiv za druge korisnike na Internetu.

Cene hostinga su vrlo različite, pa čak i u istim zemljama, i zavise od velikog broja parametara: veličina prostora na disku, broj subdomena, broj baza podataka, broj mail adresa, podršku i sl. Ove cene su poslednjih godina vidno niže nego ranije, što je posledica pojavljivanja većeg broja provajdera. Tako se danas prosečan hosting za sajt može naći i za desetak EUR-a, što je cena za godinu dana korišćenja.

1.12. Domen

Domen (*Domain name*) je jedinstvena adresa koja se koristi da bi se identifikovao računar tj. mrežni uređaj na Internetu. Domen se formira u skladu sa pravilima DNS-a i predstavlja jedan deo DNS-a. *Domain name* se najviše koristi za potrebe adresiranja i predstavlja *Internet Protocol* resurs. U tom smislu to može biti računar korisnika, web server ili web sajt na nekom web serveru.

Domain identifikator definiše tip institucije ili organizacije (.biz, .com, .edu, .org, .net) kojoj neki web sajt pripada. Na taj način, korisnik je, i pre pregledanja sadržaja sajta, inicijalno informisan o tipu, vlasniku ili mogućem sadržaju sajta.

Svaka država ima zvanično telo koje se jedinstveno bavi registracijom domena, i vodi računa da sve Internet adrese u toj državi budu jedinstvene i pripadaju odgovarajućim *Domain identifikatorima*. Ovo se realizuje na bazi dokumentacije vlasničke i osnivačke strukture kao i zvanično dobijenoj šifri delatnosti za obavljanje pojedinih poslova za dato preduzeće ili pojedinca. U Srbiji, te poslove obavlja Registar nacionalnog Internet domena Srbije (<http://www.rnids.rs>). Korisnik sam može na sajtu RNIDS-a proveriti zauzetost pojedinih domena, videti različite vrste statistika zauzetosti i sl. i na kraju registrovati svoj domen za sajt. Često u realnim uslovima, ove poslove administracije besplatno obavlja Internet provajder kod koga se zakupljuje hosting.

Svaka država ima svoje nacionalne domene, dok na svetskom nivou postoje internacionalni domeni.

Nacionalni domeni u Republici Srbiji su: .rs, .co.rs, .org.rs, .edu.rs, .in.rs, .ac.rs, .gov.rs. Bitno je naglasiti da je Rusija postala prva država koja je registrovala domen i na ćirilici „рф“ i to 12.5.2010. a da je odmah iza toga Srbija bila druga zemlja koja je registrovala svoj ćirilični domen „срб“, i to 3.5.2011. Danas imamo nekoliko ćiriličnih nacionalnih domena koji su u upotrebi: .срб, .np.срб, .opr.срб, .obp.срб, .od.срб, .ak.срб, .ynp.срб. Na ovaj način, pored domena .rs i .срб, svi ostali se nazivaju poddomeni, jer pripadaju jednom od ova dva nacionalna domena.

Tabela 1.2. Nazivi srpskih poddomena (izvor RNIDS).

.rs domen	.срб domen	Obrazloženje
.co.rs	.np.срб	Privreda, preduzetnik, preduzeće
.org.rs	.opr.срб	Organizacija
.edu.rs	.obp.срб	Obrazovanje
.in.rs	.od.срб	Predlog od, fizička lica
.gov.rs	.ynp.срб	Organi uprave
.ac.rs	.ak.срб	Akadska mreža

Internationalnih domena ima relativno puno i imaju vrlo širok spektar u odnosu na šta su grupisani. Najpoznatiji internacionalni domeni su: .com, .net, .org, .info,

.biz, .co, .me, .tv, .cc, .mobi, .name, .tel, .at, .be,... Neki od internacionalnih domena i njihova značenja su:

.aero – vazdušni prevoz

.biz – poslovne firme

.com – poslovne organizacije

.coop – kooperacije

.gov – vladine organizacije

.info – pružaoci informacija

.mil – vojska

.museum – muzeji

.name – pojedinci

.net – mreže računara

.org – neprofitne organizacije i fondacije.

U operativnom smislu, nakon kreiranja web sajta, programer treba da ima domen i hosting, da bi njegov sajt bio javno dostupan na Internetu. U zavisnosti od izabranog domena, cena može biti i besplatna, ili se kretati od 10 do 100 EUR-a na godišnjem nivou.

Korisnik može zakupiti određenu adresu u nekom domenu, pod uslovom da ispunjava uslove, u slučaju određenog tipa domena (.org, .edu, ...) i/ili da je domen slobodan. Zakupljivanjem adrese sa domenom (npr. *proba1.rs*) korisnik ima pravo da je koristi narednih godinu dana, i nakon toga treba da obnovi zakup svoje adrese sa domenom, ili se ona, nakon godinu dana, vraća u spisak dostupnih i može je zakupiti neko drugi, bez prava preče kupovine starog vlasnika. Zakupom adrese na jednom domenu, ne rezervišu se niti automatski zakupljuju adrese na drugim domenima (npr. *proba1.net*, *proba1.com*, ...).

Ukoliko se to želi, za svaki od ovih domena, se mora izvršiti posebna registracija i plaćanje.

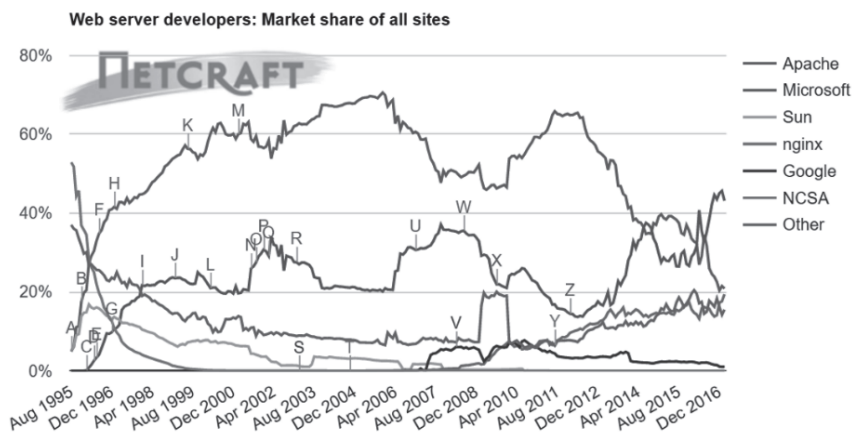
Dobijanjem adrese sa domenom, programer može koristiti i usluge subdomena, u obimu koliko je hostingom odobreno. Subdomen ili poddomen, je produbljivanje hijerarhije, u odnosu na primarnu adresu na konkretnom domenu. Tako na primer, za sajt *proba1.rs*, unutar istog ili drugog hostinga, može da podrži i još nekoliko drugih nezavisnih i nekorelisanih sajtova istog vlasnika hostinga. Najčešće imena poddomena definiše vlasnik, i ona se najčešće ne naplaćuju dodatno. Primer nekih poddomena za sajt *proba1.rs* mogao bi biti *en.proba1.rs* ili *www.proba.rs* ili *mail.proba.rs*.

Treba naglasiti da je i *www* poddomen nekog sajta, i da je preporuka da svaki sajt ima verziju adresa sa i bez *www*.

1.13. Web server

Web server je specijalizovani softver koji se fizički nalazi instaliran na nekom računaru, koga takođe žargonski zovemo web server. Primarna uloga web servera je da se na njemu hostuju web sajtovi ili web aplikacije i na taj način postanu dostupne na Internetu. Web server se instalira na računaru koji mora da ima statičku (nepromenljivu) IP adresu, da bi bio jedinstveno prepoznatljiv na Internetu. Primarna karakteristika web servera je da ume da prihvata *http request*-ove i da generiše *http response*-e. Glavna funkcionalnost web servera je da po dobijanju *http request*-a ispravno protumači zahtev, pronade željeni resurs, procesira ga i pripremi za slanje u formi *http response*-a. Na primer, to podrazumeva da kada browser kroz *http request* zahteva sadržaj određene web strane, web server treba da tu stranicu pronade, pripremi za slanje i pošalje, a nakon toga isporuči sve druge fajlove koje ta strana sadrži a koje će browser zahtevati (npr. slike, audio fajlove i sl.), da bi korisnik u browser-u video celokupnu web stranu.

Trenutno tri najpopularnija web servera su: Apache, Microsoft -IIS i Igor Sysoev - nginx. Prema podacima sa sajta <http://news.netcraft.com> zastupljenost web servera, za aktivne web sajtove, na početku 2014. godine je: Apache (54,5%), Microsoft (11,61%) IIS, Nginx (11,97%) dok je Google-ov GWS zastupljen sa 8.54%. Prema podacima sa sajta <http://w3techs.com/>, na početku 2015. zastupljenost je: Apache (58,7%), Nginx (23%), Microsoft (13,2%) IIS, dok je Google-ov server zastupljen sa 1.3%.



Slika 1.11. Zastupljenost web servera po godinama. (izvor <http://news.netcraft.com>)

Jedna od ključnih razlika između web servera su uslovi korišćenja, cene i dostupnost pojedinih provajdera koji nude korišćenje pojedinih web servera. Trenutno najpopularniji Apache je besplatan, pod određenim licencama, dok je recimo Microsoft IIS, kao vlasnički kod, među skupljima i manje zastupljenim u ponudi provajdera, jer najčešće mora imati podršku i drugih licenciranih softvera (operativnog sistema, baze podataka, i sl.).

Obzirom na pojavu novih web servera i kvalitet usluge istih, na slici 1.11. je ilustrovana zastupljenost pojedinih web servera za nekoliko prethodnih godina, na nivou celog sveta, što ukazuje na velike fluktuacije na tržištu.

1.14. Web pretraživač

Web pretraživač (*Web search engine*) je namenjen pretrazi sadržaja na World Wide Web i FTP serverima. Tri najpopularnija web pretraživača su: Google, Yahoo i Bing. Zastupljenost ovih pretraživača se drastično promenila u poslednjih 10-ak godina, tako da sada Google zauzima neprikosnoveno prvo mesto. Prema podacima sa sajta <http://gs.statcounter.com/>, za mart 2017. godine njihova zastupljenost je:

Google (92.58%), Bing (2.86%), Yahoo (2.14%), Baidu (0.98%), i YANDEX RU (0.37%).

Sve što je do sada objašnjeno kao pojam je ukazivalo na softver, hardver ili definiciju. Iako se i za web pretraživač može reći slično, treba imati na umu da je reč o veoma velikom broju hardvera, često širom sveta, servera, baza podataka, ljudi i slično, što je neuporedivo kompleksnije od bilo čega ranije pomenutog.

U tom smislu, engleski termin *Web search engine* mnogo bolje opisuje suštinu tj. da se radi o “mašineriji”, nego srpski koji često ljude pogrešno navede na pomisao da je web pretraživač recimo klasičan softver kao što je Internet Explorer ili Chrome.

Formalno, princip rada web pretraživača se može opisati kroz tri faze:

1. Web crawling (*spider*)
2. Indeksiranje
3. Pretraga

Iako je princip rada web pretraživača uvek po malo „misteriozno nepoznat“ osnovna logika rada je relativno poznata. Naime, web pretraživač ima svoje *spider*-e, specijalizovane softvere, koji su zaduženi da „šetaju i istražuju“ Internet. Ova faza rada i prikupljanja informacija naziva se **crawling**. Kada *spider* dođe na određeni web sajt, on prvo analizira početnu (*index*) stranicu sajta. Programer u principu ne zna kada će *spider* doći, koliko će se zadržati, koje informacije će analizirati, koje druge stranice će posetiti i šta će od tih informacija

vratiti npr. Google-u, ako je to njegov *spider*. Nema tačnog recepta da se *spider* privuče da dođe na sajt ili da se zna da li je i kada bio, da li se i koliko puta vratio i sl.

Međutim, deljenjem informacija velikog broja web programera koji na empirijskim primerima utvrđuju šta su unutar stranica napisali od koda i kako je to u krajnjoj fazi verifikovano od strane web pretraživača, veliki broj neformalnih zaključaka se može doneti u vezi sa radom *spider*-a.

Dolaskom *spider*-a, analizira se sajt počev od index strane, pa preko linkova, i do nekih od ostalih stranica (ako *spider* zaključi da za to ima potrebe). *Spider* analizira kompletan kod, i nosi informacije o imenu stranice, sadržaju, tipu informacija, slikama na strani, njihovim imenima, naslovima i sadržaju tekstova, velikom broju informacija koje programer piše za potrebe web servera ili namenski za web pretraživač (skriveni tagovi, atributi tagova, meta podaci, podaci u fajlu robots.txt, itd.). Cilj *spider*-a je da sakupi što više podataka kojima će „razumeti i opisati“ kvalitet, sadržaj, pouzdanost i relevantnost neke web strane i celog sajta u odnosu na temu koju taj sajt želi da promoviše, slika 1.12.



Slika 1.12. Grafički prikaz rada spider-a.

Sve ove informacije, *spider* vraća web pretraživaču. Zna se da se *spider* retko ili nikada ne vraća na sajt koji je procenjen kao „loš“, čime se taj sajt u principu „izbacuje“ iz trke za dobro pozicioniranje u kasnijim rezultatima pretrage koje realizuje neki korisnik.

Nakon *crawling* faze, kada *spider* vrati informacije pretraživaču, ovi podaci se na neki način analiziraju, filtriraju i na bazi njih se u bazu podataka web pretraživača upisuje URL adresa sajta, informacije o kvalitetu strane, informacije o strani (kratak opis strane koju je definisao programer), konkretna ocena te strane (*PageRank*), ključne reči koje se smatraju relevantnim za tu stranicu (u zavisnosti šta je programer definisao da su ključne reči i onoga šta se nalazi kao sadržaj te strane) i sl. što zavisi od konkretnog web pretraživača. Nama je i ovaj skup informacija relativno nepoznat ali se ovo pretpostavlja na bazi empirijskih provera. Ova faza kada web pretraživač memoriše informacije o određenom sajtu i ima ih definisane u svojoj bazi podataka se naziva **indeksiranje**.

■ **Napomena:** Žargonski, indeksiranje predstavlja „pravi početak života“ nekog sajta, jer tada on postaje potpuno dostupan svima koji pretražuju Internet, a ne samo našim rođacima kojima telefonom javimo URL adresu sajta, pa tako saznaju za njega.

U užem smislu, web pretraživač ne pretražuje Internet, nego informacije koje su nakon dobijanja od strane *spider*-a i analize upisane u bazu podataka web pretraživača. Iz tog razloga, svima je cilj da web pretraživač što bolje oceni dati sajt i da ga korisniku u postupku pretrage prikaže na što višem mestu.

Nakon indeksiranja, korisnik koji dođe na sajt web pretraživača, npr. *www.google.com*, dobije relativno prostu web formu, koju koristi za pretragu. Unosom određenih reči koje su mu od interesa, i klikom na taster *Search*, ove reči se prosleđuju pretraživaču. Tada nastupa treća funkcija web pretraživača – **pretraga**. Na bazi unetih reči od strane korisnika, sa jedne strane, i informacija u bazi podataka, dobijenih u fazi indeksiranja, web pretraživač pronalazi sve one sajtove koje smatra da su relevantni za unete ključne reči. Sve ove sajtove pretraživač sortira, u odnosu na ranije definisani kvalitet, prioritet i relevantnost tih stranica, i kao rezultat pretrage prikazuje korisniku. Ove informacije su naravno sortirane počev od one koja je procenjena kao najrelevantnija, da bi se korisniku što brže ponudilo ono što je on želeo, slika 1.13.



Slika 1.13. Grafički prikaz postupka pretrage u web pretraživačima.

Cilj svakog web programera je da se sajt što bolje „pozicionira“ što znači da se za određene ključne reči baš taj sajt pojavi među prvih n sajtova koje će pretraživač ponuditi korisniku. Ovo je posebno bitno kada se zna da statistika ukazuje na to da najveći broj korisnika gleda rezultate pretrage samo na prvoj i drugoj strani, dok drastično opada broj onih koji gledaju dalje stranice. Tada korisnik najčešće unosi nove ključne reči i započinje novu pretragu. Tako se

smatra da ako imamo sajt koji je za određene ključne reči na npr. 189 strani koje vrati pretraživač, da taj sajt „skoro da ne postoji“ i da je „nevidljiv“.

Poseban skup tehnika, SEO (*Search engine optimization*), se koristi sa ciljem da se sajt što bolje „pripremi“ za dolazak *spider*-a i za kasnije „ostavljanje dobrog utiska“ u fazi indeksiranja, čime se takav sajt kandiduje da bude na prvim stranicama pretrage, što je svim programerima cilj.

1.15. Oblast delovanja web dizajnera

Čak i ako se strogo definišu programski jezici potrebni za izradu web sajtova, pa u skladu sa tim i tipovi sajtova, ostaje vrlo nedefinisano šta tačno podrazumeva posao web dizajnera. Ovo je posledica vrlo intenzivnog razvoja u ovoj oblasti i velikih fluktuacija opisa poslova koje se pridružuju web dizajneru.

Inicijalno, web dizajn se više tretirao kao umetnička disciplina, sa ciljem da dizajner definiše koji deo sajta se gde nalazi, koje pozadinske boje, slike, veličine teksta i slično će gde i kako biti zastupljene. Vremenom se ovaj posao sa idejnog spustio na nivo operativnog, pa je dizajner to trebao većim delom da pripremi u formi slika, slice-ova, fontova i slično u nekom softveru za grafiku (npr. Photoshop), pa da sve to dostavi programeru. Programer je onda, na bazi tih materijala, formirao kod kojim se to generiše i dobija konačni oblik sajta. Danas je ovaj celokupan postupak dodeljen web dizajneru. Od njega se očekuje da definiše strukturu sajta, kompletnu grafiku, izgled, i sve to integriše u HTML i CSS kod, da bi dobio template web stranice. Na bazi ovog template-a, dalje se formiraju sve druge web stranice sajta tako što se u njih ubacuje konkretan korisnički sadržaj, a osnovna struktura i dizajnerska forma se ne menjaju.

Obzirom da je primera HTML-a i CSS-a, relativno jednostavna, sve češće ove poslove web dizajnera obavlja lice koje je u umetničkoj oblasti i ne mora nužno da bude programer. Uloga programera, od sada web programera, je da na bazi definisanog template-a dalje dodaje programski kod, klijentski ili serverski, kojim se dodatno programiraju sve definisane funkcionalnosti web sajta.

U ovoj knjizi, u njenom prvom delu, obrađuje se sve što podrazumeva poslove web dizajnera, dok se poslovi web programera obrađuju u drugom delu knjige.

II HTML

HyperText Markup Language

POJAVA web-a kakvog danas poznajemo, nije se desila tako daleko kao što je to slučaj sa drugim oblastima koje se najčešće izučavaju u toku školovanja. Pre samo više od 25 godina, web ne samo da nije postojao u ovakvom obliku nego nije postojao uopšte. Tek nakon 1990. godine počinje se sa prvim aktivnostima na ovu temu, bez jasnog cilja ka čemu se tačno ide i šta će na kraju biti termin „web“, kakvim ga mi danas poznajemo. Formalno, tek na prelazu između 1991. ka 1992. godini, pojavljuje se prva verzija HTML-a. Tek tada su se i zvanično stekli uslovi da se ovaj jezik značajno primenjuje i razvija, a samim tim i web.

Međutim, HTML je samo faza u razvoju i istraživanju šire grupe jezika, *markap jezika*, pa ćemo se kratko osvrnuti na sam početak.

2.1. ML - Markup Language

Markup languages (markap jezici) su kreirani sa ciljem da se standardizuje način kreiranja, opisa, obrade i prikaza teksta. Na taj način dobijen je tekstualni fajl koji će se koristiti kao standardizovani ulaz/izlaz iz pojedinih aplikacija, softvera, uređaja, ... i biti prilagođen što široj grupi medijuma (različitim tipovima hardvera, softverima, operativnih sistema, ...). Obzirom da je cilj bio vrlo ambiciozan, a evidentno postoji velika razlika u tipovima, vrstama i načinima rada različitih medijuma, markap jezici moraju imati vrlo jednostavnu sintaksu i strukturu da bi bili „svima razumljivi“, pa se krenulo od termina GML (*Generalized Markup Language*).

■ **Napomena:** Neki izvori navode da je nastanak markap jezika posledica redukcije troškova koji su se pojavili u IBM-u u toku šezdesetih godina, a bili su posledica velike količine tehničke dokumentacije koja je većim delom bila slična za pojedine proizvode. Zato su počeli da tekstualnim oznakama označavaju (opisuju) pojedine podatke, dokumente ili delove istih. Ovo je omogućilo bržu pretragu i dohvaćanje određenog tipa sadržaja (npr. poglavlja, tabele, zaglavlja, zajedničke delove, posebno bitne celine i sl.). Tako je GML (*Generalized Markup Language*) dobio prve konkretne primene.

Kako to nisu programski jezici, pa ne mogu postojati promenljive, konstante, petlje i slično, za osnovnu gradivnu strukturu markap jezika je izabran *tag*. Na engleskom jeziku, reč *tag* može imati i značenja: etiketa i podsetnica, što jasno ukazuje na ideju prvobitne potrebe za označavanjem. Tag predstavlja jedinstvenu oznaku sa određenim značenjem koje se odnosi na neki deo teksta. Na taj način jedan markap jezik ima veliki skup tagova kojima se može opisati ili kreirati željeni sadržaj.

Markup jezici se pišu kao običan tekst, pa su tako i tagovi određeni tekstualni simboli. Tagovi su karakteristični za sve markap jezike, ali svi jezici među sobom imaju velike razlike. U tom smislu, prvo treba razumeti potrebu za tim jezicima, njihovu namenu, pa se onda operativno posvetiti njihovom učenju i primeni.

Postoje dva tipa markap jezika:

1. Specific Markup Languages i
2. Generalized Markup Languages.

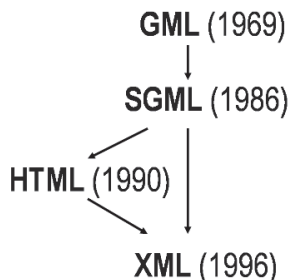
Specific Markup Languages se razvijaju za posebne (vrlo specifične) namene, aplikacije ili uređaje i za druge potrebe najčešće ne mogu biti korišćeni. Ovakvi jezici ne moraju biti kompatibilni sa drugima jer se koriste za specifične potrebe, ali moraju biti standardizovani. Standardizacija je bitna da bi aplikacije ili uređaji za koje su namenjeni znali kako da ih “razumeju”.

Generalized Markup Languages služe da istovremeno opišu strukturu i značenje teksta u fajlu, ali po pravilu ne definišu kako će se tekst koristiti i na koji način će se prikazivati. Ovi jezici nisu namenjeni za specifične aplikacije nego su dovoljno uopšteni da se mogu koristiti u velikom broju različitih aplikacija. Njihovo teorijsko razvijanje počelo je još 1969. godine. Epilog događaja je da su se iz SGML-a razvila dva različita jezika: HTML i XML.

HTML je nastao kada je Tim Berneres Lee izabrao mali skup oznaka iz SGML skupa koji je korišćen na CERN-u sa ciljem formatiranja dokumenata. Tako je softver koji je ovo znao da protumači nazvan HTML pregledač. Sa druge strane, ovaj konačni skup oznaka (tagova) je bio toliko mali da nije omogućavao dovoljno fleksibilan rad za različite specifične potrebe, dok je SGML bio preglomazan. To je uzrokovalo da se, pored HTML-a, zahteva kreiranje opet novog jezika sa sledećim uslovima:

- ✓ Mora biti primenljiv na Internetu kao i HTML
- ✓ Mora podržati veliki broj različitih specifičnih primena
- ✓ Treba da bude lak, jednostavan i razumljiv i za čoveka i za mašinu
- ✓ Struktura i dizajn moraju biti precizni i definisani.

Sve ovo je formalizovano u periodu od 1996. do 1998. godine kada *World Wide Web Consortium* objavljuje prve **XML** preporuke. Na taj način, danas možemo da konstatujemo da je razvoj ovih markap jezika išao tokom kao na slici 2.1. dok se u međuvremenu pojavio veoma veliki broj drugih specifičnih markap jezika koji su u svakodnevnoj upotrebi, ali u manjem obimu od navedenih.



Slika 2.1. Istorijski razvoj markap jezika.

Rezime!

HTML (*HyperText Markup Language*) nastao je krajem 1990. godine kao naslednik SGML-a. Predstavlja vrlo striktnu formu SGML-a i ima tačno definisan broj tj. skup tagova koji su na raspolaganju programeru. Svaki tag ima striktnu sintaksu i definisan način delovanja na podatke. Pored toga većina tagova ima i grafičku interpretaciju, pa je HTML zamišljen sa ciljem da grafički uredi i strukturiira podatke.

XML (*Extensible Markup Language*) nastao je nakon 1996. godine a standardizovan tek 2008. godine (peto izdanje verzije 1.0). Za razliku od HTMLa, XML nije zamišljen i inicijalno ne podržava grafički prikaz podataka. Njegov primarni cilj je da omogući opisivanje podataka, njihovo struktuiranje i da omogući proširivost u broju tagova. Na taj način XML dokument može opisati samog sebe, organizaciono urediti veliki broj podataka (npr. kao baza podataka) i daje mogućnost da se definiše proizvoljno veliki broj tagova sa proizvoljnim imenima.

Zajedničko i za HTML i XML je da su oni čisto tekstualni fajlovi, laki za učenje, čitanje i razumevanje a podjednako razumljivi i čoveku i mašini, što je i bio cilj markap jezika.

2.2. Istorijski razvoj HTML-a

Prva verzija HTML-a, označena sa HTML 1, pojavila se tek 1991. godine i trajala samo još u toku 1992. godine. Već tada je bilo jasno da je to vrlo „siromašna“ verzija jezika sa mnogo manjkavosti. Na primer, nije se mogao definisati željeni font stranice, pozadinska boja, kreirati tabele, forme, itd.

Iz tog razloga, već 1993. godine nastaje HTML 2, koji takođe traje samo dve godine. U ovoj verziji pojavile su se „primitivne“ forme (neka polja su mogla da se kreiraju, ali nije moglo da se izvrši *submit*-ovanje forme), mogla je da se podesi pozadinska boja ili slika, i mogle su da se kreiraju tabele. Napredak koji je kreiran za te dve godine od verzije HTML 1 se smatra vrlo velikim, ali svakako nedovoljnim.

U naredne dve godine (1995-1996.), pojavljuje se HTML 3.2. Interesantno je da ne postoji verzija HTML 3. Ovo je posledica toga što *World Wide Web Consortium* - **W3C** (Internacionalna organizacija za standardizaciju) nije na vreme dobila zahtev za standardizaciju. U ovoj verziji podržan je rad sa stilovima (prvi oblik CSS-a) i povećan broj atributa kojima se lakše uređuje stranica. Rad sa *frame*-ovima još nije bio podržan, iako su ga tadašnji browser-i već podržavali. Sa druge strane CSS je bio podržan u verziji HTML 3.2 ali ne i u tadašnjim browser-ima. Treba naglasiti da se u tom periodu, pored početka razvoja CSS 1, razvija i klijentski skript jezik *Java Script*.

U toku 1997. godine, zvanično se pojavljuje i HTML 4. Ova verzija HTML-a je imala vrlo dobre karakteristike, jer je realizovana sa dosta promena i poboljšanja u odnosu na ranije verzije, pa je uz svoje manje modifikacije trajala do 2009. godine. Ključna stvar je što je sve formatiranje pojedinih elemenata i cele strane „izmešteno“ u CSS. U ovom periodu kompletno se standardizuje i CSS 2. Paralelno sa tim pojavljuje se i XHTML 1.0 kojim se omogućava razmena i prikaz podataka iz XML-a. Razvija se DOM i DOM 2 čime se omogućava interakcija sa objektima u HTML-u, XHTML-u i XML dokumentima. Dodatna

novina je što se objektima u DOM strukturi počinje pristupati korišćenjem metoda. U ovom vrlo produktivnom periodu, 2003. godine, pojavljuje se i Web 2.0. a već 2005. godine i AJAX. Sve ove tehnologije kreiraju dinamički i interaktivni Web kakvim ga mi danas poznajemo. Kvalitet HTML 4 je time bio dodatno potvrđen obzirom da je mogao da podrži i implementira sve promene u drugim jezicima i tehnologijama.

Periodu od 12 godina, koliko je HTML 4 bio aktivno u upotrebi, logično je morao doći kraj obzirom na popularnost Web-a i sve veće korisničke zahteve. Tako se 2009. godine pojavljuje HTML 5 i CSS 3, koji se smatraju revolucijom u smislu uvedenih promena i načina rada. O ovome će detaljnije biti reči u Poglavlju V i Poglavlju VI.

2.3. Princip funkcionisanja web stranice

Web strana je običan tekstualni fajl koji je napisan primenom jezika HTML i CSS. Pored ovoga, web strana može imati i nadogradnju u smislu implementacije klijentskih ili serverskih jezika. Iako su i klijentski i serverski jezici kategorizovani kao skript ili programski jezici, koji su višeg ranga od markap jezika, oni ne mogu da realizuju ni jednu stvar koju HTML može. Na taj način, iako žargonski kategorizovan kao jezik najnižeg prioriteta, prost i lak, HTML je i dalje ključni za prikaz, organizaciju sadržaja i formatiranje web strane.

Ekstenzija HTML stranice može biti *html* ili *htm*. Snimanjem fajla sa ovom ekstenzijom, operativni sistem prepoznaje koji podrazumevani softver treba da se aktivira kada se klikne na ovakav tip fajla. U slučaju, fajlova koji su *html* ili *htm*, podrazumevani softver za njihovo otvaranje je podrazumevani browser na tom operativnom sistemu.

HTML je jezik koji ima tačno definisan skup, tj. imena, tagova koja su nam na raspolaganju i pomoću njih se kreira grafika svakog sajta. Različitost u grafičkom izgledu različitih sajtova je posledica kreativnosti, dizajna, broja i redosleda primene pojedinih HTML tagova.

Ključno je razumeti da HTML - *HyperText Markup Language* predstavlja „prost“ tekst. Međutim ovaj tekst nije kao tekst koji čitamo u dnevnim novinama. Iz tog razloga u nazivu HTML nalazi se reč *Hyper*. Ona ukazuje da ovaj tekst ima viši nivo značenja, a ne samo sadržaj za korisnika. Taj viši nivo se odnosi na način prikaza sadržaja i dodatne informacije za browser i server, a za šta su zaduženi tagovi. Pored toga, *HyperText* ukazuje, da se za razliku od običnog teksta, mogu koristiti i *HyperLink*-ovi. Linkovi predstavljaju mogućnost da se klikom miša na neki tekst, oblast ili sliku, od servera može tražiti potpuno drugi tekstualni fajl (drugi HTML fajl) čime se dobija prikaz potpuno nove web stranice.

U skladu sa opisom u Poglavlju 1.9., prikaz web strane kod korisnika, započinje unosom URL adrese u browser, kada se klikom na Enter, kreira *HttpRequest*-a.

Kao odgovor servera, na prosleđeni *HttpRequest*, HTTP protokolom se šalje tekstualni, HTML, fajl koji se nalazi na web serveru, i na kraju prikazuje u browser-u.

Kada browser dobije ovaj HTML kod, browser počinje da čita jednu po jednu liniju koda, i da u skladu sa imenima, tj. značenjem, tagova realizuje ono što su njihove funkcije. Na taj način, jedan od prvih koraka može biti prikaz sadržaja u zaglavlju browser-a, definisanje dimenzija strane, strukture, zahtevi da server isporuči propratni CSS fajl, u kome se bliže definiše način prikaza podataka u tom HTML fajlu, realizacija stilova i na kraju prikaz sadržaja web strane. Sve ove informacije se nalaze unutar HTML strane i propratnog CSS fajla. Međutim, unutar HTML stranice se inicijalno nalazi samo tekstualni sadržaj, dok se svi drugi sadržaji (slike, video, audio fajlovi i sl.) nalaze na web serveru. HTML kodom se definiše na kom mestu i sa kojim dimenzijama će se prikazati određeni fajl (npr. slika), kao i putanju do tog fajla na serveru (gde se taj fajl fizički nalazi).

Ceo postupak komunikacije browser-a sa serverom, u zavisnosti od broja fajlova koji se pojavljuju unutar HTML koda, se realizuje HTTP protokolom bez intervencije korisnika. Zbog ovog postupka koji se realizuje u više koraka, u slučaju da se koristi sporija Internet veza, učitavanje strane ili nekih njenih delova, ne deluje da se realizuje trenutno, što i jeste realno stanje stvari.

Kada se sadržaj HTML strane učitava u browser, u skladu sa napisanim kodom, browser će sadržaj te HTML strane grafički vizuelizovati. Korisniku se tako web strana prikazuje grafički dok je u pozadini originalni HTML kod. Za svaku web stranu koja se prikaže u browser-u može se videti i njen originalni HTML kod. Ovaj kod se nikako ne može sakriti od strane programera, već se prikazuje u originalnom obliku kao što je na web serveru.

U različitim browser-ima, se prikaz sadržaja HTML strane dobija na različite načine, ali za najveći broj njih se dobija desnim klikom miša na prikazani sadržaj sajta i izbor opcije *View page source*. Klik mišem mora biti van slike ili animiranog dela sajta.

Drugi način, da se ovaj posao obavi, je da se u meniju browser-a izabere odgovarajuća opcija (npr. u *Internet Explorer*-u treba izabrati *View/Source*).

Izborom opcije *View page source*, može se videti ceo HTML kod, tj. originalni izgled web strane. Ovaj kod u zavisnosti od kompleksnosti, obima strane, sadržaja, dizajna i strukture može biti manji ili veći.

Na ovaj način HTML kod stranice je kompletno dostupan i krajnjem korisniku.

Ovaj način komunikacije ne zavisi od tipa sajta, korišćenog programskog jezika, operativnog sistema i sl. Jedina razlika je što se kôd koji se dobija od web servera isporučuje u originalu, ako je to klijentski jezik, i kao takav je vidljiv korisniku, tj. preveden u HTML kod, u slučaju serverskih jezika, kada se originalni serverski kod ne može videti.

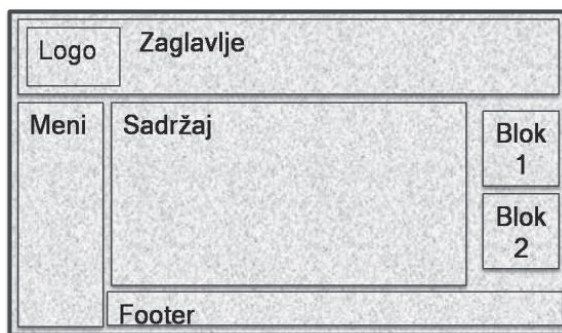
2.4. Početak izrade web sajta

Iako je HTML jezik namenjen za kreiranje web strana, pa samim tim i sajta, postupak izrade web sajta ne počinje od HTML-a. Pre svega potrebno je:

1. Osmisliti organizaciju web stranica (blokove i regione),
2. Definirati dizajn pojedinačnih delova strane,
3. Osmisliti pojedinačne stranice sajta (broj, imena, organizaciju),
4. Predvideti tip navigacije među stranicama i
5. Definirati sadržaje pojedinačnih stranica.

Ceo ovaj postupak je, za početnike, najbolje prvo uraditi na papiru, pa tek tada krenuti sa kreiranjem konkretnog HTML koda. Ove faze se mogu shvatiti kao projektni zadaci, kojima se opisuje kako će se koristiti tj. pisati HTML kod. U suprotnom, nemoguće je praviti sajt ako ne znamo šta se želi postići kao cilj.

1. Definisanje organizacione strukture web strane je prvi korak u pravljenju *template*-a, tj. bazične stranice na kojoj će se menjati samo sadržaj, dok struktura, zaglavlje, meni, itd. može ostati isti na svim stranicama.



Slika 2.2. Primer grafičke organizacije web stranice.

Ovaj postupak podrazumeva da se tačno definišu oblasti/blokovi (*divovi*) u kojima će biti pojedini delovi sajta (npr. zaglavlje, logo, meni, sadržaj, baneri, itd.). Nakon toga, blokovi (*divovi*) se po potrebi grupišu u veće *divove* tj. regione, zbog stabilnosti prikaza sadržaja u browser-u.

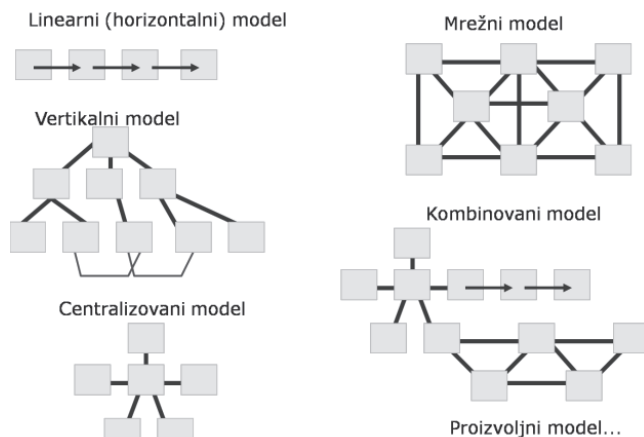
2. Dizajn pojedinačnih delova web strane podrazumeva:

- a. Definisanje pozadine stranice (boje ili slike),
- b. Definisanje pozadine pojedinačnih blokova (*divova*),
- c. Font teksta
- d. Boju teksta
- e. Stilove za prikaz sadržaja
- f. Strukturu podsadržaja, ...

3. Kreiranje pojedinačnih stranica sajta Pre početka izrade, mora se znati koje stranice sajta treba da postoje, čemu će služiti, sa kojim funkcionalnostima i kako se one zovu. Pored same ideje o konceptu sajta i rasporedu sadržaja po stranicama, ovo je kasnije potrebno i za sam HTML kod potreban za realizaciju navigacije.

Za svaku stranicu potrebno je definisati njenu organizaciju. Nije dobro imati mnogo različitih organizacija web stranica unutar web sajta. Preporuka je imati maksimalno dva različita dizajna (za prvu i ostale strane sajta). Takođe se ne preporučuje više od dva fonta i četiri boje po strani.

4. Navigacija: Kada su poznate sve stranice, treba da se definiše navigacija. Navigacija podrazumeva način povezivanja stranica pomoću linkova. Navigacija se najčešće realizuje kroz vertikalnu ili horizontalnu navigaciju na sajtu, i žargonski se naziva meni, ali može biti realizovana i kao link koji je neka reč, slika ili oblast u samom sadržaju strane. Na slici 2.3. prikazani su neki od standardnih modela navigacije.



Slika 2.3. Tipovi organizacije navigacije među web stranicama.

5. Sadržaji pojedinačnih stranica: Kada su definisane organizacije za svaku od web strana, mora se definisati i njihov sadržaj. To se radi pre pisanja samog koda, koji treba da se piše na bazi sadržaja konkretne strane. Sadržaj je najbitniji deo sajta. Primarno je bitan za korisnika, ali i za SEO tj. web pretraživače, pa treba pažljivo birati reči, strukture, slike, attribute, tagove i slično. Ako se ovo ne uradi unapred, planski, nego u toku pisanja HTML koda, takav sajt i stranicu je mnogo teško napraviti SEO optimalnom.

Ako su svi ovi postupci, u planiranju izrade sajta realizovani, može se početi sa pisanjem konkretnog HTML i CSS koda, da bi se sajt sa idejnog i papirnog oblika napravio u elektronskom obliku.

2.5. Struktura jezika HTML

Kao što je ranije napomenuto, osnovni gradivni element u HTML-u je tag. Tagovi mogu biti “upareni” ili “neupareni” tj. “zatvarajući” i “samozatvarajući”. Ova podela je izvršena u odnosu na oznake kojima se tagovi otvaraju odnosno zatvaraju. Upareni tagovi imaju oznaku za otvaranje i oznaku za zatvaranje taga:

otvaranje taga: `<ime_taga>` tj. zatvaranje taga: `</ime_taga >`. Na primer `<p>` i `</p>`.

Neupareni tagovi imaju samo jednu oznaku, kojom se tag istovremeno otvara i zatvara: `<ime_taga />`. Na primer `<br/>` ili `<img />`.

HTML jezik nije **case sensitive**, što znači da se imena tagova mogu pisati i velikim i malim i kombinovanim veličinama slova (*title = Title = tiTLe*), ali je preporuka da se koriste mala slova.

Pri prikazivanju stranica browser-i se oslanjaju na imena tagova, jer se time precizno definiše kako određeni sadržaj treba prikazati. Svaki tag, opisan svojim imenom, ima precizno definisanu ulogu. Kod uparenih tagova funkcija koju tag obavlja počinje od mesta gde je tag otvoren i završava se mestom gde je zatvoren, a sadržaj na koji tag deluje je sadržaj između otvorene i zatvorene oznake taga. Izvan zatvorenog taga efekat tog taga ne postoji. Tako se definiše i pojam **HTML elementa** koji je skup otvorenog taga, njegovog sadržaja i zatvorenog taga:

`<ime_taga> sadržaj_elementa </ime_taga>`

Tako na primer, tag *i*, kojim se postiže da se tekst prikaže kao *Italic*, i koji deluje na tekst *Pera*, u kodu se piše kao element

`<i>Pera</i>` a u browser-u prikazuje kao *Pera*

Kao što se vidi, oznake za otvaranje i zatvaranje tagova se u browser-u ne prikazuju, ali je njihov efekat na sadržaj elementa (u ovom slučaju *Italic*) postignut. U sklopu opštih pravila, važno je napomenuti da se napisani HTML kod realizuje redosledom kojim je napisan, odozgo na dole, i s leva na desno.

Pored taga, HTML se strukturno opisuje i atributima. Atribut se piše unutar imena taga, i služi da bliže definiše način delovanja taga ili opiše tag. Imena atributa su precizno definisana, kao i to koji atribut se može koristiti sa kojim tagom. U jednom tagu može biti više atributa. Sintaksa zahteva da nakon definisanja imena atributa, postoji operator dodele =, i unutar znakova navoda vrednost atributa. Tako na primer, atribut *class*, sa vrednošću *proba*, u tagu *i*, piše se kao: `<i class="proba">Pera</i>`.

Svaki tag ima tačno definisan spisak atributa koji se mogu pisati unutar njega. Spisak svih atributa, u HTML 4, po pripadnosti tagovima može se detaljno pročitati na adresi: <http://www.w3.org/TR/html4/index/attributes.html>.

Već je naglašeno da svaki tag deluje na sadržaj koji se nalazi unutar tog elementa. Taj sadržaj može biti tekst za korisnika, ali može biti i drugi tag. U slučaju da se kao sadržaj elementa nalazi novi element, kaže se da imamo ugnježdene elemente, i tada svaki od tagova deluje sa svojom funkcijom koju on realizuje. Tekst na koji deluje više ugnježđenih tagova tj. elemenata se podvrgava uticaju svih tih tagova. Treba voditi računa da se kod ugnježđenih elemenata prvo zatvara poslednje otvoreni tag. Tako na primer, ako pored taga *i*, želimo da deluje i tag *b* (boldovanje teksta), pišemo `<i><b>Pera</b></i>` u browser-u prikazuje kao **Pera**. U opšte pojmove o strukturi jezika treba svrstati i komentare. Komentari postoje u svakom programskom jeziku, i služe da pomognu programerima sa opisima koji se napisani. Komentari su delovi web stranice koji se ne interpretiraju, a samim tim i ne prikazuju korisniku. Uloga komentara je lakše snalaženje i navođenje unutar koda. Komentari su skoro obavezni kada više ljudi uređuje sajt, ili kada se koristi neki od skript jezika. Sintaksa komentara je

```
<!-- tekst komentara -->
```

Tekst komentara može biti u jednom ili više redova i može biti običan tekst ili bilo koji kod.

2.6. HTML tagovi

Kao što je naglašeno HTML ima konačno definisan skup tagova koji su na raspolaganju web dizajneru i web programeru. Ovaj skup je definisan standardom za svaku od verzija HTML-a. U tabeli 2.1. dat je listing svih originalnih tagova koji pripadaju HTML 4 i HTML 5. Treba napomenuti da pored originalnih tagova u HTML 5, on podržava i najveći broj tagova iz HTML 4, sem grupe tagova koji su u tabeli 2.1. navedeni u trećoj koloni.

Osnovni tagovi koji opisuju bazičnu strukturu HTML dokumenta tj. web stranice su:

html – HTML stranica uvek počinje tagom `<html>`, a završava se tagom `</html>`, čime se ukazuje na početak i kraj primene HTML jezika.

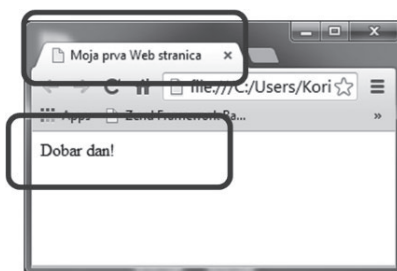
head - Upareni tag `<head>` definiše zaglavlje HTML stranice, u kome se nalaze informacije o samom dokumentu (naslov, opis, ključne reči, ime autora, itd.) i gde se pišu svi skript jezici (npr. Java Script) i veza sa drugim eksternim fajlovima (CSS, JS, ...). Sadržaj koji se piše u tagu `<head>` se najčešće ne prikazuje korisniku, i više je namenjen kao instrukcija za browser.

body – Upareni tag `<body>` je namenjen za pisanje koda koji treba da se prikaže korisniku. Ceo sadržaj stranice, nalazi se u telu dokumenta koje uokviruje element `<body>`. U stranici sme da postoji samo jedan par tagova `<body></body>`.

Skoro svi tagovi koji su namenjeni korisniku, tj. vidljivi u browser-u, nalaze se u tagu `<body>`. Jedan od retkih izuzetaka je regularni tag `<title>` koji se piše u `<head>` tagu, i definiše naslov strane, koji se prikazuje u zaglavlju browser-a. Ovaj izuzetak je zbog toga što prikazani sadržaj nije u telu sajta, već pripada zaglavlju browser-a. Na bazi ova četiri taga može se kreirati prva web strana:

```
<html>
  <head>
    <title> Moja prva Web stranica </title>
  </head>
  <body>
    Dobar dan!
  </body>
</html>
```

Ovaj kod potrebno je otkucati u bilo kom tekstualnom editoru (preporuka *Notepad++*), i snimiti kao *index.html*. Nakon snimanja, operativni sistem će prepoznati da se radi o *html* ekstenziji fajla, i automatski ikonicu promeniti u ikonicu browser-a, koji je na tom računaru izabran kao podrazumevani. Klikom na fajl, automatski će se otvoriti podrazumevani browser, i prikazati sadržaj web strane, slika 2.4.



Slika 2.4. Prikaz grafičke vizuelizacije HTML koda u browser-u

Dobijena web strana nije reprezentativna, ali pokazuje način kako se dolazi do web strane. Dalje učenje i primena HTML koda omogućava da se manipuliše sadržajem strane. Treba napomenuti, da iako u datom kodu nigde ništa eksplicitno nije napomenuto u vezi stila teksta, u browser-u postoje podrazumevane vrednosti koje se automatski primenjuju na sajt i to:

- ✓ Font: TimesNewRoman,
- ✓ Veličina: 12px,
- ✓ Boja teksta: Crna i
- ✓ Boja pozadine: Bela.

Sve ove karakteristike se mogu definisati i promeniti u samom HTML kodu, ali ako se ne navedu, koriste se ove inicijalne definisane od strane browser-a. U nastavku će se postepeno uvoditi pojedini, češće korišćeni tagovi, i dati primeri

za njihovu primenu. Učenjem ostalih tagova sadržaj web stranice se kvalitativno povećava i približava zamišljenom dizajnu i strukturama koje vidamo u svakodnevnim realnim sajtovim.

Bez obzira koliko neki sajt bio manje ili više lep, obiman, atraktivan ili ne, svaki od njih koristi isti konačni skup HTML tagova koji je dat u tabeli 2.1. pa će se sada ti tagovi grupisati, i objasniti, u skladu sa efektima njihovog delovanja.

Tabela 2.1. Spisak HTML4 i HTML5 tagova.

HTML 4			HTML 5	Nisu podržani u HTML 5
<!-->	<h1>-	<script>	<article>	<acronym>
<!DOCTYPE>	<h6>	<select>	<aside>	(<abbr>)
<a>	<hr>	<small>	<audio>	<applet>
<abbr>	<html>		<bdi>	(<object>)
<address>	<i>		<canvas>	<basefont>
<area>	<iframe>	<style>	<datalist>	<big>
		<sub>	<details>	<center>
<base>	<input>	<sup>	<dialog>	<dir>()
<bdo>	<ins>	<table>	<embed>	
<blockquote>	<kbd>	<tbody>	<figcaption>	<frame>
<body>	<label>	<td>	<figure>	<frameset>
 	<legend>	<textarea>	<footer>	<noframes>
<button>		<tfoot>	<header>	<tt>
<caption>	<link>	<th>	<keygen>	
<cite>	<map>	<thead>	<main>	
<code>	<menu>	<title>	<mark>	
<col>	<meta>	<tr>	<menuitem>	
<colgroup>	<noscript>	<u>	<meter>	
<dd>	<object>		<nav>	
		<var>	<output>	
<dfn>	<optgroup>		<progress>	
<div>	<option>		<rp>	
<dl>	<p>		<rt>	
<dt>	<param>		<ruby>	
	<pre>		<section>	
<fieldset>	<q>		<source>	
<form>	<s>		<summary>	
<head>	<samp>		<time>	
			<track>	
			<video>	
			<wbr>	

2.7. Rad sa tekstom u HTML-u

2.7.1. Prikaz srpskih slova

Problem sa srpskim jezikom se javlja zbog toga što je podrazumevani kodni raspored, na tastaturi, u samom tekstualnom editoru i browser-u, engleski. Iz tog razloga, sva slova koja pripadaju i engleskom jeziku regularno se prikazuju, ali postoji problem sa „srpskim slovima“. Ovaj problem rešava se kroz liniju koda kojom se browser-u ukazuje da prilikom prikaza treba da koristi drugi kodni raspored. Za srpsko latinično i ćirilično pismo, koristi se *UTF-8*. Ova linija koda se piše u *head* sekciji, i odnosi se na kompletni tekstualni sadržaj web strane i to u formi *meta* taga kao:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

tj. u celokupnom kodu kao:

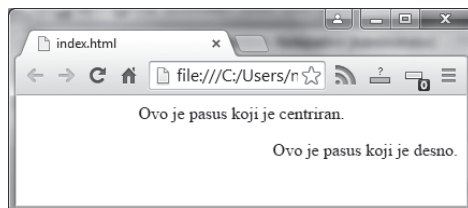
```
<html>
<head>
  <title>Definisanje tipa html-a i prikaza nacionalnih karaktera</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
  Tekst sa srpskim slovima. <br/><br/><!-- Ovo je komentar -->
  Sada se vide i slova Č, Š, Đ, Ć i Ž<br/>
  Кирилица такође: Ч, Ш, Ђ, Ћ, Љ, Њ, Џ и остала.
</body>
</html>
```

Na ovaj način, ukucavanjem srpskih ćiriličnih i latiničnih slova u kod, u browser-u se oni regularno prikazuju.

2.7.2. Rad sa pasusima

Regularni tag `<p>` omogućava kreiranje pasusa, koji prikaz sadržaja uvek započinje u novom redu. U okviru taga `<p>` može se navesti atribut *align*, koji određuje horizontalno poravnanje paragrafa; vrednost ovog atributa može biti jedna od sledećih: *left*, *center*, *right*.

```
<html>
<body>
  <p align="center">Ovo je pasus koji je centriran.</p>
  <p align="right">Ovo je pasus koji je desno.</p>
</body>
</html>
```



Slika 2.5. Prikaz uticaja atributa `align` na sadržaj taga `p`

Vrlo sličan efekat u browser-u kao i tag `<p>` ima i upareni tag `<blockquote>`, sa ciljem da ukaže da je deo teksta unutar njega preuzet od nekog drugog izvora, na šta se ukazuje atributom `cite`.

```
<html>
<body>
  <blockquote cite="http://fakultet.rs/studiranje/obavestenja">
    Svečana dodela diploma održaće se u sredu 15.4. sa početkom u 10h.
  </blockquote>
</body>
</html>
```

2.7.3. Prelazak u novi red i načini ispisa unutar reda

Efekat prelaska teksta u novi red postiže se neuparenim tagom `<br/>`. Svaki ovaj tag ima efekat kao što taster *Enter* ima u npr. *Word*-u.

Podrazumevano, struktura teksta koji se unosi u tekstualni editor se ne zadržava u načinu prikaza u browser-u. Tako prelazak u novi red sa *Enter*, ili upotrebe većeg broj *Space*-ova, ima efekat u editoru ali ne i u browser-u. Upotrebom uparenog taga `<pre>` omogućava se da se u browser-u zadrži identičan odnos pojedinih karaktera kao i u tekstualnom editoru.

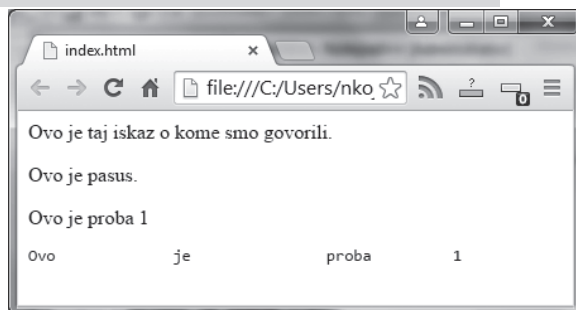
```
<html>
<body>
Ovo je taj

iskaz o kome

smo govorili.<br />

<p>Ovo je pasus.</p>
Ovo      je      proba      1<br />

<pre>Ovo      je      proba      1 </pre>
</body></html>
```



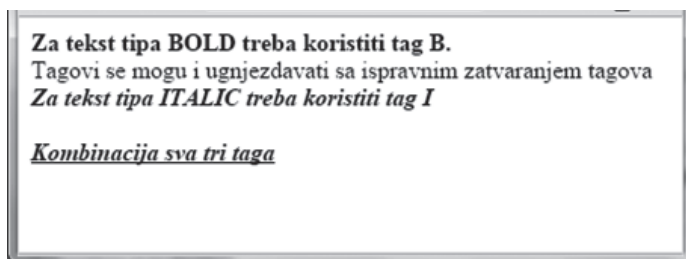
Slika 2.6. Prikaz efekta taga `br` i `pre` u browser-u

2.7.4. Stilizovanje teksta

Pomoću HTML tagova može se postići kvalitetno formatiranje teksta, vrlo slično osnovnim funkcionalnostima u *Word*-u. Međutim, još pogodniji način za formatiranje je kroz CSS koji se obrađuje u narednom poglavlju.

Stilizovanje teksta u formi: *Bold*, *Italic* i *Underline* je pomoću uparenih tagova `<b>`, `<i>` i `<u>`.

```
<html>
<body>
  <b> Za tekst tipa BOLD treba koristiti tag B. </b><br/>
  Tagovi se mogu i ugnjezdavati sa ispravnim zatvaranjem tagova <br/>
  <b> <i> Za tekst tipa ITALIC treba koristiti tag I </i> </b> <br/> <br/>
  <u> <i><b> Kombinacija sva tri taga</u> </i> </b>
</body>
</html>
```



Slika 2.7. Prikaz efekta tagova b, u i i u browser-u

Postoji i grupa drugih tagova koji su inicijalno definisani za druge namene, ali im je grafička interpretacija skoro identična sa nekim od pomenutih. Ovo je urađeno sa ciljem da se programeru, ili višem programskom jeziku, ukaže na neke razlike i detalje, iako korisnik to vidi na isti način.

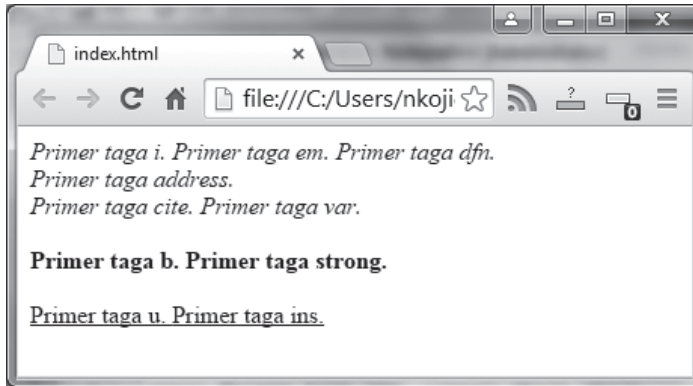
Tako na primer upareni tagovi `<em>` (za naglašavanje dela teksta), `<dfn>` (za definicije i pojmove), `<address>` (za navođenje kontakt podataka i adrese), `<cite>` (za citiranja) i `<var>` (za definisanje promenljivih) imaju isti grafički prikaz kao i tag `<i>`. Sa druge strane, tag `<strong>` ima isti grafički prikaz kao i tag `<b>` dok tag `<ins>` ima isti grafički prikaz kao i tag `<u>`.

```
<html>
<body>
  <i> Primer taga i.</i>
  <em> Primer taga em. </em>
  <dfn> Primer taga dfn. </dfn>
  <address> Primer taga address.</address>
  <cite> Primer taga cite. </cite>
  <var> Primer taga var. </var> <br/> <br/>
  <b> Primer taga b.</b>
  <strong> Primer taga strong. </strong> <br/> <br/>
```

```

<u> Primer taga u. </u>
<ins> Primer taga ins. </ins>
</body>
</html>

```



Slika 2.8. Prikaz efekta tagova *i*, *em*, *dfn*, *address*, *cite* i *var* u browser-u

Upareni tagovi `<code>` (za definisanje delova koda), `<samp>` (za definisanje izlaza iz softvera) i `<kbd>` (za označavanje sadržaja unetog sa tastature), imaju vrlo sličnu grafičku interpretaciju (tip kompjuterskog koda) i ređe se koriste u praksi. Upareni tag `<font>` nema grafičku interpretaciju, ali se koristi da se kroz njegove attribute formatira tekst, slično kao i u Word-u, slika 2.9.

```

<font color="boja">
<font size="veličina">
<font face="ime fonta">

```



Slika 2.9. Parametri za stilizovanje teksta u Word-u

Boja u HTML-u može biti ili engleska reč za boju ili heksadecimalni kod neke boje npr. green, blue, tj. #008000, #FFFF00 ...Veličina može biti broj od -7 do 7, ili dimenzija u pikselima, dok je ime fonta neko od definisanih fontova: Times New Roman, Arial, Helvetica...

```

<html>
<body>
  <font color="red" size="10px" face="Arial">
    Ovo je tekst crvene boje, slova su velicine 10.
  </font>
</body>
</html>

```


Entiteti se mogu koristiti na tri načina:

1. Primenom kodnog broja
2. Primenom heksadecimalnog zapisa
3. Primenom stringa

Na taj način, simbol @ se po nabrojanim načinima može definisati kao:

1. @
2. @
3. @

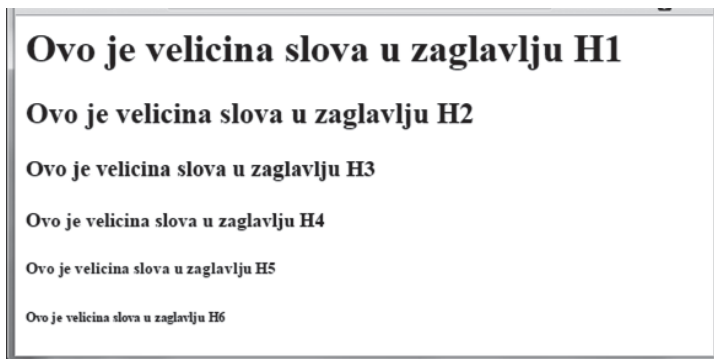
Na Internetu postoji veliki broj sajtova na kojima se mogu pronaći ove oznake, od kojih toplo preporučujem <https://dev.w3.org/html5/html-author/charref>

2.7.6. Tagovi H1-H6

Upareni tagovi `<h1>` do `<h6>` se koriste za definisanje veličine teksta. Iako je definisanje veličine teksta mnogo preciznije CSS-om, skoro svi web pretraživači očekuju H tagove. Primarna uloga H tagova je da se istaknu naslovi i podnaslovi. Postoje tagovi H1, H2, H3, H4, H5 i H6. Tag `h1` daje najveću veličinu slova, a `h6` najmanju. Svaki od ovih elemenata počinje u novom redu, a browser-i dodaju još malo praznog mesta pre njihovog prikaza. U okviru taga se može navesti atribut *align*, koji određuje horizontalno poravnanje prikaza teksta; vrednost ovog atributa može biti jedna od sledećih: *left*, *center* i *right*.

```
<html>
  <head>
    <title>Velicina slova</title>
  </head>
  <body>
    <h1>Ovo je velicina slova u zaglavlju H1</h1>
    <h2>Ovo je velicina slova u zaglavlju H2</h2>
    <h3>Ovo je velicina slova u zaglavlju H3</h3>
    <h4>Ovo je velicina slova u zaglavlju H4</h4>
    <h5>Ovo je velicina slova u zaglavlju H5</h5>
    <h6>Ovo je velicina slova u zaglavlju H6</h6>
  </body>
</html>
```

Preporuka je naslove i podnaslove pisati upotrebom tagova *h1*, *h2* i *h3*.



Slika 2.11. Prikaz efekta tagova H1 do H6 u browser-u

Pored H tagova, i upareni tag `<small>` može se koristiti za upravljanje veličinom teksta. Rezultat delovanja ovog taga je tekst manje veličine od podrazumevanog, i najčešće se koristi za footer strane i informacije o autoru, autorskim pravima i sl.

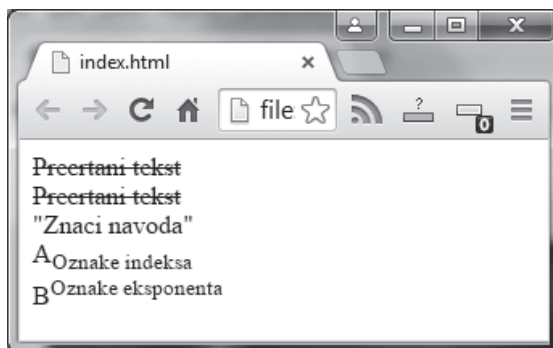
2.7.7. Efekti nad tekстом

U HTML-u se može postići veći broj efekata nad tekстом kao što je to slučaj i sa tekstualnim editorima. Iako se oni ređe koriste, u pojedinim slučajevima su neophodni. Od svih ponuđenih mogućnosti, najčešće se koristi precrtani tekst, tekst pod navodnicima i tekst u indeksu ili eksponentu. Za ove potrebe koristimo tagove *del*, *s*, *q*, *sub*, *sup*.

```
<html>
  <body>

  <del>Precrtani tekst</del>
  <s>Precrtani tekst</s>
  <q>Znaci navoda</q>
  A<sub>Oznake
  indeksa</sub>
  B<sup>Oznake
  eksponenta</sup>

  </body>
</html>
```



Slika 2.12. Prikaz efekta tagova *del*, *s*, *q*, *sub* i *sup* u browser-u

2.8. Rad sa grafikom

Rad sa grafikom spada u jedan u najkorišćenijih i najefektivnijih postupaka koji se koriste prilikom izrade web stranica i treba mu posvetiti posebnu pažnju.

2.8.1. Kreiranje linija

Neupareni tag `<hr/>` kreira jednu horizontalnu liniju visine oko 1 mm, unutar praznog reda. Moguće je podesiti debljinu korišćenjem atributa `size="n"` (n je broj piksela), dužinu pomoću atributa `width="n"`, način senčenja: `Noshade` (puna linija bez senčenja) i poravnanje pomoću atributa `align`. Dužina linije se definiše kao - apsolutna dužina ili procenat dužine celog ekrana.

```
<html>
<body>
  <hr width="100px"/>
  <hr width="75%" />
  <hr align="center" noshade size="5px" width="150px" />
</body>
</html>
```



Slika 2.13. Prikaz efekta taga `hr` i njegovih atributa u browser-u

2.8.2. Rad sa bojom

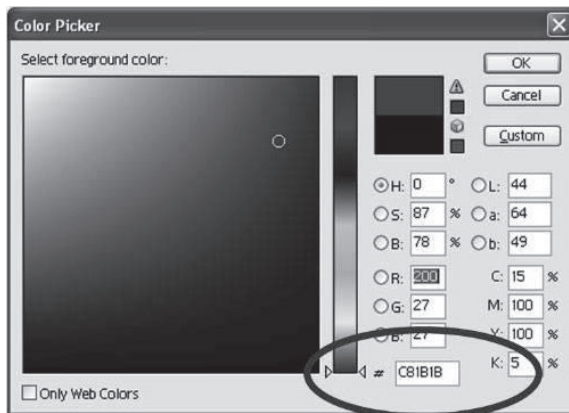
Boja se u HTML-u definiše kroz atribut `color`, ili `bgcolor`. Može biti definisana kao engleska reč (*blue, yellow, green, red, ...*) ili kao oznaka. Preciznija definicija je oznaka (heksadecimalna predstava boje), koja po pravilu počinje simbolom `#` i ima kombinaciju šest slova tj. cifara.

Ovih šest karaktera jednoznačno opisuju svaku boju na bazi RGB modela boje. RGB predstavlja aditivni model dobijanja boje na bazi međusobnog mešanja tri boje: crvene (*Red*), zelene (*Green*) i plave (*Blue*). Svaka od tri boje može biti zastupljena u konačnoj boji sa vrednošću koja je u intervalu $[0-255]$. Vrednost 0 označava da te boje nema u konačnoj mešavini, dok 255 ukazuje na njeno maksimalno moguće prisustvo. Ako se vrednosti $[0-255]$, iz decimalnog sistema, zapišu u heksadecimalnom brojnem sistemu, tada su krajnje vrednosti ovog intervala: $0_{10} = 00_{16}$ tj. $255_{10} = FF_{16}$ pa su granice u heksadecimalnom obliku $[00-FF]$. Na taj način, ako se posmatra originalna crvena boja, koja u RGB zapisu ima kompletnu zastupljenost crvene boje, dok druge dve nema, imamo zastupljenost $RGB = 255, 0, 0$. Zapis boje u heksadecimalnom obliku se dobija konverzijom svakog pojedinačnog broja iz RGB modela u heksadecimalni broj. Tako crvena boja (255,0,0) postaje `#FF0000`, plava (0,0,255) postaje `#0000FF` itd.

U ovom modelu boje bela se dobija sa maksimalnom zastupljenošću sve tri boje (255, 255, 255) pa postaje `#FFFFFF`, dok je crna odsustvo svake boje (0,0,0) pa postaje `#000000`.

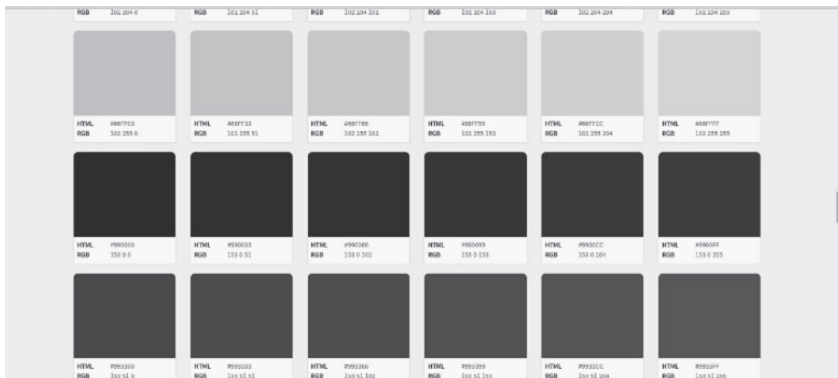
Na ovaj način svaka boja se iz RGB sistema lako može konvertovati u heksadecimalni zapis. Primeri pojedinih boja su: žuta (255,255,0) postaje #FFFF00, siva (192,192,192) #C0C0C0, ljubičasta (128,0,128) #800080, itd.

U slučaju da se želi proizvoljna nijansa boje, klikom na datu boju, u većem broju softverskih paketa koje rade sa obradom slike, ova oznaka se automatski dobija za tu nijansu boje.



Slika 2.14. Prikaz softvera za određivanje heksadecimalne oznake boje

Međutim, zbog problema koji se pojavljuje u pojedinim starijim verzijama browser-a, sa preciznošću prikaza pojedinih nijansi boja, preporučuju se takozvane sigurne boje. Postoji veći broj sajtova na kojima se može videti katalog ovih boja kao i oznaka svake od njih.



Slika 2.15. Prikaz jednog sajta za određivanje heksadecimalne oznake boje. (izvor <http://websafecolors.info/>)

2.8.3. Postavljanje pozadinske boje

Definisanje pozadinske boje cele web stranice realizuje se atributom *bgcolor*, koji se definiše u tagu *body*.


```

<html>
  <body BGCOLOR="#GGFFAA" >
    <font face="Bookman Old Style" color="#FF0000" size="5px">
      Ovo je tekst.
    </font>
  </body>
</html>

```

2.8.4. Rad sa listama

Liste podrazumevaju još jednu od standardnih oblika formatiranja teksta. Postoje dva tipa listi (oblika nabiranja): sa numeracijom (uređene) i bez numeracije (neuređene).

Upareni tag za definisanje liste sa numeracijom je `<ol>`. Unutar ovog taga nabrajaju se konkretne stavke, koje se zovu elementi liste. Upareni tag za svaki od elemenata liste je `<li>`. Inicijalna numeracija je po pravilu arapskim brojevima. U slučaju da se želi drugačiji način numeracije, koristi se tag `<ol>` sa atributom *type* i vrednošću od koje će krenuti novi način numeracije. Na primer:

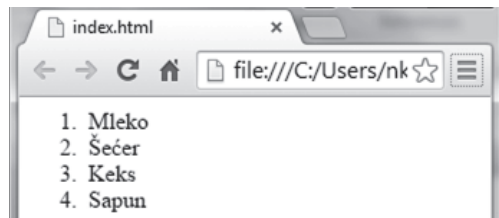
```
<ol type="a">, <ol type="A">, <ol type="i">, <ol type="I">, ...
```

Definisanjem prvog slova ili broja, ostali se automatski dodaju. Za počinjanje liste drugim brojem od 1 treba koristiti atribut *start*, npr. `<ol start="5">`.

```

<ol>
  <li>Mleko</li>
  <li>Šećer</li>
  <li>Keks</li>
  <li>Sapun</li>
</ol>

```

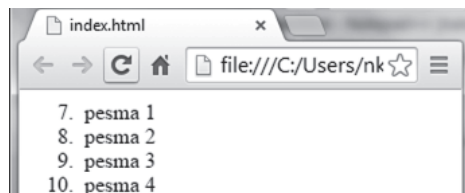


Slika 2.16. Prikaz efekta taga `ol` i `li` u browser-u

```

<ol start="7">
  <li>pesma 1</li>
  <li>pesma 2</li>
  <li>pesma 3</li>
  <li>pesma 4</li>
</ol>

```



Slika 2.17. Prikaz efekta taga `ol` i `li` sa atributom *start* u browser-u

Za označavanje elemenata neuređene liste koristi se regularni tag `<ul>`. Ovaj tip označavanja se može urediti posebnim grafičkim elementima primenom atributa *type*:

```

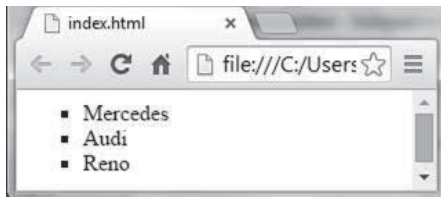
<ul type="disc">      za crnu tačku
<ul type="square">    crni kvadrat
<ul type="circle">    crna kružnica

```

```

<ul type="square">
  <li>Mercedes</li>
  <li>Audi</li>
  <li>Reno</li>
</ul>

```



Slika 2.18. Prikaz efekta taga `ul` i `li` u browser-u

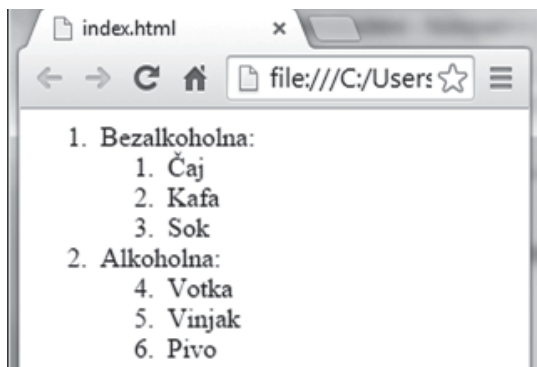
Ugnježdene liste se definišu na isti način kao i liste, ali su elementi osnovnih lista druge liste.

```

<ol start="1">
  <li>Bezalkoholna:
    <ol>
      <li>Čaj</li>
      <li>Kafa</li>
      <li>Sok</li>
    </ol>
  </li>
  <li>Alkoholna:
    <ol start="4">
      <li>Votka</li>

      <li>Vinjak</li>
      <li>Pivo</li>
    </ol>
  </li>
</ol>

```



Slika 2.19. Prikaz efekta ugnježđenih lista u browser-u

2.8.5. Definicione liste

Ove liste se koriste kod posebne vrste prikaza teksta, koji treba strukturano formatirati. Definišu se pomoću tagova `<dl>`, `<dt>` i `<dd>`. Tag `<dl>` se koristi za otvaranje i zatvaranje definicione liste. Tag `<dt>` se koristi za kreiranje naslova za stavke koje se navode u listi dok se tag `<dd>` koristi za definisanje sadržaja svake od navedenih stavki.

```

<dl>
<dt> Prvi trimestar
  <dd> Traje od xxx do
xxx</dd>
</dt>

<dt> Drugi trimestar
  <dd> Traje od xxx do xxx
</dd>
</dt>

<dt> Treci trimestar
  <dd> Traje od xxx do xxx
</dd>
</dt>

</dl>

```



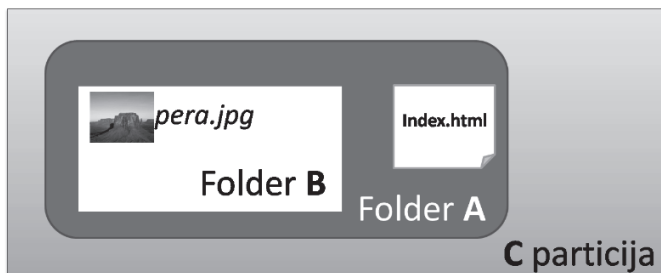
Slika 2.20. Prikaz efekta taga *dl*, *dt* i *dd* u browser-u

2.8.6. Referenciranje objekata

Fizički prostor na web serveru podjeljen je na direktorijume i foldere. Da bi se “održao” sadržaj web strane ili sajta napravljenog na personalnom računaru, koji u sebi sadrži i fajlove (npr.slike) svi fajlovi se moraju definisati relativnim ili apsolutnim putanjama.

- ✓ Relativna putanja se koristi kada se posmatra putanja od web stranice do željenog fajla tj. od foldera gde se nalazi *html* stranica koja poziva određeni fajl (npr. slika A.jpg, koja se nalazi u folderu *slike* koji je u folderu u kome i *html* stranica).
- ✓ Apsolutna putanja se koristi kada se posmatra putanja od particije diska do željenog fajla tj. bazira u odnosu na strukturu foldera na tom računaru počev od oznake particije (npr. slika B.jpg, se nalazi u folderu slike na particiji D, ili to može biti putanja do drugog sajta tj. računara).

Primer za sliku *pera.jpg*, koja je u folderu B, unutar direktorijuma A (gde je i kod stranice *index.html*), a sve ovo je na C disku je:



Slika 2.21. Struktura foldera sa slikom i HTML stranicom

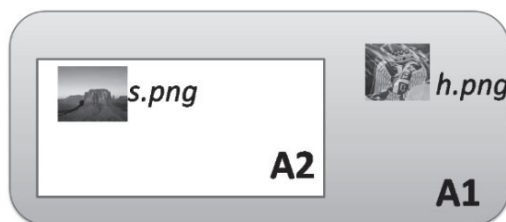
Relativna putanja (fajl koji poziva je u istoj ravni gde i folder A):
B/pera.jpg

Apsolutna putanja (fajl koji poziva je bilo gde, pa se daje puna putanja):
C:A/B/pera.jpg

U slučaju da je struktura foldera kompleksnija za apsolutnu putanju mora da se piše svaki delić putanje, dok se relativna ne menja. Obzirom da je često nepoznata struktura van *root* foldera (folder koji je od strane hostinga dodeljen za određeni sajt), apsolutnu putanju treba izbegavati za sve fajlove koji su sastavni deo sajta a fizički se nalaze hostovani sa sajtom. Izuzetak kada se apsolutna putanja koristi je kada se pristupa fajlu sa nekog drugog sajta, kao na primer

www.drugi_sajt.com/slike/pera.jpg

I u ovom slučaju, problem apsolutne putanje može biti promena lokacije i reorganizacija sajta sa koga se dohvata željeni fajl. U slučaju da se u putanji navodi podređeni folder, onda se navodi njegovo ime u putanji, dok se za nadređeni folder koristi simbol `../`. Na primer, ako je folder A2, sa slikom *s.png*, unutar foldera A1, posmatrano van pozicije foldera A1, putanja do slike je A1/A2/s.png.



Slika 2.22. Struktura foldera sa dve slike

Međutim, ako se iz pozicije folder A2, želi stići do A1, i fajla *h.jpg*, koji je u A1, putanja je `../h.jpg`. Ukoliko se želi pozicioniranje u još viši nadređeni folder, za svaki viši nivo se koristi po jedan simbol `../`. Tako za pozicioniranje tri foldera u odnosu na posmatrani pišemo `../../..` itd.

2.8.7. Rad sa slikom

Slika se definiše neuparenim tagom `<img />`. Ključni atribut taga `<img />` je *src*. Atribut *src* definiše putanju (*source*) i ime slike koja se želi prikazati. Putanja može biti ili relativna ili apsolutna:

```



  
```

Veličina slike koja se prikazuje u browser-u je originalna veličina slike *upload*-ovane slike na web server. Promena prikaza veličine u browser-u, realizuje se

atributima *width* i *height*. Međutim, na ovaj način smanjuje se samo dimenzija ali ne i “težina” slike.

```

```

Veličina slike, definisana atributima *width* i *height*, može biti apsolutna (u pikselima) ili relativna (definisana u % u odnosu na originalnu veličinu slike). U slučaju da se želi preneti fizički manja slika, potrebno je sliku posebno kreirati sa željenim dimenzijama, i manjom “težinom” i kao takvu je prikazati u browser-u. Ovakve slike se nazivaju “*thumbnail*” i najčešće su prvi prikaz slike, dok se klikom na njih prikazuje druga, veća slika, koja se posebno priprema i uploaduje na server.

Često korišćeni atributi su *alt* i *title*. Atribut *alt* se koristi da bi se umesto slike (kada postoji problem sa učitavanjem) na mestu slike prikaže tekst definisan atributom *alt*. Atribut *alt* je vrlo bitan i za web pretraživače, obzirom da oni ne razumeju sastav slike, nego sliku vide kroz tekst (definisan u *alt* atributu). Atribut *title* se koristi da bi se korisniku diskretno prikazao tekst, neposredno preko linka (bilo da je to tekstualni link ili slika), i to u trenutku prelaska miša preko linka.

Slika može imati i atribut *border*, koji definiše okvir oko slike i atribut *name*, koji se ne vidi, ali se koristi da viši programski jezici mogu da “prozovu” sliku i izvrše neku akciju nad njom. Atribut *align* se može koristiti za pozicioniranje slike.

```
<body>
```

Slika 1 – Originalna slika sa atributom *alt* koji ispisuje tekst ako nema slike


```
 <br/>
```

Slika 2 – Definisanje atributa za apsolutnu visinu i dužinu slike


```
<br/>
```

Slika 3– Definisanje atributa za relativnu visinu i dužinu slike


```
 <br/>
```

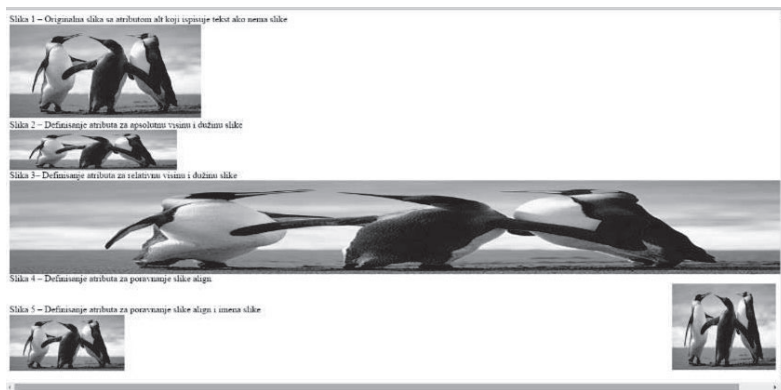
Slika 4 – Definisanje atributa za poravnanje slike *align*


```
 <br/><br/>
```

Slika 5 – Definisanje atributa za poravnanje slike *align* i imena slike


```

<br/><br/>
</body>
```



Slika 2.23. Prikaz efekta atributa *width* i *height* kod taga *img* u browser-u

Posebnu pažnju treba obratiti kod skaliranja slike i primene atributa *width* i *height*, obzirom da se lako može desiti da se dobije deformisana slika ukoliko se obe dimenzije ne promene proporcionalno.

Definisanje slike kao pozadine web strane: u HTML kodu realizuje se pomoću atributa *background*, pri čemu se dobija efekat slike kao pozadine dela ili cele web stranice

```
<body background="slika.gif">
```

Ovo se može kombinovati i sa ranije definisanim atributom *bgcolor* da bi se u slučaju da postoji problem sa učitavanjem slike pozadina opisala bojom:

```
<body background=" slika.gif" bgcolor="#333333">
```

2.9. Hyper linkovi

Hyper linkovi predstavljaju vrlo bitan segment rada HTML-a, ali i konkretne primene obzirom na potrebu navigacije među različitim web stranicama sajta. Tag za link je `<a>`. Između `<a>` i `</a>` nalazi se tekst ili slika koja se prikazuje korisniku kao vidljiv deo linka, i koji se smatra aktivnim delom linka. Klik na taj tekst ili sliku, treba da ukaže na neki fajl ili objekat (web stranu, sliku, video fajl, zip fajl, word dokument, ...) koji će se prikazati ili preuzeti. Atribut taga `<a>` kojim se definiše putanja i fajl, koji će se klikom na link “prozvati”, definisan je atributom *href*.

Primer 1: Link ka web strani istog sajta

```
<a href="home.html">Ovde pritisnuti da bi videli našu home stranu.</a>
```

Primer 2: Link ka web strani druge web lokacije

```
<a href="http://nesto.com/index.htm">Ovde pritisnuti da bi videli sajt  
nesto.com.</a>
```

Primer 3: Tekstualni link ka nekom fajlu

```
<a href="http://www.nesto.com/web/mika.zip"> Ovde pritisnuti da bi skinuli mika.zip sa sajta www.nesto.com </a>
```

Primer 4: Link (slika) ka web strani istog sajta

```
<a href="home.html">  </a>
```

Linkovi se unutar web stranice mogu naći na bilo kom mestu i u bilo kojoj formi. Međutim, *spider-i* web pretraživača, kao i viši programski jezici, trebalo bi da „razumeju“ grupe linkova koje su i ljudima logički i vizuelno povezani ili grupisani. Ovo se najčešće odnosi na grupu linkova koji su meni, vesti, najave, grupa proizvoda i sl. da bi se prikazivali na isti vizuelni način i da bi se isti efekti prikazivali nad istom grupom linkova, unutar iste web strane. Da bi programski jezici mogli da razumeju koja grupa linkova pripada onome što posmatrač sajta doživljava kao grupu linkova, linkovi se nekako moraju u kodu označiti kao elementi koji pripadaju nekoj zajedničkoj grupi. Za ove potrebe se koriste liste.

Iako se liste ređe koriste u sajtovima kao klasični vizuelni identitet liste, sa nekim od efekata *bullet-a*, njihova primena za grupisanje linkova, a kasnije i drugih elemenata web strane, je izuzetno velika. U ovom poglavlju ćemo se ograničiti na primenu lista u kombinaciji sa linkovima sa ciljem da se prikaže pravljenje jedne grupe linkova, koja će kasnije biti skoro obavezna kod forme najrazličitijih oblika menija. U narednom listingu prikazan je najčešći oblik neuređene liste koja se koristi da grupiše tekstualne linkove za potrebe navigacionog menija. Za vrednost atributa *href* iskorišćena je vrednost *#* koja omogućava da link postoji ali da u ovom slučaju klik na link ne vodi nigde. U slučaju da ima više grupa linkova, najčešće se koristi atribut *id* u tagu *ul* kojim se označavaju cele grupe linkova.

```
<ul id="meni1">
  <li><a href="#">Početna strana</a></li>
  <li><a href="#">O nama</a></li>
  <li><a href="#">Galerija</a></li>
  <li><a href="#">Kontakt</a></li>
</ul>
```

Pored referenciranja na fajlove van web strane moguće je izvršiti i referenciranja unutar web strane. Ovo se koristi kada je potrebno klikom na link „skočiti“ tj. pozicionirati se na tačno definisani deo web stranice.

Za ove potrebe koristi se specijalni karakter *#* koji se stavlja na početku sadržaja atributa *href*. Iza njega, može se definisati proizvoljno ime, kojim se opisuje naziv pozicije na koju se „skače“. Da bi se unutar strane definisalo gde počinju pojedine pozicije, koristi se atribut *name*. Tako u listingu koji sledi, klikom na link JavaScript, korisniku će se stranica sama „skrolovati“, i na vrh strane postaviti sadržaj koji kreće od dela teksta *JavaScript Ovde ubaciti malo više teksta...*

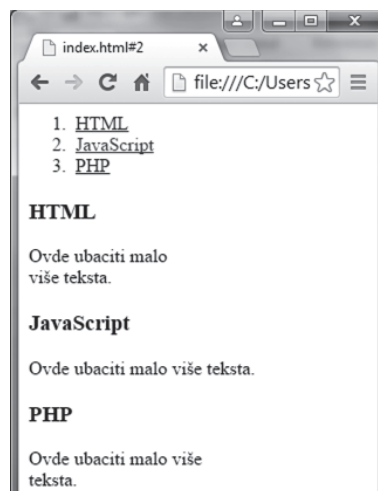
```

<html>
<body>
<ol>
<li> <a href="#1">HTML</a></li>
<li> <a href="#2">JavaScript</a></li>
<li> <a href="#3">PHP</a></li>
</ol>

<h3> <a name="1"> HTML</a> </h3>
<p>Ovde ubaciti malo<br/> više
teksta.</p>
<h3> <a name="2"> JavaScript </a>
</h3>
<p>Ovde ubaciti malo više teksta.</p>
<h3> <a name="3"> PHP</A> </h3>
<p>Ovde ubaciti malo više<br/>
teksta.</p>

</body>
</html>

```



Slika 2.24. Prikaz referenciranja unutar web stranice

2.9.1. Prikaz linkovanih dokumenta

Klikom na neki link, inicijalno se sadržaj linkovane web stranice prikazuje u istom prozoru browser-a. Međutim, primenom atributa *target*, moguće je preciznije kontrolisati gde će se prikazati linkovani sadržaj. Atribut *target* pripada tagu `<a>` i može imati više vrednosti.

```
<a href="www.nekisajt.com " target="_blank"/> Klik </a>
```

Tabela 2.2. Vrednosti atributa *target*.

Vrednost	Opis
<code>_blank</code>	Otvora linkovani document u novom prozoru ili tab-u
<code>_self</code>	Otvora linkovani document u istom prozoru (default)
<code>_parent</code>	Otvora linkovani document u roditeljskom (nadređenom) frame-u
<code>_top</code>	Otvora linkovani document u celom body-ju prozoru
<code>framename</code>	Otvora linkovani document u posebnom frame-u sa definisanim imenom

2.9.2. Formatiranje teksta linka

Link ima tri faze koje se detektuju (inicijalna boja, boja u trenutku klika, i boja nakon klika na link, koja se vidi nakon vraćanja na stranicu gde je link). Ove tri faze se mogu pojedinačno definisati proizvoljnom bojom. To se realizuje kroz tri

atributa *link*, *alink* (*active link*) i *vlink* (*visited link*), taga `<a>`. Ovi atributi se mogu definisati u tagu `<body>` i tada pravilo važi za sve linkove na toj strani.

```
<body bgcolor="#CCFFAA" text="blue" link="red" vlink="green" alink="red">
```

Pored ovoga, mogu se definisati unutar CSSa i tako bliže odrediti kada i gde se ove boje primenjuju.

2.9.3. Realizacija linka za slanje mail-a

Pravljenje linka kojim se automatski otvara *Outlook Express* (ili neki drugi podrazumevani lokalni mail klijent) je istog formata kao i klasičan link, ali je u atributu *href* definisana adresa primaoca sa **mailto:aaa@bbb.ccc** kao:

```
<a href="mailto:nkojic@singidunum.ac.rs">Kontakt</a>
```

Pored osnovne adrese mogu se dodati i ostali atributi. Odvajaju se znakom `?`, `i` & `i` nose unapred definisana imena promenljivih (*subject* i *body*).

```
<a href="mailto: nkojic@singidunum.ac.rs?subject=Mail sa sajta&body=Dobar dan">Kontakt</a>
```

Klikom na ovako kreirani link, korisniku se automatski otvara *Outlook Express* i u njega upisuju definisani podaci, ali se mail ne šalje dok korisnik klikom na *Send* to ne potvrdi.



Slika 2.25. Prikaz popunjenog mail klijenta tagom *a*

2.9.4. Dodavanje slike u URL adresu

Ova mogućnost se sve češće koristi u novijim sajtovima. Njom se omogućava da programer definiše sliku koja će se zajedno sa sajtom, učitati i prikazati ispred URL adrese sajta. Da bi se ovo omogućilo potrebno je kreirati poseban format slike *.ico*, u rezoluciji do 64x64px. Tu sliku treba staviti na web server kao i svaku drugu i jednom linijom koda, unutar *head* sekcije, je pozvati na sledeći način:

```
<link rel="shortcut icon" href="ime.ico" />
```

Rezultat je prikaz slike u zaglavlju browser-a ili pored URL adrese, što zavisi od tipa browsera.



Slika 2.26. Prikaz slike pored URL adrese u browser-u

U širem kontekstu rada sa linkovima, može se posmatrati i upareni tag `<abbr>`. Ovaj tag svoju interpretaciju u browser-u ostvaruje kada se mišem pređe preko dela teksta koji pripada sadržaju ovog elementa, kada se u posebnom *pop-up* prozoru prikazuje sadržaj atributa *title*.

Ovaj kurs se realizuje na `<abbr title="Univerzitet Singidunum">Tehničkom fakultetu Univerziteta Singidunum</abbr>` i u 2017. godini.

Ovaj kurs se realizuje na Tehničkom fakultetu Univerziteta Singidunum i u 2017. godini.

Univerzitet Singidunum

Slika 2.27. Prikaz efekta taga `abbr` u browser-u

Posebna pogodnost za linkovanje unutar delova jedne slike omogućena je primenom tagova `<map>` i `<area>`. Naime, učitavanjem originalne slike, moguće je na njoj definisati oblasti i svakoj od njih dodeliti drugu linkovanu stranu. Povezivanje slike sa oblastima se realizuje pomoću atributa *usemap*, kao u primeru. Unutar taga `<map>` definiše se proizvoljno mnogo oblasti tagom `<area>`. Svaka oblast je definisana oblikom (atribut *shape*), koordinatama koje opisuju tu oblast (atribut *coords*), alternativnim imenom i putanjom do linkovane stranice, ukoliko se klikne na tu oblast.

```

<map name="svi">
  <area shape="rect" coords="0,0,80,150" alt="Bokal" href="bokal.html">
  <area shape="circle" coords="80,70,5" alt="Tanjir" href="tanjir.html">
  <area shape="circle" coords="90,30,3" alt="Sat" href="sat.html">
</map>
```

Na ovaj način se mogu kreirati vrlo precizne krive ali je potrebno uneti veći broj koordinata u atribut *coords*.

Jedan od vrlo karakterističnih tagova je i neupareni tag `<base/>`. Ovaj tag se piše u *head* sekciji web stranice. Kreiranjem ovog taga, i kombinacijom sa njegova dva atributa (*href* i *target*), svi linkovi unutar te web strane, koji nisu označeni kao eksterni, će automatski dobiti prefiks u putanju koji je definisan u tagu *base* i atributu *href*. Isto tako, ako je definisan atribut *target*, svi linkovi na strani će se ponašati u skladu sa tim pravilom definisanim u tagu *base*.

Posmatrajmo primer listinga. Tagom *base* definisana je primarna putanja kao `www.nekisajt.com/slike/` i `target="_blank"`.

```

<html>
<head>
  <base href="www.nekisajt.com/slike/" target="_blank"/>
</head>
<body>
   <br/>
   <br/>
  <a href="prikaz.html">Prikaz 1</a>
  <a href="http://tf.singidunum.ac.rs/">Prikaz 2</a>
</body>
</html>

```

U telu web strane, učitavaju se dve slike i dva linka. Iako je putanja do prve slike definisana sa `src="1.jpg"`, zbog postojanja taga `base`, ova slika će se učitati sa putanje `www.nekisajt.com/slike/1.jpg`. Slično, za drugu sliku putanja će biti `www.nekisajt.com/slike/letovanje/2.jpg`. U slučaju prvog linka, njegova pozicija će se tražiti na lokaciji `www.nekisajt.com/slike/prikaz.html`, i sadržaj će se zbog `target="_blank"` otvoriti u posebnom prozoru browser-a.

Jedini izuzetak je lokacija za drugi link, koja će ostati kao i u originalnom kodu, `http://tf.singidunum.ac.rs/`, jer se tumači kao eksterni link sa punom apsolutnom putanjom (ključnu ulogu ima `http://`), ali i on poprima pravilo `target="_blank"`.

2.10. Rad sa multimedijalnim fajlovima

2.10.1. Pozadinski zvuk

HTML omogućava da se prilikom učitavanja sajta učita i reprodukuje audio sekvenca. Tag kojim je ovo omogućeno je `bgsound` sa sintaksom

```

<bgsound src="imeaudiofajla.format" loop="broj" />

```

gde se ime fajla definiše preko imena i formata fajla. Audio fajl se ne mora nalaziti u istom direktorijumu kao i kod web strane, ali u tom slučaju se mora zadati apsolutna ili relativna putanja do tog fajla. Formati koji su podržani u HTML-u su Wav, AU, MP3 i MIDI.

Broj koji se definiše u atributu `loop` određuje koliko puta se sekvenca ponavlja. Upisivanjem npr. broja **2**, ponoviće se dva puta, dok se stavljanjem **-1** ili **infinite** automatski ponavlja sve dok korisnik posmatra stranicu. Pored taga `bgsound`, kao i za veći broj drugih multimedijalnih fajlova, može se koristiti tag `<embed>`. Ovaj tag podržava uobičajene formate kao što su .wav, .mid, .mp3, i .au.

Primer:

```
<embed src="filename.ext" width="x" height="x" autoplay="x" hidden="x"
loop="x" volume="x"> </embed>
```

gde su sledeći atributi definisani kao:

src: URL (relativna ili apsolutna putanja) do audio fajla.

controls: *console* tj. *smallconsole*. *Console* definiše 144 piksela za širinu i 60 za visinu dok *smallconsole* definiše 144 piksela za širinu i 15 za visinu.

height i **width:** Definišu veličinu prostora za objekat *embed* taga. Kod nekih browsera je obavezan.

loop: Definiše koliko puta se sekvenca ponavlja. Ako se ništa ne definiše, izvršava se samo jednom.

autostart: Ako je *true* tada se zvuk aktivira po učitavanju stranice, dok se za *false* vrednost mora kreirati taster kojim korisnik upravlja pokretanjem

hidden: Vrednost *true* sakriva konzolu, dok je *false* prikazuje. Upotreba *hidden* i *autostart* kao *true* postiže se efekat isti kao i primenom *bgsound*.

volume: Vrednost između 0 i 100 za definisanje nivoaa zvuka.

align: poravnanje sa leve i desne strane tj. *left* i *right*.

hspace i **vspace:** horizontalni odnosno vertikalni prostor oko konzole u pikselima.

Pored ovog načina, audio fajl se u web stranu može učitati i pomoću taga *<object>* i njegovog atributa *data*, kao u listingu. Tagom *<param>* moguće je kontrolisati automatski početak reprodukcije audio fajla, kroz njegov atribut *name* i *value*.

```
<object data="pesma.wav">
  <param name="autoplay" value="true"/>
</object>
```

2.10.2. Flash objekat

Flash objekat se može uključiti u HTML kod korišćenjem taga *embed*. Kao i kod drugih objekata treba definisati putanju do objekta (sa *src*) i dimenzije (*width* i *height*). Poželjno je postaviti kvalitet koji se očekuje (*quality*), tip (*type*) i putanju za plugin-ove (*pluginspage*).

```
<embed src="http://pera.com/folder/1.swf" quality="high"
  pluginspage="http://www.macromedia.com/go/getflashplayer"
  type="application/x-shockwave-flash" width="568" height="32">
</embed>
```

2.10.3. Video fajl

U HTML 4 ne postoji posebna podrška za video fajlove. Oni se mogu prikazati upotrebom nekog od drugih softvera koji se prikazuju u browser-u, dok samostalna podrška ne postoji. Ovo je ispravljeno u HTML 5. Pored video fajla koji je „off line“, u web strani se može prikazati i integrisani prikaz video fajla „u *streaming*-u“. Ovo je najčešće realizovano upotrebom sajta *youtube.com* ili sličnih. Ova integracija moguća je na tri načina: tagom *object*, *embed* i *iframe*.

U slučaju tagova *embed* i *iframe*, koristi se atribut *src*, da se ukaže na punu apsolutnu putanju do video sekvence tj. *data* za tag *object*. U ovim primerima kontrole nad video fajlom se automatski obezbeđuju.

```
<html>
<body>
  <object data="http://www.youtube.com/embed/W7qWa52k-nE"
    width="560" height="315"></object>

  <iframe src="http://www.youtube.com/embed/W7qWa52k-nE"
    width="560" height="315" frameborder="0" allowfullscreen>
</iframe>

  <embed src="http://www.youtube.com/embed/W7qWa52k-nE"
    width="420" height="315">
</embed>
</body>
</html>
```

Organizacija sadržaja web stranice

Sadržaj web stranice i njegova organizacija imaju primarni uticaj na stav korisnika o samom sajtu. Do sada obrađeni deo HTML koda omogućava da se na stranici prikaže i formatira željeni sadržaj a sada će se posebna pažnja posvetiti organizaciji i struktuiranju tog sadržaja. Ovo je posebno važno imajući u vidu da web pretraživači imaju izrazito različite kriterijume i pravila u smislu načina organizacije sadržaja. Pored toga, dobra organizacija obezbeđuje da se željeni sadržaj prikazuje isto u svim ili većini browser-a, što inače nije slučaj zbog različitih interpretacija HTMLa u pojedinim browser-ima. Organizacija sadržaja stranice je prvi korak koji se realizuje prilikom izrade sajta, pa će se tome posvetiti posebna pažnja. Iako postoje tri različita načina (frejmovi, tabele i CSS) da se ovaj postupak uradi, i bez obzira što sva tri nisu podjednako zastupljena, svaki od njih će se posebno obraditi. Aktuelni trend je da se organizacija sadržaja realizuje isključivo pomoću CSS-a, dok rad sa tabelama treba koristiti samo za potrebe uređenja dela sadržaja strane sa podacima. Treći način, primena *frame*-ova, nije preporučljiv i nije podržana u HTML 5, ali je i dalje zastupljena u određenom procentu komercijalnih web sajtova. Bitno je napomenuti da se željena organizacija može postići na sva tri načina, i to tako da korisnik i ne može da primeti razliku. Međutim, održavanje koda, proširivanje i povezivanje sa drugim programskim jezicima, kao i kompatibilnost sa web pretraživačima drastično utiču na način kako organizacija sadržaja treba da bude realizovana.

2.11. Rad sa frame-ovima

Frame-ovi (okviri) spadaju u najstariju i sve više zapostavljenu tehniku za organizaciju sadržaja. Iako su bili vrlo rasprostranjeni u sajtovima, u poslednjoj verziji HTML 5 više nisu ni podržani na način kako je to bilo u ranijim verzijama. Prednosti upotrebe *frame*-ova je što se jedna web stranica može podeliti na proizvoljan broj oblasti (*frame*-ova) i unutar svakog učitati posebna HTML stranica, od kojih svaka može imati i svoj interni vertikalni i horizontalni *scroll*.

Međutim, *frame*-ovi se pokazuju loše kod rangiranja u pretraživačima, jer je nejasno koja stranica nosi glavnu informaciju, i šta će biti sadržaji ostalih delova koji se naknadno učitavaju u početnu stranu sajta. Dobra strana njihove primene je što su delovi stranice fizički odvojeni i struktura stranice je vrlo stabilna, ali ne uvek i estetski efikasna.

Rad sa frame-ovima se realizuje kroz dva taga `<frameset>` i `<frame>`. Tag `<frameset>` definiše raspored tj. strukturu i veličinu okvira dok tag `<frame>` definiše konkretnu web stranicu koja se učitava u odgovarajući okvir. Tako na primer, ukoliko se žele tri horizontalna okvira, od kojih je prvi širine 100px, drugi 400px a treći 200px potrebno je kreirati tag `<frameset>` kao

```
<frameset rows="100,400,200">
</frameset>
```

Za svaki okvir potrebno je definisati koja fizička HTML stranica će se u njih učitati i svakoj od njih treba definisati ime. Ovo se realizuje kroz tag `<frame>`, na sledeći način

```
<frameset rows="100,400,200">
  <frame src="meni.html" name="left">
  <frame src="telo.html" name="centar">
  <frame src="footer.html" name="bottom">
</frameset>
```

Treba napomenuti, da je uvođenjem HTML 5 tag `<iframe>` koji se odnosi na jednolinijski okvir, i dalje ostavljen u standardu, dok se klasična upotreba frame-ova polako napušta i ne treba je koristiti. Pomoću taga `<iframe>` moguće je u web stranicu sajta dodati potpuno novu web stranicu. Ova stranica može biti stranica tog sajta ili bilo koja druga, definisana apsolutnom URL adresom. U zavisnosti od dimenzija *iframe*-a browser će automatski generisati vertikalni i/ili horizontalni *scroll* ukoliko je dimenzija učitane stranice veća od dimenzija *iframe*-a.

```
<html>
<body>
  Ovo je neki tekst!
  <iframe width="500px" height="200px"
    src="http://tf.singidunum.ac.rs/">
  </iframe>
  Ovo je neki tekst!
</body>
</html>
```

Pored integrisanja celih web stranica, tag *iframe*, se koristi i u situacijama kada je potrebno integrisati specifične delove sadržaja nekih drugih sajtova ili aplikacija. Jedna od najčešćih primena je integracija *youtube* servisa za real-time distribuciju video ili audio materijala, kao u listingu.

```
<iframe width="560" height="315" src="https://www.youtube.com/
embed/_qNu_vkK5rk" frameborder="0" allowfullscreen>
</iframe>
```

2.12. Rad sa tabelama

Tabele se za razliku od frame-ova mnogo češće koriste. Iako ih ne treba koristiti za organizaciju sadržaja cele strane, i dalje su vrlo pogodne za standardno uređenje podataka u tabelarnom obliku.

Organizacija sajta tabelom podrazumeva da je cela strana jedna tabela, kojoj su ivice nevidljive. U svakoj od ćelija tabele nalaze se neki sadržaji sajta (baneri, navigacija, tekstovi, slike ...), ali i cele manje tabele da bi struktura bila “čvršća” i lakša za rad. Za velike sajtove, ova struktura je vrlo komplikovana jer postoji veliki broj ugnježđenih tabela, pa se ne preporučuje zbog čitljivosti koda i održavanja istog. Takođe se pokazuje kao loša za rangiranje u pretraživačima.

Tabele su u HTML-u kreiraju primenom nekoliko tagova. Po pravilu, tabela se sastoje od vrsta i kolona. U HTML-u tabela se sastoji od redova koji su podeljeni na polja, a prva polja svih redova čine prvu kolonu, druga polja drugu kolonu, itd.

Tagovi za rad sa tabelama su:

`<table>` - za kreiranje tabele,

`<tr>` - za kreiranje reda,

`<td>` - za polje unutar reda tj. kolonu,

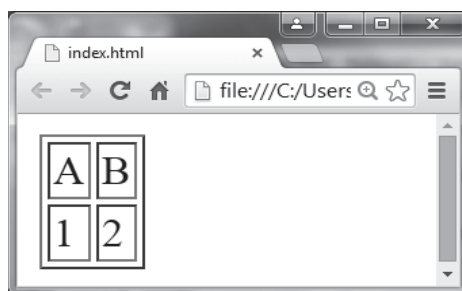
`<th>` - za definisanje zaglavlja tabele,

`<thead>`, `<tbody>` i `<tfoot>` - se koriste ka grupisanje sadržaja u zaglavlju, telu i podnožju tabele,

`<caption>` - definisanje naslova cele tabele.

Kreiranje tabele počinje tagom `<table>`, unutar koga se definiše željeni broj redova pomoću taga `<tr>`. Pojedinačne ćelije tj. kolone unutar redova definišu se pomoću taga `<td>`, pa se konkretan sadržaj piše unutar tagova `<td>`. Na taj način, tabela dimenzija 2 x 2, i sadržajem po redovima A, B i 1, 2 kreira se kao u listingu.

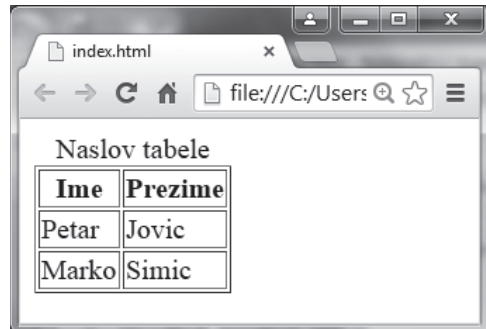
```
<table>
  <tr>
    <td>A</td>
    <td>B</td>
  </tr>
  <tr>
    <td>1</td>
    <td>2</td>
  </tr>
</table>
```



Slika 2.28. Prikaz tabele u browser-u

U slučaju da se tabeli želi definisati naslov i zaglavlje, koriste se tagovi `<caption>` i `<th>`, kao u listingu

```
<table>
  <caption>Naslov
  tabele</caption>
  <tr>
    <th>Ime</th>
    <th>Prezime</th>
  </tr>
  <tr>
    <td>Petar</td>
    <td>Jovic</td>
  </tr>
  <tr>
    <td>Marko</td>
    <td>Simic</td>
  </tr>
</table>
```



Slika 2.29. Prikaz tabele sa zaglavljem u browser-u

Atributi koje se često koriste u radu sa tabelama su:

border – definiše širinu linije u pix

colspan – koristi se za spajanje ćelija u koloni

rowspan - koristi se za spajanje ćelija u redu

cellspacing - definiše rastojanje između ćelija u tabeli

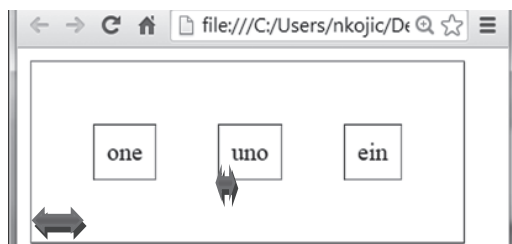
cellpadding - definiše rastojanje od ivice do sadržaja ćelije

U listingu je data primena atributa *border*, *cellspacing* i *cellpadding*. Kod većeg broja browser-a podrazumevana vrednost za atribut *border* je 1px, ako kako to ipak nije kod svih, potrebno je eksplicitno definisati.

```
<table border="1px" cellpadding
="8px" cellspacing="40px">

  <tr>
    <td>one</td>
    <td>uno</td>
    <td>ein</td>
  </tr>

</table>
```



Slika 2.30. Prikaz efekta atributa *cellpadding* i *cellspacing* u browser-u.

Upravljanje dimenzijama tabele, ili njenih ćelija, realizuje se atributima *width* i *height*. Kao i u drugim primerima, i ovde je moguće koristiti i apsolutne i

relativne dimenzije. Tako primer za relativnu i apsolutnu dužinu tabele, tj. ćelije može biti kao u primeru:

```
<table width="90%" height="30%"> - za tabelu
<td width="90%" height="30%"> - za ćeliju
```

```
<table width="750px" height="250px"> -za tabelu
<td width="50px" height="20px"> -za ćeliju
```

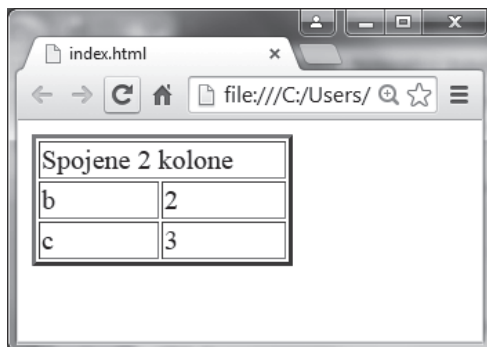
U realnim primenama, vrlo često postoji potreba za spajanjem ćelija u veće ćelije. Ovo spajanje može biti unutar kolona ili redova, čime se postiže vertikalno tj. horizontalno spajanje.

```
<table width="60%"
border="3px">
<tr>
<td rowspan="3">A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</table>
```



Slika 2.31. Prikaz efekta atributa rowspan u browser-u

```
<table width="60%"
border="3px">
<tr>
<td colspan="2">Spojene 2 kolone</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
</table>
```



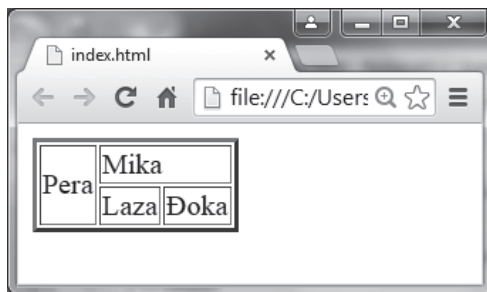
Slika 2.32. Prikaz efekta atributa colspan u browser-u

Pored pojedinačnih spajanja, kombinacijom atributa u istoj tabeli moguće je proizvoljno puno puta izvršiti spajanja unutar redova ili tabela. Primer kombinovanog spajanja je dat u listingu.

```

<table border="3px">
<tr>
  <td rowspan="2">Pera</td>
  <td colspan="2">Mika</td>
</tr>
<tr>
  <td>Laza</td>
  <td>Đoka</td>
</tr>
</table>

```



Slika 2.33. Prikaz efekta atributa *rowspan* i *colspan* u browser-u

Stilizovanje tabela realizuje se kroz attribute HTML tagova ili još preciznije pomoću CSS-a. U HTML-u se mogu realizovati osnovni stilovi primenom atributa *bordercolor* (za boju linija tabele), *b bgcolor* (za definisanje pozadinske boje) i *background* (za definisanje pozadinske slike). U zavisnosti u kom tagu se ovi atributi koriste, na različitim mestima će se njihov efekat i ispoljiti. Tako na primer, definisanje boje ivica tabele se realizuje kao

```
<table bordercolor="#00CC00">
```

dok se popuna tabele pozadinskom bojom tj. slikom realizuje kao

```

<table bgcolor="#000FF0">
<table background="adresa_foldera/pera.jpg">

```

U slučaju da se želi uticati samo na red, kolonu ili ćeliju atribut se stavlja u željeni tag. Na primer za definisanje pozadinske boje kolone tj. njenu popunu slikom koristimo

```

<td bgcolor="#000FF0">
<td background="adresa_foldera/pera.jpg">

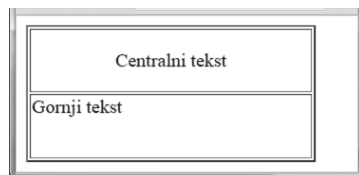
```

Pored stilizovanja u grafičkom domenu, stilizovanje se može realizovati i kroz poziciju sadržaja u tabeli tj. redu ili koloni. Za to se koristi atribut *align* (horizontalna poravnanja) tj. *valign* (vertikalna poravnanja). Vrednosti ovih atributa su definisane i mogu biti: *left*, *right*, *center* (za *align*) tj. *top*, *middle*, *baseline* i *bottom* (za *valign*).

```

<table border="2px"
width="90%" height="100%">
<tr>
  <td align="center"> Centralni tekst</td>
</tr>
<tr>
  <td valign="top"> Gornji tekst</td>
</tr>
</table>

```



Slika 2.34. Prikaz efekta poravnanja sadržaja ćelija tabele u browser-u

2.13. Organizacija sadržaja pomoću CSS-a

Pojavom HTML 4 organizacija sadržaja primenom CSS-a je postala moguća, a danas skoro da nema alternativu. Ovaj način organizacije je vrlo lak za kreiranje i održavanje, skalabilan i poželjan za sve web pretraživače. Iz tih razloga, možemo reći da je ne samo preporučljiv nego skoro i obavezan. Čak i kada je organizacija sadržaja realizovana pomoću CSS-a, pre svega je potrebno kreirati i određeni deo koda u HTML-u da bi CSS mogao da deluje. Za te potrebe koriste se dva taga: *div* i *span*.

Upareni tagovi *div* i *span* inicijalno nemaju grafičku interpretaciju u browser-u, i služe za grupisanje sadržaja na web strani. To znači da se sam tag ne vidi. *Span* je po pravilu jednoredni *div*, i češće se koristi za grupisanje teksta u jednoj liniji. Sa druge strane *div* se može koristiti za grupisanje svih vrsta sadržaja pa i drugih *div*ova.

I *div* i *span* se mogu shvatiti kao zagrade u matematici. Nemaju uloge operatora ali ih koristimo za grupisanje elemenata u iskazu i na taj način jasno definišemo redoslede prioriteta akcija. Isto tako, tag *div* koristimo da pomoću njegovih dimenzija i pozicije, definišemo jedan prostor unutar stranice. Tako *div* nema ulogu da bude vidljiv, nego da „rezerviš“ mesto gde tek treba da dođe neki sadržaj koji korisnik vidi. Na ovaj način struktura *div*ova se može zamisliti kao struktura prostorija u dokumentaciji za izradu kuće. Tačno znamo koliko prostorija ima, kolike su im dimenzije, gde se koja nalazi u odnosu na druge i sl. ali nemamo nameštaj jer ta kuća tek treba da se gradi. Isto tako se skiciranjem organizacije *div*ova pravi organizaciona struktura rasporeda unutar web strane, i svaki *div* je jedna prostorija. Sadržaj prostorije će se definisati kada se „kuća izgradi“, tj. sadržaj *div*a kada se završi prva faza pisanja HTML koda, koji kreira *div*ove. Ono što se želi prikazati na poziciji gde je neki *div* piše se unutar *div* elementa tj. otvorenog i zatvorenog taga tog *div*a. Sadržaj ovog taga je tekst, slike, linkovi, drugi *div*ovi i sl. što se kreira ranije opisanim HTML kodom.

Broj tagova *div* unutar strane nije ograničen pa samim tim ni broj logičkih celina koje se kreiraju nema limita. U realnim sajtovima, broj *div*ova je mnogo veći nego što je broj vidljivih delova stranice, što je posledica potrebe da se obezbedi „neraspadanje sajta“. Ova pojava se dešava zbog različite interpretacije pojedinih tagova, atributa i inicijalnih parametara u različitim browser-ima. Da bi se ovo izbeglo, pribegava se pravilu grupisanja *div*ova, u regione, čime se ovaj problem rešava u velikoj meri. Pravilo grupisanja je relativno lako: ukoliko u horizontalnoj strukturi, ili vertikalnoj, postoji više od jednog *div*a, gledano od krajnjih ivica browser-a, tih više pojedinačnih *div*ova se grupiše u jedan veći. Ovo pravilo se primenjuje dok se ne dođe do jednog jedinog *div*a, koji se žargonski naziva okvir (*wrapper*). Na ovaj način, ceo sajt je jedan *div*, unutar koga postoji složena unutrašnja hijerarhija.

Na primer, pretpostavimo da nam je potrebna osnovna struktura sajta kao na slici 2.35., dobijena na osnovu zahteva klijenta.

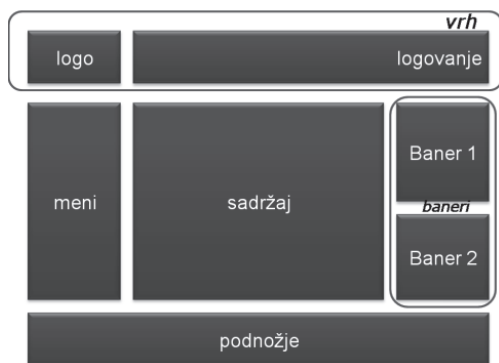
Ovakva šema treba da bude izlaz prve faze logičkog projektovanja (*Organizacija web stranice*).



Slika 2.35. Primer organizacije web stranice-u

Ono što dalje predstoji je pretvaranje ove ideje u kod. Ako pretpostavimo da je svaki od blokova jedan *div*, onda se pre izrade koda mora izvršiti grupisanje istih, u regione, po opisanom pravilu. Na taj način potrebno je prvo grupisati blokove *logo* i *logovanje*, jer se ta dva bloka nalaze u horizontalnom nizu između leve i desne ivice browser-a. Region koji ih grupiše označimo sa *vrh*. Na sličan način, sledeća manja grupa je *Baner 1* i *Baner 2*, jer su u vertikalnom nizu u visini ostalog levog dela sajta. Region koji ih grupiše označimo sa *baneri*. Grupisanje do ovog trenutka je prikazano na slici 2.36.

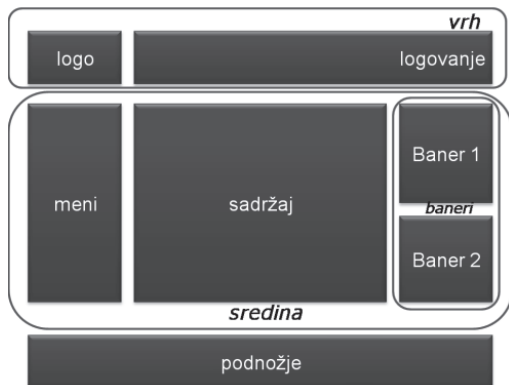
■ **Napomena:** U praksi se blokom naziva osnovna gradivna komponenta u organizaciji, tj. onaj deo čiji je sadržaj namenjen korisniku (tekst, slika, multimedija,...), i najčešće je to jedan *div* element, dok će se blokovi kojima se vrše grupisanja ovih osnovnih komponenti nazivaju regioni.



Slika 2.36. Primer grupisanja divova u prvom stepenu grupisanja

Posmatrano od vrha ka dole, *div vrh* je grupisao sve od leve do desne ivice. Međutim, u sledećem redu, horizontalno su postavljana tri diva: *meni*, *sadržaj*, *baneri*. U skladu sa pravilom, njih treba grupisati u jedan novi *div*, tj. region, sa nazivom *sredina*. *Div podnožje* je celom širinom browsera, i njega nema potreba za sada “dirati”. Novi korak u grupisanju prikazan je na slici 2.37.

Sada u sve tri horizontalne grupe postoje grupisanja od leve do desne ivice. Međutim, po vertikali postoje tri bloka: *vrh*, *sredina*, *podnožje*, koje po istom pravilima treba grupisati u jedan *div okvir*.



Slika 2.37. Primer grupisanja divova u drugom stepenu grupisanja

Na ovaj način došlo se do jednog regiona: *okvir*. Ovakav dizajn izrade strukture sajta je najčešći ali može odstupati u slučaju da su potrebni različiti efekti ili prikazi pojedinih blokova, pa se tada najčešće ne stavlja poslednji div (*okvir*.) Tako se pojedini blokovi tj. regioni mogu manje ili više razvlačiti unutar browser-a, i ne moraju biti podjednako poravnati unutar diva *okvir*.

Na ovaj način došlo se do jednog regiona: *okvir*. Ovakav dizajn izrade strukture sajta je najčešći ali može odstupati u slučaju da su potrebni različiti efekti ili prikazi pojedinih blokova, pa se tada najčešće ne stavlja poslednji div (*okvir*.) Tako se pojedini blokovi tj. regioni mogu manje ili više razvlačiti unutar browser-a, i ne moraju biti podjednako poravnati unutar diva *okvir*.

Kada je kreirana celokupna struktura divova, potrebno je napisati HTML kod koji odgovara ovoj strukturi. Pre nego se to uradi, treba naglasiti da je u postupku pisanja HTML koda potrebno omogućiti CSS kodu (koji će se kasnije napisati) da može da deluje na kreirani HTML kod, i tako se postigne konačni cilj.

Ova veza se realizuje preko HTML atributa (*id* ili *class*) taga *div* tj. *span*. Atribut *id* se koristi za svaki *div* ili *span* koji ima jedinstvene karakteristike, i označava skraćenicu od identifikator. Jedinstvenost se može ogledati u dimenziji, poziciji, boji i sl. Na taj način ako su svi divovi različiti, svako ima atribut *id*, sa posebnim imenom (npr. *id="prvi"*, *id="meni"*...). Sa druge strane, ako dva ili više divova (ili *span*-ova) imaju iste karakteristike (ali ne obavezno i sadržaj) koristi se atribut *class*. Za one divove koji imaju iste karakteristike koristi se ista vrednost atributa *class*. Tako dva diva koja imaju iste karakteristike imaju *class="oglas"*, dok neka druga dva sa drugim zajedničkim karakteristikama mogu imati *class="cena"*, itd.

U primeru sa slike 2.38, divovi za *Baner 1* i *Baner 2*, su istih dimenzija, iste pozicije (desno), iste strukture (npr. imaju samo neku sliku i link) i njih treba opisati sa *class*. Svi drugi divovi su unikatni i imaju atribut *id*.

Sada treba napisati HTML kod kada se zna kompletna struktura, imena divova i pripadnost *class* ili *id* atributa pojedinim divovima. Kod ćemo pisati gledajući sliku 2.38. u smeru odozgo na dole. Prvi div koji se vidi je div sa *id="okvir"*. Njegov HTML kod je:

```
<div id="okvir">
```

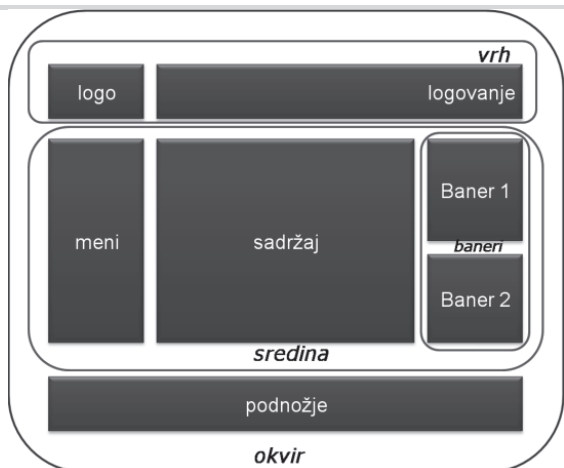
```
</div>
```

To implicira da treba kreirati tri elementa u redosledu: `<div id="vrh">`, pa `<div id="sredina">` i `<div id="podnozje">`. Srpska slova ne treba koristiti u imenima atributa ili imenima fajlova.

```
<div id="okvir">
  <div id="vrh">
  </div>
  <div id="sredina">
  </div>
  <div id="podnozje">
  </div>
</div>
```

U skladu sa dobrom praksom pisanja koda, svaki div, koji je unutar nekog drugog, se piše sa kodom koji je uvučen, da bi i vizuelno ukazao na pripadnost nadređenom divu.

Kako se unutar regiona *vrh*, nalaze dva diva: *logo* i *logovanje*, njihov kod treba pisati unutar diva *vrh*.



Slika 2.38. Primer grupisanja divova u poslednjem stepenu grupisanja

```
<div id="okvir">
  <div id="vrh">
    <div id="logo">
    </div>
    <div id="logovanje">
    </div>
  </div>
  <div id="sredina">
  </div>
  <div id="podnozje">
  </div>
</div>
```

Na sličan način u divu *sredina*, treba kreirati tri elementa: *meni*, *sadržaj* i *baneri*.

```

<div id="okvir">
  <div id="vrh">
    <div id="logo">
    </div>
    <div id="logovanje">
    </div>
  </div>
  <div id="sredina">
    <div id="meni">
    </div>
    <div id="sadrzaj">
    </div>
    <div id="baneri">
    </div>
  </div>
  <div id="podnozje">
  </div>
</div>

```

Unutar regiona *baneri*, treba upisati dva diva sa *class* atributom *baner*.

```

<div id="okvir">
  <div id="vrh">
    <div id="logo">
    </div>
    <div id="logovanje">
    </div>
  </div>
  <div id="sredina">
    <div id="meni">
    </div>
    <div id="sadrzaj">
    </div>
    <div id="baneri">
      <div class="baner">
      </div>
      <div class="baner">
      </div>
    </div>
  </div>
  <div id="podnozje">
  </div>
</div>

```

Ovim je kompletno formiran region *sredina*, a obzirom da element *div podnozje* nema unutrašnjih divova, ovim je završen kompletan HTML kod organizacije strane za strukturu na slici 2.35.

Obzirom da div nema inicijalnu grafičku prezentaciju, ceo ovaj kod pokrenut u browser-u neće prikazati ništa. On je samo prvi korak kojim se omogućava da deluje CSS kod, koji treba da definiše tačne dimenzije svakog od divova, margine, fontove, boje, itd. Čak i tada, korisnik neće videti ništa, ali je kompletna arhitektura strane definisana i čeka se sadržaj. Tom sadržaju divovi će definisati poziciju, a CSS boju, font, veličinu, margine itd.

Sadržaji konkretnih divova će se pisati između otvorenog i zatvorenog *div* taga, i biće određeni da prikažu željeni sadržaj.

2.14. HTML forme

Forma predstavlja skup grafičkih elemenata koji za cilj imaju da omoguće korisniku da određene podatke prosledi web serveru. Forma je jedini način na koji se korisnički podaci mogu proslediti web serveru sa ciljem da se oni dalje analiziraju, snime ili ažuriraju. Iz tog razloga forme predstavljaju vrlo bitan deo u formiranju interaktivnosti web sajta.

Forma je sastavljena od elemenata forme. Elementi forme se mogu grupisati kao: tekstualna polja, radio tasteri, check polja, drop down liste i tasteri.

Forma se u HTML kodu definiše uparenim tagom `<form>`. Ovaj tag nema grafičku interpretaciju, ali služi da se unutar njega definišu elementi forme, koji treba da čine jednu celinu. Na jednoj web strani može biti više formi, i one se međusobno razlikuju time što su grupisane u posebne *form* elemente.

Pored toga, tagom *form* omogućava se definisanje njegovih atributa koji su vrlo bitni za funkcionisanje forme. Atributi taga *form* su :

Action – URL adresa web stranice kojoj browser šalje sve podatke unete u formu, klikom na taster tipa *Submit*. Ako se šalje na mail, treba staviti *mailto:adresa*.

Method – je način slanja. Može biti *head*, *post* ili *get*. *Post* metod šalje podatke *HttpRequest/Responce*-om, pa su samim tim skriveni od korisnika. *Get* metod šalje podatke javnim putem, unutar URL adrese.

Title ili *id* – je ime forme, tj. njen identifikator, po kojoj joj može pristupiti viši programski jezik i po kojoj se međusobno razlikuje više formi unutar jedne web stranice.

Enctype – tip kodiranja pri slanju. Obavezan je kod uploada fajlova.

Primer upotrebe *form* taga sa njegovim atributima je dat u listingu.

```
<form name="formular_1" action="obrada.php" method="POST">
  <!-- Ovde dolaze elementi forme -->
</form>
```

Tag za najveći broj elemenata forme je `<input />`. Ovo je jedan od neuparenih tagova. Da bi se elementi forme, koji se definišu tagom `input`, međusobno razlikovali, koristi se atribut `type`. U zavisnosti od elementa forme, postoji još puno atributa koji se često koriste u realnim upotrebama: `id`, `name`, `size`, `maxlength`, `disabled`, `multiple`, `checked`, `value`, ...

2.14.1. Tekstualna polja

Tekstualna polja služe za unos teksta (*string*). U zavisnosti od vrednosti atributa `type`, imaju različitu grafičku interpretaciju, i namenu.

Prvi tip tekstualnog polja je `type="text"`. Ovo je najčešće korišćen tip tekstualnog polja. Grafička prezentacija ovog tipa polja je prikazana na slici 2.39.



Slika 2.39. Primer inicijalno popunjenog tekstualnog polja

```
<form>
<input type="text" id="ime" name="polje1" value="Ovo je inicijalni tekst"/>
</form>
```

Atribut `value` definiše inicijalno vidljiv sadržaj tekstualnog polja koji je editabilan. Sadržaj atributa `id` i `name` se ne prikazuju korisniku i služe višem programskom jeziku da “komunicira” sa elementom forme.

Često korišćeni atributi su `maxlength` i `size`. Pomoću `maxlength` definiše se maksimalna dužina sadržaja koji se može uneti, a `size` definiše dužinu tekstualnog polja koja se prikazuje korisniku.

```
<input type="text" id="ime" maxlength="10" size="30" />
```



Slika 2.40. Primer ograničenja dužine sadržaja u tekstualnom polju

Drugi tip tekstualnog polja je `type="password"`. Sadržaj ovog elementa forme se ne prikazuje u originalu, nego se svaki karakter menja nekim grafičkim znakom, u zavisnosti od browser-a (npr. zvezde ili tačke). Ovaj element forme se koristi za unos šifri.

```
<form>
<input type="password" id="ime2" name="polje2" value="Ovo je tekst"/>
</form>
```

Treći tip tekstualnog polja je *type*="file". Ovaj element se koristi za selekciju nekog fajla na računaru, tj. definisanje putanje do njega. Različiti browser-i ga različito grafički prikazuju, što je prikazano na slici 2.41. Ovaj element forme se koristi kod uploada fajlova.

```
<form>
<input type="file" name="ime" id="ime_id" />
</form>
```



Slika 2.41. Primer prikaza polja za izbor fajla u različitim browser-ima.

Četvrti tip tekstualnog polja je *type*="hidden". Sadržaj ovog elementa forme kao i sam element se ne prikazuje u browser-u, iako postoji u kodu. Iako to može delovati kao beskorisno, ovo polje se vrlo često koristi u realnim sajtovima. Ovaj element forme se koristi za smeštanje nekih informacija, koje programer želi da postoje na strani a da ih korisnik ne vidi, i koristi se kod rada sa višim programskim jezicima.

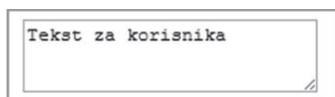
```
<form>
  <input type="hidden" id="ime1" name="polje1" value="Ovo je tekst"/>
</form>
```



Slika 2.42. Izgled browser-a prilikom upotrebe skrivenog polja

Peti tip tekstualnog polja je poseban tag *<textarea>*. Ovo je upareni tag i istorijski gledano je „malo stariji“. Sadržaj ovog elementa forme se koristi za kreiranje višerednog tekstualnog polja. Broj redova i kolona definiše se atributima *cols* i *rows*. Kod većeg broja browsera, atributi *cols* i *rows* definišu samo inicijalne dimenzije koje se prikazuju, dok se „razvlačenjem“ polja na donji desni ugao, njegove dimenzije mogu proizvoljno podešavati u browser-u korisnika.

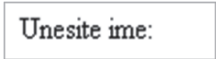
```
<form>
  <textarea id="ime2" name="polje2" cols="24" rows="3">
    Tekst za korisnika
  </textarea>
</form>
```



Slika 2.43. Prikaz višerednog tekstualnog polja u browser-u

Tag *label* se vrlo često koristi u strukturi web formi, iako u definiciji forme njoj striktno i ne pripada. Ovaj tag služi da se kroz njega prikaže statički tekst, koji korisnik ne može da menja. Namena ovog teksta je da definiše ime podatka kojim se u formi unosi (npr. tekst *Ime*, pre tekstualnog polja za unos imena korisnika.) Iz tog razloga ne spada striktno u elemente forme. Međutim, tag *label* je u višim programskim jezicima postao standard, naročito kada se radi o upravljanju sadržajem i pozicioniranju teksta, pa se preporučuje njegovo korišćenje i u HTML-u.

```
<label> Unesite ime: </label>
```



Slika 2.44. Prikaz taga *label* u browser-u.

Kako u formi često ima više elemenata forme, a samim tim i više *label*-a, poželjno je koristiti atribut *for*, kojim se određena *label*-a povezuje sa odgovarajućim elementom forme, koji opisuje.

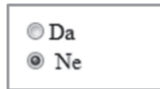
```
<label for="ime"> Unesite ime: </label>  
<input type="text" id="ime" value="ime" />
```

2.14.2. Radio tasteri

Radio tasteri su elementi forme koji se aktiviraju klikom miša i imaju isprogramiranu logiku. Ova logika omogućuje da se izborom jednog tastera, drugi selektovani taster automatski deselektuje. Na taj način samo jedan taster u grupi radio tastera može biti selektovan.

Radio taster se realizuje pomoću taga *input*, uz *type="radio"*. Atributom *checked* određeni taster se inicijalno označava selektovanim. Obzirom da u jednoj strani može biti više grupa radio tastera (npr. za pol, za bračni status, posao i sl.) potrebno je definisati grupu i radio tastere koji pripadaju toj grupi. Ovo se realizuje atributom *name*. Tako svi tasteri koji pripadaju jednoj grupi moraju imati isti sadržaj atributa *name*, da bi logika unutar browser-a kontrolisala da samo jedan taster u grupi može biti aktivan u jednom trenutku. Iako vrednost atributa *name* mora biti ista za sve elemente grupe, pojedinačni tasteri se međusobno razlikuju primenom atributa *id*, čija vrednost mora biti drugačija za svaki radio taster.

```
<form>  
  
<input type="radio" name="grupa1" id="broj1" value="ABC"/>Da <br/>  
<input type="radio" name="grupa1" id="broj2" value="DEF" checked/> Ne  
  
</form>
```

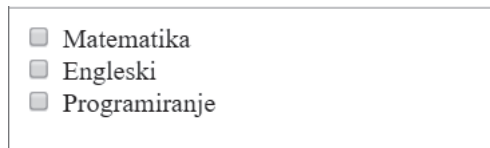


Slika 2.45. Prikaz grupe radio tastera u browser-u

2.14.3. Check polja

Check polja (*checkbox*) predstavljaju elemente forme koji se, kao i radio tasteri, označavaju (selektuju) klikom miša. Za razliku od radio tastera, kod kojih samo jedan može biti selektovan, kod *check polja* se može selektovati proizvoljan broj elemenata. Iz tog razloga nema inicijalne pripadnosti grupi, pa svaki *checkbox* ima drugačiju vrednost atributa *name* (ovo je poželjno), a naravno i atributa *id* (što važi uvek kao pravilo). Kao i kod radio tastera i *check polja* se mogu grupisati u logičke grupe. Pišu se u tagu *input*, a definišu se atributom *type="checkbox"*.

```
<form>
  <input type="checkbox" name="chbM" id="chbMa" value="Mat" />
  Matematika <br/>
  <input type="checkbox" name="chbE" id="chbEn" value="Eng"/>
  Engleski<br/>
  <input type="checkbox" name="chbP" id="chbPr" value="Prog"/>
  Programiranje
</form>
```



Slika 2.46. Prikaz grupe check polja u browser-u

2.14.4. Dropdown lista

Dropdown lista („padajuća lista“) je skup stavki objedinjenih u jednu celinu. Inicijalno, iz ove grupe, može se izabrati samo jedna od stavki. Dodatno, posebnim atributom *multiple*, može se omogućiti izbor više stavki istovremeno. Svaka pojedinačna stavka se piše unutar elementa *<option>*. Ove stavke su objedinjene u celinu pomoću elementa *<select>*.

Atributi *dropdown* liste, tj. taga *select*, su: *size* (broj inicijalno vidljivih opcija), *id*, *name*, *multiple*, *selected* (inicijalno vidljiva vrednost), *value* – ime/vrednost koja se vraća serveru,...

```

<form>

<select name="lista">
  <option value="pera">Petar</option>
  <option value="mika">Milenko</option>
  <option value="laza">Lazar</option>
</select>

```



Slika 2.47. Prikaz standardne dropdown liste u browser-u

```

</form>

<form>
<select name="lista" size="2">
  <option value="pera">Petar</option>
  <option value="mika">Milenko</option>
  <option value="laza">Lazar</option>
</select>
</form>

```



Slika 2.48. Prikaz dropdown liste sa atributom size u browser-u

```

<form>
<select name="lista" multiple>
  <option value="pera">Petar</option>
  <option value="mika">Milenko</option>
  <option value="laza">Lazar</option>
</select>
</form>

```



Slika 2.49. Prikaz dropdown liste sa svojstvom multiple u browser-u

Poseban oblik stilizovanja prikaza sadržaja dropdown liste je primenom taga *optgroup*. Ovim tagom moguće je grupisati elemente *option* unutar iste *dropdown* liste. Grupisanje se realizuje na nekoj logičkoj osnovi, dok se prikaz fizički razlikuje od klasične *dropdown* liste. Tag *optgroup* ima atribut *label* kojim se definiše podnaslov, posle koga se prikazuju elementi liste, uvučeni u odnosu na levu ivicu. Na taj način korisniku se unutar jedne liste prikazuje grafičko grupisanje elemenata, sa nazivom grupe i samog elementa. Izbor je moguć samo na nivou elementa dok se naziv grupe ne može selektovati.

```

<select>
  <optgroup label="Prva godina">
    <option value="matematika">Matematika</option>
    <option value="web_dizajn">Web dizajn</option>
    <option value="baze_podataka">Baze podataka</option>
  </optgroup>
  <optgroup label="Druga godina">
    <option value="SQL_server">SQL server</option>
    <option value="web_programiranje">Web programiranje</option>
    <option value="PHP">Web programiranje PHP 1</option>
  </optgroup>
</select>

```



Slika 2.50. Prikaz dropdown liste sa tagom *optgroup* u browser-u

2.14.5. Tasteri

Postoje tri različita tipa tastera, i potpuno se razlikuju po svojim funkcionalnostima. Svi se pišu u tagu `<input/>`, ali sa različitim atributima.

Svakom od tastera može se definisati inicijalni tekst, atributom *value*, koji se prikazuje kao tekst na samom tasteru. U jednoj formi može biti više tastera, čak i istog tipa.

Svakom tasteru se može dodeliti ime (atribut *name* ili *id*) da bi viši programski jezik mogao da „komunicira“ sa tasterom.

a) Taster tipa *submit* se koristi da se klikom na njega svi podaci iz forme pošalju ka stranici definisanoj u atributu *action*. Forma treba da ima ovaj taster ili taster tipa *button*.

```
<input type="submit" id="taster1" name="ime1" value="Prijava">
```

b) Taster tipa *reset* služi da se klikom na njega svi popunjeni podaci u formi ponište i forma vrati u inicijalno stanje.

```
<input type="reset" id="taster2" name="ime2" value="Ponisti">
```

c) Taster tipa *button* nije inicijalno isprogramiran i nema logiku rada kao tasteri tipa *submit* i *reset*. On se koristi kada postoji potreba da se klikom na taster realizuje neka posebna aktivnost. U tom slučaju programer treba da isprogramira rad samog tastera. Ovo se najčešće realizuje u Java script-u.

```
<input type="button" id="taster3" name="ime3" value="Izracunaj">
```

Kombinacijom elemenata forme, u proizvoljnom broju i redosledu, dobija se finalni izgled forme. Primer forme dat je u listing.

```

<html>
<head>
  <title>Prijavni formular</title></head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<form name="kreiranjemaila" method="post" action="">
  <b>Lični podaci</b><br/><br/>
  Ime: <input type="text" name="ime" /> <br/> <br/>
  Prezime: <input type="text" name="prezime" /> <br/> <br/>
  Pol:<br/>
  Muški <input type="radio" name="pol" value="pol" checked />
  Ženski <input type="radio" name="pol" value="pol" /> <br/> <br/>
  Adresa: <input type="text" name="adresa" /> <br/> <br/>
  Poštanski broj: <input type="text" name="postBroj" size="5"
maxlength="5" /> <br/><br/>
  Datum rođenja:
  <select name="mesec">
    <option>Januar</option>
    <option selected>Mesec</option>
    <option>Februar</option>
    <option>Mart</option>
    <option>Novembar</option>
    <option>Decembar</option>
  </select>
  <select name="dan">
    <option>1</option>
    <option selected>Dan</option>
    <option>2</option>
    <option>3</option>
    <option>29</option>
    <option>30</option>
  </select>
  Plaćanje:
  Keš<input type="checkbox" name="kes" value="checkbox" />
  Elektronski<input type="checkbox" name="elektronski" /><br/><br/>

  <input type="submit" name="potvrđi" value="POTVRDI">
  <input type="reset" name="odustani" value="ODUSTANI">
</form>
</body>
</html>

```

Lični podaci

Ime:

Prezime:

Pol:
Muški ☒ Ženski ☐

Adresa:

Poštanski broj:

Datum rođenja: Mesec Dan

Plaćanje: Keš ☐ Elektronski ☐

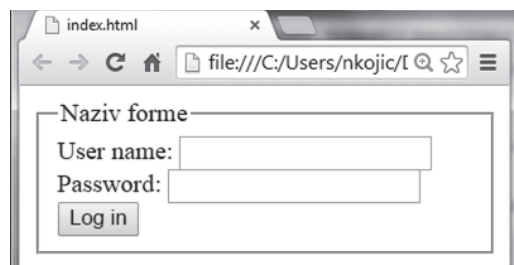
POTVRDI ODUSTANI

Slika 2.51. Primer forme u browser-u

2.14.6. Dizajn forme

Kao i drugi elementi HTML-a, tako se i elementima forme najbolje grafički upravlja pomoću CSS-a. Međutim, u HTML-u ima dva taga koji mogu da ponude određeni dizajn forme. Tag `<fieldset>` omogućava da se oko forme definiše okvir koji ima isključivo estetski karakter. Sadržaj ovog taga treba da bude unutar taga `form`. Tag `<legend>` služi da se u `<fieldset>` definiše ime koje se nalazi u gornjem levom uglu.

```
<form name="formular" method="post" action="pera.php">
<fieldset>
<legend>Naziv forme</legend>
  User name: <input type="text" name="user" /><br/>
  Password: <input type="text" name="pass" /><br/>
  <input type="submit" value="Log in" />
</fieldset>
</form>
```



Slika 2.52. Prikaz stilizovanja forme u browser-u

2.14.7. Organizacija elemenata forme

Elementi forme se u browser-u ponašaju kao i klasični karakteri, tj. prikazuju se na prvom slobodnom mestu u liniji koda u kojoj su napisani. Ovim se često dobija „nazubljen“ i vizuelno ne stilizovan prikaz elemenata forme. Jedan od načina da se elementi forme prikažu u organizovanom obliku je upotreba tabela. Treba naglasiti da je upotreba tabele u ovakve svrhe, sa stanovišta SEO-a, dozvoljena, ali ne i za organizaciju cele web stranice.

Posebnu pažnju treba posvetiti mestima gde se otvara tag `form` u odnosu na tagove tabele. Preporuka je da se za svaku željenu kombinaciju, takav tip ugnježdavanja tagova proveri unutar sajta <https://validator.w3.org/> kroz opciju *Validate by Direct Input*. U primeru je data forma za registraciju korisnika, čiji su elementi organizovani kroz tabelu.

```

<html>
  <head> <title>HTML formular</title></head>
  <body> <form action="#" method="POST">
    <table>
      <tr><th colspan="2" align="center">Registracija korisnika</th></tr>
      <tr><td>Ime:</td><td><input type="text" name="tbIme" id="tbIme" /> </td>
      </tr>
      <tr><td>Prezime:</td><td><input type="text" name="tbPrezime"
      id="tbPrezime" /> </td></tr>
      <tr><td>Datum rođenja:</td><td align="right">
        Dan: <select name="ddlDan" id="ddlDan">
          <option value="0">Izaberi....</option>
          <option value="1">1</option>
          <option value="2">2</option>
          <option value="3">3</option></select><br/>
        Mesec:<select name="ddlMesec" id="ddlMesec">
          <option value="0">Izaberi...</option>
          <option value="1">Januar</option>
          <option value="2">Februar</option></select><br/>
        Godina:<select name="ddlGodina" id="ddlGodina">
          <option value="0">Izaberi...</option>
          <option value="1990">1990</option>
          <option value="1991">1991</option>
          <option value="1992">1992</option></select></td>
      </tr>
      <tr><td>Pol:</td><td>
        <label for="muski">Muški</label> <input type="radio" name="rbPol"
        id="muski" /><br /><label for="zenski">Ženski</label><input type="radio"
        name="rbPol" id="zenski" /></td></tr>
      <tr><td>Korisničko ime:</td>
        <td><input type="text" name="tbKorisnickoIme" id="tbKorisnickoIme" />
        </td></tr>
      <tr><td>Lozinka:</td>
        <td><input type="password" name="tbLozinka" id="tbLozinka" /></td>
        </tr>
      <tr><td>Lozinka ponovo:</td><td>
        <input type="password" name="tbLozinkaPonovo" id="tbLozinkaPonovo" />
        </td></tr>
      <tr><td><input type="submit" name="btnRegistruj" id="btnRegistruj"
        value="Registracija" /></td><td>
        <input type="reset" name="btnPonisti" id="btnPonisti" value="Poništi" /></td>
        </tr>
    </table>
  </form>
</body>
</html>

```

Interpretacija ovakvog koda u browser-u je prikazana na slici 2.53. Očigledno je da je ovakav oblik organizacije, iako se ivice tabele ne prikazuju korisniku, daleko efektivniji nego što je to slučaj sa formom na slici 2.52.

Registracija korisnika

Ime:	
Prezime:	
Datum rođenja:	Dan: Izaberi... ▼ Mesec: Izaberi... ▼ Godina: Izaberi... ▼
Pol:	Muški ☐ Ženski ☐
Korsničko ime:	
Lozinka:	
Lozinka ponovo:	
Registracija Poništi	

Slika 2.53. Prikaz organizacije elemenata forme

U ovom primeru treba primetiti načine definisanja atributa name i id. Uobičajena praksa kod programera, ali ne i pravilo, je da se u kodu naglašava o kom elementu forme se radi i da se njegovo ime dodatno istakne. Tako se za tekstualna polja često koristi prefiks *tb*, za padajuću listu *ddl*, radio taster *rb*, checkbox taster *cb* i za taster *btn*. Ime elementa forme, koje se piše iza jednog od navedenih prefiksa, se započinje velikim slovom, da bi se jasnije uočilo. Tako se ime tekstualnog polja za ime, definiše kao:

```
name="tbIme"
```

dok je npr. padajuća lista za mesec definisan sa imenom

```
name="ddlMesec"
```

Preporuka je da se početnici navikavaju na ovakav način pisanja koda, iako je on duži, i možda inicijalno teži, ali vremenom se on usvoji i postane lepa praksa.

2.15. Meta tagovi

Meta tagovi označavaju veliku grupu neuparenih tagova *meta*. Ovi tagovi se pišu unutar *head* sekcije, i treba da opišu neki od karakterističnih termina namenjenih browser-u ili *spider*-u. Ovi podaci se odnose na sadržaj web strane, autora, intelektualna prava, opis strane, ključne reči, mail-u autora, kodnom rasporedu, informacijama za *spider*-a itd.

Postoji veliki broj meta tagova, i oni se međusobno razlikuju svojim atributima. Skup atributa koje je moguće koristiti prilikom kreiranja meta taga su: *charset*, *content*, *http-equiv*, *name* i *scheme*.

U realnim primenama najčešća je upotreba sledećih meta tagova (u HTML 4 sintaksi):

1. Meta tag za definisanje ključnih reči web stranice. Npr. definisanje ključnih reči web, programer, dizajn i sajt bi bilo

```
<meta name="keywords" content=" web, programer, dizajn, sajt "/>
```

2. Meta tag za definisanje opisa web stranice. Npr. opis stranice „Naučite da napravite sajt. Kursevi web programiranje i web dizajna!“ je

```
<meta name="description" content=" Naučite da napravite sajt. Kursevi web programiranje i web dizajna!"/>
```

3. Meta tag za definisanje kodnog rasporeda, čime se omogućava prikaz srpskih ćiriličnih i latiničnih slova:

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
```

ili

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250" />
```

4. Meta tag za definisanje kontakt podataka o autoru sajta:

```
<meta name="author" content="mailto: nkojic@singidunum.ac.rs " />
```

5. Meta tag za autorska prava:

```
<meta name="copyright" content="Moja firma @ 2010" />
```

Pored ovih najčešće korišćenih *meta* tagova, koriste se i:

1. Automatsko „refresh-ovanje“ sadržaja web stranice, npr. svakih 1800 sekundi:

```
<meta http-equiv="Refresh" content="1800" />
```

2. Definisanje meta taga za apstrakt web strane

```
<meta name="abstract" content="Preduzeće posluje više od 10 godina i ima vrlo dobre rezultate kako u zemlji tako i u inostranstvu!" />
```

3. Prikaz sadržaja u skladu sa standardom određene verzije browser-a

```
<meta http-equiv="X-UA-Compatible" content="IE=8" />
```

4. Meta tag za opis lokacije

```
<meta name="zipcode" content="11000" />
```

5. Meta tag za opis datuma objave nekog sadržaja

```
<meta http-equiv="expires" content="Tue, 28 Feb 2015 08:30:00 GMT"/>
```

6. Meta tag za definisanje jezika u kome je pisan sadržaj

```
<meta name="language" content="spanish" />
```

7. Meta tag za davanje instrukcija *spider*-u o indeksaciji web stranice:

```
< meta name ="robots" content ="noindex, follow" />
```

Tag *Robots* služi da pretraživaču ukaže da li datu stranu treba indeksirati (vrednost: *index*) ili ne (*noindex*), kao i da li da prati linkove definisane na strani ili ne (*follow/nofollow*). Proširenje meta taga *Robots* može biti fajl *Robots.txt* koja mora biti unutar *root*-a sajta. Ovim se pretraživačima može zabraniti indeksiranje strane što je korisno kod zaštite sadržaja.

2.16. xHTML

Sve što je do sada objašnjeno u vezi sa HTML-om, opisuje strukturu i način korišćenja ovog markap jezika. Ono što je primarna karakteristika HTML-a je da je baziran na tagovima, i da služi za prikaz teksta, slika, linkova,... Tagovi kojima se ovo postiže su unapred definisani, sa unapred definisanim pripadajućim atributima. Ovakva striktna struktura, definisani skup tagova i njihovih primena, je posledica činjenice da je HTML verzija ranije opisanog SGML-a. Sa druge strane, načini primene HTML-a su vrlo fleksibilni: moguće je otkucati pogrešno ime taga, moguće je ne zatvoriti ispravno tag, upisati pogrešno ime atributa, dodeliti atribut tagu koji ga ne podržava, upisati pogrešno vrednost atributa, pogrešno ugnjezditi elemente,... i kod će se opet prikazati. Browser će prikazati sve što „ikako“ može uz „pretpostavke“ kako bi to trebalo da bude, gde god je to moguće. Ako se ovome doda da pojedini browser-i rade na računarima, neki na prenosivim uređajima, mobilnim telefonima itd. prikaz sadržaja web strane u ovim uslovima može drastično da se razlikuje, što je vrlo loše.

Suprotno ovim karakteristikama, definiše se drugi bitan markap jezik XML. Sa jedne strane, on nema ograničen broj i imena tagova, pa programeru daje mogućnost da sam uredi i definiše sve što mu je potrebno. Primarno, XML-om se može definisati bilo koja vrsta podataka ili sadržaja koje želi da se podeli na Web-u. Iako vrlo fleksibilan u mogućnostima, XML je baziran na vrlo striktnim sintaksnim pravilima, sa mogućnošću parsiranja dokumenta. Za razliku od HTML-a, kod XML-a se ne sme dogoditi bilo koji oblik neregularnog zapisa (pogrešno zatvaranje taga, ugnježdavanja, dodela vrednosti i sl.).

Kako je mogućnost rada sa deljenjem bilo kod tipa sadržaja, koju donosi XML, vrlo poželjna u odnosu na „ograničeni“ HTML, napravljeno je jedno „hibridno“ rešenje: xHTML. Tako je xHTML skraćenica od *EXtensible HyperText Markup Language*, i predstavlja poseban markap jezik. Ovaj jezik je skoro identičan HTML-u, samo što je striktniji u smislu dozvoljenih grešaka i sintakse. Kao takav, predstavlja XML aplikaciju, i podržan je u skoro svim većim browser-ima.

xHTML se prvi put pojavljuje 2000. godine, i od tada ima vrlo dinamično napredovanje.

Primarna ideja kod xHTML je unapređenje mogućnosti HTML-a primenom mogućnosti XML-a. Na taj način, kod xHTML-a se može upotrebiti i neki novi tag, atribut, podatak,... pre nego se on npr. standardizuje i postane sastavni deo HTML-a.

Ovo je vrlo pogodno imajući u vidu da je vrlo malo potrebno promeniti u dosadašnjem načinu pisanja HTML-a, a za uzvrat dobiti nove mogućnosti za primenu jezika.

Tako se kod xHTML-a pravila mogu definisati u nekoliko koraka:

1. Definisanje „tipa dokumenta“ kroz DOCTYPE, na samom početku koda, je obavezno
2. Tag *html* mora imati atribut *xmlns*
3. Tagovi *html*, *head*, *title* i *body* su obavezni
4. Svi tagovi moraju biti ispravno ugnježdeni
5. Svi tagovi moraju biti ispravno zatvoreni (npr. `<br/>`)
6. xHTML dokument mora imati tačno jedan root element
7. Imena tagova i atributa treba da se pišu malim slovima
8. Vrednosti atributa moraju biti unutar znakova navoda
9. Skraćivanja u pisanju koda nisu dozvoljena npr. pogrešna primena *checked* bi bila

```
<input type="checkbox" id="a1" checked />
```

a ispravna

```
<input type="checkbox" id="a1" checked="checked" />
```

■ **Napomena:** Detaljno objašnjenje pojedinih navedenih termina koji pripadaju XML-u će biti dato u poglavlju VII, kada se XML bude obrađivao.

Prednost upotrebe xHTML-a je što se njime umnogome rešava problem različitog prikaza sajta u različitim browser-ima i rada sa višim jezicima.

Imajući u vidu da je u toku ovog poglavlja najveći broj ovih pravila već bio sugerisan i za pisanje HTML-a, jedina novina se može pronaći u tačkama 1. i 2.

Tako na primer, dati kod predstavlja jedan ispravan primer xHTML-a.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Naslov stranice</title>
</head>
<body>
    <p> Ovo je neki pasus</p> Toliko <br />Nenad
</body>
</html>

```

Kao što je prikazano u listingu, u odnosu na dosada obrađeni HTML, dodat je samo deo `<!DOCTYPE...>` i atribut u `html` tag `<html xmlns="http://www.w3.org/1999/xhtml">`. Ako za sada pretpostavimo da je sadržaj ove dve linije koda uvek isti, onda se zaključuje da kreiranje XHTML-a ne predstavlja problem, nakon poznavanja jezika HTML.

III CSS

Cascading Style Sheets

CSS predstavlja skup pravila kojima se bliže uređuje stilizovanje i formatiranje dokumenata napisanih primenom markap jezika. CSS nije ni programski ni markap jezik, već predstavlja tekstualni fajl kojim se uređuje grafički prikaz markap jezika. Efekat delovanja CSS-a vidi se u nekom softveru a najčešće browser-u.

CSS se pojavio kao podrška i pomoć programerima, nakon standardizacije verzije HTML-a 3.2. U ovoj verziji HTML-a, pored osnovnih tagova, pojavljuje se i upareni tag *font* namenjen za stilsko uređivanje sadržaja web stranice. Tag *font* je delovao kroz svoje atribute, dok on sam nema grafičku interpretaciju. Ovo je posebno bilo važno obzirom da su, u to vreme, različiti fontovi, boje, veličine i stilovi postali vrlo atraktivni i često korišćeni prilikom izrade web sajtova. Zbog velike „količine“ stilova, upotreba taga *font* je postala vrlo robusna, a stranice preopterećene kodom, pa je i njihovo održavanje postalo vrlo komplikovano.

U tom cilju, kreiran je CSS sa idejom da postojećem HTML kodu i njegovom sadržaju obezbedi dve stvari: stilizovanje sadržaja i organizaciju sadržaja.

Stilizovanje je trebalo da omogući lako definisanje stilova i upravljanje estetikom prikaza sadržaja (boja, font, veličina, linije, rastojanja, pozadine stranica, formatiranje...). Sa druge strane, organizacija sadržaja treba da uredi

pozicioniranje (poravnanje) pojedinih blokova ili regiona (divova) unutar web strane, i da se na taj način izbegne upotreba tabela ili frejmova.

Iz tog razloga, ova dva aspekta treba potpuno odvojeno posmatrati i realizovati.

U prvoj fazi kreiranja sajta (Poglavlje 2.4.), u postupku organizacije web stranice, HTML kodom se definišu svi potrebni divovi, a CSS-om njihova veličina i pozicija. Veličina i pozicija se inicijalno definišu početnom skicom i idejnim rešenjem, tj. gde ti blokovi treba da se nađu i kako su međusobno postavljeni. Tu ideju, primenom koda, CSS treba i formalno da omogući. Nakon pisanja ovog dela CSS koda, koji pripada organizaciji sadržaja, prelazi se u drugu fazu.

Ova faza podrazumeva da se unutar kreiranih blokova, koji su sada ispravno pozicionirani, unese sadržaj za korisnika (Poglavlje 2.4. faza Sadržaji pojedinačnih strana). Ovaj sadržaj definiše se HTML-om, ali se njegov kompletan stilski dizajn definiše CSS-om. Ovo je druga navedena uloga CSS-a: stilizovanje sadržaja.

Obe ove faze se realizuju kroz objedinjeni CSS fajl i deluju na istu HTML stranicu, ali sa različitim ciljem i efektima.

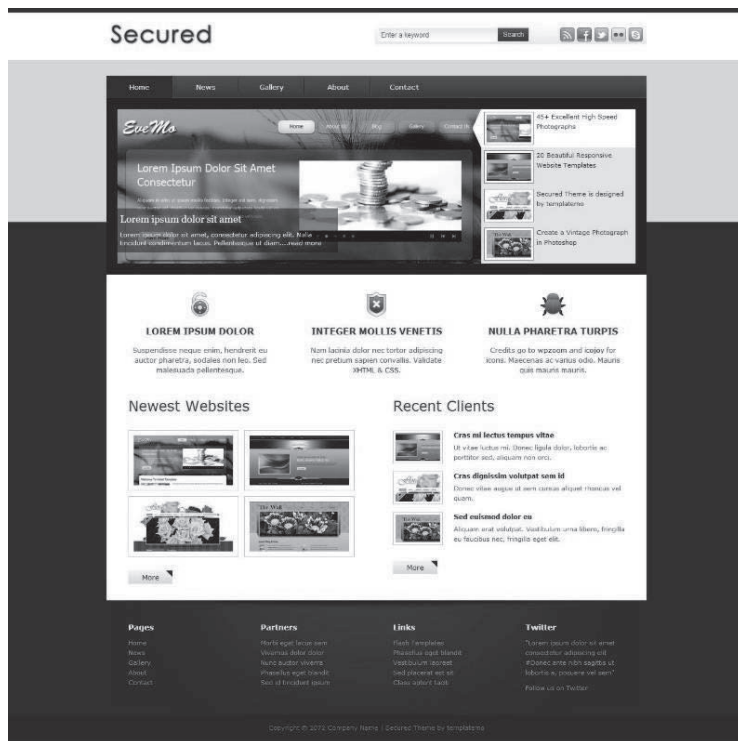
Istorijski posmatrano, prva verzija CSS-a, CSS 1, je realizovana 1996. godine. Ova verzija podržavala je osnovno stilizovanje (boja, font i pozadinska slika). Iste godine, razvijen je Internet Explorer 3, koji je bio prvi komercijalni browser koji je podržao CSS, iako još nije postao W3C preporuka. Nakon IE3, Netscape Navigator 4.0 je bio sledeći browser koji je podržao CSS, i tako su polako počeli i drugi. Iako deluje progresivno, ovaj postupak je išao sa dosta problema, pa je čak i naredna verzija IE4 imala problema u potpunoj primeni CSS-a.

CSS 2 se pojavljuje 1998. godine i donosi velika poboljšanja u odnosu na karakteristike CSS 1. Verzija CSS 2 omogućava pozicioniranje (poravnanje) elemenata, podršku za fontove koji se preuzimaju iz nezavisnih fajlova, formatiranje stranice za štampu itd. Na taj način, HTML dobija podršku za koju se nije moglo ni pretpostaviti u kom smeru će krenuti. Sa druge strane, čak i tada samo Opera 6 i Netscape 6 imaju potpunu podršku za CSS 2, dok je drugi imaju delimično ili čak uopšte nemaju. U tom periodu CSS počinje sve više da se primenjuje i na stilizovanje XML dokumenta, što mu daje potpuno novu dimenziju i pojačava ulogu CSS-a.

Tek posle 2011. godine se i zvanično pojavljuje CSS 3, koji je doneo velike promene u odnosu na svoje prethodnike. CSS 3, je za razliku od prethodnika podeljen u module, kojih ima preko pedeset. Sada su na raspolaganju nove mogućnosti u domenu animacije i tranzicije, naprednih selektora, sadržaja, gradijenata, web fontova, ivica, multimedije, pozadina, rada sa kolonama, 3D animacija itd.

Sve ovo ukazuje na veliki potencijal koji CSS ima, i opravdava razlog zašto skoro svi web pretraživači „vole“ organizovanje i stilizovanje sadržaja web stranice primenom CSS-a.

Da bi se bolje razumeo uticaj CSS-a na neki realni web sajt, posmatrajmo sajt na slici 3.1, koji je organizovan i stilizovan primenom CSS-a.



Slika 3.1. Prikaz sajta organizovanog CSS-om

Iako je u štampanoj, crno-beloj, verziji teško kompletno dočarati efekte boje, i ovakav primer će biti više nego ilustrativan. Ovaj sajt predstavlja standardno moderan dizajn sajta sa klasično stilizovanim sadržajem. Treba primetiti različite veličine i fontove teksta, pozicije i grupisanja slika, veličine slika, raspored ikonica, dizajn, poziciju i raspodelu boja, poziciju i dizajn footer-a... Ono što se želi istaći je uloga koju CSS ima u ovom konkretnom sajtu. CSS je ovde korišćen i za organizaciju sadržaja i za stilizovanje, što znači da istovremeno grupiše i pozicionira pojedine delova sajta i uređuje način prikaza. Ako se ovom sajtu ukine ceo CSS kod, dobija se sajt kao na slici 3.2.



Slika 3.2. Prikaz sajta sa slike 3.1. bez uticaja CSS koda

Skoro da je nemoguće prepoznati originalni sajt:

1. Pozadinska boja i centralizovana struktura ne postoje,
2. Ikonice ka socijalnim mrežama ne da nisu samo na desnoj strani, nego su sada i vertikalno poredane
3. Horizontalni navigacioni meni skoro da ne postoji
4. Centralni slajder se “raspao”

5. Kompletan centralni deo sajta je “raspadnut”

6. Stilovi, u smislu boje, fonta, pozicije teksta i slično su nestali, ...

Sve ovo ukazuje da je HTML definisao samo vrlo grub originalni i nepromenljiv sadržaj: tekst, linkove, slike, ... ali da je CSS definisao sve ostalo: poziciju gde se sadržaj nalazi (gore, dole, centar, ...), poravnanje, margin-e, padding-e, pozadinsku boju, boju teksta, promenu inicijalnog fonta, veličine i boje, veličina i pozicije slika, ...

Iz tog razloga, CSS je postao skoro neprikosnoven u primeni jer su njegovi efekti vrlo uočljivi, a pored toga je programerima umnogome ubrzao i olakšao rad. Zato se u realnim web sajtovima mnogo više vremena i pažnje posvećuje CSS-u nego inicijalnom HTML-u.

3.1. Struktura CSS fajla

Već je napomenuto da CSS nije programski jezik već se može reći da je to skup pravila. U tom smislu, CSS je običan tekst i ukoliko se piše u posebnom fajlu ima ekstenziju *.css*. Inicijalno je definisano da se ceo CSS sadržaj bazira samo na dva gradivna bloka: *selektor* i *deklaracija*.

Selektor ukazuje na koji HTML element (ili element drugog markap jezika) se primenjuje određeno pravilo. Na taj način, selektor je veza između originalnog HTML koda i CSS-a.

Deklaracija definiše, jedno ili više, svojstava i njegovih vrednosti. Svojstvo ukazuje na efekat koji se želi postići u konačnom izgledu.

■ **Napomena:** Žargonski se kaže da CSS-a selektor kaže *gde* a svojstvo *kako* treba delovati!

Svojstvo je definisano određenom rezervisanom rečju, koja mora biti podržana u određenoj verziji CSS-a. Tako na primer, neka od svojstava CSS-a, su: *font-size*, *color*, *margin*, *padding*, *text-align*, *border*, ... Kao što im samo ime ukazuje, svojstva definišu na koji stilski detalj će se vršiti uticaj. Međutim, intenzitet tog uticaja, ili efekat, definiše *vrednost svojstva*. Tako na primer, vrednost svojstva *font-size* može biti 18px, a svojstva *color* može biti *green*. Operator dodele vrednosti u CSS je *:*. Tako svojstvo sa njegovom vrednošću pišemo kao

```
font-size: 18px
```

Ovaj „paket“ se može ponoviti više puta za više svojstava i međusobno se razdvaja tj. završava sa *;*, a piše unutar *{ }*. Ime selektora, na koji se svojstvo ili grupa svojstava, odnosi, piše se ispred *{ }*.

Tako je kompletna sintaksa npr. selektora *p*, sa svojstvima *font-size* i *color*, i njihovim vrednostima 18px tj. *green* data sa:

```
p{  font-size: 18px;
    color: green; }
```

Ovaj kod ukazuje da kod svakog elementa `p`, prilikom prikaza njegovog sadržaja, treba primeniti veličinu teksta od 18px i tekst obojiti zelenom bojom. Kao što se vidi, CSS ne definiše koji je to tekst, koliko ga ima i slično (za šta je zadužen HTML) nego samo način njegovog prikaza u stilskom domenu.

Ukoliko neki sadržaj na sajtu nije obuhvaćen selektorom i svojstvima CSS-a, on će se prikazivati po inicijalnim pravilima browser-a (podrazumevanim vrednostima browser-a). Kako ove vrednosti nisu iste za sve browser-e, i njihove verzije, skoro je obavezno da se sve eksplicitno definiše unutar CSS-a. Ovo se odnosi i na one selektore i njihove vrednosti koje često smatramo podrazumevanim, ali se zbog mogućih različitosti browser-a, može dogoditi da se isti sajt različito interpretira u različitim browser-ima (npr. margine 0px, padding 0px, nema pozadine, nema okvirnih linija, ...). Iz ovog razloga, u profesionalnim sajtovima, veliki početni deo koda CSS fajla, je definisanje ovih podrazumevanih vrednosti (čime se browser-ima ne dozvoljava da koriste svoje podrazumevane vrednosti), pa tek onda kreće operativni deo koda kojim programer definiše ono što želi da postigne.

Spisak svojstava, i njihovih mogućih vrednosti, kao i njihova pripadnost pojedinim selektorima, su definisana pravilima tj. standardima određene verzije CSS-a, i ima ih relativno puno, pa je najbolje polako ih usvajati kroz praksu.

3.2. CSS selektori

Kao što je rečeno, prvi korak u pisanju CSS-a je definisanje selektora, koji treba da ukažu na koji HTML element će delovati CSS kod. Selektori se mogu realizovati kroz nekoliko različitih oblika (elementi, atributi, tipovi, vrednosti atributa itd.). U prvoj podeli, selektore možemo da podelimo na: *ime taga*, atribut *id* ili atribut *class*.

3.2.1. Selektor u formi imena taga

Ovi selektori se odnose na originalno ime HTML taga. Tako na primer ako se definiše određeni stil, sa selektorom koji je tag *h1*, kao

```
h1{  font-size: 18px;
    color: green;
    text-align: right; }
```

sadržaj svih *h1* elemenata u toj web strani, bez obzira na njihov broj i poziciju, će biti u veličini 18px, zelene boje i poravnat uz desnu stranu.

Ovo je prvi pokazatelj skraćivanja pisanja koda primenom CSS-a, jer je na samo jednom mestu moguće definisati stil za sve npr. *h1* tagove, bez obzira koliko ih u stranici ima, što nije slučaj ako se koristi tag *font*.

Dozvoljeno je dodatno skraćivanje ukoliko postoji više različitih elemenata na koje treba primeniti isti stil. Tako ako se ista grupa pravila želi primeniti na sve *h2* i *p* tagove, to je moguće skraćeno pisati kao:

```
h2, p{ font-size: 18px;
      color: green;
      text-align: right; }
```

Često korisnici greše kod upotrebe zareza između selektora iz prethodnog primera. Naime, kako je rečeno, zarez služi da razdvoji više stilova sa istim deklaracijama. Sa druge strane izostanak zareza ima potpuno drugo značenje, tj. ukazuje na pripadnost nadređenom elementu unutar DOM strukture. Tako npr. ako imamo neuređenu listu i definišemo sledeći CSS kod

```
ul { list-style-type: none;
    }
ul li { display: block;
      float: left;
    }
ul li a {color: #f2f2f2;
    }
```

Definisali smo stil za celu listu, kroz selektor *ul*, za pripadajuće elemente liste kroz selektor *ul li*, i za elemente koji se nalaze u pojedinim elementima liste, što je u ovom slučaju link, sa selektorom *ul li a*. Ovako definisani selektori se odnose na sve liste, elemente liste i sadržaje elemenata liste u toj web strani.

3.2.2. Selektor u formi atributa id

Ovaj selektor koristi vrednost atributa *id*, u bilo kom od HTML tagova. Tako na primer ako se definiše element `<div id="primer1"> </div>`, selektor je string *primer1*, uz prefiks #, koji se koristi kada je u pitanju atribut *id*. Tako je stil koji se definiše u CSS-u za ovaj selektor

```
#primer1 {
    float: left;
    color: red; }
```

U ovom primeru, sav sadržaj koji se nalazi unutar `<div id="primer1"> </div>` će biti podvrgnut definisanom CSS stilu. To može biti običan tekst, ili grupa drugih ugnježđenih elemenata sa njima pripadajućim sadržajima. Na sve njih će se primeniti definisani stil. Primena ovog stila, ne sprečava mogućnost da se

nekim drugim selektorom, nad sadržajem unutar posmatranog elementa, dodatno definiše još neki stil.

Selektor u formi atributa *id* se koristi kada se definiše jedinstven skup pravila. Na taj način HTML element mora imati unikatnu vrednost atributa **id**, u toj stranici, čime to postaje njegov jedinstveni identifikator.

3.2.3. Selektor u formi atributa class

Suprotno od upotrebe atributa *id*, atribut *class* omogućava da se isto pravilo u CSS-u može dodeliti većem broju istih, ili različitih, elemenata koji imaju istu vrednost atributa *class*. Prilikom korišćenja atributa *class* koristi se vrednost ovog atributa i . kao prefiks.

Tako na primer, možemo imati dva *div* elementa i *p* element, koji imaju istu vrednost atributa *class*.

```
<div class="primer2"> ABC</div>
<div class="primer2"> 123</div>
<p class="primer2"> www ooo ccc</p>
```

a samo jedan CSS stil kojim se upravlja sadržajem sva tri elementa

```
.primer2 {    float: left;
              color: red;    }
```

Obzirom da je ponekad selekcija imenom taga vrlo gruba, jer se odnosi na sve te elemente, dozvoljeno je da se izvrši dodatno „filtriranje“ selektovanih elemenata. Tako ako se, na primer, neki stil ne želi primeniti na sve *p* elemente redom, nego samo na neke, željenim tagovima se definiše atribut *class* sa istom vrednošću.

Na taj način, selekcijom taga *p* i atributa *class*, određeni stil se primenjuje na one *p* elemente koji imaju *class* atribut sa tom vrednošću (npr. *.primer3*).

```
p.primer3 {    float: left;
               color: red;    }
```

U realnim primenama atribut *class* se često koristi sa više od jedne vrednosti. Na taj način programer može da definiše nekoliko različitih stilova i da ih proizvoljno kombinuje. Na primer, ako se definišu dva stila: *prviStil* i *drugiStil*, i primene se kombinovano u jednom divu pomoću *class="prviStil drugiStil"* dobija se zbirni efekat kao na slici 3.3.

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    .prviStil { color: #FF00CC; }
    .drugiStil { background-color: yellow; }
</style>
</head>
<body>
    <div class="prviStil drugiStil">Neki sadrzaj </div>
</body>
</html>

```



Slika 3.3. Grafički prikaz delovanja CSS-a preko više vrednosti atributa class

U slučaju da se koristi više od jedne vrednosti u atributu *class*, stilovi se mogu proizvoljno kombinovati. Ovakve primene su vrlo česte kada se definiše „ugnježdavanje“ i kada postoji jedan stil koji važi za sve, pa za stavku unutar prve stavke, pored osnovnog stila dodaje se još jedan, i tako redom, da bi se opšte karakteristike zadržale, a pojedine stavke međusobno razlikovale.

Međutim, programeru je na raspolaganju dodatna mogućnost selekcije. Tako, ako se u CSS-u definiše poseban stil, koji je dobijen kombinacijom postojećih, npr. *.prviStil.treciStil*, njegovo pravilo će se primeniti na sve elemente gde se unutar višestrukih vrednosti atributa *class* između ostalih, nalaze i ta dva stila. Primer ovakve upotrebe je dat u listingu.

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    .prviStil { color: #0000FF;}
    .drugiStil { background-color: yellow;}
    .treciStil { border: 3px solid #FF0000;}
    .prviStil.treciStil { font-style: italic;font-weight: bold;}
</style>
</head>
<body>
    <div class="prviStil drugiStil">Neki sadrzaj </div>
    <div class="prviStil drugiStil treciStil">Neki sadrzaj </div>
    <div class="prviStil treciStil">Neki sadrzaj </div>
</body>
</html>

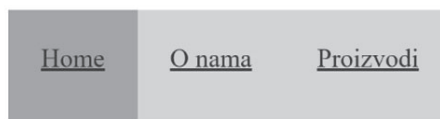
```



Slika 3.4. Grafički prikaz filtriranja CSS-a sa više vrednosti atributa *class*

Na sličan način kao u ranijem primeru, stavljanjem selektora u formi atributa *id* može se precizno selektovati veća grupa elemenata, a oni dalje prozivati po „dubini“. U primeru neuređene liste posmatrajmo sledeći kod:

```
<html>
<head>
<style>
#meni { list-style-type: none;}
#meni li {
    display: block;
    height:30px;
    float: left;
    padding:20px;
    background-color: #dddddd;
}
#meni .active { background-color: blue;}
#meni li a {
    color: red;
}
</style>
</head>
<body>
    <ul id="meni">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#">O nama</a></li>
        <li><a href="#">Proizvodi</a></li>
    </ul>
    <br/><br/><br/>
    <ul id="najava">
        <li>Vesti 1</li>
        <li>Vesti 2</li>
        <li>Vesti 3</li>
    </ul>
</body>
</html>
```



- Vesti 1
- Vesti 2
- Vesti 3

Slika 3.5. Izgled browser-a za za dati kod

U ovom primeru postoje dve liste na istoj stranici, ali se stavljanjem selektora *#meni* pristupa tačno prvoj, dok druga može imati potpuno drugačija pravila. Takođe, sa selektorom *#meni li*, se pristupa svim *li* elementima ove liste. Ako se jednom elementu ove liste, i to prvom, stavi posebna klasa *class="active"* onda se pristup samo ovom elementu liste vrši selektorom *#meni .active*. A kada bi hteli da pristupimo samo linku ovog elementa selektor bi bio *#menu .active a*. Na

ovaj način kombinacijom selektora moguće je vrlo precizno i jednoznačno pristupiti svakom elementu strane, ili većoj grupi njih istovremeno.

3.2.4. Selektor u formi specifičnog atributa

Pored navedenih načina, selekciju je moguće realizovati i tzv. specifičnim atributom, tj. proizvoljnim atributom drugačijim od *id* i *class*. Treba voditi računa da je ovo moguće realizovati u Internet Explorer-u samo ako je definisan *!DOCTYPE*.

Selektor kojim se selektuju elementi sa njihovim atributima se realizuje pomoću sintakse *[attribute]*. Tako na primer, svi *p* elementi koji imaju atribut *align* se selektuju sa:

```
p[align] { background-color: red; }
```

Obzirom da je ovo relativno gruba selekcija, dozvoljeno je da se ista dodatno profilise. Ovim je omogućeno da se pored definisanja imena atributa definiše i vrednost tog atributa tj. sintaksom *[attribute=value]*.

Tako na primer, svi *p* elementi koji imaju atribut *align* sa vrednošću *right*, se selektuju sa:

```
p[align="right"] { background-color: red; }
```

Česta primena ovakve selekcije je kod selektovanja elemenata forme:

```
input[type="text"] { background-color: yellow;}
input[type="radio"] { background-color: yellow;}
```

Dodatno olakšavajuća okolnost je da se kao selektor koristi atribut i njegov sadržaj koji nije kompletno definisan, nego jednim svojim delom. Tako je moguće selektovati sve elemente koji u vrednosti atributa imaju određenu reč. Sintaksa za ovu primenu je *[attribute~=value]*. Tako na primer, svi *a* elementi koji imaju atribut *alt* unutar čijih vrednosti se nalazi reč *Srbija*, se selektuju sa:

```
a[alt~="Srbija"] { background-color: red;}
```

Ukoliko uslov nije da sadržaj bude jedinstvena reč, nego bilo koji deo sadržaja, tj. skup karaktera, koristi se sintaksa *[attribute*=value]*. Sada je moguće pronaći i sadržaj kao što je i „*bila jedn*“.

```
a[alt*="bila jedn"] { background-color: red;}
```

Slično ovoj mogućnosti, moguće je selektovati sve tagove, sa definisanim atributima, kod kojih sadržaj počinje određenim sadržajem. Ukoliko se želi definisati tačno prva reč, iza koje ide *space* ili -, koristi se sintaksa *[attribute|=value]* dok ako to ne mora biti cela reč već samo određeni skup

karaktera koristimo sintaksu $[attribute^=value]$. Identično, moguće je definisati skup karaktera kojim će se vrednost atributa završiti kao $[attribute\$=value]$.

Primer za selektovanje svih *a* elemenata, koji kao sadržaj *alt* atributa imaju, na početku reč *Srbija*, imaju prva tri karaktera *Srb*, tj. poslednja tri karaktera *ija*, definiše se respektivno kao:

```
a[alt/="Srbija"] {
    background-color: red;
}
a[alt^="Srb"] {
    background-color: red;
}
a[alt$="ija"] {
    background-color: red;
}
```

3.3. Povezivanje HTML koda sa CSS kodom

CSS kod se može povezati sa originalnim HTML kodom, na koji treba da deluje, na četiri načina:

1. Pisanjem unutar HTML fajla (u liniji HTML taga)
2. Pisanjem unutar HTML fajla (interni CSS kod)
3. Pisanjem van HTML fajla (eksterni CSS kod)
4. Kombinovanjem bilo koja od ova tri načina.

1. Ukoliko se CSS kod piše u liniji taga, onda se kao veza HTML-a i CSS-a koristi HTML atribut **style**. Na taj način sav HTML kod, uključujući pravila za definisanje HTML atributa *style*, se piše po pravilima HTML-a, a vrednost ovog atributa je kod koji se piše po pravilima CSS-a.

```
<p style="font-size: 20px; color:red; background-color:#FFFF00;"> Kreiranje
CSS stilova u liniji. </p>
```

Tako se kao operator dodele za atribut *style* koristi =, a za svojstvo *color* ∴. Upotrebnom CSS-a na ovaj način, ne koristi se selektor, jer je to tag unutar koga se piše kod.

■ **Napomena:** Žargonski kažemo da je unutar taga i atributa *style* ceo CSS svet!

2. Ukoliko se CSS kod piše kao interni, to znači da je sastavni deo originalne HTML stranice. Interni CSS kod, tada se piše u HEAD sekciji u posebnom tagu `<style> </style>`. Unutar taga *style* važe sva do sada definisana pravila CSS-a. Obzirom da je sada kod van linije taga, neophodno je pisati i selektor da bi se znalo na koga se dati stil želi primeniti.

```

<head>
  <style type="text/css">
    h1 {color:blue; font-size:20px; font-family: verdana;}
    .ime { font-size: 30pt; color: blue;}
    #spolja{ width:1000px; height:500px; background-color:#CCC333;
      margin:10px 20px; border:1px dotted #000222; padding:5px; }
  </style>
</head>
<body>
  <h1>Neki naslov</h1>
  <div id="spolja"> Tekst </div>
  <p class="ime"> </p>
</body>

```

Prednost internog CSS-a, u odnosu na kod u liniji taga, je da se na jednom mestu može napisati kod koji se može pozvati proizvoljan broj puta unutar jedne web stranice. Tako se HTML kod „čisti“ od CSS koda, i CSS kod optimalno tj. minimalno piše, unutar jedne web strane.

3. Međutim, problem sa internim CSS kodom, je što se taj kod mora identično pisati na svakoj strani sajta, ukoliko te stranice imaju iste stilove (što je realan slučaj jer sajt ima isti dizajn za veći broj stranica). U tom slučaju koristi se eksterni CSS fajl. Tada sve stranice sajta učitavaju sadržaj jednog istog eksternog *css* fajla, čiji sadržaj postaje njihov sastavni deo, na isti način kao da je pisan unutar svake od njih ponaosob.

Ovo je najčešći način profesionalne upotrebe, jer se svaka naredna stilska korekcija na celom sajtu radi centralizovano na samo jednom mestu – eksternom CSS fajlu. Način pisanja eksternog CSS-a je identičan sa do sada definisanim CSS pravilima. Nakon definisanja CSS koda, fajl se snima sa ekstenzijom *css*, npr. *proba.css*.

■ **Napomena:** Ime eksternog CSS fajla nikako ne bi trebalo da bude *css*, tj. *css.css*, jer to pojedini browser-i neće da prihvate.

Obzirom da je ovaj CSS fajl fizički odvojen od HTML stranice, mora se napraviti veza između njih. To se realizuje unutar HEAD taga u posebnom tagu *link*. Unutar ovog taga koristi se atribut *href* da se ukaže na putanju do eksternog CSS fajla.

```

<head>
  <link rel="stylesheet" type="text/css" href="fajlovi/proba.css">
</head>

```

Na ovaj način je HTML stranica potpuno „oslobođena“ CSS koda. Upotrebom eksternog CSS-a, moguće je u jednoj stranici učitati više različitih CSS fajlova, sa više pojedinačnih *link* tagova. Na taj način, svi sadržaji eksternih fajlova se pojedinačno učitavaju i postaju dostupni originalnoj HTML stranici.

```

<html>
<head>
  <link rel="stylesheet" href="stil1.css" type="text/css">
  <link rel="stylesheet" href="stil2.css" type="text/css">
</head>
<body>
  <span class="naslov">Ispis definisan klasom naslov.</span><br/>
<table border="2px">
  <tr><td class="podnaslov">Prikaz u tabeli <br/>podnaslov</td></tr>
</table>
  <p class="telo">Centralni deo ispisan je klasom telo.</p>
</body></html>

```

4. Kombinacija ova tri pravila omogućava da se unutar jedne stranice može koristiti bilo koja njihova interna kombinacija. To se realno retko koristi, ali je efekat potpuno isti bez obzira na način na koji se kod piše.

Važno je napomenuti da je kod rada sa CSS-om vrlo bitan redosled učitavanja tj. kasnijeg pozivanja deklaracija u CSS-u. Naime, ako se u *head* sekciji web strane učitavaju dva eksterna CSS fajla, gde je na prvom mestu *stil1.css* a zatim *stil2.css*, i ako se u oba definiše isti selektor, recimo tag *p*, sa istim svojstvom, recimo *color*, ali sa različitim vrednostima (npr. u fajlu *stil1.css* je definisano `p{color:red;}` a u fajlu *stil2.css* imamo `p{color:blue;}`) doći će do promene jednog od pravila. Pravilo koje je poslednje definisano će biti pravilo koje će da važi. U ovom slučaju boja testa u tagu *p* bi bila plava. Ukoliko bi se redosled učitavanja ova dva fajla, u *head* sekciji, promenio na način da se prvo učitava *stil2.css*, pa *stil1.css*, poslednje pravilo bi bilo pravilo iz *stil1.css* pa bi boja teksta bila crvena.

■ **Napomena:** U stručnoj zajednici se mogućnost CSS-a da u delovima istog koda menja pravila koja su ranije definisana, i da se to može posmatrati i selektovati po dubini DOM-a, ili na specifičnim mestima koja su selektorima definisana, uz ključnu mogućnost da se više stilova može primeniti na isti deo HTML koda, smatra korenom reči *Cascading*, koja je prva reč u akronimu CSS. Håkon Wium Lie opisuje termin "cascade" kao "Proces kombinovanja nekoliko stilova i rešavanje konflikata između njih".

Pored opisanih načina integracije HTML-a sa CSS-om, postoji i dodatni način kojim se CSS fajl može povezati sa eksternim CSS-om. Ovo se realizuje primenom pravila *@import* u opštoj sintaksi

```
@import url;
```

Ako se pretpostavi da postoji HTML strana, gde se učitava eksterni CSS fajl *style1.css* i u njemu definiše neki deo CSS koda kojim se stilizuje HTML kod, ali pored toga i učitava novi eksterni CSS fajl *style2.css* pomoću

```
@import url("style2.css");
```

U kome se dodatno definiše neki CSS kod, efekat na HTML stranu će biti kumulativan, tj. delovaće sva pravila koja su i u fajlu *style1.css* i u fajlu *style2.css*.

U listingu koji sledi dati su opisani primeri *index.html*, *style1.css* i *style2.css*

Listing index.html

```
<html>
<head>
    <link href="style1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <p> Primer sadrzaja </p>
</body>
</html>
```

Listing style1.css

```
@import url("style2.css");
body{ background-color:#f2f2f2; }
```

Listing style2.css

```
p{    font-size:24px;
      color:red;
      font-weight:bold; }
```



Slika 3.6. Grafički prikaz web strane sa dva CSS fajla

3.4. Svojstva u CSS-u

Postoji veliki broj svojstava u CSS-u. Svojstva su ključni elementi na koji CSS deluje i pomoću kojih se postižu efekti u browser-u. U različitim verzijama CSS-a pojavljuju se različita svojstva, i treba biti obazriv kod primena, obzirom da svojstva imaju tačno definisana imena, sintaksu i moguće vrednosti. Kao i sve u Web-u, i broj svojstava se konstantno povećava i unapređuje. Negativna posledica ove pojave je činjenica da starije verzije browser-a neko od svojstava ne mogu da realizuju. Slično se dešava i sa pojedinim verzijama browser-a kada se radi o novijim svojstvima. To programeru često stvara vrlo velike prepreke da sajt koji je napravljen bude prikazan, u više različitih browser-a ili njihovih verzija, na identičan način.

U ovoj knjizi obradiće se najčešće korišćena svojstva i za neka prikazati njihove primene. Detaljnija primena, na operativnom realizuje se u sklopu laboratorijskih

vežbi za ovaj predmet. Svojstva će se inicijalno grupisati po načinu ili mestu na koja deluju. Prikaz svojstava će biti dat u formi tabele, gde se u prvoj koloni nalazi ime svojstva, u drugoj opis tog svojstva a u trećoj koloni moguće vrednosti tog svojstva.

U ovom poglavlju obrađivaće se samo CSS 1 i CSS 2 svojstva, dok su CSS 3 svojstva data u posebnom poglavlju, sem kada je CSS 3 proširenje skupa svojstava toliko minimalno, da je većina sličnih svojstava nasleđena iz ranijih verzija CSS-a.

3.4.1. Svojstva za pozadinu

<i>Naziv svojstva</i>	<i>Opis svojstva</i>	<i>Vrednost svojstva</i>
<i>background</i>	<i>Koristi se za objedinjavanje svih svojstava pozadine u jednu deklaraciju</i>	<i>inherit</i>
<i>background-color</i>	<i>Definiše pozadinsku boju</i>	<i>color transparent initial, inherit</i>
<i>background-image</i>	<i>Definiše pozadinsku sliku</i>	<i>url('URL') none initial, inherit</i>
<i>background-position</i>	<i>Definiše početnu poziciju pozadinske slike</i>	<i>left top left center left bottom right top right center right bottom center top center center center bottom x% y% xpos ypos initial, inherit</i>
<i>background-repeat</i>	<i>Omogućava automatsko ponavljanje iste pozadinske slike više puta</i>	<i>repeat repeat-x repeat-y no-repeat initial, inherit</i>
<i>background-attachment</i>	<i>Definiše da li se pozadinska slika pomera sa ostatkom stranice ili je fiksna</i>	<i>scroll, fixed local initial, inherit</i>

Kod većine svojstava postoje dve zajedničke vrednosti: *initial* i *inherit*. *Initial* definiše vrednost svojstva na njegovu podrazumevanu vrednost dok, *Inherit* dobija vrednost svojstva od nadređenog taga (roditeljskog taga).

Primer upotrebe objedinjenog svojstva *background*:

```
body { background: #CCFFCC url('slika.png') no-repeat fixed center; }
```

■ **Napomena:** Putanja koja se definiše unutar url("") i koja ukazuje na neku sliku, treba da bude apsolutna ili relativna. Ukoliko je relativna, piše se u odnosu na lokaciju gde se nalazi CSS fajl, a ne HTML stranica.

Primer definisanje pozadinske boje na tri različita načina:

```
body { background-color: red;
      background-color: #FF0000;
      background-color: rgb(255,0,0); }
```

Primer definisanja slike kao pozadine web stranice:

```
body{ background-image: url('slika.jpg');
      background-attachment: fixed;
      background-position: center; }
```

Definisanjem pozadinske slike kroz CSS, omogućava se da se preko nje regularno prikaže sadržaj koji je definisan u tagovima HTML-a. Kada se slika učita pomoću HTML taga `<img />`, onda se ona već posmatra kao sadržaj, pa se podrazumevano ne može novi sadržaj staviti preko nje, dok se učitavanjem pomoću CSS-a „doživljava“ kao pozadina.

Vrlo česta primena CSS-a, u smislu pozadinske slike je multipliciranje uzorka slike tj. *slice*-a. *Slice*, ili uzorak slike, je po pravilu slika koja je vertikalno ili horizontalno orijentisana, i vrlo je “tanka”. Teorijski može biti i 1px, jer je tada “težina” te slike vrlo mala, a CSS kod je može multiplicirati onoliko puta koliko je potrebno, i stvoriti efekat kao da se učitava velika slika. Posmatrajmo teksturu slike 3.7.a. koja ima linearni gradijent dve boje (što je često u realnim upotrebama). Pretpostavimo da je ova slika veličine 800px x 70px, i da se koristi u zaglavlju sajta. Umesto učitavanja ove velike slike, dovoljno je napraviti njen vertikalni uzorak, slika 3.7.b., koji CSS kodom treba kopirati potreban broj puta.



Slika 3.7. a) Originalna slika



b) vertikalni uzorak slike

Kod primene CSS za multipliciranje slike, pored uštede u veličini slike, dobija se i još jedna prednost. Naime, originalna slika ima svoje dimenzije, i u slučaju da je rezolucija ekrana korisnika veća nego što je slika, razvlačenjem slike smanjuje se njen kvalitet i postiže loš vizuelni efekat. Upotrebom CSS koda, proces multiplikacije se realizuje onoliko puta koliko je potrebno za datu rezoluciju čime se kvalitet slike ne menja, jer se ona više puta fizički kopira, a ne “razvlači”.

Primer definisanja pozadinske slike koja se automatski kopira duž *x*-ose:

```
body{
    background-image:url('slika.jpg');
    background-repeat: repeat-x;
}
```

Polazeći od ograničenja od 20 *http_request*-ova po strani, jedan od načina da se on smanji, kada su u pitanju slike, je da se umesto većeg broja manjih slika, naročito kada su one tematske i služe kao ikonice ili pozadine manjih blokova, učita samo jedna, na kojoj se nalazi više manjih slika. Ako se pođe od slike sa datim skupom ikonica



Slika 3.8. Objedinjena slika sa većim brojem manjih slika

i ako se u web strani prikazuju čak i samo dve ikonice, potrebno je istu sliku učitati kao pozadinsku sliku u dva bloka i promenom pozicije slike, kreirati utisak da se vidi samo jedna konkretna ikonica na željenom mestu, a „troši“ se samo jedan *http_request*. Za dati HTML kod

```
<html>
<head>
    <link href="style1.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <div id="prvi"></div>
    <p> Ovo je neki tekst </p>
    <div id="drugi"></div>
</body>
</html>
```

kreirati CSS kod, kojim se učitavanje pozadinske slike prikazuje na dva različita načina, sa istim efektom, kao

```
#prvi{
  width:70px;
  height:70px;
  background-image: url("1.png");
  background-repeat: no-repeat;
  background-position: -60px -60px; }
#drugi{
  width:80px;
  height:80px;
  background: url("1.png") no-repeat 10px -130px; }
```

Na ovaj način dobiće se dva bloka, koji su sadržaj stranice, i u kojima je ista pozadinska slika, ali sa utiskom kod korisnika da su to različite slike.



Slika 3.9. Prikaz browser-a kada se pozadinska slika učitava u dva bloka sa pomerajem

Pored ovakvog načina da se optimalno učitava više slika, učitavanjem jedne veće, postoje i drugi alternativni način. Jedan od njih je upotreba *web icon*-a. Ovakve ikonice su sastavni deo nekog CSS koda koji je prvo potrebno preuzeti, i integrisati u sajt kao eksterni CSS kod, i zatim ga implementirati na željeno mesto. Upotrebom *web icon*-a programer ima na raspolaganju veliki broj standardnih ikonica, koje se „doživljavaju“ kao tekst i koje su prepoznatljivije.

Postoji više načina kako se mogu upotrebiti pa će se sada prikazati upotrebe *Font Awesome icons* i *Google icons*-a. I jedna i druga primena najčešće koriste tag *i* ili *span* za njihovu interpretaciju u sajtu, uz prethodno učitavanje odgovarajućeg eksternog CSS fajla, kao u primerima. Za obe primene potrebno je pronaći veliki broj skraćenica za ponuđene ikonice na zvaničnim sajtovima proizvođača.

Listing: Primer upotrebe Font Awesome icons

```
<html>
<head>
<style>
  span{width:50px;}
</style>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/
  /libs/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
```

```

<body>
  <span class="fa fa-arrow-circle-o-right"></span>
  <span class="fa fa-bar-chart"></span>
  <span class="fa fa-bus"></span>
  <span class="fa fa-gift"></span>
  <span class="fa fa-home"></span>
</body>
</html>

```



Slika 3.10. Prikaz browser-a kada se koriste Font Awesome icons



Slika 3.11. Prikaz browser-a kada se koriste Google icons

Listing: Primer upotrebe Google icons

```

<html>
<head>
  <link    rel="stylesheet"    href="https://fonts.googleapis.com/
icon?family=Material+Icons">
</head>
<body>
  <i class="material-icons" style="font-size:36px;">access_alarms</i>
  <i class="material-icons" style="font-size:36px;">android</i>
  <i class="material-icons" style="font-size:36px;">attach_file</i>
  <i class="material-icons" style="font-size:36px;">shopping_cart</i>
  <i class="material-icons" style="font-size:36px;">tag_faces</i>
</body>
</html>

```

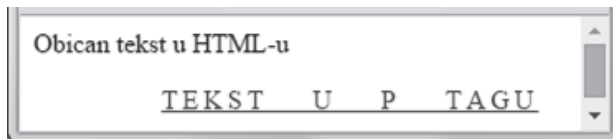
3.4.2. Svojstva za tekst

U CSS-u postoji veliki broj svojstava za tekst, i imena svojstava prilično jasno ukazuju na njihovu funkciju. Skoro identične funkcionalnosti nudi i najveći broj softvera za editovanje teksta, pa je skoro sve što se može u njima, može urediti i primenom CSS-a.

<i>Naziv svojstva</i>	<i>Opis svojstva</i>	<i>Vrednost svojstva</i>
<i>color</i>	<i>Definiše boju teksta</i>	<i>color</i>
<i>text-decoration</i>	<i>Definiše dekoraciju teksta</i>	<i>none</i> <i>underline</i> <i>overline</i> <i>line-through</i> <i>initial, inherit</i>
<i>letter-spacing</i>	<i>Definiše rastojanje između karaktera u tekstu</i>	<i>normal</i> <i>length</i> <i>initial, inherit</i>
<i>line-height</i>	<i>Definiše visinu karaktera u teksta</i>	<i>normal</i> <i>number</i> <i>length, %</i> <i>initial, inherit</i>
<i>text-align</i>	<i>Definiše horizontalno poravnanje teksta</i>	<i>left, right</i> <i>center</i> <i>justify</i> <i>initial, inherit</i>
<i>text-indent</i>	<i>Definiše uvlačenje prve linije teksta</i>	<i>Length, %</i> <i>initial, inherit</i>
<i>text-transform</i>	<i>Omogućava rad sa malim tj. velikim slovima</i>	<i>none</i> <i>capitalize</i> <i>uppercase</i> <i>lowercase</i> <i>initial, inherit</i>
<i>white-space</i>	<i>Omogućava kontrolu nad praznim prostorom unutar teksta</i>	<i>normal</i> <i>nowrap</i> <i>pre, pre-line</i> <i>pre-wrap</i> <i>initial, inherit</i>
<i>word-spacing</i>	<i>Definiše rastojanje između reči u tekstu</i>	<i>normal</i> <i>nowrap</i> <i>pre</i> <i>pre-line</i> <i>pre-wrap</i>

U ovom objedinjenom primeru, veći broj svojstava za tekst je dodeljen elementu *p*. Radi demonstracije, prvi deo teksta na slici 3.12. je kreiran u HTML-u, kao običan tekst van elementa, dok je drugi deo definisan kao sadržaj elementa *p*. Na taj način CSS kod deluje na drugi deo teksta, sa svim svojstvima objedinjeno.

```
p {
  color: red;
  text-align: center;
  text-decoration: underline;
  letter-spacing: 3px;
  text-indent: 60px;
  text-transform: uppercase;
  word-spacing: 20px; }
```



Slika 3.12. Efekat svojstava za tekst u CSS-u

3.4.3. Svojstva za fontove

Pored toga što je primarni utisak korisnika o sadržaju sajta baziran na načinu kako je tekstualni sadržaj predstavljen, a koji će se definisati svojstvima za fontove, ova svojstva su dodatno dobila na obimu korišćenja kada je HTML tag *font*, u verziji HTML 5, izbačen. Obzirom da je do sada, kroz tag *font* programer delovao na stilizovanje tekstualnog sadržaja, a da sada taj tag više nije podržan, svo stilizovanje je dodeljeno CSS-u i njegovim svojstvima za rad sa fontovima.

Karakteristike ovih svojstava su da većina počinje sa rečju font, a njima će biti omogućeno delovanje na font, veličinu, debljinu, stil teksta i sl.

Naziv svojstva	Opis svojstva	Vrednost svojstva
<i>font</i>	Definiše sva svojstva u vezi fonta u objedinjenoj formi	<i>font-style</i> <i>font-variant</i> <i>font-weight</i> <i>font-size/line-height</i> <i>font-family</i> <i>initial, inherit</i>
<i>font-family</i>	Definiše font teksta	<i>family-name</i> <i>initial, inherit</i>
<i>font-size</i>	Definiše veličinu teksta	<i>medium</i> <i>xx-small</i> <i>x-small</i> <i>small</i> <i>large</i> <i>x-large, xx-large</i> <i>smaller, larger, length</i> <i>%</i> <i>initial, inherit</i>

<i>font-style</i>	Definiše stil prikaza teksta	<i>normal</i> <i>italic</i> <i>oblique</i> <i>initial, inherit</i>
<i>font-variant</i>	Omogućava umanjeni prikaz velikih slova	<i>normal</i> <i>small-caps</i> <i>initial, inherit</i>
<i>font-weight</i>	Definiše debljinu teksta	<i>normal</i> <i>bold, bolder, lighter</i> <i>100, 200, 300, 400</i> <i>500, 600, 700, 800</i> <i>900</i> <i>initial, inherit</i>
<i>line-height</i>	Definiše visinu linije teksta	<i>normal</i> <i>number</i> <i>percentage</i> <i>length</i> <i>inherit, inherit</i>

Primer objedinjavanja kroz svojstvo *font*.

```
p { font: italic bold small/1.9em Verdana, sans-serif; }
```

■ **Napomena:** U realnim primenama, svojstva *font-family* i *font-size* se definišu na jednom mestu, na nivou cele stranice, a po potrebi se u pojedinim delovima stranice redefinišu tj. menjaju. Ovim se postiže definisana uniformnost izgleda cele stranice a minimalno pisanje koda.

Objedinjeni primer više svojstava za stilizovanje, dodeljen elementu *p* je:

```
p { font-family: "Times New Roman", Georgia, Serif;
    font-size: 200%;
    font-style: italic;
    font-variant: small-caps;
    font-weight: bold;
    line-height: 1.5;
}
```



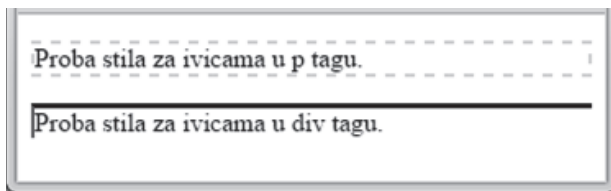
PROBA STILA ZA RAD SA FONTOVIMA.

Slika 3.13. Prikaz primene CSS svojstava za rad sa fontovima

3.4.4. Svojstva za linije

Rad sa ivicama (*border*) omogućava vizuelno isticanje pojedinih ivičnih linija sa ciljem naglašavanja nekog dela sadržaja ili fizičke podele neke površine. CSS ima veći broj svojstava za rad sa ivicama, i sve počinju sa rečju *border*. Pojedina svojstva se primenjuju automatski na sve ivice, dok je eksplicitnim navođenjem moguće izdvojiti pojedine ivice (npr. *border-right*). Primer primene ovih svojstava je:

```
p { border: 2px dashed #FFCCCC; }
#proba{ border-left: 2px solid green;
        border-top-style: solid;
        border-right-color: red; }
```



Slika 3.14. Prikaz primene CSS svojstava za rad sa ivicama

Skup svih svojstava za linije sa njihovim opisima je dat sa:

Naziv svojstva	Opis svojstva	Vrednost svojstva
<i>border</i>	Definisanje svih svojstava za ivice	<i>border-width</i> <i>border-style</i> <i>border-color</i> <i>initial, inherit</i>
<i>border-color</i>	Definisanje boje svih ivica	<i>color</i> <i>transparent</i> <i>initial, inherit</i>
<i>border-style</i>	Definisanje stilova svih ivica	<i>none</i> <i>hidden</i> <i>dotted, dashed, solid</i> <i>double, groove, ridge</i> <i>inset, outset</i> <i>initial, inherit</i>
<i>border-width</i>	Definisanje debljine svih ivica	<i>medium</i> <i>thin, thick</i> <i>length</i> <i>initial, inherit</i>
<i>border-AAA</i>	Definisanje svih svojstava za određeni deo ivice	AAA može biti: <i>bottom, left, right, top</i>

<i>border-AAA-color</i>	Definisanje boje za određeni deo ivice	AAA može biti: <i>bottom, left, right, top</i>
<i>border-AAA-style</i>	Definisanje stilova za određeni deo ivice	
<i>border-AAA-width</i>	Definisanje debljine za određeni deo ivice	

3.4.5. Svojstva za liste

Iako možete pomisliti da se svojstva za liste koriste više naučno nego praktično, grešite. Ova svojstva su vrlo često korišćena jer se njima dodatno stilizuju stavke menija i linkova u menijima, definišu drugačije slike od podrazumevanih u listama, drugačiji oblici nabravanja i sl.

<i>Naziv svojstva</i>	<i>Opis svojstva</i>	<i>Vrednost svojstva</i>
<i>list-style</i>	Objedinjeno definisanje svih svojstva o listama	<i>list-style-type</i> <i>list-style-position</i> <i>list-style-image</i> <i>initial, inherit</i>
<i>list-style-image</i>	Definisanje slike kao oznake elementa liste	<i>none, url</i> <i>initial, inherit</i>
<i>list-style-position</i>	Definisanje pozicije oznake elemenata liste	<i>Inside, outside</i> <i>initial, inherit</i>
<i>list-style-type</i>	Definisanje tipova oznaka elemenata liste	<i>disc</i> <i>armenian</i> <i>circle</i> <i>decimal</i> <i>decimal-leading-zero</i> <i>georgian, hebrew</i> <i>hiragana, hiragana-iroha</i> <i>katakana</i> <i>katakana-iroha</i> <i>lower-alpha</i> <i>lower-greek</i> <i>lower-latin</i> <i>lower-roman</i> <i>square</i> <i>upper-alpha</i> <i>upper-latin</i> <i>upper-roman</i> <i>initial, inherit</i>

Primeri upotrebe nekih od ovih svojstava su:

```
list-style: square outside;
```

```
li { list-style-image: url(images/myface.jpg); }
li { list-style-position: inside; }
```

U realnim sajtovima, navigacioni meniji se pišu u obliku neuređene liste. Razlog za ovakav oblik pisanja leži u kasnijim primenama skript jezika i modifikaciji originalnog HTML koda u neku atraktivniju formu menija (padajući meni, ekspanzirajući meni, animirani,...). Međutim, stavljanjem linkova u elemente liste, inicijalno se dobijaju grafički simboli tačke. Da bi se ovo sakrilo, a meni i dalje ostao u formi liste, koristi se svojstvo *list-style-type*. Primer vertikalnog menija dat je u listingu.

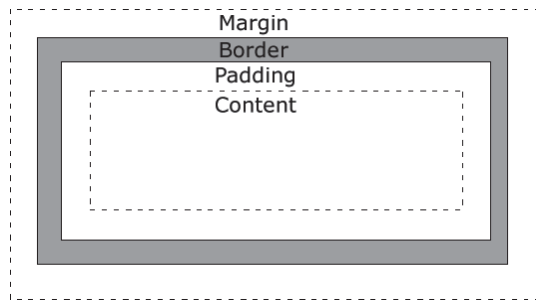
```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    margin: 0px;
    padding: 0px;
}
</style>
</head>
<body>
<ul>
    <li><a href="prvi.html">Prvi</a></li>
    <li><a href="drugi.html">Drugi</a></li>
    <li><a href="treci.html">Treci</a></li>
    <li><a href="cetvrti.html">Cetvrti</a></li>
</ul>
<h2> Ovo je neki nastavak teksta </h2>
</body>
</html>
```



Slika 3.15. Prikaz primene CSS-a za izradu vertikalnog menija

3.4.6. Svojstva za pozicioniranje i prikaz

Grupa svojstava za pozicioniranje (poravnanje) i prikaz imaju ključnu ulogu u postupku organizacije sadržaja sajta. Najčešće korišćena svojstva su: *margin*, *padding*, *display*, *float*, *position*, *width*, *height*, ... Iako većina imena ovih svojstava odgovara terminima koju su od ranije poznati, posebnu pažnju treba posvetiti razlikovanju uloge *margin* i *padding* svojstva, koji se često mešaju. Ako se posmatra određeni sadržaj unutar neke web strane, i ako taj sadržaj ima okvirnu liniju, tada je *padding* rastojanje tog sadržaja od okvirne linije, a *margin* rastojanje okvirne linije, sa svim unutrašnjim sadržajem, od granica web stranice.



Slika 3.16. Grafički prikaz uticaja svojstva *margin* i *padding* u odnosu na ivicu

Generalno, i *margin* i *padding*, imaju četiri vrednosti, koje redom označavaju poziciju gore, desno, dole i levo (smer kazaljke na satu počev od gore) tj. *top*, *right*, *bottom*, *left*. Tako na primer, sintaksa

```
margin:10px 5px 15px 20px;
```

označava vrednosti margina *margin-top: 10px*, *margin-right: 5px*, *margin-bottom: 15px* i *margin-left: 20px*. Međutim, u realnim uslovima, često su vrednosti pojedinih margina iste, ili se ne definišu eksplicitno, pa se uvodi skraćeni zapis.

```
margin:10px 5px 15px;
```

Sada su definisane sledeće vrednosti: *margin-top: 10px*, *margin-right: 5px* i *margin-bottom: 15px*. U slučaju sve vrednosti:

```
margin:10px 5px;
```

definisane su sledeće vrednosti: *margin-top: 10px* i *margin-bottom: 10px* dok su *margin-right: 5px* i *margin-left: 5px*. Međutim, u slučaju da se definiše jedna vrednost:

```
margin:10px;
```

definisane su sledeće vrednosti: *margin-top: 10px*, *margin-right: 10px*, *margin-bottom: 10px* i *margin-left: 10px*.

Pored svojstava *margin* i *padding*, često se mešaju svojstva *visibility* i *display*. Naime, i realnim sajtovima često se nameće potreba da se pojedini delovi sajta sakriju (dok se ne klikne na nešto ili dok se ne plati, istekne neko vreme i sl.). Sakrivanje se formalno može uraditi na dva načina:

1. Svojstvo *visibility* sa vrednošću *hidden*
2. Svojstvo *display* sa vrednošću *none*

Iako ova dva svojstva realizuju sakrivanje određenog sadržaja, oni to ne rade na isti način:

- ✓ Upotrebom `{visibility:hidden;}` objekat koji se selektuje se sakriva, ali se njegovo mesto i dalje čuva u istim dimenzijama koliki je i objekat,
- ✓ Upotrebom `{display:none;}` nema čuvanja prostora i ceo ostali sadržaj sajta (koji je desno ili ispod) se pomera za taj prostor (u levo ili na gore).

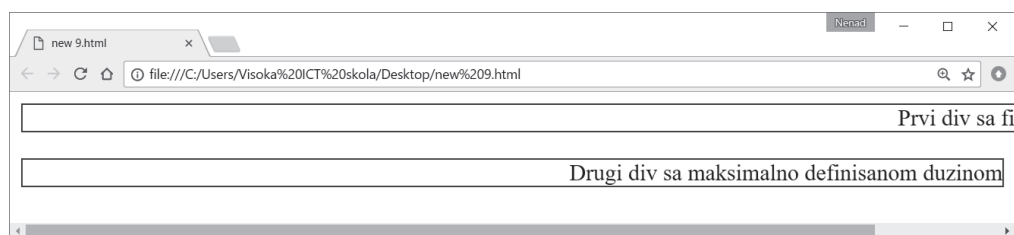
Kod početnika se takođe često uočava pogrešna upotreba svojstva *float*. Naime, vođeni svakodnevnim načinom rada u postupku pozicioniranja (poravnanja) nekog objekta, često se očekuje da se i poravnanje u sajtu radi levo/desno, gore/dole. Međutim, svojstvo *float* ima vrednosti *left* i *right*, ali **ne** i nešto tipa: *top*, *up*, *bottom*, *down*, i sl.

Naziv svojstva	Opis svojstva	Vrednost svojstva
<i>clear</i>	Definiše kako je novi blok postavljen nakon pozicioniranog bloka, tj. sa koje strane elementa se ne nastavlja prethodno poravnanje	<i>none</i> <i>left</i> <i>right</i> <i>both</i> <i>initial, inherit</i>
<i>margin</i>	Definiše veličinu praznog prostora oko elementa, sa spoljašnje strane ivice. Svaka od četiri margine: <i>top</i> , <i>left</i> , <i>bottom</i> i <i>right</i> , mogu se definisati nezavisno.	<i>auto</i> <i>length(px, cm, pt, ...)</i> <i>%</i> <i>inherit</i>
<i>display</i>	Definiše kako će se neki HTML element prikazivati	<i>inline</i> <i>block</i> <i>flex</i> <i>inline-block</i> <i>inline-flex</i> <i>inline-table</i> <i>list-item</i> , <i>run-in</i> <i>table</i> , <i>table-caption</i> <i>table-column-group</i> <i>table-header-group</i> <i>table-footer-group</i> <i>table-row-group</i> <i>table-cell</i> <i>table-column</i> <i>table-row</i> <i>none</i> <i>initial, inherit</i>
<i>float</i>	Definiše poziciju bloka	<i>none</i> <i>left</i> , <i>right</i> <i>initial, inherit</i>

<i>height</i>	<i>Definisanje visine elementa</i>	<i>auto, length % initial, inherit</i>
<i>left</i>	<i>Definisanje pozicije elementa sa leve strane</i>	<i>auto, length % initial, inherit</i>
<i>overflow</i>	<i>Definiše mogućnost da se element nađe preko određenog sadržaja</i>	<i>visible, hidden scroll, auto initial, inherit</i>
<i>padding</i>	<i>Definisanje svih padding svojstava u jednoj deklaraciji. Može biti sa specificiranjem: padding-bottom, padding-left, padding-right, padding-top</i>	<i>length % initial, inherit</i>
<i>position</i>	<i>Definiše tip pozicioniranja elementa</i>	<i>static, absolute fixed, relative initial, inherit</i>
<i>right</i>	<i>Definiše desno poravnanje elementa</i>	<i>auto, length % initial, inherit</i>
<i>top</i>	<i>Definiše gornje poravnanje elementa</i>	<i>auto, length % initial, inherit</i>
<i>visibility</i>	<i>Definiše vidljivost elementa</i>	<i>visible hidden collapse initial, inherit</i>
<i>width</i>	<i>Definisanje širine elementa</i>	<i>auto, length % initial, inherit</i>
<i>vertical-align</i>	<i>Definisanje vertikalnog poravnanja elementa</i>	<i>baseline length % sub, super top, text-top middle bottom text-bottom initial, inherit</i>
<i>z-index</i>	<i>Definiše redosled u pozicioniranju i prikazivanju elemenata</i>	<i>auto number initial, inherit</i>

Jedno od vrlo čestih primena je svojstvo *width* i *height*. Ova dva svojstva treba dodeliti svakom bloku i regionu kod kojih je moguće predvideti obim sadržaja. Iz tog razloga svojstvo *width* se često koristi sa apsolutnom vrednošću, jer jasno definiše veličinu nekog bloka, ali i sa relativnom kada je potrebno da se prilagodi rezoluciji ekrana na kome sa sajt gleda. Jedno od „među rešenja“ kod svojstva *width*, između apsolutne i relativne dimenzije je svojstvo *max-width*. Ovo svojstvo obezbeđuje da se za vrednosti veće od definisanog *max-width* blok ponaša kao da ima apsolutnu širinu, a za manje relativnu i da se prilagođava rezoluciji tj. dužini browser-a. U primeru su data dva diva sa *width* i *max-width* svojstvima i prikaz u browser-u kada je dužina browser-a manja od definisane vrednosti ovih atributa, gde se vidi da se jedan blok srazmerno smanjuje a drugi ostaje sa fiksnom dužinom.

```
<html>
<head>
<style>
div{
    border: 1px solid red;
    margin: auto;
    text-align:right;
}
#prvi {
    width:800px;
}
#drugi {
    max-width:800px;
}
</style>
</head>
<body>
    <div id="prvi">Prvi div sa fiksnom duzinom</div> <br/>
    <div id="drugi">Drugi div sa maksimalno definisanom duzinom</div>
</body>
</html>
```



Slika 3.17. Grafički prikaz uticaja svojstva *width* i *max-width*

Primer upotrebe svojstva *display*, i različitih vrednosti *block* tj. *inline*, prikazano je u listingu.

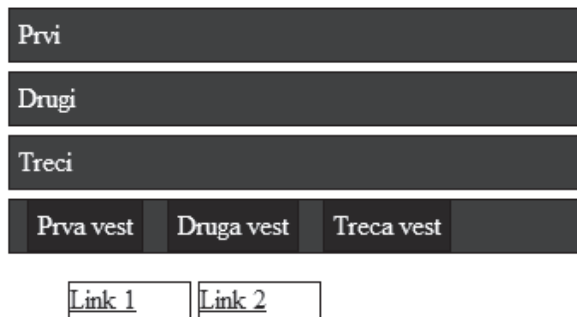
```

<html>
<head>
<style type="text/css">
    div{ display:block; background-color: #FF0000; color:#FFF; border:1px
    solid black; margin: 5px; padding: 5px;}
    li { display:inline-block; width:70px; border:1px solid #000;}
    .vest{ display: inline; width:150px; background: #0000FF; color:#FFF;}
</style>
</head>
<body>

    <div id="prvi" class="blok">Prvi</div>
    <div id="drugi" class="blok">Drugi</div>
    <div id="treći" class="blok">Treći</div>

<div id="omot">
    <div class="vest">Prva vest</div>
    <div class="vest">Druga vest</div>
    <div class="vest">Treća vest</div>
</div>
<ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
</ul>
</body>
</html>

```



Slika 3.18. Prikaz primene svojstva *display*.

Obzirom da svojstva koja se koriste za pozicioniranje i prikaz imaju vrlo veliku ulogu u konačnom vizuelnom identitetu sajta, i da se definišu za skoro svaki element, često dolazi do ponavljanja velike količine koda. Ovo se najčešće odnosi na svojstva *margin* i *padding*, ali u zavisnosti od koncepta stranica i na druga svojstva. U cilju redukovanja linija koda koje se pojavljuju dodeljene svakom ili skoro svakom elementu, koristi se univerzalni selektor * (*Asterisk*). Upotrebom ovog selektora, selektuje se svaki element koji se nalazi unutar dokumenta, tj. web stranice. Tako se jednom linijom koda npr.:

```
* { margin: 0px; padding: 0px; }
```

svim elementima strane definiše da su sve margine i svi padding delovi setovani na vrednost 0px. Po potrebi, u pojedinim elementima ovo se može predefinisati i postaviti druga vrednost od one koja je unutar univerzalnog selektora.

■ Važno! Treba naglasiti da se ukupna širina elementa renderovanog u obliku bloka, po W3C standardu, zavisi od veličine margina, ivica (*border*), *padding*-a i širine sadržaja (*width*).

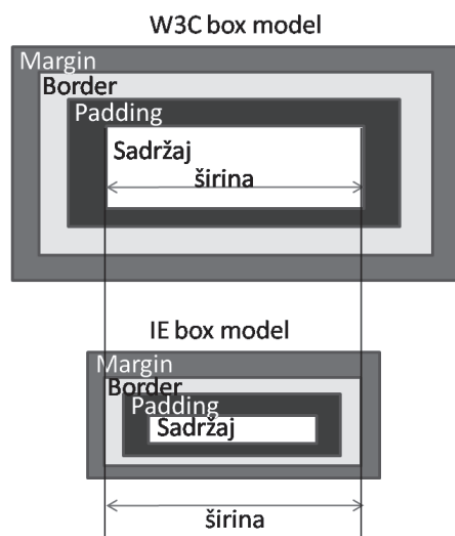
Tako, na primer, ako element ima sledeće vrednosti svojstava:

```
{width:250px; padding:10px; border:5px solid gray; margin:10px; }
```

širina elementa je 300px.

Ovih 300px je dobijeno kao suma: 250px (*width*) + 20px (left padding + right padding) + 10px (left border + right border) + 20px (left border + right border)

Ovaj način računanja ne važi za *Internet Explorer*. Naime ovaj browser računa širinu bez uključivanja margina u ukupan zbir, tako da bi ukupna širina elementa u gornjem primeru bila 280px.



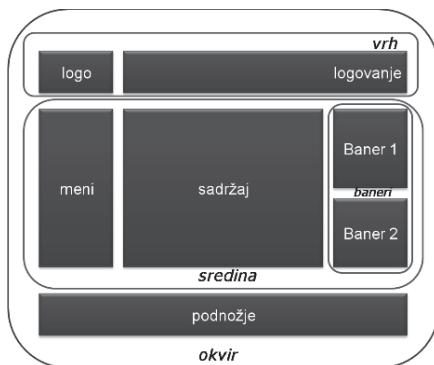
Slika 3.19. Razlika interpretacije box modela (izvor:Wikipedia)

3.5. Primer organizacije sajta primenom CSS-a

U poglavlju 2.13. objašnjena je logika koja se primenjuje kod pisanja HTML koda kada je potrebno web stranicu organizovati CSS-om. U ovom delu na praktičnim primerima pokazaće se objedinjen primer HTML-a i dodatnog CSS-a potrebnog da se kreira jedna web strana.

3.5.1. Primer 1

Polazeći od strukture koja je ranije definisana, slika 3.20. prvo je kreiran HTML kod, kojim su definisani divovi i njima pripadajući atributi *id* i *class*, preko kojih će CSS delovati.



Slika 3.20. *Struktura web stranice koju je potrebno organizovati CSS-om*

Primarni cilj CSS koda je da kreirane HTML blokove uredi na način kako je to predviđeno slikom 3.20. Za te potrebe koristiće se eksterni CSS fajl, koji je nazvan *style.css*. Obzirom da divovi inicijalno nemaju svoju grafičku interpretaciju u browser-u, kao sadržaj svakog od njih definišaće se kratak tekst (Logo, Logovanje, Meni, Sadržaj, Baner, ...), da bi se u browser-u lakše pratila njihova pozicija. Polazeći od već kreiranog HTML koda za ovu strukturu, Poglavlje 2.13., ove izmene označene su boldovanih tekstem.

```
<html>
<head>
    <link rel="stylesheet" href="style.css" type="text/css" />
</head>
<body>
<div id="okvir">
<div id="vrh">
    <div id="logo">
        Logo
    </div>
    <div id="logovanje">
        Logovanje
    </div>
</div>
<div id="sredina">
    <div id="meni">
        Meni
    </div>
    <div id="sadržaj">
        Sadržaj
    </div>
```

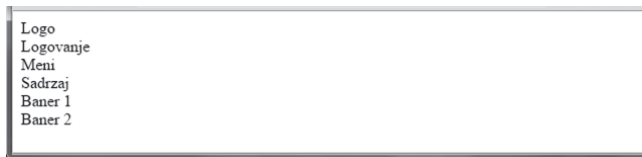


```

<div id="baneri">
  <div class="baner">
    Baner 1
  </div>
  <div class="baner">
    Baner 2
  </div>
</div>
<div id="podnozje">
</div>
</body>
</html>

```

Ukoliko se ovaj kod pokrene u browser-u, dobija se izgled kao na slici 3.21. Kako CSS još nije kreiran, ni jedan od blokova nema instrukcije gde da se prikaže, pa se podrazumevano prikazuju jedan ispod drugog, uz levu ivicu browser-a.



Slika 3.21. Prikaz koda u browser-u vez delovanja CSS koda

Prvi korak je da se kompletna web stranica, tj. blok *#okvir*, definiše i pozicionira. Potrebno je definisati širinu web strane, što zavisi od analize ciljne grupe koja sajtu pristupa i njihove najčešće korišćene rezolucije monitora, ali ćemo za početak pretpostaviti da većina koristi širinu 1024px. U tom slučaju izabraćemo da sajt bude širok 1000px, i da kompletan sadržaj bude centralno pozicioniran u odnosu na browser. Širina, bloka *#okvir*, definiše se pomoću svojstva *width*, dok se efekat centriranja postiže sa *auto* marginom, tj. *margin:0px auto*;. Da bi se pozicija bloka bolje uočavala, u početku ćemo svakom bloku definisati boju ivica (*border*), što će se kasnije skloniti. Ako ove tri linije koda definišemo kao stil unutar *#okvir*{}, i to snimimo u fajl *style.css* dobijamo drugi efekat u browser-u, slika 3.22.

```

#okvir {
  width:1000px;
  margin:0px auto;
  border: 1px solid #000000; }

```



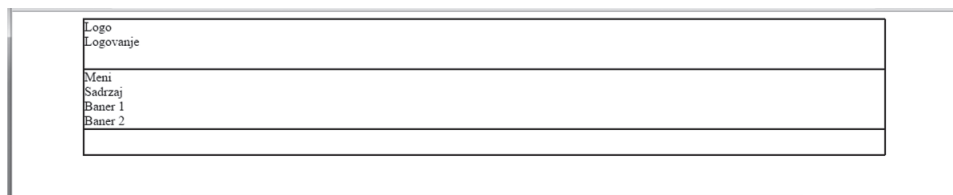
Slika 3.22. Prikaz koda u browser-u nakon definisanja stila za *#okvir*.

Vidi se da je ceo sadržaj centralno pozicioniran, da je definisana širina sajta manja od rezolucije monitora, i da je blok prikazan sa crnom okvirnom linijom. Unutrašnji sadržaj je i dalje poredan jedan ispod drugog, i prilepljen uz levu ivicu bloka `#okvir`.

Na sličan način definišu se stilovi i za `#vrh`, `#sredina` i `#podnožje` koji su tri taga ugnježdene u blok `#okvir`. Svakom od njih je definisana širina od 1000px, u skladu sa predloženom organizacijom na slici 3.20, kao i ivica, da bi se bolje videli. Međutim, ono što ih razlikuje je svojstvo *height*. Ovo svojstvo treba da se definiše kod svih blokova koji imaju definisan obim sadržaja, i koji nije planiran da se menja. Tamo gde se očekuje promena sadržaja, a najčešće je to centralni deo sajta, ovo svojstvo se ne definiše. U suprotnom, definisanjam visine bloka, većom nego što je sadržaj tog bloka, prikazivao bi se previše visok i polu prazan blok, tj. u suprotnom, može doći do odsecanja unetog HTML sadržaja, jer je on veći od definisane visine. U ovoj strukturi, pretpostavimo da blok *sadržaj* treba da bude taj koji će imati promenljiv obim sadržaja, pa se njemu neće definisati *height*. Međutim, svi regioni unutar kojih je ovaj blok ugnježđen, po pravilu, i iz istog razloga, ne treba da imaju limitiranu visinu da bi dozvolili bloku *sadržaj* da se eventualno „visinski širi“. Drugim blokovima i regionima, definišaće se, za sada proizvoljna visina. Proširenje CSS fajla je dato listingom.

```
#okvir {width:1000px;
        margin:0px auto;
        border: 1px solid #000000; }
#vrh { width:1000px;
        height: 60px;
        border: 1px solid #000000; }
#sredina {width:1000px;
          border: 1px solid #000000; }
#podnožje { width:1000px;
             height: 30px;
             border: 1px solid #000000; }
```

Ponovnim učitavanjem web strane, dobija se izgled kao na slici 3.23. Sada se vidi da su i tri unutrašnja bloka dobila svoje dužine.

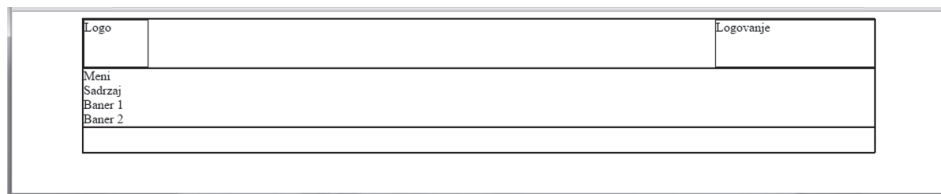


Slika 3.23. Prikaz koda u browser-u bez delovanja CSS koda

Na sličan način, dodajmo sada CSS kod za dva unutrašnja bloka `#vrh`, tj. `#logo` i `#logovanje`. Pored standardnih svojstava, ova dva bloka moraju da imaju i svojstvo *float*. Obzirom da ova dva bloka nemaju maksimalne dužine, tj. da su i

jedan i drugi unutar drugog bloka, mora se naglasiti ono što je i šemom definisano: ko je levo a ko desno. Kako je levo blok `#logo`, on će imati `float:left`, tj. `#logovanje` će imati `float:right`. Širine ova dva bloka, za sada, definišimo proizvoljnim vrednostima, npr. 80px i 200px, dok visina može biti 58px.

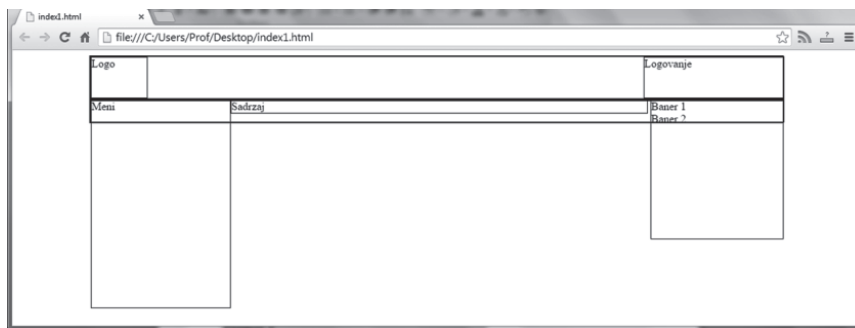
```
#logo {width:80px;
      height: 58px;
      border: 1px solid #000000;
      float:left; }
#logovanje { width:200px;
             height: 58px;
             border: 1px solid #000000;
             float:right; }
```



Slika 3.24. Prikaz koda u browser-u sa stilom za `#logo` i `#logovanje`

Sada je organizacija regiona `#vrh` završena, i sledeći je region `#sredina`. U njemu se nalaze tri bloka, jedan pored drugog: `#meni`, `#sadržaj` i `#baneri`. Već je objašnjeno da za `#sadržaj`, za sada ne treba stavljati `height`. Sva druga svojstva su već korišćena i poznata, i data u listingu. Bitno je naglasiti da suma dužina sva tri bloka, mora biti jednaka ili manja od dužine regiona u kome se nalaze (`#sredina`). Jedina dodatna dilema koja se nameće je `float`, obzirom da imamo tri bloka u nizu. Za sada usvojimo pravilo da su skroz levi i desni, `left` i `right`, a da srednji teorijski može biti i jedno i drugo. Iako praktično postoje neke potrebe da se ovo razlikuje, sada ćemo staviti da je i srednji blok `left`, što znači da se „lepi sa svoje leve strane“ u odnosu na prethodnika, tj. blok `#meni`. Ovaj dodatak CSS koda je dat u listingu.

```
#meni {
  width:200px;
  height: 300px;
  border: 1px solid #000000;
  float:left;}
#sadržaj {
  width:600px;
  border: 1px solid #000000;
  float:left;}
#baneri {
  width:190px;
  height: 200px;
  border: 1px solid #000000;
  float:right;}
```

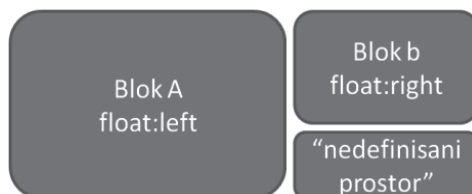


Slika 3.25. Prikaz koda u browser-u nakon dodavanja stilova za `#meni`, `#sadrzaj` i `#baneri`.

Kao što se vidi na slici 3.25. blokovi `#meni`, `#sadrzaj` i `#baneri` su sada postavljeni onako kako smo želeli, uz napomenu da visina `#sadrzaj` nije fiksno definisana, pa je zato onolika koliko u njemu trenutno ima HTML sadržaja. Međutim, ono što se jasno uočava je „raspadanje stranice“ jer je blok `#podnozje` sada prešao preko tri centralna bloka. Ova greška je vrlo česta i namerno je ostavljena da bi se ukazalo kako se do nje dolazi, kako se manifestuje i kako je sada treba rešiti.

Blok `#podnozje` je CSS-om definisan da bude sa određenom širinom, visinom i okvirom. Međutim, visinska pozicija bloka se ne definiše, nego se formira u skladu sa ostatkom blokova u web stranici. Tako se ni visinske pozicije za sve druge blokove nisu definisale, već su se blokovi „slagali“ učitavanjem redosleda HTML koda, onako kako je pisan. Međutim, iako je blok `#podnozje` potpuno definisan, sam za sebe, i redosledom HTML koda pisan iza bloka `#sadrzaj`, njegova pozicija je sada „ugrožena“. Ovo je posledica činjenice da se unutar strukture stranice teorijski nalazi neki prostor, koji je ostao nedefinisan, i gde sledeći blok pokušava da se „postavi“, slika 3.26.

Ovo se rešava posebnim svojstvom `clear`. Ovo svojstvo zabranjuje da se elementu na koji se odnosi, nešto definiše u odnosu na levu ili desnu stranu. Tako se primena ovog svojstva kreira kroz prazan div, koji sam po sebi pokušava da se „umetne u nedefinisanu prostor“.



Slika 3.26. Struktura blokova koja omogućava „nedefinisanu prostor“

Svojstvom `clear`, taj prazan div ne dozvoljava da neko nastavi da se postavlja pored njega (npr. levo i desno), što znači da je u toj visini on poslednji, a svi ostali idu „ispod“. Žargonski, kaže se da „*cleaner*“ počisti sva pozicioniranja u tom redu, i dalje sve počinje od novog reda, kao da je to novi početak. Tako je potrebno div „čistač“ staviti svaki put, kao poslednji u nizu, kada imamo više

elemenata **jedan pored drugog**, koji mogu biti i sa različitim visinama, širinama i/ili pozicioniranjima.

Treba naglasiti da u web strani ne treba da ostane ni jedan piksel prostora koji nije definisan ili „pokriven površinom nekog bloka“. Ukoliko se ovo desi, u kombinaciji sa svojstvom *float*, lako može da dođe do „raspadanja sajta“. Upotrebom *cistac*-a sav „nedefinisani“ prostor, u posmatranoj liniji, se „popunjava“ i sprečava se svaki neželjeni prikaz koda koji je ispod te linije koda.

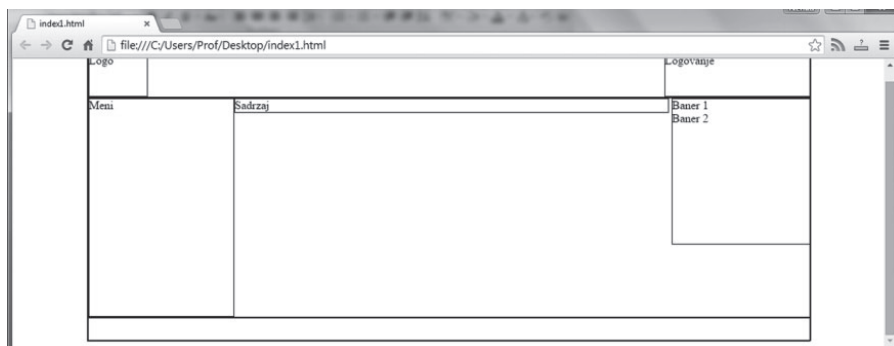
U ovom slučaju, stavićemo `<div id="cistac"> </div>`, unutar bloka `#sredina`, iza poslednjeg bloka `#baneri`, kao u boldovanom delu listinga.

```
<div id="sredina">
  <div id="meni">
    Meni
  </div>
  <div id="sadrzaj">
    Sadrzaj
  </div>
  <div id="baneri">
    <div class="baner">
      Baner 1
    </div>
    <div class="baner">
      Baner 2
    </div>
  </div>
  <div id="cistac">
  </div>
</div>
```

Za pomenuti div, u CSS fajlu treba napisati i njegov stil, `#cistac`.

```
#cistac { clear:both; }
```

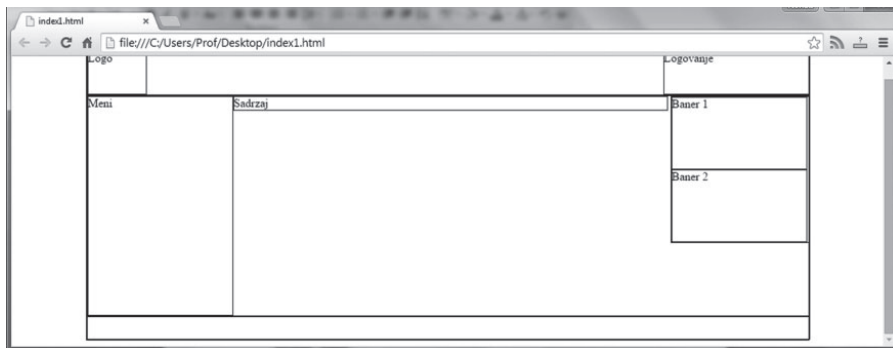
Pokretanjem ovog koda, u browser-u se primećuje da je problem loše pozicioniranog bloka `#podnozje`, rešen.



Slika 3.27. Prikaz koda u browser-u nakon dodavanja stilova za `#cistac`

Poslednji po hijerarhiji ugnježdavanja je region `#baneri`. Unutar njega se nalaze dva bloka, istog stila, `#baner`, pa su definisani atributom `class`. Dodavanjem stila i za njih, dobija se struktura stranice kao na slici 3.28.

```
.baner {width:186px;
        height: 98px;
        border: 1px solid #000000; }
```



Slika 3.28. Prikaz koda u browser-u nakon dodavanja stilova za `.baner`

Na kraju ove prve faze organizacije sadržaja stranice, dobili smo strukturu u web stranici onakvu kakva je crtežom, slika 3.20, bila definisana. Objedinjeni CSS fajl je dat u listingu.

style.css

```
#okvir { width:1000px;
        margin:0px auto;
        border: 1px solid #000000;}

#vrh {  width:1000px;
        height: 80px;
        background: url(uzorak.jpg) repeat-x center top;}

#sredina { width:1000px;
        border: 1px solid #000000;}

#podnozje { width:1000px;
        height: 30px;
        border: 1px solid #000000;}

#logo {  width:105px;
        height: 80px;
        float:left;
        background:url(logo.jpg) no-repeat center 0;}

#logovanje { width:250px;
        height: 53px;
        float:right;
        text-align:right;
        padding-top:5px;}

#meni {  width:200px;
        height: 300px;
```

```

border: 1px solid #000000;
float:left;}
#sadrzaj {
width:600px;
border: 1px solid #000000;
float:left;}
#baneri {
width:190px;
height: 200px;
border: 1px solid #000000;
float:right;}
#cistac { clear:both; }
.baner {
width:186px;
height: 98px;
border: 1px solid #000000; }

```

Sada kada je organizacija strane završena, u realnim sajtovima se dodaje sadržaj u skladu sa realizovanim fazama: dizajn i definisanje sadržaja strana. Iako ovaj primer ima za cilj da praktično pokaže organizaciju stranice, iskoristiće se za izradu proste web stranice. Cilj ovog nastavka je da se praktično pokaže primena još nekih CSS svojstava i potreba za korigovanjem inicijalnog CSS koda za organizaciju u skladu sa sadržajem.

Za početak, posmatrajmo region *#vrh*. U bloku *#logo*, potrebno je učitati logo sajta. Ovo se može realizovati dodanjem linije koda u stil *#logo*.

```
background:url(logo.jpg) no-repeat center 0;
```

Dalje, dodaćemo pozadinu regionu *#vrh*, kao multipliciranu sliku.

```
background: url(uzorak.jpg) repeat-x center top;
```

Shodno dostupnim slikama koje su korišćene, tj. njihovim dimenzijama, konačno su modifikovana ova tri stila, kao u listingu.

```

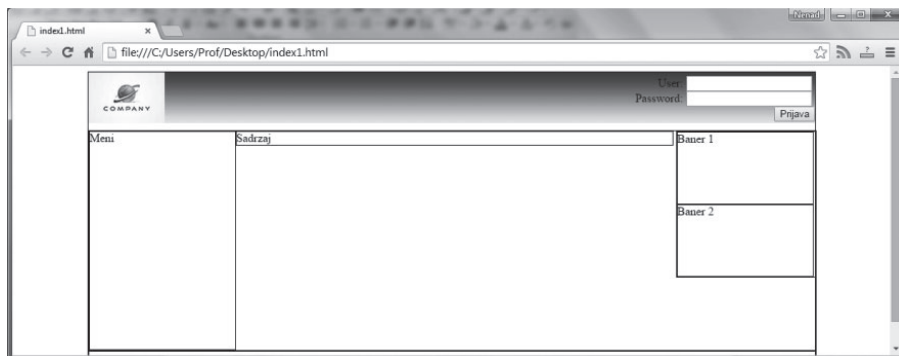
#vrh { width:1000px;
height: 80px;
background: url(uzorak.jpg) repeat-x center top; }
#logo {width:105px;
height: 80px;
float:left;
background:url(logo.jpg) no-repeat center 0; }
#logovanje {
width:250px;
height: 53px;
float:right;
text-align:right;
padding-top:5px; }

```

Sledeći korak je dodavanje sadržaja u blok *#logovanje*, što je forma za logovanje, koja se piše u HTML-u. Sadržaj ovog bloka u HTML fajlu je označen boldovanim tekstom.

```
<div id="logovanje">
  <form name="logovanje" method="post" action="">
    User: <input type="text" name="user" /> <br/>
    Password: <input type="text" name="password" /> <br/>
    <input type="submit" name="potvrdi" value="Prijava" />
  </form>
</div>
```

Snimanjem i učitavanjem stranice, dobija se izgled kao na slici 3.29. Kao što se primećuje, samo je promenjen deo regiona *#vrh*, dok je ostali deo nepromenjen.



Slika 3.29. Prikaz stranice nakon dodavanja dela sadržaja u bloku *#vrh*

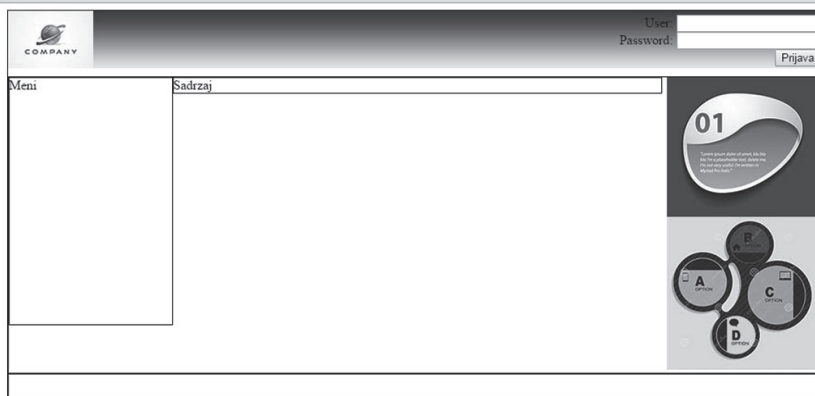
Ako se dalje pretpostavi da su baneri dve slike, u skladu sa njihovim dimenzijama, bilo je potrebno korigovati dimenzije stilova za baner, da bi se dobio lepši estetski utisak, bliži strukturi na slici 3.20. Kao što je napomenuto, ove dimenzije su prilikom kreiranja početnog CSS fajla stavljene proizvoljno, a sada poprimaju svoj finalni oblik.

```
#baneri {
  width:190px;
  height: 360px;
  float:right;
}
#cistac {
  clear:both;
}
.baner {
  width:186px;
  height: 170px;
}
```

Pored izmena u CSS-u, potrebno je i u HTML kodu dodati dve linije koda kojima se učitavaju dve slike za banere (*baner1.jpg* i *baner2.jpg*).

```
<div id="baneri">
    <div class="baner">
        
    </div>
    <div class="baner">
        
    </div>
</div>
```

Dodavanjem banera, i pokretanjem stranice, dobija se izgled kao na slici 3.30.



Slika 3.30. Prikaz stranice nakon dodavanja dela sadržaja u bloku #baneri

Sledeći blok je blok #meni. U ovaj blok treba staviti linkove za navigaciju. Po pravilu, meni se piše u formi neuređene liste, s tim što se grafički efekat elemenata liste (*list-style-type: none;*) može ukloniti u CSS-u, kao i inicijalno podvlačenje linkova (*text-decoration: none;*). Listing korigovanog dela CSS-a je dat u listingu.

```
#meni ul.navigacija {
    margin: 5px;
    padding: 0; }
#meni ul.navigacija li {
    list-style-type: none;
    padding: 10px; }
#meni ul.navigacija li a {
    color: #000333;
    font-size: 18px;
    text-decoration: none; }
```

Ostaje da se doda i sadržaj menija. Definisanjem elemenata liste, kao linkova, kreiramo meni sa željenim brojem linkova. Ovde je izabrano da bude njih šest.

```

<div id="meni">
<ul class="navigacija">
  <li><a href="index.html">Home</a></li>
  <li><a href="1.html">Prva</a></li>
  <li><a href="2.html">Druga</a></li>
  <li><a href="3.html">Treci</a></li>
  <li><a href="4.html">Cetvrti</a></li>
  <li><a href="5.html">Peta</a></li>
</ul>
</div>

```

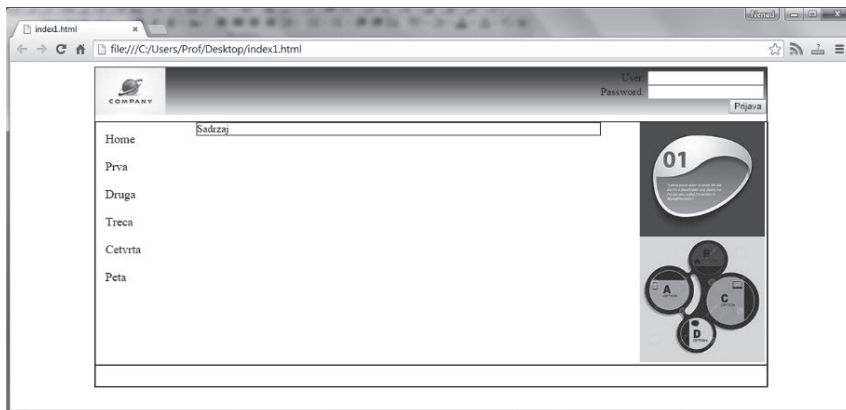
Obzirom da je dužina teksta linkova relativno mala, korigovaćemo širinu bloka `#meni` u CSS-u. Tako će stil `#meni` biti kao u listingu.

```

#meni {width:150px;
      height: 300px;
      float:left; }

```

Izgled stranice, nakon dodavanja dela za meni, je prikazan na slici 3.31.



Slika 3.31. Prikaz stranice nakon dodavanja dela sadržaja u bloku `#meni`

Obzirom da sada ivice blokova počinju da budu suvišne, jer su služile samo da vidimo pozicije blokova dok nije postojao njihov sadržaj, one se mogu ukloniti iz svih stilova. Pored toga, u blok *podnozje* ćemo dodati sliku kroz CSS, da bi kasnije u HTML preko nje mogli da definišemo neki tekst (tekstualni meni, autora, godinu, autorska prava i sl.). Deo korigovanog CSS-a je dat u listingu.

```

#okvir {width:1000px;
      margin:0px auto;}
#podnozje { width:1000px;
            height: 64px;
            background:url(dno.jpg) no-repeat center 0;}
#sredina {
      width:1000px;}

```

Na slici 3.32. dat je prikaz stranice sajta nakon ovih korekcija.

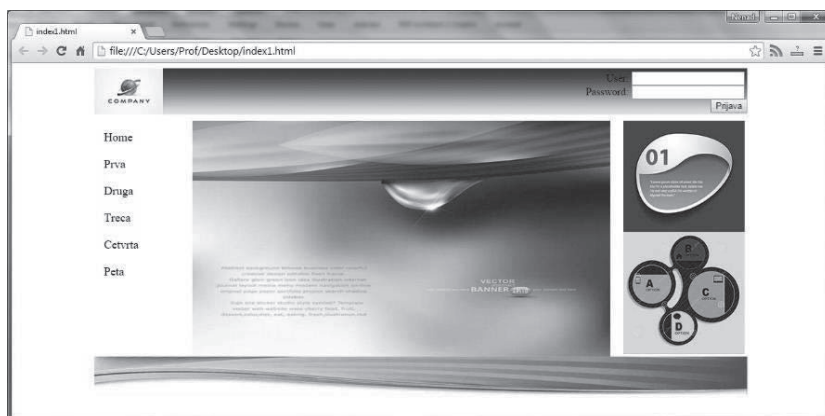


Slika 3.32. Prikaz stranice nakon dodavanja dela sadržaja u bloku #podnozje

U realnim sajtovima, centralni sadržaj bi bio skup ugnježenih blokova sa naslovima, podnaslovima, slika i sl. Međutim, za potrebe ovog prostog primera, učitaćemo u sadržaj samo jednu sliku, *sredina.jpg*.

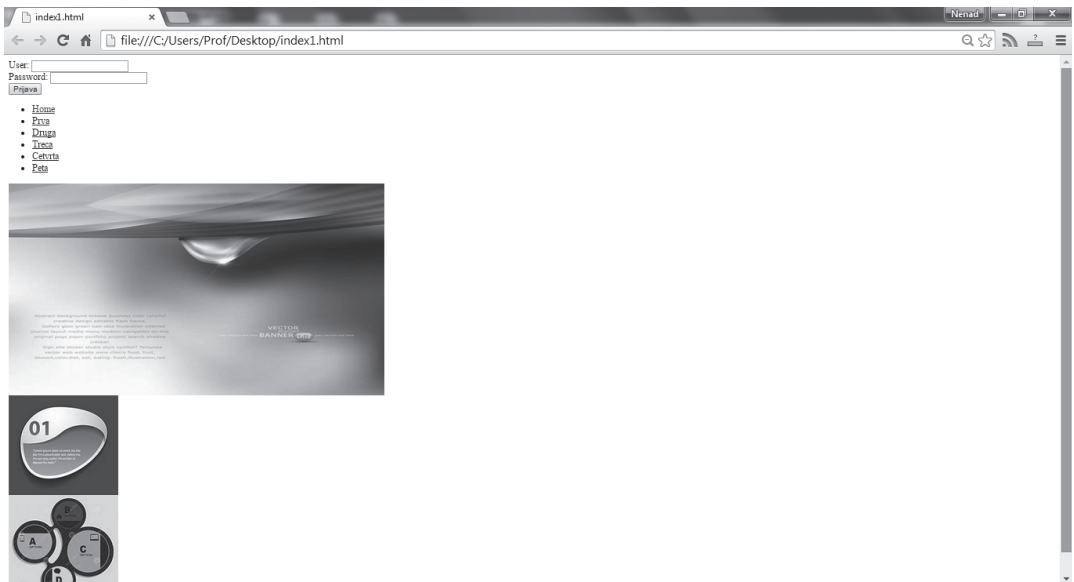
```
<div id="sadrzaj">
    
</div>
```

Konačni izgled stranice sajta, sa strukturom sa slike 3.20. i osnovnim sadržajem je prikazan na slici 3.33. Kao što je pokazano, celokupna organizacija stranice, dizajn sadržaja i grafika su realizovani pomoću CSS-a, dok je u HTML-u pisan samo osnovni deo sadržaja.



Slika 3.33. Prikaz koda u browser-u bez delovanja CSS koda

Ako bi se sada ovom sajtu, u HTML kodu izbrisala linija koda kojom se učitava eksterni CSS fajl, dobili bi izgled kao na slici 3.34.



Slika 3.34. Prikaz konačne verzije web stranice bez delovanja CSS koda

Kao što je sada i očekivano, isključivanjem CSS-a, prvo se izgubilo poravnanje blokova, pa je sav sadržaj uz levu ivicu browser-a. Sve slike učitane kroz CSS, sada se ne vide (logo, pozadina blokova *#vrh* i *#podnozje*). Meni se prikazuje sa podvučenim linkovima i tačkama, a elementi forme nisu desno poravnati. Na ovaj način, se uočava da je uloga CSS-a izuzetno velika u konačnom dizajnu jednog sajta.

Cilj ovog primera je da ukaže na osnovne principe i pravila kako treba kreirati HTML fajl koji se organizuje i stilizuje primenom CSS-a, dok je realna primena dodatno proširena CSS kodom za prefinjenije i detaljnije uticaje na pojedine korisničke sadržaje, i to se obrađuje unutar Praktikum.

Konačni izgled HTML fajla i CSS koda je dat u narednom listing.

index.html

```
<html>
<head>
<link rel="stylesheet" href="style.css" type="text/css" />
</head>
<body>
<div id="okvir">
<div id="vrh">
  <div id="logo">
  </div>
  <div id="logovanje">
    <form name="logovanje" method="post" action="">
      User: <input type="text" name="user" /> <br/>
      Password: <input type="text" name="password" /> <br/>
      <input type="submit" name="potvrdi" value="Prijava" />
    </form>
  </div>
</div>
</div>
```

```

        </form>
    </div>
</div>
<div id="sredina">
    <div id="meni">
<ul class="navigacija">
    <li><a href="index.html">Home</a></li>
    <li><a href="1.html">Prva</a></li>
    <li><a href="2.html">Druga</a></li>
    <li><a href="3.html">Treci</a></li>
    <li><a href="4.html">Cetvrti</a></li>
    <li><a href="5.html">Peta</a></li>
</ul>
    </div>
    <div id="sadrzaj">
        
    </div>
    <div id="baneri">
        <div class="baner">
            
        </div>
        <div class="baner">
            
        </div>
    </div>
<div id="cistac"> </div>
</div>
    <div id="podnozje">
    </div>
</div>
</body>
</html>

```

style.css

```

#okvir {width:1000px;
        margin:0px auto;}
#vrh { width:1000px;
        height: 80px;
        background: url(uzorak.jpg) repeat-x center top;}
#sredina {width:1000px;}
#podnozje { width:1000px;
            height: 64px;
            background:url(dno.jpg) no-repeat center 0;}
#logo {width:105px;
        height: 80px;
        float:left;
        background:url(logo.jpg) no-repeat center 0;}

```

```
#logovanje { width:250px;
              height: 53px;
              float:right;
              text-align:right;
              padding-top:5px;}
#meni {width:150px;
       height: 300px;
       float:left;}
#sadrzaj {width:640px;
         float:left;}
#baneri {width:190px;
         height: 360px;
         float:right;}
#cistac { clear:both; }
.baner {width:186px;
        height: 170px;}
#meni ul.navigacija { margin: 5px;
                     padding: 0;}
#meni ul.navigacija li { list-style-type: none;
                        padding:10px;}
#meni ul.navigacija li a {color: #000333;
                        font-size: 18px;
                        text-decoration: none;}
```

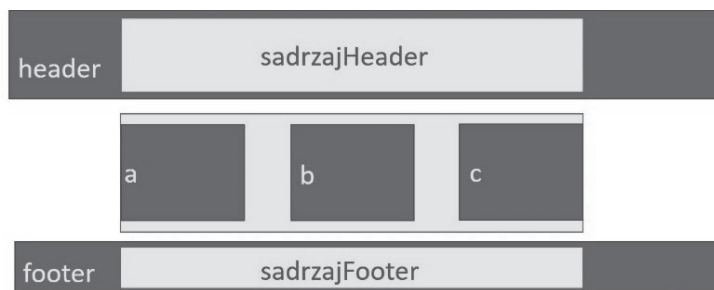
3.5.2. Primer 2

Prethodni primer je prikazao strukturu sajtova kod kojih regioni i blokovi imaju istu širinu, pa se na kraju postupka grupisanja dobija jedan region (omot). Kod sajtova novijeg datuma popularne su organizacije gde su širine pojedinih regiona celom širinom browser-a, dok su neki drugi centralno pozicionirani. U prvom primeru, pretpostavimo da se kreće od šematske strukture kao na slici 3.35., gde postoje tri bloka. Prvi i poslednji blok su postavljeni celom dužinom browser-a, koji je označen isprekidanim linijama. Srednji blok treba da bude centriran po horizontali.



Slika 3.35. Organizacija web strana sa blokovima koji su celom dužinom ekrana

Ovakva organizacija omogućava unos sadržaja u centralni blok, ali najčešće se očekuje da i sadržaji u gornjem i donjem bloku takođe budu u ravni sa sadržajem u centralnom bloku. Na ovaj način se kod promene rezolucije ekrana sav sadržaj prikazuje kao da ima zamišljeni region, sa fiksnom širinom, koji drži sav sadržaj sajta, a pozadinske boje gornjeg i donjeg bloka se „razvlače“ i popunjavaju levi i desni deo u odnosu na sadržaj. Da bi se ovo postiglo, potrebno je dodati posebne blokove koji će biti unutar gornjeg i donjeg bloka, i služiti da centriraju njihov unutrašnji sadržaj po horizontali, a u skladu sa srednjim blokom (sadržajHeader i sadržajFooter). Neka su imena gornjeg, srednjeg i donjeg bloka respektivno *header*, *okvir* i *footer*. Dodatno, pretpostavimo da je unutar bloka okvir sadržaj organizovan u tri bloka: *a*, *b* i *c*. Šematski prikaz takve organizacije tada postaje kao na slici 3.36.:



Slika 3.36. Modifikacija strukture sa slike 3.35.

HTML kod ovakve strukture je dat sa:

```
<html>
  <head>
    <title>Relativna duzina</title>
    <link rel="stylesheet" type="text/css" href="style.css"/>
  </head>
  <body>
    <div id="header">
      <div id="sadržajHeader">
        <p>Ovo je header!</p>
      </div>
    </div>
    <div id="okvir">
      <div id="a"></div> <div id="b"></div> <div id="c"></div>
    </div>
    <div id="footer">
      <div id="sadržajFooter">
        <p>Ovo je footer!</p>
      </div>
    </div>
  </body>
</html>
```

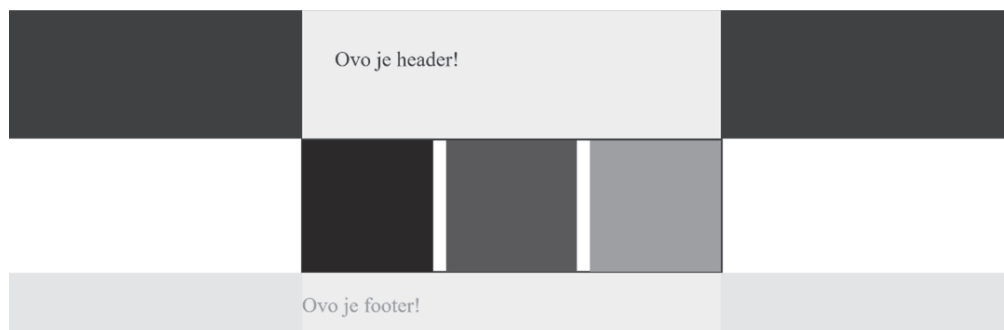
CSS kod definisani HTML kod je dat sa:

```

*{    margin:0px;
      padding:0px; }
p{    color:orange;}
#a, #b, #c{
      width:100px;
      height:100px;
      border:0px solid red;
      float:left; }
#a{    background-color:blue; }
#b{    margin-left:10px;
      background-color:green; }
#c{    margin-left:10px;
      background-color:orange;
}
#okvir{
      width:320px;
      height:100px;
      border:1px solid red;
      margin: 0px auto;}
#header{
      width:100%;
      height:100px;
      background-color:red;}
#footer{
      width:100%;
      height:50px;
      background-color:yellow;}
#sadrzajHeader{
      width:320px;
      height:100px;
      background-color:#f2f2f2;
      margin: 0px auto;}
#sadrzajHeader p{
      color:red;
      padding-left:25px;
      padding-top:30px;}
#sadrzajFooter{
      width:320px;
      height:50px;
      background-color:#f2f2f2;
      margin: 0px auto;}
#sadrzajFooter p{
      padding-top:15px;}

```

Dodatno, ukoliko se sadržaj bloka *sadržajHeader* želi pomeriti po dužini ili centrirati po visini, potrebno ga je staviti u neki tag kojim će se CSS-om upravljati. U skladu sa datim listingom dobija se sledeće prikaz:



Slika 3.37. Prikaz web stranice nakon modifikacija HTML i CSS koda

3.6. Pozicioniranje elemenata

Jedan od često korišćenog skupa svojstva u CSS-u su i svojstva za pozicioniranje elemenata. Ovo pozicioniranje istovremeno može da definiše i poziciju elementa unutar strane (u odnosu na granice browser-a, rezoluciju i sl.), međusobnu poziciju dva elementa (jedan iza drugog) ali i način prikaza elementa ukoliko je njegov sadržaj veći nego što je predviđeno.

Pozicioniranje elemenata se može realizovati primenom metoda i svojstava za pozicioniranje. Poželjno je prvo definisati metod za pozicioniranje, pa tek onda svojstvo, jer neka svojstva rade različito u zavisnosti koji metod se koristi.

Postoje ukupno četiri metoda za pozicioniranje (definiše se u odnosu na neki referentni sistem):

- Statičko pozicioniranje - ***position:static***
- Relativno pozicioniranje - ***position:relative***
- Apsolutno pozicioniranje - ***position:apsolute***
- Fiksno pozicioniranje - ***position:fixed***

Statičko pozicioniranje je podrazumevani oblik i primenjuje se ako programer ne definiše drugačije. Ovo pozicioniranje ne primenjuje ni jedan poseban tj. specifičan oblik pozicioniranja, već se elementi prikazuju redosledom pisanja koda u skladu sa inicijalnom veličinom elementa i same strane. Žargonski se kaže da elementi koji su statički pozicionirani nisu ni pozicionirani, jer se njihova pozicija prepušta bazičnom skupu podrazumevanih pravila. Kod ovog tipa pozicioniranja, svojstva za pozicioniranje (*top*, *left*, *bottom* i *right*) nemaju uticaja.

Relativno pozicioniranje je inicijalno isto kao i statičko. Međutim, ako se uključe i svojstva za pozicioniranje može se postići potpuno drugačiji efekat (pomeranje elementa na neku stranu ili preklapanje drugog elementa). Ovo pozicioniranje postavlja element relativno u odnosu na poziciju koju bi taj element imao da je statički pozicioniran. Posmatrajmo listing. Div *id="static"*, je postavljen onako kao što bi bio da statičko pozicioniranje nije eksplicitno ni navedeno. Pošto je prvi po redosledu pisanja, prikazuje se dužinom browser-a, od početka stranice, slika 3.38.

```

<html>
<head>
<style>
    #static {
        position: static;
        border: 1px solid;}
    #relative1 {
        position: relative;
        border: 1px solid;}
    #relative2 {
        position: relative;
        border: 1px solid;
        top: -30px;
        left: 40px;
        width: 300px;
        background-color: yellow;}
</style>
</head>
<body>
<div id="static">Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda static! Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda static!</div> <br/>

<div id="relative1">Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative! Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative!</div><br/>

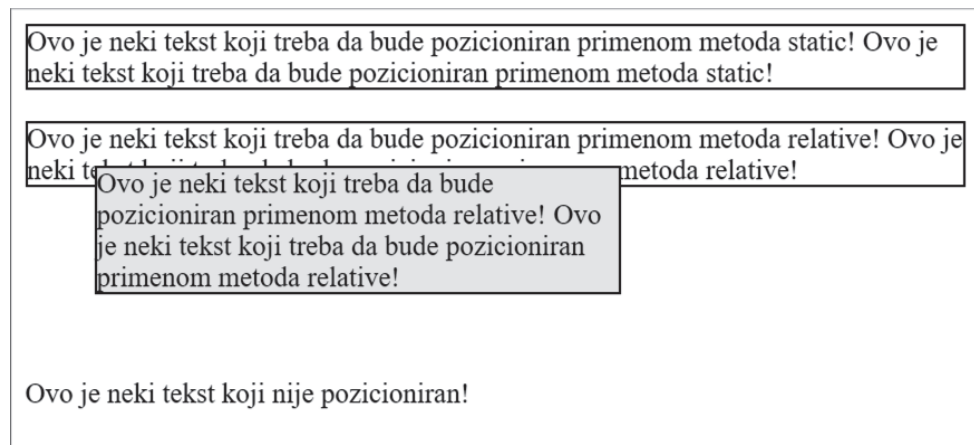
<div id="relative2">Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative! Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative!</div><br/>

<div id="nesto">Ovo je neki tekst koji nije pozicioniran!</div><br/>
</body>
</html>

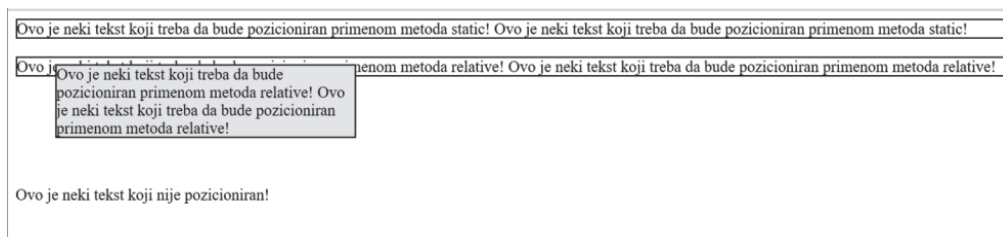
```

Postavljanjem drugog diva *id="relative1"*, koji samo ima definisano metod *relative*, bez upotrebe dodatnih svojstava, dobija se isti efekat kao i kod statičkog,

samo što se sadržaj prikazuje neposredno ispod, jer je i njegov kod napisan kao sledeći po redosledu.



Slika 3.38. Prikaz pozicioniranja metodom static i relative kada je dužina browser-a manja od maksimalne



Slika 3.39. Prikaz pozicioniranja metodom static i relative kada je dužina browser-a jednaka maksimalnoj

Postavljanjem trećeg diva `id="relative2"`, koji pored definisanog metoda relative, ima i svojstva za pozicioniranje `top: -30px;` i `left: 40px;`, element će se u odnosu na svoju očekivanu poziciju (kao da je bio statički pozicioniran) pomeriti za 30px na gore i 40px u desno. Na ovaj način element je prešao preko prethodnog, a mogao je u zavisnosti od vrednosti svojstava da pređe preko bilo kog drugog ili čak da izađe van vidljivog dela browser-a, i tako postane nevidljiv za korisnika.

Bitno je napomenuti da se ovakvim pozicioniranjem, prostor za element u browser-u, zauzima kao da je statički pozicioniran, iako se on prikazuje drugačije. Tako na primer, ako se neposredno iza ovog, relativno pozicioniranog, bloka definiše element (`id="nesto"`), koji nije eksplicitno pozicioniran, njegov prikaz će početi od mesta koje bi bilo kao i da je div `id="relative2"` statički pozicioniran. To znači da korisnik u prikazu vidi element `id="relative2"` pomeren na gore, ali je taj element rezervisao prostor unutar web strane kao da nije ni pomeran. Ova rezervacija prostora se odnosi na sav sadržaj koji dolazi nakon ovog elementa, slika 3.39.

Apsolutno pozicioniranje podrazumeva da se element pozicionira relativno ali u odnosu na svog roditelja (jednog od nadređenih) elementa. **Ovaj roditelj element je prvi nadređeni element koji ima pozicioniranje drugačije od statičkog.** U slučaju da takav roditelj element ne postoji, uzima se tag `<html>`.

Kod apsolutnog pozicioniranja može doći do potpuno drugačijeg prikaza u odnosu na podrazumevani prikaz elemenata. Iz ovog razloga, sam dokument (stranica) i svi drugi elementi se ponašaju kao da apsolutno pozicionirani element ne postoji i za njega se ne rezerviše prostor, kao u slučaju relativno pozicioniranog elementa. Žargonski kažemo kao da ne pripada sajtu i da je u potpuno paralelnoj ravni sa početnim sajtom.

Posmatrajmo naredni kod. Elementu `div id="absolute"` definisano je apsolutno adresiranje. Obzirom da je prvi roditelj ovog elementa tag `<body>`, i da on nema pozicioniranje drugačije od statičkog, za roditelja se uzima tag `<html>`. O odnosu na ovog roditelja posmatraju se svojstva za pozicioniranje `top: 10px;` i `right: 20px;`. Na taj način element koji je apsolutno pozicioniran će biti od ivica browser-a udaljen 10px odozgo i 20px od desne strane, čime će prekriti postojeći sadržaj.

Primetiti da unutar web strane, mesto na kome je kod ovog apsolutnog elementa napisan, nije rezervisano za njegov sadržaj, i nastavljeno je sa prikazivanjem sadržaja koji je napisan iza njega (`div id="nesto2"`).

```
<html>
<head><style>
    #static {
        position: static;
        border: 1px solid; }
    #relative1 {
        position: relative;
        border: 1px solid; }
    #relative2 {
        position: relative;
        border: 1px solid;
        top: -30px;
        left: 40px;
        width: 300px;
        background-color: red; }
    #absolute {
        position: absolute;
        border: 1px solid;
        top: 10px;
        right: 20px;
        width: 200px;
        height: 100px;
        background-color: yellow; }
</style></head>
```

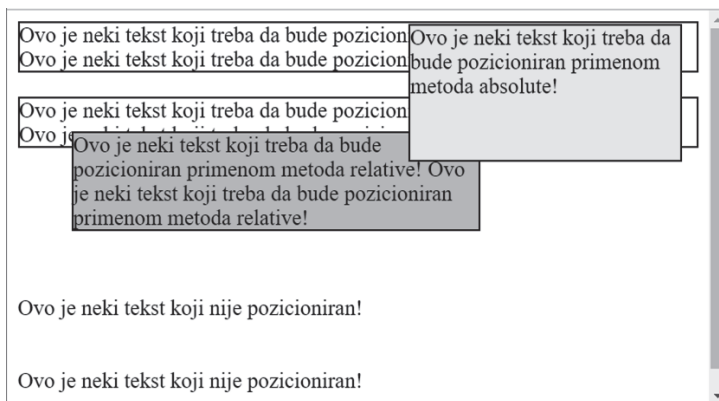
```
<body>
<div id="static">Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda static! Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda static!</div> <br/>
```

```
<div id="relative1">Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda relative! Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda relative!</div><br/>
```

```
<div id="relative2">Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda relative! Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda relative!</div><br/>
```

```
<div id="nesto">Ovo je neki tekst koji nije pozicioniran!</div><br/>
<div id="absolute">Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda absolute!</div><br/>
```

```
<div id="nesto2">Ovo je neki tekst koji nije pozicioniran!</div><br/>
</body>
</html>
```

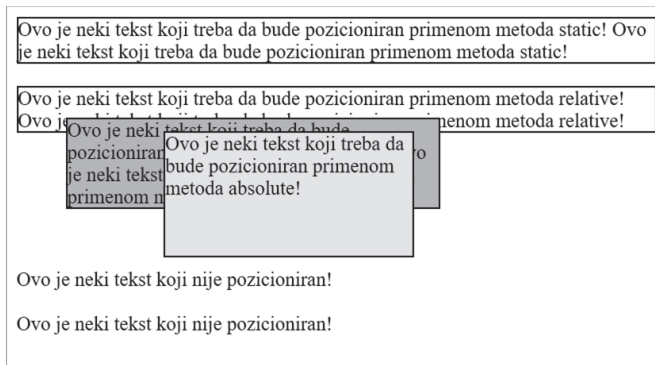


Slika 3.40. Prikaz pozicioniranja metodom absolute, primer 1

Ukoliko bi se element koji je apsolutno pozicioniran postavio unutar prethodnog elementa, tako što se iskopira na drugo mesto, kao u delu koda

```
<div id="relative2">Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative! Ovo je neki tekst koji treba da bude pozicioniran primenom
metoda relative!
<div id="absolute">Ovo je neki tekst koji treba da bude pozicioniran
primenom metoda absolute!</div><br/>
</div><br/>
```

tada bi roditelj bio element `id="relative2"`, pa bi u odnosu na ovog roditelja bila realizovana svojstva za pozicioniranje `top: 10px; i right: 20px;`.



Slika 3.41. Prikaz pozicioniranja metodom absolute, primer 2

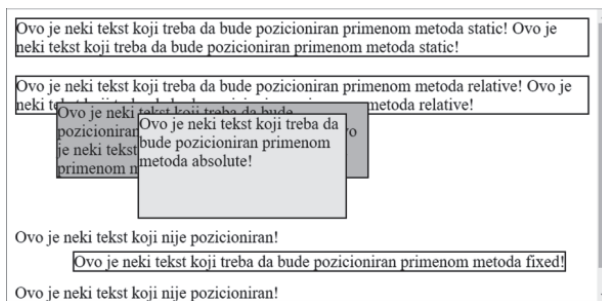
Fiksno pozicioniranje je relativno pozicioniranje u odnosu na ivice browser-a, i dok se veličina browser-a ne promeni, ne menja se ni pozicija fiksno pozicioniranog elementa. Ova pozicija je fiksna čak i u toku skrolovanja stranice. Kod ovog metoda pozicioniranja koriste se sva svojstva za pozicioniranje. Ako se postojećem kodu doda još jedan element, koji je fiksno pozicioniran, *id="fixed"*, tada se ovaj blok pozicionira u odnosu na dimenzije browser-a, tj. po 30px od donje i desne ivice.

```
<!DOCTYPE html>
<html>
<head>
<style>
#static { position: static;
          border: 1px solid; }
#relative1 { position: relative;
             border: 1px solid; }
#relative2 {
  position: relative;
  border: 1px solid;
  top: -30px;
  left: 40px;
  width: 300px;
  background-color: red; }
#absolute {
  position: absolute;
  border: 1px solid;
  top: 10px;
  right: 20px;
  width: 200px;
  height: 100px;
  background-color: yellow; }
#fixed {
  position: fixed;
  border: 1px solid;
  bottom: 30px;
```

```

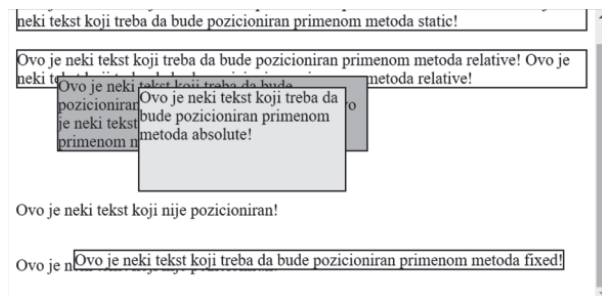
    right: 30px;
    background-color: white; }
</style>
</head>
<body>
    <div id="static">Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda static! Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda static!</div> <br/>
    <div id="relative1">Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda relative! Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda relative!</div><br/>
    <div id="relative2">Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda relative! Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda relative!
    <div id="absolute">Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda absolute!</div><br/>
    </div><br/>
    <div id="nesto">Ovo je neki tekst koji nije pozicioniran!</div><br/>
    <div id="fixed">Ovo je neki tekst koji treba da bude pozicioniran
    primenom metoda fixed!</div><br/>
    <div id="nesto2">Ovo je neki tekst koji nije pozicioniran!</div><br/>
</body>
</html>

```



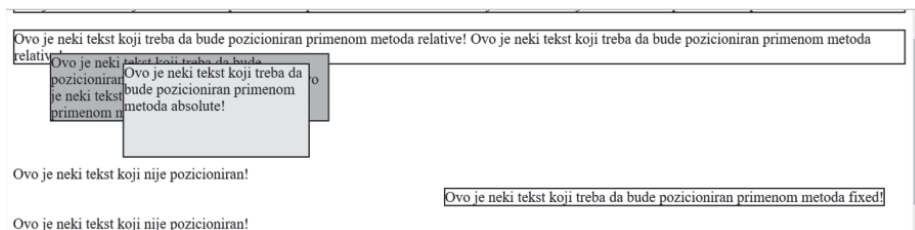
Slika 3.42. Prikaz pozicioniranja metodom *fixed*, primer 1

Skrolovanjem stranice na dole, pozicija fiksnog bloka i dalje ostaje po 30px od donje i desne ivice, bez obzira kako se ostatak sadržaja web strane prikazuje.



Slika 3.43. Prikaz pozicioniranja metodom *fixed*, primer 2.

Pozicija ovog bloka će vizuelno izgledati drugačije (jer se menja način prikaza ostalog sadržaja) samo ako se veličina browser-a promeni, ali i dalje ostaje po 30px od donje i desne ivice.



Slika 3.44. Prikaz pozicioniranja metodom *fixed*, primer 3.

■ **Napomena:** Internet Explorer podržava fiksno pozicioniranje samo ako postoji !DOCTYPE.

Preklapanje elemenata

Pored opisanih metoda pozicioniranja, CSS omogućava i preklapanje elemenata, čime se pozicioniranje posmatra u odnosu na “dubinu” web strane. Ovo se postiže sa svojom **z-index**. Na ovaj način postiže se upravljanje redosledom elemenata koji se prikazuje jedan preko drugog tj. iza drugog. Upravljanjem **z-index**-om moguće je samo ako se ne koristi statičko pozicioniranje.

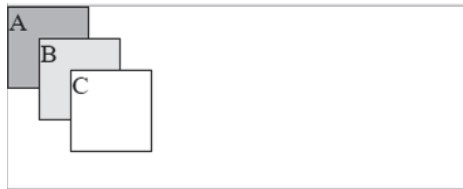
```
<!DOCTYPE html>
<html>
<head><style>
    #prvi { position: absolute;
            border: 1px solid;
            left: 0px; top: 0px;
            width:50px; height:50px;
            z-index: -1;
            background-color: red;}
    #drugi {position: absolute;
            border: 1px solid;
            left: 20px; top: 20px;
            width:50px; height:50px;
            z-index: -1;
            background-color: yellow;}
    #treći { position: absolute;
            border: 1px solid;
            left: 40px; top: 40px;
            width:50px; height:50px;
            z-index: -1;
            background-color: white;}
</style>
</head>
```



```

<body>
  <div id="prvi">A</div>
  <div id="drugi">B</div>
  <div id="treći">C</div>
</body>
</html>

```



Slika 3.45. Primer preklapanja elemenata primenom z-index-a

3.7. Uslovni komentari

Uslovni komentari (*Conditional comments*) služe isključivo za browser Internet Explorer, od verzije 5 do verzije 9, sa ciljem da daju dodatne instrukcije tim browser-ima.

Veliki problem kod CSS-a je što različiti browser-i različito interpretiraju pojedina CSS svojstva. Ovaj problem u Internet Explorer-u se rešava primenom uslovnih komentara. Na ovaj način, programer po potrebi piše npr. dva eksterna CSS fajla: jedan sa svojstvima i vrednostima namenjenim grupi browser-a a drugo za IE. Da bi kod “znao” kada koji CSS fajl da pozove, potrebno je dodati kod za prepoznavanje IE, i učitavanje CSS namenjenog za IE. Kako ne postoji posebna komanda za ovakvu akciju, koristi se proširena verzija komentara: uslovni komentar.

Primer primene uslovnog komentara za IE verzija 8, kojim se učitava eksterni CSS fajl, *primer1.css*, dat je sa

```

<!--[if IE 8]>
  <link rel="stylesheet" type="text/css" href="primer1.css" />
<![endif]-->

```

U slučaju da se želi pisati različit CSS kod za različite verzije IE, ova grupa koda se ponavlja za svaku verziju IE-a. U slučaju da se neke posebne napomene žele uraditi od verzije IE10, to je moguće uraditi primenom skript jezika.

3.8. Jedinice mere u CSS-u

Iako u realnom životu za dužinu imamo usvojenu samo jednu jedinicu mere (metar), u CSS-u postoji veliki broj različitih jedinica mera kojima se definiše dužina. Ovim jedinicama definišu se vrednosti velikog broja svojstava (npr.

width, height, margin, padding, font-size, border, ...). Po pravilu vrednost svojstva koje opisuje dužinu treba da ima brojnu vrednost i jedinicu mere, bez međusobnih razmaka (npr: 19%, 25px, 2.5em, ...). Za pojedina svojstva, ove vrednosti mogu imati i negativne vrednosti.

U CSS-u postoje dva tipa mernih jedinica: apsolutne i relativne.

- ✓ Apsolutne mere imaju unapred definisanu veličinu i način prikaza u browser-u, i u primeni su: centimetar (*cm*), milimetar (*mm*), inč (*in*), tačka (*pt* gde je $1pt = 1/72in$) i picas (*pc*, gde je $1pc = 12pt$).
- ✓ Relativne mere nemaju unapred definisanu veličinu i njihova konačna veličina se računa u odnosu na neko drugo svojstvo koje definiše dužinu. Ovaj tip mera je vrlo zahvalan kada se radi o medijumima sa promenljivim rezolucijama. U primeni su: *em* (relativni odnos prema svojstvu elementa *font-size*. Npr. 2.5em znači 2.5 puta veće od trenutne veličine fonta u elementu), *ex* (predstavlja relativni odnos prema visini trenutnog fonta), *ch* (relativna mera u odnosu na širinu, sa nultom referentnom vrednošću), *rem* (relativna mera u odnosu na veličinu fonta *root* elementa), *vw* (1% širine prikaza u browser-u), *vh* (1% visine prikaza u browser-u), *vmin* (1% minimalne širine prikaza u browser-u), *vmax* (1% maksimalne širine prikaza u browser-u) i *px* ($1px = 1/96in$).

Primeri upotrebe različitih jedinica mere:

```

<!DOCTYPE html>
<html>
<head>
<style>
    #prvi { font-size: 1.5cm;}
    #drugi {font-size: 40px;}
    #drugi p {font-size: 0.6em;}
</style>
</head>

<body>
<div id="prvi">Tekst velicine 1.5cm, definisanih sa font-size: 1.5cm;</div>
<div id="drugi">

Tekst velicine 40px, definisanih sa font-size: 40px;
    <p>Tekst velicine 24px, definisanih sa font-size: 0.6em; (0.6*40=24)<p>
</div>

<div> Tekst podrazumevane velicine </div>
</body>
</html>

```



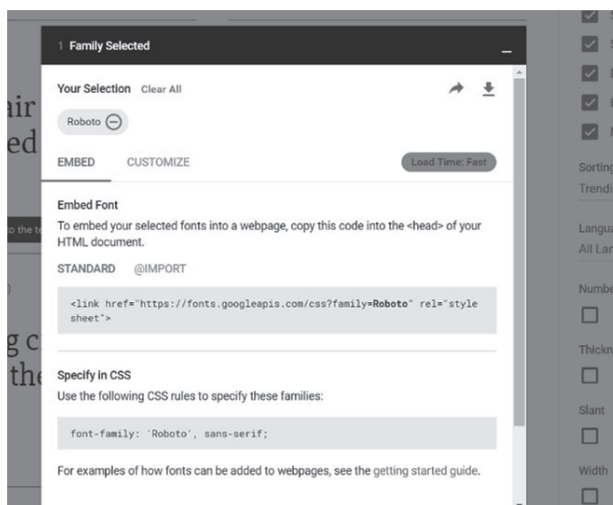
Slika 3.46. Prikaz teksta primenom različitih mernih jedinica

3.9. Primeri parcijalnih primena

Nakon do sada obrađenih poglavlja, u kojima su data pravila, imena svojstava i njegove očekivane vrednosti, u ovom poglavlju obrađiće se neki parcijalni primeri kombinacije opisanih svojstava tj. selektora koji se često koriste u realizaciji realnih sajtova. Ovi primeri su međusobno nepovezani i mogu se proizvoljno kombinovati sa drugim primerima, primenama i organizacijama sadržaja unutar sajtova. Ovi primeri su izdvojeni kao odgovori na najčešća pitanja studenata i dilema na koje su nailazili kod ovih primena.

Primer 1: Učitavanje eksternog fonta

Pored fontova koje korisnici dobijaju sa operativnim sistemom, sve češće se koriste i fontovi koji su naknadno kreirani i imaju savremeni dizajn. Ove fontove programer može koristiti samo ako ih korisnik ima na svom računaru da bi kod korisnika mogli da se prikažu u browser-u. Kako je teško pretpostaviti da li svi korisnici imaju određeni font, često se koristi princip da se neki poseban font isporuči korisniku zajedno sa web sajtom, putem *http_request*-a. Na netu ima puno sajtova sa kojim se mogu preuzeti posebno kreirani fontovi. Jedan od najkorišćenijih je <https://fonts.google.com/>.



Slika 3.47. Prikaz prozora nakon selekcije željenog fonta

Na ovom sajtu, nakon izbora nekog željenog fonta, npr. *Roboto*, potrebno je kliknuti na +, u gornjem desnom uglu, i u posebnom prozoru koji se pojavi prikazaće se deo koda koji je potreban za programera da ga integriše u svoj sajt. Za font *Roboto* ovi podaci izgledaju kao na slici 3.94. Prva linija koda je link ka eksternom fajlu, koji je font, na adresi <https://fonts.googleapis.com>, i koji treba staviti u *head* sekciju web strane. Na taj način font će se isporučiti korisničkom računaru i biti dostupan za upotrebu. Da bi ovako dostupan *font* mogao da se upotrebi u sajtu potrebno je da se u CSS kodu definiše svojstvo *font-family: 'Roboto', sans-serif*;, i eksterni font će se prikazati kod korisnika.

U primeru datog koda korišćen je interni CSS zbog sveobuhvatnog koda, ali se u realnim primenama savetuje upotreba eksternog CSS-a.

```
<html>
<head>
<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
<style>
    p{font-family: 'Roboto', sans-serif;}
</style>
</head>
<body>
    <p> Proba </p>
</body>
</html>
```

Primer 2: Izrada horizontalnog menija

Ranije je napomenuto da sve oblike menija treba praviti u formi neuređene liste. Polazeći od HTML liste, najčešće se prvo prozivanjem selektora *ul* neutrališu vidljivi buleti elemenata liste. Zatim se svakom elementu liste definiše da će imati strukturu bloka i blokovi poređaju horizontalno sa *float:left*. Dalje se prozivanjem linkova, tj. taga *a*, definiše njihov *padding* kojim se definiše površina pravougaonika u kome se prikazuje link.

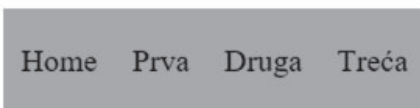
Često se destilizuje link sa podvučenim tekstom pomoću *text-decoration:none*. Inicijalno će elementi liste, tj. menija, biti neznatno odvojeni. Često je to poželjno, ali kada nije definiše se negativna margina da bi se oni još više približili ili delimično preklopili. Vrlo profesionalni pristup zahteva i definisanje hover efekta, i diskretnu promenu ili pozadinske boje i/ili boje teksta linka da bi korisnik imao grafički efekat kada se kreće preko linkova u meniju.

Ukoliko se definiše *li a:hover* kao selektor, to znači da će se prelaskom miša preko linka, baš na tom linku promeniti neko od definisanih svojstava. U ovom slučaju pozadinska boja i boja teksta.

```

<html>
<head>
<style>
ul{ list-style-type: none; }
li{ display: block;
    float:left; }
li a{ color: #000000;
    padding: 18px 10px;
    text-decoration: none;
    background-color: #00ff00;
    margin-left:-2px; }
li a:hover{ background-color: gray;
    color: white; }
</style>
</head>
<body>
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="prva.html">Prva</a></li>
        <li><a href="druga.html">Druga</a></li>
        <li><a href="treca.html">Treća</a></li>
    </ul>
</body>
</html>

```



Slika 3.48. Prikaz menija iz Primera 2.



Slika 3.49. Prikaz menija iz Primera 1 kada se mišem pređe preko linka Prva.

Primer 3: Stilizovanje forme

Stilizovanje elemenata forme i cele forme je postalo skoro obavezan stilski detalj u modernim sajtovim. Najčešće se formi dodeljuje neka pozadinska boja, zaobljene ivice elemenata forme kao i same pozadine, upravljanje dužinom elemenata forme, ali i upotreba *padding*-a kojim se uneti sadržaj u npr. tekstualna polja dodatno pomera u odnosu na ivice elemenata i na taj način dodatno stilizuje. Za ove potrebe koriste se standardna svojstva ali je karakteristična upotreba selektora kojim se selektuje više elemenata forme, kao na primer `input[type=text]` za selekciju svih tekstualnih elemenata na toj stranici. Primer stilizovanje jedne forme je dat u listingu:

```

<html>
<style>
input[type=text], select {
    width: 100%;
    padding: 12px 20px;
    margin: 10px 0;
    border: 1px solid gray;
    border-radius: 4px;
    box-sizing: border-box;}
input[type=submit] {
    width: 100%;
    background-color: #4CAF50;
    color: white;
    padding: 14px 20px;
    margin: 10px 0;
    border: none;
    border-radius: 4px;
    cursor: pointer;}
#form {
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;}
</style>
<body>
<div id="form">

    <form action="/action_page.php">
        <label for="ime">Ime</label>
        <input type="text" id="ime" name="tbime" placeholder="Uneti ime..">

        <label for="prezime">Prezime</label>
        <input type="text" id="prezime" name="tbprezime" placeholder="Uneti
prezime...">

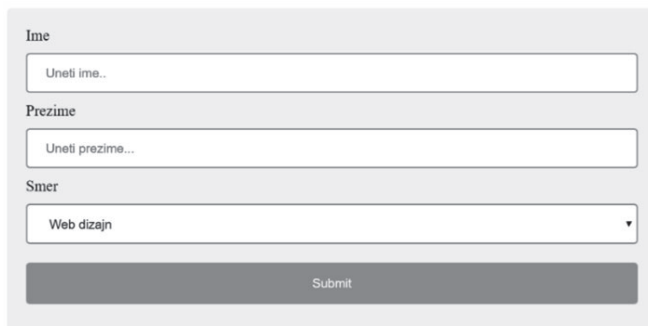
        <label for="smer">Smer</label>
        <select id="smer" name="dldsmer">
            <option value="1">Web dizajn</option>
            <option value="2">Osnove programiranja</option>
            <option value="3">Matematika</option>
        </select>

        <input type="submit" value="Submit">
    </form>
</div>

</body>
</html>

```

■ **Napomena:** Atribut placeholder pripada HTML-u 5 dok svojstvo border-radius i box-sizing pripada CSS 3.



A stylized form with a light gray background. It contains three input fields stacked vertically, each with a label above it. The first field is labeled 'Ime' and has the placeholder text 'Uneti ime...'. The second field is labeled 'Prezime' and has the placeholder text 'Uneti prezime...'. The third field is labeled 'Smer' and has a dropdown menu with the selected option 'Web dizajn'. Below the input fields is a dark gray 'Submit' button.

Slika 3.50. Prikaz stilizovane forme iz Primera 3.

IV HTML 5

HyperText Markup Language ver. 5

HTML 5 je trenutno najnovija verzija HTML-a. Pojavila se 2009. godine kao naslednik HTML 4.01 i XHTML-a, koji datiraju od 1997. tj. 1999. godine. Obzirom na veliki period od prethodne verzije, bilo je očekivano da će se nova verzija pojaviti sa velikim brojem izmena, iako se realno nije očekivalo onoliko koliko ih je HTML 5 doneo.

HTML 5 je dobio veliki broj novih tagova i atributa, izgubio neke od starijih HTML 4 tagova i atributa ali i modifikovao primene i mogućnosti nekih od postojećih tagova. Međutim ključna promena je u načinu na koji se posmatra uloga HTML-a kao jezika u kreiranju web stranica. Naime, novi HTML ima neuporedivo veću interakciju i nadležnosti na klijentskoj strani kada je reč o CSS-u i integraciji sa Java script-om. HTML 5 dolazi sa API-jima od kojih su pojedini bili konceptualno nezamislivi u ranijem periodu (rad sa podacima, geo lokacije, rad sa regularnim izrazima, grafikom, upravljanje multimedijom, ...). Sve ove novine, a naročito one koje se odnose na grafički prikaz pojedinih elemenata web strane, su mnogo atraktivnije i primamljivije za krajnjeg korisnika.

Nezvanično, HTML 5 matematički možemo zamisliti kao

$$HTML\ 5 \approx HTML\ 5 + CSS\ 3 + JavaScript$$

Na ovaj način HTML 5 je alat za:

- ✓ Označavanje - *Markup* (HTML 5)
- ✓ Prikazivanje - *Presentation* (CSS 3)
- ✓ Interakciju - *Interaction* (DOM, Ajax, APIs)

Treba naglasiti da HTML 5 nije novi markap jezik, tj. da je način rada i najveći deo postojećih tagova, atributa i sl. zadržan, ali da predstavlja vrlo kvalitetnu i revolucionarnu nadogradnju HTML-a.

Ključni problem u primeni HTML 5, u ovom trenutku, je što veliki broj browser-a, tj. njihovih starijih verzija, ne može da ispravno interpretira sve novine koje HTML 5 donosi, pa to predstavlja veliki problem za programere. U tom smislu, treba doneti tešku odluku da li sajt raditi sa velikim brojem HTML 5 elemenata, što je diktirano strukturom ciljne grupe korisnika, tehničkim predispozicijama istih, tj. neophodnošću da se pojedine stvari rade ili u HTML 4 ili HTML 5 verziji, imajući u vidu da neki korisnici neće moći da vide sajt onako kako je to zamišljeno. Trenutno se najčešće koristi jedna hibridna verzija, tj. koriste se tagovi HTML 5 koji nemaju grafičku interpretaciju (*header*, *nav*, *article*, ...) i neće mnogo narušiti koncept, ukoliko ne mogu da se prikažu, a doprinose boljoj organizaciji sadržaja i predstavljanja strane web pretraživaču. Ovo je posebno bitno ako se inicijalno ne zna koja je struktura korisnika kojima je sajt namenjen, pa se čeka da se vremenom, preko alata za analitiku, stekne uvid u tehničke karakteristike *browser*-a koje koriste posetioци sajta. Iako je HTML 5 doneo velike novine, ukupan efekat na kvalitet i dizajn web stranice dodatno podržavaju unapređeni DOM, API-ji i CSS 3, koji omogućavaju podršku za novu eru u web dizajnu.

4.1. Ključne novine u HTML 5

Kao što je navedeno, HTML 5 dolazi sa nizom novina od kojih se ključne sledeće:

1. Novi elementi za definisanje strukture web stranice (*<header>*, *<footer>*, *<nav>*...)
2. Novi elementi u formama i mogućnost validacije na klijentskoj strani
3. Rad sa multimedijalnim fajlovima (audio i video kontrole)
4. API-ji za grafiku i crtanje
5. Skladištenja podataka
6. Geo-lokacija
7. Mogućnost *Drag & Drop*-a
8. *Offline* aplikacije, ...

Pored ovih formalnih novina, uočava se i novi skraćeni oblik pisanja pojedinih tagova (npr. *meta*, *doctype*, ...). Interesantno je da se više ne mora koristiti osnova struktura tagova za web stranu (*<html>*, *<head>* i *<body>*) i da browser-i sami prepoznaju tagove koji pripadaju ovim tagovima, i ove strukturne tagove dodaju u kod strane. Dozvoljeno je i pisanje vrednosti atributa bez upotrebe znakova navoda. Međutim, lično ne preporučujem da se ove fleksibilne opcije koriste, dok to ne postane jedini oblik standardizovanog zapisa.

4.2. HTML 4 u odnosu na HTML 5

Neke od primarnih karakteristika HTML 4 koje se drugačije od HTML 5 su:

- ✓ Razumljiv svim browser-ima
- ✓ Nije zamišljen za upravljanje audio i video fajlovima
- ✓ Ne podržava skladištenje, van lokalnog računara ili lokalne mreže
- ✓ Ne podržava validaciju podataka
- ✓ Ne podržava interaktivno crtanje
- ✓ Ne odgovara zahtevima savremenih korisnika
- ✓ Nema kontrolu sintakasnih grešaka

Sa druge strane, HTML 5 ima sledeće karakteristike:

- ✓ Nije razumljiv svim browser-ima
- ✓ Napravljen za rad sa audio i video fajlovima
- ✓ Podržava skladištenje van mreže
- ✓ Donosi nove tagove, attribute i funkcionalnosti
- ✓ Omogućava da se posebnim tagovima definiše tip i namena korisničkog sadržaja
- ✓ Podržava interaktivno crtanje
- ✓ Podržava različite tipove unosa u formulare i validaciju istih
- ✓ Ima mogućnost nekog oblika detektovanja grešaka

Pored ovih suštinskih razlika, postoji razlika i u skupu tagova, atributima i njihovih vrednosti u ove dve verzije. Pojedini tagovi iz HTML 4 ne postoje u HTML 5, dok su nekim tagovima proširene mogućnosti primene (npr. *a*, *hr*, *cite*, *strong*).

Spisak tagova koji ne postoje u HTML 5, i koji ili potpuno nestaju ili se njihova uloga prebacuje na delovanje CSS-a su:

```
<acronym> (<abbr>)
<applet>(<object>)
<basefont> (CSS)
<big>(CSS)
<center>(CSS)
<dir>(<ul>)
<font>(CSS)
<frame>
<frameset>
<noframes>
<tt>(CSS)
```

Posebna novina je proširenje mogućnosti elemenata formi, koje su sada dobile nove tipove elemenata. Tako se sada kao vrednosti atributa *type*, taga *input*, pojavljuju i vrednosti:

color	range
date	search
datetime	tel
datetime-local	time
email	url
month	week
number	

Novi tagovi, koje donosi HTML 5, su:

<section>,	<progress>,
<article>,	<meter>,
<aside>,	<time>,
<hgroup>,	<ruby>,
<header>,	<rt>,
<footer>,	<rp>,
<nav>,	<wbr>,
<figure>,	<canvas>,
<figcaption>,	<command>,
<video>,	<details>,
<audio>,	<summary>,
<source>,	<datalist>,
<embed>,	<keygen> i
<mark>,	<output>

Pojedini novi tagovi imaju mnogo širi oblik delovanja nego što je to do sada bio slučaj. Tako na primer, tag *<canvas>*, formalno predstavlja jedan novi tag, iako se pomoću njega može isprogramirati veliki broj grafičkih oblika i animacija primenom Java script-a, što mu daje beskonačno mnogo različitih načina prikaza i primena.

U daljem tekstu će se objasniti načini implementacije novina koje donosi HTML 5, s tim što sve što je zadržano iz verzije HTML 4, a objašnjeno u Poglavlju II, podrazumeva i važi na isti način kao što je tamo opisano.

4.3. Primene HTML 5 na početku web strane

Kako je pisanje HTML 5 u skladu sa definisanim sintaksom XHTML, to svaka stranica mora da počne definisanjem „tipa dokumenta“. Posmatrajmo kako je ta linija koda izgledala u HTML 4 verziji ili XHTML 1.0, respektivno:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

HTML 5 uvodi drastično skraćenje i sintaksa je definisana sa

```
<!DOCTYPE html>
```

Vidi se da ranije verzije DOCTYPE-a uključuju u sebe celu URL adresu ukazujući na određeni fajl koji browser-u treba da pomogne u definisanju načina kako da se određeni HTML, CSS i DOM tretira i prikazuje. Na ovaj način, isti kod, u različitim browser-ima, će se tretirati i prikazivati na isti način, što je primarni cilj svakog web programera.

Nakon definisanja ove prve linije koda, programer se već u *head* sekciji susreće sa drugim novinama. Naime, definisanje kodnog rasporeda, kojim se omogućava prikaz karakterističnih slova i simbola, ranije je definisan kao

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

dok se u HTML 5 definiše skraćeno kao

```
<meta charset="utf-8">
```

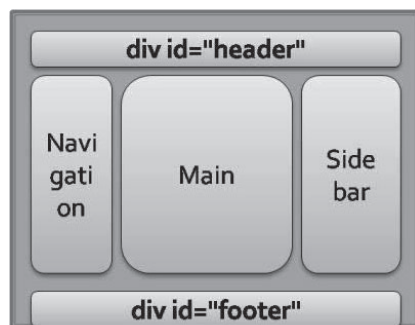
Očigledno je da je jedna od ideja HTML 5 bila skraćivanje pojedinih delova koda koji su se vrlo često koristili u većem broju sajtova, čime se programerima olakšao budući rad. Ova logika primenjena je i na delove koji pripadaju vidljivoj delu sajta, tj. *body* tagu, što će se sledeće prikazati.

4.4. Primene HTML 5 za organizaciju strane

U HTML 5 se pojavljuju novi elementi kojima se vrši struktuiranje sadržaja web strane. Do sada se to realizovalo korišćenjem taga *div*, i njemu pridruženim CSS-om. Ono što je pokazala statistika je da imena atributa *id* ili *class*, u velikom broju imaju iste ili slične sadržaje, tj. imena (*wrapper*, *body*, *main*, *navigation*, *menu*, *footer*, *top*, *banner*, *content*, *left side*, ...). Kako su programeri često pisali ovakav kod, sa jasnom željom da se ukaže na koji deo sadržaja stranice se misli, a kako to zbog jezičkih barijera ni web pretraživačima nije najjasnije, u HTML 5 su dodati novi tagovi koji će jednoznačno objediniti sve korišćene varijacije. Tako sada imamo tagove: *header*, *nav*, *article*, *section*, *aside*, *footer* koji vrlo jasno treba da definišu vrstu sadržaja koja je u njima.

Na primer, posmatrajmo klasičnu strukturu nekog sajta datu sa organizacionom šemom i HTML kodom.

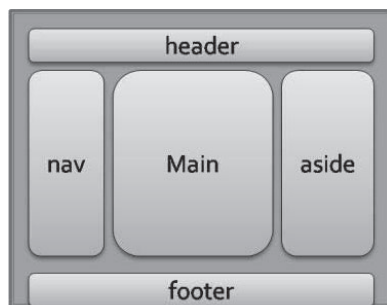
```
<body>
  <div id="header">...</div>
  <div id="navigation">...</div>
  <div id="main">...</div>
  <div id="sidebar">...</div>
  <div id="footer">...</div>
</body>
```



Slika 4.1. Primer organizacione strukture sajta

Primenom novih HTML 5 tagova za strukturu stranice to sada može da izgleda kao:

```
<body>
  <header>...</header>
  <nav>...</nav>
  <div id="main">...</div>
  <aside>...</aside>
  <footer>...</footer>
</body>
```



Slika 4.2. Primer organizacione šeme sajta primenom HTML 5 tagova

Na ovaj način i čoveku i mašini je mnogo jednostavnije i intuitivnije da razume pozicije pojedinih delova stranica i njihovog sadržaja.

→*Header* tag, unutar *body* taga, definiše zaglavlje dokumenta, ali sada i određene sekcije ili manje logičke celine unutar web stranice. Ako se posmatra standardna primena ovog taga, tada se unutar njega može definisati npr. *h1* tag i neki drugi tag ili slika, ukazujući na neku celinu.

```
<header>
  <h1>Ovo je header</h1>
  <p class="tagline">Ovo je dodatni tekst u header-u</p>
</header>
```

→Tag *nav* definiše deo sajta u kome se nalazi navigacija. Najčešće je to HTML neuređena lista linkova, da bi se pomoću CSS-a i jQuery-ja mogla estetski lako urediti i/ili kasnije animirati. Primer jednog *nav* taga mogao bi da bude:

```
<nav>
  <ul>
    <li> <a href="index.html"> Početna strana </a> </li>
    <li> <a href="galerija.html"> Slike</a> </li>
    <li> <a href="lokacija.html"> Lokacija</a> </li>
  </ul>
</nav>
```

→*Footer* tag, unutar *body* taga, definiše kraj dokumenta ili sekcije. Tipično, sadrži tekst i/ili tekstualnu navigaciju. Ranijim primenama, kao i kod taga *header*, podrazumeva jedan tag *footer* (kao dete taga *body*), dok ih sada može biti više, ako su to oznake završetka pojedinih sekcija, kao sadržaja centralnog dela sajta. Primer jednog *footer* taga bi mogao biti

```
<footer class="stil1">
  Ovo je footer
</footer>
```

Obzirom da pretraživači “vole” tekstualnu navigaciju u footer-u, često se koristi kombinacija *nav* i *footer* taga, u podnožju web stranice

```
<footer >
  <p>
    Footer text i vlasnička prava
  </p>
  <nav>
    <h6>Quick Links</h6>
    <ul>
      <li>
        <a href="#">link 1</a>
      </li>
    </ul>
  </nav>
</footer>
```

Section i article

Ova dva nova taga treba da ukažu na logički način grupisanja sadržaja namenjenog korisniku. Iako nije striktno definisano kada ide jedan a kada drugi, dosadašnje iskustvo definiše sledeće preporuke:

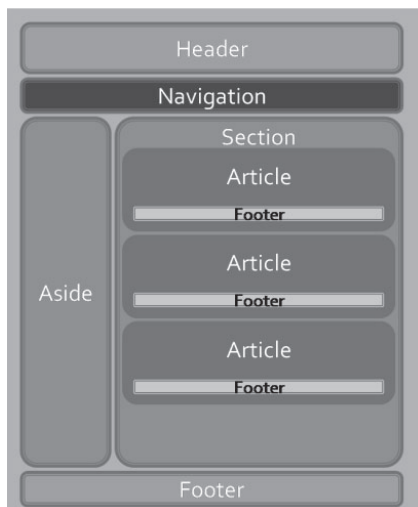
Tag *section* treba koristiti kada se strana ili deo strane logički strukturiše, tj. deli, i tada tag *section* definiše jednu užu, specifičnu celinu.

Tag *article* se koristi kada je sadržaj koji on sadrži jedinstven i koncizno definisan i skoro unikatan.

Možda je najbolje poređenje primene ova dva taga sa štampanim novinama: U novini postoji više „sekcija“ (Zabava, Kultura, Vesti iz zemlje, Pitanja čitalaca i sl.). Svaka od ovih sekcija ima više članaka (*article*) unutar sebe (festivali, muzika, koncerti, posete, konferencije, moda,...). Na taj način unutar taga *section* ima jedan ili više taga *article*. Međutim, neki članak (*article*) može imati više segmenata (Više pitanja i odgovora o gajenju cveća), pa je unutar njega novi *section*, sa novim skupom tagova *article* (Konkretno pitanje čitaoca i odgovor urednika). Obzirom da bi *article* trebao da ima neki konkretan i jedinstven sadržaj, često je njegov sastavi deo i neki *header*, sadržaj i *footer*. Primer ovakvog uređenja je dat u listingu.

```
<article>
  <header>
    <h1> Plac na Avali </h1>
    <p> Agencija Placevi</p>
  </header>
  <p> Prodajem plac na Avali .... </p>
  
  <footer>
    Za dodatne informacije pogledajte sajt <a href=""> aaa </a> ili
    pozovite na 222222
  </footer>
</article>
```

Na sličan način, unutar jednog *section* može se naći više *article* od kojih svaki ima samo npr. *footer*, Slika 4.3.



Slika 4.3. Primer primene tagova *section* i *article*

→Tag *aside* se često koristi da se u njega stavi sadržaj koji najčešće stoji skroz levo ili desno unutar stranice. Tako u njemu može biti celokupna navigacija sa tagom *nav*, baneri, slike, reklame, ili bilo šta se treba odvojiti od centralnog dela sajta.

→Pored ovih „globalnih“ organizacionih tagova, u ovu grupu se može svrstati i tag *hgroup*. Samo ime taga ukazuje da se ovim tagom vrši grupisanje više *h* tagova. Ovo je posebno bitno imajući u vidu da SEO „voli“ *h* tagove, a da se često uz naslov grupiše i njegov pod naslov u drugoj veličini *h* taga. Primer *hgroup* taga može biti:

```
<hgroup>
  <h2> Sutra novi popusti </h2>
  <h4> Patike i cipele po dodatnih 20% popusta</h4>
</hgroup>
```

4.5. Primene HTML 5 za vizuelne efekte

Pored standardnih HTML 4 tagova, kojima se određeni deo sadržaja na neki način grupiše ili odvaja od drugih, HTML 5 je ponudio i neke nove tagove za potrebe isticanja teksta ili informacija na stranici.

→Tagom *mark* postiže se efekat isticanja teksta, na isti način kao kada se markerom deo teksta oboji u odštampanoj strani. Primer primene *mark* elementa dat je u listingu.

```
<body>
  Deo teksta koji je <mark> markiran </mark>
</body>
```

Deo teksta koji je markiran

Slika 4.4. Primer primene taga *mark*

→Novi tag *progress* omogućava da se u grafičkom režimu prikazuje napredak određenog procesa. Atributima *max* i *value*, definiše se koji deo maksimalne dužine će biti obojen drugom bojom.

```
Napredak <progress value="70" max="100"></progress>
```

Napredak 

Slika 4.5. Primer primene taga *progress*

→Vrlo slično tagu *progress*, funkcioniše i novi tag *meter*. Ovaj tag za razliku od taga *progress*, koristi se kada se zna i minimalna i maksimalna vrednost unutar koje se pokazuje neki obojeni deo. Definisanjem minimalne vrednosti 0 a maksimalne 10, ukoliko se želi prikazati vrednost 7, pišemo:

```
<meter min="0" max="10" value="7"></meter>
```



Slika 4.6. Primer primene taga *meter* sa *min="0"*

dok se definisanjem minimalne vrednosti na 5 ima

```
<meter min="5" max="10" value="7"></meter>
```

čime se i grafički prikaz menja. Naime sada je dužina obojenog dela kraća, jer je iscertavanje od vrednosti 5, a ne od vrednosti 0.



Slika 4.7. Primer primene taga *meter* sa *min="5"*

U HTML 4 je slika bila celina za sebe i kreirana kroz tag *img*. U realnim sajtovima, slika ima neki opis, naziv ili objašnjenje koji se prikazuje korisniku, neposredno iznad, pored ili ispod slike. Međutim, taj tekst nikako nije kodom povezan sa slikom, i ne čini logičku celinu, sem što se korisniku prikazuje kao tekst fizički blizu slike, pa ga doživljavamo kao njen opis.

U HTML 5, ovo je promenjeno, obzirom na zastupljenost slika u web stranicama i potrebe da se slika opiše tekстом kako korisniku tako i web pretraživaču. Za ove potrebe definisana su dva taga: *figure* i *figcaption*.

→*Figure* je tag koji treba da ukaže na neki objekat (slika, video, animacija, flash, ...), dok tag *figcaption* služi da prikaže ime objekta ili opis objekta. Pozicija taga *figcaption* može biti ispred ili iza tog objekta.



Slika 4.8. Primer primene tagova `figure` i `figcaption`.

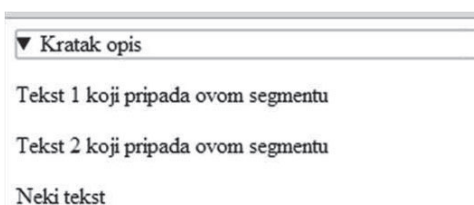
```
<body>
<figure>
  <figcaption>Ime slike</figcaption>
  
</figure>
</body>
```

→Dva nova taga, *details* i *summary*, omogućavaju inicijalno grupisanje i sakrivanje teksta, koja se nalazi u tagu *details*, sa izuzetkom da se prikazuje samo tekst koji je u tagu *summary*. Prikaz preostalog teksta se realizuje klikom miša na uokvireni tekst.

```
<body>
<details>
  <summary>Kratak opis</summary>
  <p> Tekst 1 koji pripada ovom segmentu </p>
  <p> Tekst 2 koji pripada ovom segmentu </p>
</details>
<p> Neki tekst </p>
</body>
```



Slika 4.9. Primer primene tagova *details* i *summary*, sa redukovanim prikazom



Slika 4.10. Primer primene tagova *details* i *summary*, sa kompletnim prikazom

→Tag *bdi* služi za realizaciju bidirekcionog teksta. Ovim se postiže ispravan prikaz pisama koja se realizuju s leva na desno (npr. srpski) ali i s desna na levo (npr. arapski).

→Tagovi *ruby*, *rt* i *rp* se najčešće koriste kada se pojavi potreba za dodatnim tumačenjem ili opisom pojedinih termina, fraza ili simbola u nekom od npr. azijskih jezika.

→Tag *wbr* daje mogućnost browser-u da može da reč ili deo teksta, koji je dug, prelomi u novi red, na određenom mestu, gde se to tagom predloži. Ovo se radi u situacijama kada nismo sigurni gde će se prelom realizovati, pa se ovim tagom „sugeriše“ gde bi to trebalo da se desi.

→Posebna novina je tag *command* i *menu*, koji je zamišljen da se unutar njega definišu paketi alati tj. meniji za određenu aplikaciju, sa prikazom ikonice koji

su pridruženi toj aplikaciji. Klik na neku od ikonica poziva script koji dalje realizuje određenu akciju. Problem ovog taga je što ga do sada podržava samo jedan browser, pa su primene do sada bile male.

4.6. HTML 5 tagovi za forme

U HTML 5 se pojavljuju tri nova taga za potrebe formi: *datalist*, *keygen* i *output*.

→Tag *datalist* najviše odgovara HTML 4 tagu *select*, i koristi se u kombinaciji sa tagovima *option*. Ovaj tag nema grafičku vizuelizaciju, i služi da ponudi definisane opcije kao sadržaj tekstualnog polja. To zahteva definisanje tekstualnog polja i njegovo „vezivanje“ sa listom (*datalist*). Ova veza je preko atributa *list* (u *input* tagu) i atributa *id* (u *datalist* tagu).

```
<form>
  <input type="text"
list="moja_lista" />
  <datalist id="moja_lista" >
    <option value="ABV" />
    <option value="123" />
    <option value="XYZ" />
  </datalist>
</form>
```



Slika 4.11. Primer primene taga *datalist*

→Glavni razlog za uvođenje taga *keygen* je omogućavanje sigurnog načina prenosa podataka (npr. autentifikacije korisnika)

```
<keygen name="security" />
```

Ovaj tag definiše dva ključa, trenutkom submit-ovanja, i to javni i privatni. Privatni se skladišti lokalno, dok se javni šalje ka web serveru (npr. javni se može koristiti za pravljenje klijentskog sertifikata za autentifikaciju korisnika).

→Tag *output* omogućava prikaz neke vrednosti koja je dobijena kao rezultat određene operacije. U HTML 4, ovo je moglo da se realizuje jedino primenom JavaScript-a, dok je sada ta integracija spuštена na novo HTML-a. Primer integracije sa tipom *range*, sa ciljem prikaza trenutne vrednosti grafičkog klizača, dat je u listingu:

```
<form oninput="vrednost.value=opseg.value" >
  <input type="range" name="opseg" id="opseg" />
  <output id="vrednost"> Izaberite vrednost </output>
</form>
```



Slika 4.12. Primer primene taga *output*

4.7. HTML 5 tag <time>

Tag *time*, koristi se za definisanje datuma i vremena koje je namenjeno mašini (web pretraživaču, skript jezicima, web servisima, ...) i korisnicima sajta. Ovaj tag omogućava jedinstveni zapis kojim se datum tj. vreme istovremeno zapisuje u formatu koji je prilagođen i softveru i korisniku. Atribut ovog taga koji sadrži ključni sadržaj je *datetime*.

Sintaksa za definisanje datuma je YYYY-MM-DD (godina-mesec-dan), dok je za vreme HH:MM:SS (sat-minut-sekund).

Oznaka *T* koristi se da bi se razdvojili datum i vreme, ukoliko se oba prikazuju, dok na kraju ide oznaka (*Z*) koja je oznaka vremenske zone.

Primeri primene taga *time*:

```
<time datetime="2012-11-15"> 15. Novembar 2012 </time>
<time datetime="15:00"> 15 časova </time>
<time datetime="2012-11-15T15:00"> 15. Novembar 2012 </time>
<time datetime="2012-11-15T15:00:22.4 +00:00"> 15. Novembar 2012
</time>
```

→Tag *time* ima i dodatni atribut *pubdate* koji ukazuje na datum objavljivanja određenog teksta, vesti ili sadržaja. Iako se piše u tagu *time*, i u skladu sa do sada opisanom sintaksom, on ne ukazuje na trenutno vreme nego na trenutak u prošlosti ili budućnosti, kada se određena vest želi označiti kao objavljena tj. „aktivna“.

```
<article>
  <header>
    <h1> Plac na Avali </h1>
    <p> Agencija Placevi</p>
    <time datetime="2012-11-15T15:00" pubdate> 15. Novembar 2012
  </time>
  </header>
  <p> Prodajem plac na Avali .... </p>
  
  <footer>
    Za dodatne informacije pozovite na 222222.
  </footer>
</article>
```

4.8. Novi tipovi taga input

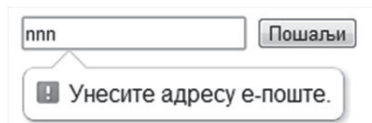
Tag `<input />` je u HTML 5 proširen sa velikim brojem novih vrednosti atributa *type*. Ovim novim tipovima, HTML 5 dobija potpuno novu dimenziju u odnosu na HTML 4, i elementi formulara dobijaju mogućnost ugrađene validacije unetog sadržaja. U najvećem broju novih tipova, ime tipa jasno ukazuje na sadržaj koji je dozvoljen da se unese. → Novi tipovi su: *search*, *email*, *url*, *tel*, *datetime*, *date*, *month*, *week*, *time*, *datetime-local*, *number*, *range* i *color*.

Tako na primer polje koje je kodom definisano da se koristi za pretragu sadržaja se kreira kao:

```
<input type="search" />
```

Često korišćeno polje je tipa *email*. Definisanjem ovog tipa polja, nakon submit-ovanja forme, vrši se validacija unetog sadržaja. Ukoliko uneti sadržaj ne odgovara opštem formatu email adrese, prijaviće se greška, u grafičkom okruženju, koje je već isprogramirano. Pored mogućnosti da se unese jedna email adresa, moguće je uneti i grupu adresa. Ovo je moguće ukoliko se u tagu *input* definiše atribut *multiple*, a unete email adrese međusobno razdvajaju pomoću , (simbola za zarez).

```
<form>
  <input type="email" name="usermail" />
  <input type="submit" value="Пошаљи" />
</form>
```



Slika 4.13. Primer primene atributa *type="email"*

Slično kao i tip *email*, na Internetu se često zahteva unos URL adrese. Polje tipa *url* omogućava automatsku proveru unetog sadržaja, nakon submit-ovanja forme. Unosom pogrešnog formata, prikazuje se unapred isprogramirano grafičko okruženje o pogrešno unetoj URL adresi.

```
<form>
  <input type="url" name="url" />
  <input type="submit" value="Пошаљи" />
</form>
```



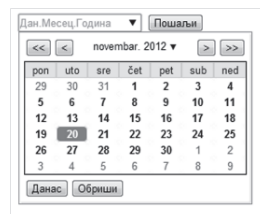
Slika 4.14. Primer primene atributa *type="url"*

Poseban tip tekstualnog polja je i *tel*. Ovaj tip podataka dozvoljava vrlo fleksibilan format, jer su formati telefonskog broja vrlo različiti.

```
<input type="tel" name="tel" />
```

Tip *date* je vrlo dugo očekivan jer je grafički prikaz kalendara i izbora datuma, pre HTML 5, realizovan primenom velikog broja linija koda viših programskih jezika. Sada je to moguće realizovati samo jednom linijom HTML koda.

```
<form>
  <input type="date" name="proba"/>
  <input type="submit" value="Пошаљи" />
</form>
```



Slika 4.15. Primer primene atributa *type="date"*

Slično tipu *date*, tip *time* omogućava formatiran unos zapisa o vremenu.

```
<form>
  <input type="time" name="proba"/>
  <input type="submit" value="Пошаљи" />
</form>
```



Slika 4.16. Primer primene atributa *type="time"*

→Na isti način kao i *date* tj. *time*, koriste se i *datetime*, *month* ili *week*. Njihov cilj je da se ograniči i ispravno definiše unos datuma i vremena, tj. meseca ili sedmice. Za veliki broj browser-a postoji problem za prikaz ovih tipova *input* taga, pa se još uvek ređe koriste u praksi.

Tip *number* omogućava da se u polje upiše samo broj, koji se u pojedinim browser-ima može klikom na strelice automatski inkrementirati (povećati) ili dekrementirati (smanjiti).

```
<form>
  <input type="number" name="broj" />
  <input type="submit" />
</form>
```



Slika 4.17. Primer primene atributa *type="number"*

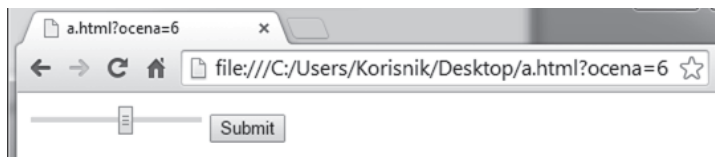
Kao atributi ovog taga, na raspolaganju su *max*, *min*, *step* i *value*, i kod svih je sadržaj neka numerička vrednost.

→ Tip *range* omogućava da se pomoću klizača automatski izabere određena vrednost koja je u definisanom opsegu (atributi *min* i *max*). Submit-ovanjem forme odabrana vrednost se automatski prosleđuje web serveru što se može videti u URL adresi.

```
<form>
  <input type="range" min="1" max="10" name="ocena" />
  <input type="submit" />
</form>
```



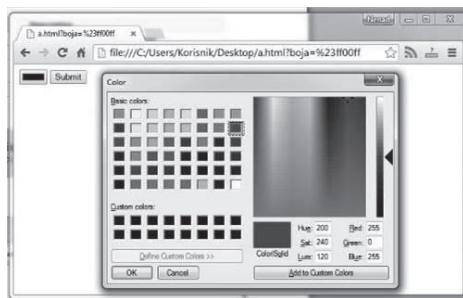
Slika 4.18. Primer primene atributa *type="range"*



Slika 4.19. Primer primene atributa *type="range"* sa prenosom vrednosti GET metodom (*ocena=6*)

→ Jedno od vrlo atraktivnih grafičkih okruženja pripada tipu *color*. Ovaj tip tekstualnog polja omogućava da se izborom boje, u grafičkom okruženju, web serveru pošalje heksadecimalni predstavnik izabrane boje. Ovo se može videti nakon submit-ovanja forme u URL adresi.

```
<form>
  <input type="color" name="boja" />
  <input type="submit" />
</form>
```



Slika 4.20. Primer primene atributa *type="color"*

4.9. Novi atributi

Pored novih tagova i opisanih vrednosti atributa *type*, HTML 5 je doneo i nove attribute. Ovi atributi su dodeljeni različitim tagovima, i u pojedinim situacijama, omogućavaju komforniji rad korisnika i posao web programera.

Atribut *autocomplete*

Ovaj atribut omogućava da se korisniku ponudi sadržaj, ili njegov deo, koji je ranije uneo, ukucavanjem prvih karaktera tog sadržaja. Može se primeniti na *input* ili *form* tag. Vrednost za aktivaciju je *on* a zabranu *off*.

```
<form >
    <input type="text" name="opseg" autocomplete="on" />
</form>
```



Slika 4.21. Primer primene atributa *autocomplete*

Atribut *autofocus*

Ovaj atribut omogućava da se po učitavanju stranice izvrši automatski fokus na određeni element forme (npr. kursor trepće u definisanom *text* polju) i na taj način se korisniku skreće pažnja da se od njega očekuje unos sadržaja u tom elementu forme.

```
<form >
    <input type="text" name="opseg" autofocus />
</form>
```

Atribut *form*

Ovaj atribut omogućava da se element koji se fizički nalazi van neke forme, tj. taga *<form>* i *</form>*, pridruži baš toj formi. Ovo je moguće realizovati tako što će se formi definisati atribut *id*, a elementu koji treba da se toj formi pridruži, atribut *form*, sa vrednošću prethodno definisanog atributa *id*.

```
<form id="forma1">
<input type="text" name="opseg" autocomplete="off" autofocus />
</form>

<input type="text" form="forma1" />
```

Sa HTML 5 pojavljuje se potpuno drugačija logika definisanja pojedinih atributa elemenata forme, koji su do sada isključivo pripadali tagu *form*.

Tako se uvodi i pojam *override*, u slučaju kada element forme ima, poseban, novi atribut. Na ovaj način, iako nadređeni tag *form* ima neko pravilo, element forme može to da promeni (*override*-uje), samo za sebe.

Do sada su se kao atributi taga *form* pojavljivali *action* i *method* a sada je moguće definisati i *target* i *validate*. Ovo je moguće uraditi kroz nove attribute taga *input*:

formaction (*type*="submit" i *type*="image")

formmethod (*type*="submit" i *type*="image")

formnovalidate (*type*="submit")

formtarget ()

→Drugi način, pored neuparenog taga *input*, da se u HTML-u kreira taster, je tag *button*. Ako se posmatra upareni tag *button*, koji može biti tipa *button*, *reset* i *submit*, i ako se želi kreirati tip *submit*, sa tekstom *Prijava*, definiše se sledeći kod:

```
<button type="submit">Prijava</button>
```

Ukoliko se kreira forma sa ovakvim tasterom, tada imamo

```
<form action="1.php" method="get">
  Telefon: <input type="text" name="tel"/>
  <button type="submit">Prijava</button>
</form>
```

Ako se formi doda još jedan taster, *Prijava 2*, kojim se želi postići slanje podataka ka drugoj stranici web servera, npr. *2.php*, tada imamo

```
<form action="1.php" method="get">
  Telefon: <input type="text" name="tel"/>
  <button type="submit">Prijava</button>
  <button type="submit" formaction="2.php"> Prijava 2</button>
</form>
```

Ako se sada istoj formi doda treći taster, *Prijava 3*, moguće je podesiti da se podaci forme pošalju na novu stranicu *3.php*, ali i drugim metodom, npr. *post*.

```
<form action="1.php" method="get">
  Telefon: <input type="text" name="tel"/>
  <button type="submit">Prijava</button>
  <button type="submit" formaction="2.php"> Prijava 2</button>
  <button type="submit" formmethod="post" formaction="3.php"> Prijava
3</button>
</form>
```

Primer uključivanja atributa *formnovalidate*, kojim se zaobilazi provera podataka pre slanja serveru, realizovaće se tako što se npr. tasteru *Prijava 2*, doda atribut *formnovalidate*. Na taj način, klikom na bilo koji drugi tester u formi, sem *Prijava*

2, podaci se prvo proveravaju (u skladu sa napisanim skript jezikom ili HTML 5 mogućnostima), pa tek ako su regularni, šalju serveru. Inače, „zaobilazjenje“ provere je moguće uraditi na nivou celog formulara pomoću *formnovalidate* ali i na nivou samo određenog elementa forme primenom atributa *novalidate*.

→Istovremeno, tasteru *Prijava*, dodelićemo atribut *target*, koji može imati vrednosti: *_blank*, *_self*, *_parent* i *_top*. Ovim atributom definiše se mesto gde se nakon klika na određeni taster dalje prikazuje sadržaj koji je definisan za prikaz. Podrazumevano je to u istom prozoru browser-a, tj. aktivna je vrednost *_self*, dok se sada može staviti da to bude u npr. novom prozoru, pomoću vrednosti *_blank*.

```
<form action="1.php" method="get">
  Telefon: <input type="text" name="tel"/>
  <button type="submit" formtarget="_blank">Prijava</button>
  <button type="submit" formaction="2.php" formnovalidate > Prijava 2
</button>
</form>
```

Atributi *min*, *max* i *step*

Ovi atributi definišu opseg dozvoljenih vrednosti prilikom unosa sadržaja od strane korisnika. Primenili smo ih ranije kod elemenata forme tipa *range* ili *number*. Pored toga, mogu se primeniti i kod tipa *date*:

```
<input type="date" min="2014-11-01" max="2019-11-30" />
```

Pored atributa *min-max*, uveden je i atribut *step* kojim se definiše „korak promene“ neke vrednosti, klikom na taster, koja se računa automatski. Najčešće se primenjuje kod tipa *number* ili *time*.

Atribut *multiple*

Ovaj atribut omogućava da se u postupku submit-ovanja, iz jednog elementa forme, prosledi više parametara. Koristi se za tip *file* i *email*, gde se razdvajanje, pojedinačnih fajlova, vrši zarezom.

```
<input type="file" multiple="multiple" />
```

Atribut *pattern*

Ovaj atribut omogućava da programer sam napiše svoj regularni izraz (pravilo po kome će se vršiti validacija nekog unetog sadržaja), direktno u HTML kodu. Podrazumevano, mogu da se koriste već definisana pravila koja su automatski dodeljena novim tipovima *input* taga (*email*, *number*, *time*, ...). Ako se na primer

želi ograničiti unos samo na četiri cifre, pri čemu prva ne može biti 0, može se definisati sledeći kod:

```
<input pattern="[1-9][0-9]{3}" name="cifre" />
```

Detaljnija pravila za pisanje *pattern*-a i regularnih izraza obrađuju se u kursu Web programiranje.

Atribut *placeholder*

Ovaj atribut koristi se za definisanje nekog kraćeg teksta koji će se prikazati korisniku u nekom elementu forme (za duži tekst koristiti atribut *title*). Ovaj tekst je obično bleđe boje od inicijalne i nestaje kada korisnik mišem klikne u to polje da unese svoj sadržaj.

```
<input type="text" name="ime" placeholder="Unesite ime">
```

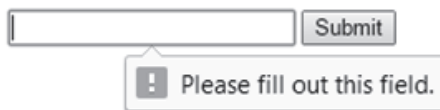


Slika 4.22. Primer primene atributa *placeholder*

Atribut *required*

Ukoliko se ovaj atribut definiše u nekom elementu formulara, onda je označeno da je obavezan unos sadržaja u taj element. Ukoliko korisnik ipak ne unese sadržaj, a pokuša da submit-uje formu, doći će do prijavljivanja greške.

```
<form>
  <input type="text" name="ime" required />
  <input type="submit" />
</form>
```



Slika 4.23. Primer primene atributa *required*

Atribut *readonly*

Ovaj atribut je vrlo sličan atributu *disabled*. Na ovaj način korisniku se onemogućava da edituje neki element forme. U najvećem broju browser-a se ovo polje prikazuje kao i regularno editabilno polje, ali klikom na njega ne može se uneti sadržaj. Sa druge strane, atributom *disabled*, polje se automatski oboji sivom bojom i vizuelno se naglašava da je reč o nekom izuzetku.

```
<form>
  <input type="text" id="ime1" />
  <input type="text" id="ime2" readonly />
  <input type="text" id="ime3" disabled />
  <input type="submit" />
</form>
```

Slika 4.24. Primer primene atributa `readonly` i `disabled`

4.10. Rad sa audio i video fajlovima

U HTML 4 mogućnost reprodukcije multimedije bila je većinski bazirana na plugin-ovima, što može da predstavlja problem ukoliko kod korisnika u browser-u to nije instalirano. Generalno, podrška u HTML 4 za korisničko upravljanje multimedijalnim sadržajem skoro da nije ni postojala u obliku kako to korisnik očekuje. Druga mogućnost bila je da se u web stranicu „uključi“ (*embed*) neki objekat koji je dobijen sa drugog sajta (npr. *youtube.com*) pa samim tim da se dobije i odgovarajuće grafičko okruženje kojim će se tim multimedijalnim sadržajem upravljati, obzirom da se upravljačke kontrole nisu obezbeđivale HTML kodom.

Dodatni problem je sa video formatima, kada se kontrola video fajla mogla raditi samo u slučaju kada postoji konkretno okruženje ili format čiji *player* to omogućava. Iz tog razloga najčešći formati su bili Flash ili Quicktime.

Formati koje podržava HTML 5 su:

- ✓ Audio (.ogg, .mp3, .wav)
- ✓ Video (.mpg4, .ogg, .webm)

→U HTML 5 uveden je novi tag *audio*, za potrebe reprodukcije audio fajlova. Osnovni atribut je *src*, da bi se definisala putanja od željenog fajla zaključno sa imenom i ekstenzijom istog, i atribut *controls*, kojim se audio kontrole za korisnika prikazuju vidljivim (*play/pause*). Pored ovih atributa, na raspolaganju su i *autoplay* (kojim se audio sekvenca automatski reprodukuje po učitavanju), *loop* (kojim se audio sekvenca automatski ponovo reprodukuje nakon prethodnog završetka) i *preload* (definiše da li će se i kako audio sekvenca učitati nakon prikaza web strane).

```
<audio src="pesma1.mp3" controls="controls">
  Ova verzija browser-a ne podržava audio tag!
</audio>
```



Slika 4.25. Primer primene taga *audio*

→ Za reprodukciju video fajlova uveden je novi HTML 5 tag koji se zove *video*. Ovaj tag omogućava prikaz i kontrolu video fajla. U skladu sa željenim dimenzijama video fajla, definišu se atributi *width* i *height*, kao i opisani atribut *controls*. Kod taga *video*, koristi se tag *source*, kojim se definiše putanja do video sekvence, pomoću atributa *src*.

```
<video width="320px" height="240px" controls="controls">
  <source src="movie.mp4" type="video/mp4" />
  Ova verzija browser-a ne podržava audio tag!
</video>
```



Slika 4.26. Primer primene taga *video*

Na sličan način kao i kod taga *audio*, na raspolaganju su i atributi: *autoplay*, *loop* i *preload*. Specifičnost taga *video* su i dva dodatna atributa: *muted* (definiše se da je audio izlaz video signala isključen) i *poster* (daje mogućnost postavljanja slike, dok se video fajl preuzima ili dok korisnik ne klikne na taster *play*).

4.11. Rad sa Canvas-om

Jedan od novih API-ja, u HTML 5, nazvan *Canvas* (platno), omogućava da se direktno u sajtu vrši crtanje, prikaz i animacija različitih objekata i oblika, primenom Java skript-a. Canvas omogućava da se u dvodimenzionom prostoru vrši renderovanje nekih objekata i slika u skladu sa Java Script metodama koji su na raspolaganju za te potrebe.

Iako postoji veliki broj ovih metoda oni se u ovoj knjizi neće obrađivati jer to pripada materiji kursa Web programiranje. Neke od njih mogu se pogledati na sajtu:

http://www.w3schools.com/tags/ref_canvas.asp.

Primer prikaza nepravilnog oblika sa ispunom boje dat je u listingu:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
  body {
    margin: 0px;
    padding: 0px;
  }
</style>
</head>
<body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
  <script>
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');

    // begin custom shape
    context.beginPath();
    context.moveTo(170, 80);
    context.bezierCurveTo(130, 100, 130, 150, 230, 150);
    context.bezierCurveTo(250, 180, 320, 180, 340, 150);
    context.bezierCurveTo(420, 150, 420, 120, 390, 100);
    context.bezierCurveTo(430, 40, 370, 30, 340, 50);
    context.bezierCurveTo(320, 5, 250, 20, 250, 50);
    context.bezierCurveTo(200, 5, 150, 20, 170, 80);

    // complete custom shape
    context.closePath();
    context.lineWidth = 5;
    context.fillStyle = '#8ED6FF';
    context.fill();
    context.strokeStyle = 'blue';
    context.stroke();
  </script>
</body>
</html>
```



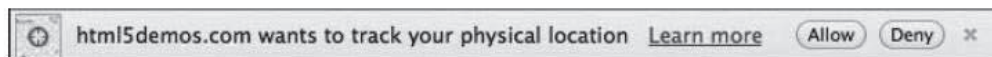
Slika 4.27. Primer primene taga canvas.

<http://www.html5canvastutorials.com/tutorials/html5-canvas-shape-fill/>

4.12. Rad sa geolokacijom

Geolocation omogućava posetiocima sajta, da uz njihovu saglasnost, pratite njihovu lokaciju. Praćenje može da se realizuje preko:

- ✓ IP adrese
- ✓ Konekcije na wireless mrežu
- ✓ Primopredajnika mobilne mreže
- ✓ GPS podrške na uređaju



Slika 4.28. Primer traženja dozvole za praćenje lokacije

Geolocation API može da se koristi primenom tri metoda:

- ✓ *getCurrentPosition()*
- ✓ *watchPosition()*
- ✓ *clearPosition()*

Princip rada:

1. Proveriti da li je *Geolocation* podržan od strane browser-a
2. Ako je podržan, potrebno je pozvati metod *getCurrentPosition()*
3. Ako je pozivanje metoda *getCurrentPosition()* bilo uspešno, kao rezultat će se dobiti koordinate korisnika
4. Nakon toga, metodom *showPosition()* moguće ih je prikazati npr. *Latitude* i *Longitude*

```
<script>
var x=document.getElementById("demo");

function getLocation(){
if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition(showPosition);
}
else{
    x.innerHTML="Geolocation nije podrzan u ovom browser-u";
}}
function showPosition (){
    x.innerHTML="Geografska širina je:" + position.coords.latitude + " a
                geografska dužina je:" + position.coords.longitude;
}
</script>
```

Metod `getCurrentPosition()` vraća objekat, u kome se uvek nalaze geografska dužina i širina, ali je pored njih moguće dobiti i sledeća svojstva:

Svojstvo	Opis
<code>coords.latitude</code>	Geografska širina kao decimalni broj
<code>coords.longitude</code>	Geografska dužina kao decimalni broj
<code>coords.accuracy</code>	Preciznost pozicije
<code>coords.altitude</code>	Nadmorska visina
<code>coords.altitudeAccuracy</code>	Preciznost nadmorske visine
<code>coords.heading</code>	Azimut
<code>coords.speed</code>	Brzina u m/s
<code>timestamp</code>	Datum i vreme generisanja odgovora

4.13. Web storage API

Web Storage API je standard za lokalno snimanje podataka, koji je takođe velika promena koju donosi HTML 5 u odnosu na ranije verzije. U verziji HTML 5 postoje dve vrste skladištenja podataka:

- ✓ *session Storage*
- ✓ *Local Storage*

Snimanje u sesijama je snimanje podataka na strani servera, dok se lokalno skladištenje realizuje na strani korisničkog računara. Do sada je skladištenje na strani klijenta bilo u formi kolačića (tekstualni fajl veličine do 4kB). Problem sa kolačićima je što se šalju serveru sa svakim *HTTP request*-om, dok se kod *Local Storage* podaci snimaju i stoje na klijentskom računaru sve dok ih neko ne zatraži. Druga bitna razlika, je što je kod *Local Storage* omogućeno skladištenje u fajlu veličine 5MB.

Ukoliko se koristi *web storage*, podaci se snimaju u formi *key/value*. Tako *key* najčešće ukazuje na neku promenljivu a *value* sadržaj koji joj se dodeljuje i koji se želi zapamtiti (npr. key: *email*, value: *pera@mika.com*). Skladištenje *key/value* para (bez obzira da li je to *local* ili *session storage*) se realizuje pomoću metoda *setItem()* dok je dohvaćanje *value*, na bazi poznatog *key*, pomoću *getItem()*. Na primer, ako se snima vrednost *10*, za ključ "*primer*" sintaksa kod je

```
localStorage.setItem("primer", "10");
```

Dohvaćanje tako skladištene vrednosti realizuje se kao:

```
var size = localStorage.getItem("primer");
```

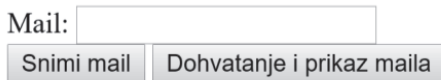
Moguće je koristiti i skraćenu sintaksu

```
var size = localStorage["size"];
```

U HTML 5 uvedena je mogućnost da se pomoću JavaScript-a komunicira sa bazom podataka, što je sledeći oblik komunikacije sa podacima koji do sada nije postojao u takvom obliku. Ova mogućnost se ranije vezivala samo za serverske jezike, a od sada to može da se uradi i preko klijentskog jezika. Primer komunikacije Local Storage-a sa JavaScript-om je dat u primeru snimanja maila korisnika.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Local Storage i Java Script</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/
  jquery.min.js"></script>
<script type="text/javascript">
if(localStorage){
  $(document).ready(function(){
    $(".snimi").click(function(){
      var mail = $("#mail").val();
      localStorage.setItem("mail-korisnika", mail);
      alert("Uneti mail je snimljen na Local Storage"); });
    $(".dohvati").click(function(){
      alert("Vas mail je: " + localStorage.getItem("mail-korisnika")); });
  });
} else{ alert("GRESKA! Browser ne podrzava rad sa Local Storage-ima!"); }
</script>
</head>
<body> <form>
  <label>Mail: <input type="text" id="mail"></label><br/>
  <button type="button" class="snimi">Snimi mail</button>
  <button type="button" class="dohvati">Dohvatanje i prikaz
maila</button>
</form> </body>
</html>
```

Sa datim kodom korisniku se prikazuje sadržaj browser-a kao na slici 5.29.



The image shows a web interface with a label 'Mail:' followed by a text input field. Below the input field are two buttons: 'Snimi mail' and 'Dohvatanje i prikaz maila'.

Slika 4.29. Primer upotrebe Local Storage-a

Korisnik može popuniti sadržaj u tekstualnom polju, i klikom na taster Snimi mail upisati uneti sadržaj u *Local Storage*. Na taj način, čak i nakon zatvaranja browser-a, i njegovog ponovnog učitavanja, klikom na taster se može dohvatiti snimljeni sadržaj i u ovom primeru ispisati korisniku, kao potvrda da je dohvaćeni sadržaj onaj koji je ranije upisan.

V CSS 3

Cascading Style Sheets ver. 3

CSS 3 je trenutno najnovija verzija CSS-a. Počeci CSS 3 se vezuju za 1998. godinu, iako se preporuke, i sam razvoj, dešavaju i dan danas. Da bi se bolje ukazalo na razlike i poboljšanja u odnosu na CSS 2, u CSS 3 specifikaciji su formirana posebna dokumenta, nazvana moduli. Svaki modul je logička celina koja sadrži opise, skup svojstava i pravila za određenu ciljnu grupu elemenata. Kako se moduli konstantno proširuju i dodaju novi, pretpostavlja se da će se od određenog trenutka samo proglasiti da je počela nova era CSS-a, tj. CSS 4, obzirom na veliki obim promena, mada se za sada o tome samo „priča“.

Postoji veliki broj modula koji su zvanično objavljeni i još više njih koji su „kandidati“. Samo do sredine 2012. godine *CSS Working Group* je objavio preko pedeset modula. Neki od popularnijih modula su: *Box Model*, *Selectors*, *Backgrounds and Borders*, *Text Effects*, *2D/3D Transformations*, *Animations*, *Multiple Column Layout*, *User Interface*, *Speech module*, *Hyperlink Presentation*, *Image Values and Replaced Content*.

CSS 3 predstavlja skup ranijih i dodatnih svojstava, kao i selektora, pomoću kojih se može vršiti stilizovanje tj. dizajn i organizacija prikaza sadržaja markap jezika. CSS 3 donosi proširenje skupa postojećih svojstava ali uvodi i potpuno nove

selektore tj. pseudo klase. Celokupni skup promena je relativno veliki i vrlo atraktivan za korisnika, pa je CSS 3 vrlo brzo postao popularan i primenljiv u većini komercijalnih sajtova. CSS 3 „pati“ od istog problema kao i HTML 5, a to je što u ovom prelaznom periodu postoji dosta svojstva koje pojedini browser-i ne mogu da interpretiraju na pravi način. Sa druge strane, postoje svojstva koja su neki browser-i implementirali i pre nego što su ona postala sastavni deo zvaničnog standarda, jer su procenjena kao vrlo korisna. U oba slučaja, situacija da mogućnosti, tj. svojstva, koja su data na raspolaganje programerima i mogućnost da to korisnici vide, u browser-ima, nije usaglašena dovodi do svesnog izbegavanja upotrebe pojedinih svojstava čime se kvalitet sajtova ipak umanjuje.

U skladu sa modulima, i zastupljenošću pojedinih svojstava, u ovom poglavlju obradiće se pojedina svojstva koja se pojavljuju kao novine u CSS 3, i koja su podržana u većini značajnijih browser-a.

5.1. Svojstvo border

Tri najpopularnija svojstva dodeljena ivicama (*border*) su: *border-radius*, *box-shadow*, *border-image*.

Svojstvo *border-radius* omogućava da se određenom elementu, kome se može upravljati ivicama, uglovi prikazuju zaobljeni, a ne inicijalno pod pravim uglom. Ovo je trenutno vrlo aktuelno u komercijalnim sajtovima, pa se čak šta više „pravi uglovi“ izbegavaju.

```
div {
  border: 2px solid #a1a1a1;
  background: yellow;
  width: 500px;
  border-radius: 25px;
}
```

Ovo je neki tekst u divu sa zaobljenim ivicama!

Slika 5.1. Primer primene svojstva *border-radius*

Svakom uglu neke figure se može definisati posebno zaobljenje, ili svima isto, po pravilu skraćenog pisanja, koje je objašnjeno na primeru margina.

```
border-radius:10px 20px 30px 40px;
border-radius:10px 20px;
border-radius:20px;
```

Pre CSS 3, efekat zakrivljenih ivica je realizovan umetanjem slika na kojima su ivice nacrtane zakrivljene, što je u mnogome usložnjavalo količinu i složenost

koda. Sada je to jednostavno, primenom samo jedne linije koda. Zakrivljenja se često primenjuju i kod elementa forme.

Svojstvo *box-shadow* omogućava definisanje senke na pojedinim objektima. Senka je kao efekat vrlo atraktivna, a pojavljuje se od ranije i u većini softvera za obradu teksta, pa je sada to moguće primeniti i u prikazu sadržaja unutar web strane.

```
div {
  width: 500px;
  height: 200px;
  background-color: gray;
  box-shadow: 15px 15px 10px
  #ffcc00;
}
```



Slika 5.2. Primer primene svojstva *box-shadow*

Svojstvo *border-image* omogućava da se umesto linija koje su korišćene, kao ivice, koristi slika.

```
<!DOCTYPE html>
<html>
<head>
<style>
div { border: 20px solid ;
      width: 250px;  }
      #uokviriti { border-image: url(1.jpg) 65 65 round; }
      #rasireno { border-image: url(1.jpg) 45 45 stretch; }

</style>
</head>
<body>

  <div id="uokviriti">Ovo je tekst sa slikom !</div><br/>
  <div id="rasireno">Ovo je tekst sa slikom !</div>
  <br/>Slika<br/> 

</body>
</html>
```



Slika 5.3. Primer primene svojstva *border-image*

Pored do sada opisanih svojstava, CSS 3 omogućava da se svako svojstvo eksplicitno definiše za određeni browser. Ovo zahteva da svaki browser ima svoju oznaku, i da se ona navede ispred konkretnog svojstva. Sastavni deo oznake su crtice ispred i iza oznake. Spisak oznaka značajnijih browser-a je:

Prefiks/Browser	Mozilla	Chrome/Safari	Microsoft	Opera
<i>prefix</i>	-moz-	-webkit-	-ms-	-o-

Primer primene svojstva za konkretan browser je:

```
-ms-border-image: url(1.png) 30 30 round;
-webkit-border-image: url(1.png) 30 30 round;
-o-border-image: url(1.png) 30 30 round;
```

Ukoliko se prilikom pisanja CSS-a koristi specificiranje imena browser-a, gde ne postoji ograničenje u broju browser-a koji se mogu nabrojati, potrebno je nakon tog skupa definisati i standardno CSS svojstvo, koje će se primenjivati za sve druge browser-e (one koji nisu navedeni ili za one browser-e koji su navedeni ali za njihove starije verzije). Npr:

```
div{
  -ms-border-image: url(1.png) 30 30 round;
  -webkit-border-image: url(1.png) 30 30 round;
  -o-border-image: url(1.png) 30 30 round;
  border-image: url(1.png) 30 30 round;
}
```

5.2. Svojstvo background

CSS 3 omogućava nekoliko novih svojstava za upravljanje pozadinom web stranice. Neka od novih svojstava su: *background-origin* i *background-size*

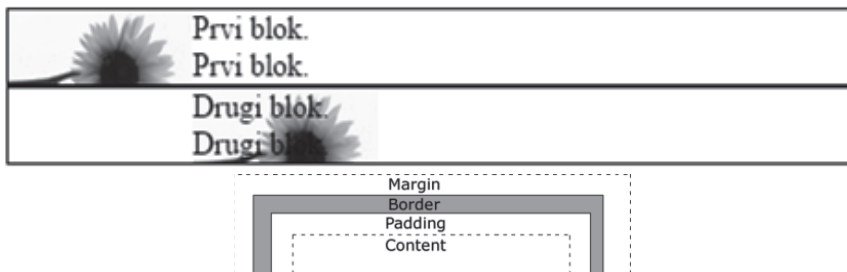
→Svojstvo *background-origin* omogućava definisanje pozadine koja će se dodeliti jednoj od tri karakteristične oblasti: delu za definisanje ivica, delu za *padding* i delu za sadržaj tj. *border-box*, *padding-box* i *content-box*. Na taj način, na primer, ista pozadinska slika učitana u određenom div-u, se ne prikazuje na istom mestu ako se tom divu definišu različite vrednosti svojstva *background-origin*.

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
div { border: 1px solid black;
      background-image: url('1.jpg');
      padding-left: 100px;
      background-repeat: no-repeat; }
#a{background-origin: border-box;}
#b{background-origin: content-box;}

</style>
</head>
<body>
    <div id="a">    Prvi blok.<br/>Prvi blok.    </div>
    <div id="b">    Drugi blok.<br/>Drugi blok.    </div>

</body>
</html>
```



Slika 5.4. Primena svojstva *background-origin* uz podsećanje o delovima box modela

Svojstvo *background-size* omogućava definisanje veličine pozadinske slike. Ranije se slika morala unapred fizički pripremiti u dimenzijama koje se žele, a sada se to može regulisati svojstvom. Vrednosti ovog svojstva mogu biti apsolutne i relativne veličine.

```
<!DOCTYPE html>
<html>
<head><style>
    div { background: url(1.jpg);
          background-size: 50px 33px;
          background-repeat: no-repeat; }
</style></head>

<body>
    <div id="a">
        Prvi blok.<br/>Prvi blok.
    </div>

    <br/>Originalna slika <br/>
    

</body>
</html>
```



Slika 5.5. Primer primene svojstva *background-size*

Pored do sada opisanih svojstava za pozadinu, u CSS 3 je moguće definisati i više od jedne pozadinske slike. Ovo se realizuje nabrojanjem URL adresa, tj. putanja do svake od slika, i odvajanjem zarezom.

```
<!DOCTYPE html>
<html>
<head>
<style>
div { background: url(1.jpg), url(2.jpg);
      background-repeat: no-repeat; }
</style>
</head>
<body>
<div id="a">
    Prvi blok.<br/><br/>
    Prvi blok.<br/><br/>
    Prvi blok.<br/><br/>
    Prvi blok.<br/>
</div>
</body>
</html>
```



Slika 5.6. Primer definisanja više pozadinskih slika istovremeno

Rad sa pozadinom, u CSS 3, doneo je i mogućnost da se kodom definiše gradijent boje. Ovo je do sada bilo moguće samo primenom nekih od alata za obradu slike. Problem je što se na ovaj način prenosi slika, koja je svakako „teža“ od teksta, i što kod razvlačenja slike dolazi do gubitka rezolucije. Kada se to radi kodom, onda browser generiše tu boju, i rezolucija se zadržava, a slika ne mora da se prenosi sa web servera. U CSS 3 su definisana dva tipa gradijenta:

1. Linearni gradijent (*linear-gradient (down/up/left/right/diagonally)*) i
2. Radijalni gradijent (*radial-gradient*).

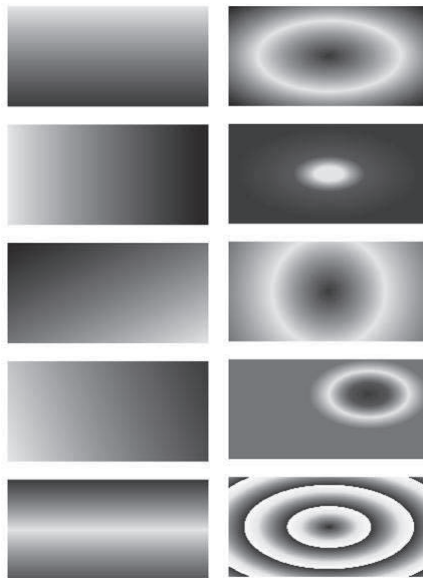
Primeri primene linearnog gradijenta:

```

<!DOCTYPE html>
<html>
<head>
<style>
div{height: 100px; width: 200px;}
  #a{background: linear-gradient(yellow, red);}
  #b{background: linear-gradient( to left, blue, yellow);}
  #c{background: linear-gradient( to bottom right, blue, yellow);}
  #d{background: linear-gradient(70deg, yellow, red);}
  #e{background: linear-gradient(red, yellow, green);}

</style>
</head>
<body>
  <div id="a"></div><br/> <div id="b"></div><br/>
  <div id="c"></div><br/> <div id="d"></div><br/>
  <div id="e"></div><br/>
</body>
</html>

```



Slika 5.7. Različite vrste primene a) linearnog i b) radijalnog gradijenta

Kao i kod primene softverskih alata za obradu slike, sada je i kod CSS 3 moguće definisati transparentnost kod kreiranja gradijenta. Tako na primer, definisanjem poslednjeg parametra u funkciji *rgba()*, između 0 i 1, pored tri parametra za RGB komponentu boje, dobija se efekat transparentnosti. Vrednost 0 ukazuje na potpuno transparentnost dok 1 na potpunu boju.

Dodatno, moguće je definisati „uzorak“ kompozicije boje, koji će se ponavljati onoliko puta koliko je definisana veličina određenog bloka. Ove se može realizovati pomoću *repeating-linear-gradient*.

```
<style>
div{height: 100px; width: 200px;}
#a1{background: linear-gradient(to right, rgba(125,50,100,0),
  rgba(250,100,200,1));}
#b1{ background: repeating-linear-gradient(green, yellow 15%, blue 25%);}

</style></head>
<body>
  <div id="a1"></div><br/>
  <div id="b1"></div><br/>
</body>
```



Slika 5.8. Primer primene vrednosti *repeating-linear-gradient*

Radijalni gradijent je takođe postao standardni efekat u softverskim alatima za obradu slike, a sada je moguće kreirati ga pomoću CSS-a. Ovaj gradijent svoj efekat realizuje počev od lokalnog centra, pa ka „u polje“. Pozicija centra, redosled boja, njihova zastupljenost u efektu i slično, može se modifikovati različitim vrednostima *radial-gradient*, kao u primeru, slika 5.7 b).

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div{height: 100px; width: 200px;}
    #a{background: radial-gradient(red, yellow, blue);}
    #b{background: radial-gradient(yellow 10%, green 25%, red 65%);}
    #c{background: radial-gradient(circle, red, yellow, gray);}
    #d{background: radial-gradient(closest-side at 70%
35%,red,green,yellow,gray);}
    #e{background: repeating-radial-gradient(red, yellow 20%, white
30%);}
  </style></head>

  <body>
    <div id="a"></div><br/> <div id="b"></div><br/>
    <div id="c"></div><br/> <div id="d"></div><br/>
    <div id="e"></div><br/>
  </body>
</html>
```

5.3. Svojstva za tekst

CSS 3 omogućava nekoliko novih svojstava za stilizovanje teksta. Na žalost, veći broj ovih svojstava još uvek nema podrške u trenutnim verzijama browser-a, dok neka tek polako dobijaju mogućnost da se primene. Ova svojstva treba da približe stilizovanje teksta mogućnostima koja su na raspolaganju u kvalitetnijim tekstualnim editorima i na koji su korisnici štampanih izdanja već navikli, a koja do sada nisu lako mogla da se postignu u web strani. Neka od novih svojstava su:

1. *hanging-punctuation*: omogućava postavljanje znaka interpunkcije van krajnje linije definisanog bloka u kome je sadržaj. Moguće vrednosti su *none|first|last|allow-end|force-end|initial|inherit*
2. *punctuation-trim*: omogućava primenu funkcije *trim* na interpunkcijske znake koji se nađu na početku ili kraju linije
3. *text-align-last*: omogućava definisanje tipa poravnanja za poslednju liniju sadržaja u nekom bloku,
4. *text-emphasis*: omogućava stilizovanje teksta,
5. *text-justify*: omogućava preciznije definisanja svojstava poravnanja kada je poravnanje za tekst već podešeno. Moguće vrednosti su: *auto|inter-word|inter-ideograph|inter-cluster|distribute| kashida| trim| initial| inherit*,
6. *text-outline*: omogućava definisanje kontura teksta
7. *text-overflow*: definiše način prikaza teksta ukoliko je dužina teksta veća od oblasti u kojoj je namenjen da se prikaže. Moguće vrednosti su: *clip|ellipsis|string|initial|inherit*.

```
<!DOCTYPE html>
<html>
<head>
<style>
p{   width: 150px;
      border: 1px solid;
      white-space: nowrap;
      overflow: hidden; }
#a {text-overflow: clip;}
#b {text-overflow: ellipsis;}
</style>
</head>
<body>
    <p id="a"> Ovde je neki tekst u tagu p i prikazuje se sa ivicom!</p>
    <p id="b"> Ovde je neki tekst u tagu p i prikazuje se sa ivicom!</p>
</body></html>
```

Ovde je neki tekst u tag

Ovde je neki tekst u ...

Slika 5.9. Primer primene svojstva *text-overflow*

8. *text-shadow*: omogućava kreiranje senke nad tekstem. Moguće vrednosti su: *h-shadow v-shadow blur-radius color|none|initial|inherit*.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p{ text-shadow: 3px 5px #28F40C;
      font-size:48px;}
  </style>
</head>
<body>
  <p>Ovo je neki tekst!</p>
</body>
</html>
```

Ovo je neki tekst!

Slika 5.10. Primer primene svojstva *text-shadow*

9. *text-wrap*: omogućava definisanje načina prelamanja teksta. Moguće vrednosti su: *normal|none|unrestricted|suppress*.

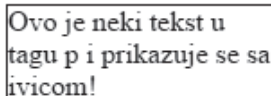
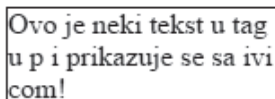
10. *word-break*: omogućava definisanje preloma reči. Moguće vrednosti su: *normal|break-all|keep-all|initial|inherit*.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p{ width: 150px;
      border: 1px solid;
    }
    #a {word-break: keep-all;}
    #b {word-break: break-all;}
  </style>
</head>
<body>
  <p id="a"> Ovo je neki tekst u tagu p i prikazuje se sa ivicom!</p>
```

```

<p id="b"> Ovo je neki tekst u tagu p i prikazuje se sa ivicom!</p>
</body>
</html>

```

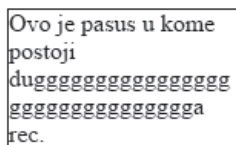
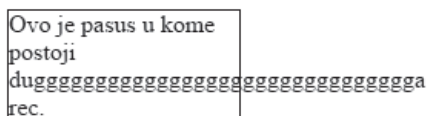
Slika 5.11. Primer primene svojstva `word-break`

11. `word-wrap`: omogućava prelom dugih reči u više linija teksta. Moguće vrednosti: `normal`|`break-word`|`initial`|`inherit`.

```

<!DOCTYPE html>
<html>
<head>
  <style>
    p{      width: 150px;
            border: 1px solid;}
    #prelom_reci {word-wrap: break-word;}
  </style>
</head>
<body>
  <p id="prelom_reci"> Ovo je pasus u kome postoji
dugggggggggggggggggggggggggggggggggggga rec.</p>
  <p > Ovo je pasus u kome postoji dugggggggggggggggggggggggggggggggggga rec.</p>
</body>
</html>

```

Slika 5.12. Primer primene svojstva `word-wrap`.

5.4. Svojstvo @font-face

Pre desetak godina, kada su fontovi za tekstove počeli da zauzimaju sve bitniju ulogu u stilizovanju teksta, na računarima korisnika se nalazio relativno mali skup fontova. Dolaskom novih operativnih sistema i softvera, koje korisnik instalira, dobijali su se i novi fontovi. U tom periodu, bilo je vrlo rizično koristiti bilo koji font, jer je bilo pitanje da li baš taj font korisnik ima na svom računaru. Tako je uveden termin „*web safe fonts*“ tj. fontovi koji će sigurno moći da se prikažu kod korisnika, onako kako je dizajner predvideo.

Pojavom CSS 3, ovaj složen problem je potpuno rešen, i sada je dozvoljeno da programer koristi bilo koji font. Ovo je omogućeno posredstvom svojstva *@font-face*, kojim se određeni font sada automatski preuzima sa web servera, kao i web strana, i ne očekuje se da od ranije postoji na računaru korisnika.

Programer može sam kreirati proizvoljan font ili koristiti neki ranije kreiran, i njega primeniti na stilizovanje teksta unutar web strane. Font može biti kreiran kao jedan od sledećih formata: *TrueType Fonts (TTF)*, *OpenType Fonts (OTF)*, *The Web Open Font Format (WOFF)*, *SVG Fonts/Shapes* i *Embedded OpenType Fonts (EOT)*.

Primena proizvoljnog fonta se realizuje u dva koraka: Prvo definisati ime fonta i putanju do fajla u kome je font, primenom *@font-face* (npr. *mojFont*), a zatim nekom bloku web strane definisati da mu to bude font (npr. *font-family:mojFont*).

```
@font-face{
    font-family: mojFont;
    src: url('fontovi/fontTest.ttf'), url('fontovi/fontTest.eot') }

p{    font-family:mojFont; }
```

5.5. Rad sa tranzicijama

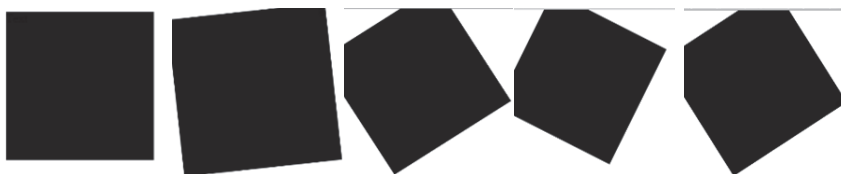
Tranzicije omogućavaju vremenski vidljivu promenu jednog CSS stila u drugi, primenom samo CSS koda. Na ovaj način se postiže efekat kretanja, promena, animacije i slično ali bez upotrebe Java Script-a, Action script-a ili Flash-a. Svojstva za rad sa tranzicijama su:

1. *transition*: objedinjavanje četiri pojedinačna svojstva za tranziciju, u jedinstvenu celinu
2. *transition-delay*: definiše za koliko će se sekundi efekat aktivirati
3. *transition-duration*: definiše posle koliko milisekundi će se efekat završiti
4. *transition-property*: definiše ime svojstva na koji se primenjuje efekat
5. *transition-timing-function*: definiše brzinu realizacije efekta.

Ovi efekti se najbolje primećuju ako se definiše događaj kojim se ove promene iniciraju. U primeru je izabrano da to bude *hover*, tj. trenutak prelaska miša preko određene oblasti.

```
<!DOCTYPE html>
<html>
<head>
<style>
div { width: 200px;
      height: 200px;
      background: blue;
      transition: width 3s, height 3s, transform 4s; }
div:hover { width: 500px;
            height: 500px;
            transform: rotate(200deg);}

</style>
</head>
<body>
    <div>Text</div>
</body>
</html>
```



Slika 5.13. Primer primene svojstva transition

5.6. Rad sa sadržajem u više kolona

Formatiranje sadržaja, a najčešće teksta, u više kolumni prikaz, je već postalo standard u štampanim medijima, i ima ugrađenu podršku u alatima za obradu teksta. Do sada se takva stvar u web stranicama realizovala kreiranjem više blokova, koji su pozicionirani jedan pored drugog, a tekst „ručno cepao“ tj. dodeljivao svakom od njih. CSS 3 nam omogućava da se jednim svojstvom, u jednoj liniji koda, sadržaj automatski podeli u više kolona. Širine kolona se automatski menjaju sa promenom veličine browsera, i sadržaj se dinamički ažurira. Ovo se postiže svojstvom *column-count*. Kao i kod drugih CSS 3 svojstava, poželjno je kod svih svojstava stavljati i prefikse za pojedine browser-e, jer se još uvek dešava da isto svojstvo bez prefiksa tj. oznake browser-a neće uvek da radi.

```
<!DOCTYPE html>
<html>
<head>
```

```

<style>
#a1 {-webkit-column-count: 2;}
</style>

</head>
<body>
<p id="a1">Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo
je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo
je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!
</p>
</body>
</html>

```

Ovo je neki tekst!Ovo je neki tekst!Ovo	tekst!Ovo je neki tekst!Ovo je neki
je neki tekst!Ovo je neki tekst!Ovo je	tekst!Ovo je neki tekst!Ovo je neki
neki tekst!Ovo je neki tekst!Ovo je neki	tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki	tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki	tekst!Ovo je neki tekst!Ovo je neki tekst!

Slika 5.14. Primer primene svojstva *column-count*

Pored kreiranja kolona, CSS 3 omogućava i dodatno stilizovanje. Svojstvo *column-gap* omogućava da se definiše širina prostora među kolonama. Na ovaj način se dodatno može urediti način prikaza teksta.

```

<!DOCTYPE html>
<html>
<head>

<style>
#a1 { -webkit-column-count: 4;
      column-gap: 20px;}
</style>

</head>
<body>
<p id="a1">
Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je
neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo
je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!
</p>
</body>
</html>

```

Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki
tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki
tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki
tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	tekst!
tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	
tekst!Ovo je neki	tekst!Ovo je neki	tekst!Ovo je neki	

Slika 5.15. Primer primene svojstva *column-gap*

Iako se sa dva opisana svojstva može u potpunosti postići efekat koji imamo u npr. novinama, CSS 3 omogućava i dodatno stilizovanje među kolonskog prostora. Naime, svojstvo *column-rule* omogućava da se za ovaj prostor definiše širina, stil i boja koja će se koristiti. Najčešća primena je postavljanje vertikalnih linija, koje imaju određenu boju, i koje vizuelno dodatno dele sadržaj na logičke celine.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    #a1 { -webkit-column-count: 4;
          -webkit-column-gap: 10px;
          -webkit-column-rule: 2px outset red;}
  </style>
</head>
<body>
<p id="a1">
Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je
neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo
je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki
tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!
</p>
</body>
</html>
```

Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!	neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!	tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!	je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!Ovo je neki tekst!
--	---	--	--

Slika 5.16. Primer primene svojstva *column-rule*

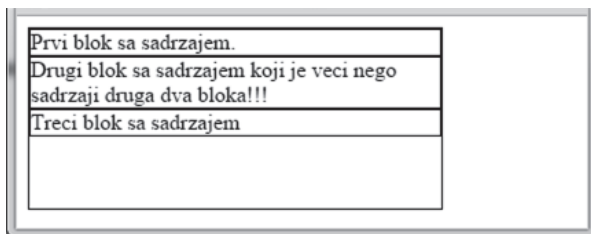
5.6.1. Svojstvo *flex*

Iako je CSS doneo velike novine i olakšanja kod rada sa više kolona, opisni način nije jedini koji korisniku prikazuje prelom teksta po vertikali. Naime, CSS 3 omogućava i primenu novog svojstva *flex* (*display: flex*). Ovo svojstvo može da definiše dužinu, širinu, prikaz i prelom nekog sadržaja, i to relativno u odnosu na ostale sadržaje unutar istog bloka ili regije.

Posmatrajmo listing koda kojim se kreiraju tri div elementa sa različitim sadržajem.

```
<html>
<head>
<style>
#sadrzaj {      width: 300px;
                 height: 130px;
                 border: 1px solid black; }
.kolone { border: 1px solid black; }
</style>
</head>
<body>
<div id="sadrzaj">
    <div class="kolone">Prvi blok sa sadrzajem.</div>
    <div class="kolone">Drugi blok sa sadrzajem koji je veci nego sadrzaji
druga dva bloka!!!</div>
    <div class="kolone">Treci blok sa sadrzajem</div>
</div>
</body>
</html>
```

Obzirom da za blokove nije definisan poseban *display*, oni će se podrazumevano prikazati kao na slici 5.17.



Slika 5.17. Prikaz sadržaja bez primene svojstva *flex*

Međutim, ako se div-u sa *id*=“sadrzaj” definiše *display: flex*; tj. deo CSS koda promeni sa:

```
#sadrzaj {
  width: 300px;
  height: 130px;
  border: 1px solid black;
  display: flex;
}
```

tri unutrašnja elementa sa korisničkim sadržajem će se složiti jedan pored drugog, kao na slici 6.24. Efekat svojstva *flex* se ovako automatski realizuje, kao da je on definisan sa vrednošću *auto*. Pored ove vrednosti moguće je koristiti i vrednosti 0 i 1.

Prvi blok sa sadržajem	Drugi blok sa sadržajem koji je veći nego sadržaji druga dva bloka!!!	Treći blok sa sadržajem
------------------------	---	-------------------------

Slika 5.18. Prikaz sadržaja sa primenom svojstva *flex* i vrednošću *auto*

Po pravilu svojstvo *flex* se definiše na mestu gde je korisnički sadržaj, što je u ovom slučaju div sa *class="kolone"*, tako da je prikazani listing poželjno modifikovati sa:

```
.kolone {
  border: 1px solid black;
  flex: auto; }
```

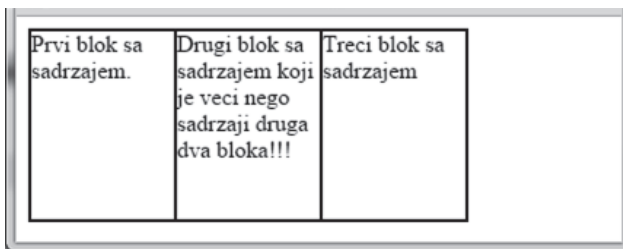
Vrednost *auto*, automatski podešava prikaz sadržaja unutar kolona, koji se završavaju na istoj visini browser-a. Ovo je nekada poželjno, ali često je ovakav prikaz posledica „veštačkog razvlačenja teksta“, pa se češće koristi vrednost 1. U tom slučaju, ako se kompletan stil u datom listingu zameni sa:

```
<style>
#sadrzaj {
  width: 300px;
  height: 130px;
  border: 1px solid black;
  display: flex; }
.kolone {
  border: 1px solid black;
  flex: 1; }
</style>
```

dobija se efekat kao na slici 6.25. gde je sadržaj prikazan u tri kolone, ali su dužine sadržaja u kolonama shodne unetom sadržaju a širine kolona automatski podešene na iste širine.

Posebnu pažnju treba posvetiti svojstvu **justify-content**, kojim se vrlo lako postiže centriranje sadržaja ili ravnomerno raspoređivanje većeg broja blokova duž linijskog prikaza, u smislu međusobnog rastojanja, što menja ručno definisanje margina i padding-a.

```
justify-content: center;
justify-content: space-between;
```

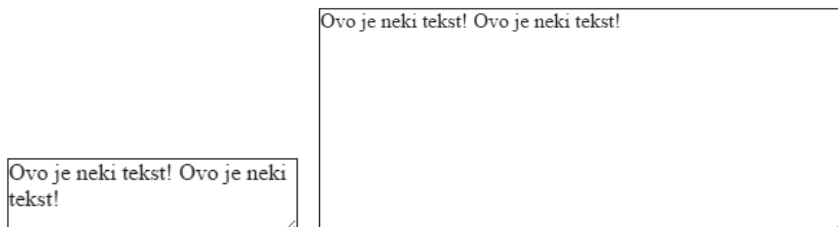


Slika 5.19. Prikaz sadržaja sa primenom svojstva *flex* i vrednošću 1

5.7. Svojstvo *resize*

Ovo svojstvo omogućava korisniku da određeni element stranice prilagodi svojim potrebama (u smislu dimenzija). Primer dat u listingu pokazuje primenu svojstva *resize* na jedan `div` element. Promena dimenzija se realizuje „razvlačenjem“ donjeg desnog ugla elementa.

```
<!DOCTYPE html>
<html>
<head><style>
#a3 {
  border: 1px solid;
  width: 200px;
  height: 50px;
  resize: both;
  overflow: auto;}
</style></head>
<body>
  <div id="a3">Ovo je neki tekst! Ovo je neki tekst!</div>
</body>
</html>
```



Slika 5.20. Primer primene svojstva *resize*

Pored opisanih svojstava postoje još neka nova svojstva i još više mogućih vrednosti istih, ali se oni neće obrađivati na ovom nivou. Dodatno, svojstva koja još uvek nisu podržana ni u jednom od poznatijih browser-a su namerno minimizirana.

Iako Vam se može učiniti da je ovo „nepovoljan“ period jer puno stvari još uvek nije podržano u browser-ima, možda to treba posmatrati sa druge strane: Vi ste direktni učesnici razvoja i usvajanja novih jezika i veština, i možete da sagledate problem u tranziciji iz jedne u drugu tehnologiju i razumeti u kom smeru se dalje može očekivati razvoj Web-a. Isto tako, svojstva ili pojedine primene, koje budu komplikovane ili problematične se često modifikuju ili izbacuju, što možete direktno da pratite i aktivno učestvujete u tome.

5.8. Primeri primene HTML 5 i CSS 3 koda

Primer 1: Primer implementacije svojstva *border-radius* u realnim primenama

Listing index.html

```
<body>
    
    
    
</body>
```

Listing style.css

```
*{    margin: 5px;
      padding: 5px;  }
.slika1{
    background-color: #eb4800;
    border-radius: 50%;
    padding: 25px;}
.slika2{
    background-color: #eb4800;
    border-radius: 30px 10px;
    padding: 25px; }
.slika3{
    background-color: #eb4800;
    border-radius: 10px 30px 0px;
    padding: 25px; }
```



Slika 5.21. Primer primene svojstva *border-radius* u različitim oblicima

Primer 2: Primer upotrebe linearnog gradijenta*Listing index.html*

```
<body>
  <div id="prvi">
  </div>
  <div id="drugi">
  </div>

</body>
```

Listing style.css

```
div{ width:300px;
      height:200px;
      float:left; }
#prvi{ background: linear-gradient(blue, yellow); }
#drugi{ background: linear-gradient(to right, #00ff00 , #f2f2f2); }
```

**Slika 5.22.** Primer primene svojstva `background` i vrednosti `linear-gradient`**Primer 3:** Primer realizacije neaktivnog tastera*Listing index.html*

```
<!DOCTYPE html>
<html>
<head>
<style>
  .taster { background-color: #929292;
            color: white;
            padding: 15px 32px;
            text-align: center;
            font-size: 18px;
            margin: 5px 3px; }
  .neaktivno { cursor: not-allowed; }

</style>
</head>
<body>
  <button class="taster neaktivno">NEAKTIVNO</button>

</body>
</html>
```

**Slika 5.23.** Primer realizacije neaktivnog tastera

VI UVOD U JAVASCRIPT

Šta je, čemu služi i kako se koristi?

POČETAK učenja JavaScript-a, uvek nameće pitanje šta prvo treba reći i šta je od čega bitnije. Ja ću se opredeliti za ime i pisanje imena ovog jezika, koji će biti tema većeg dela ove knjige, pa tek onda definisanju pojma i namene samog jezika.

JavaScript je, pod ovim imenom, nastao u decembru 1995. godine. Pre toga, njegovo ime je bilo *LiveScript*, ali su njegovi osnivači Netscape i Sun Microsystems odlučili da od tog trenutka krene sa novim imenom, iako je nastao relativno kratko pre toga. Različiti izvori ukazuju da je ovo ime izabrano iz čisto marketinških razloga, i da je trebalo da da „vetar u leđa“ novom jeziku na bazi popularnog i vrlo korišćenog jezika Java. Ovaj jezik razvila je kompanija Netscape Communications, Inc. koja je bila jedan od lidera u domenu web-a u to vreme. Jezik je razvijen paralelno sa novom verzijom Netscape-ovog web servera i trebalo je da omogući programerima lakše upravljanje i povezivanje stranica sa drugim servisima i serverskim jezicima koji komuniciraju sa bazama podataka. Paralelno sa ovom namenom, novi jezik je trebalo da omogući lakše i brže upravljanje klijentskim delom web stranice, u cilju provera informacija koje

korisnici unose, npr. *username* i *password*, kako u smislu da li su unete, tako i u smislu provere da li je format unetog sadržaja dobar, i da se to ne šalje web serveru pre nego se sve provere ne završe na klijentskoj strani.

Novo ime *JavaScript* je zbunilo mnoge programere, jer su mislili da ima direktne veze sa Javom tj. korišćenim Java apletima, ali i web dizajnere koji su strahovali od složenog i striktnog jezika Java. I dan danas postoje programeri koji misle da su ova dva termina povezana, ali je njihova veza jedino u sintaksi koji su zajedno preuzeli iz baznog jezika C, na kome su napisani. U tom smislu JavaScript i programski jezik Java, imaju neuporedivo više razlika nego sličnosti, i ne treba ih mešati.

Oznaka *Script* treba da vrlo korektno ukaže da se radi o tzv. skript jeziku. Skript jezici po pravilu ukazuju na „niži nivo programiranja“ u odnosu na klasične programske jezike. Ovi jezici su namenski manje kompleksni, sa manje strogom i prostijom sintaksom, lakši za razumevanje, najčešće platformski nezavisni i lako se mogu integrisati u „više programske jezike“. Skript jezici se češće interpretiraju nego što se kompajliraju, ne traže striktno posebna razvojna okruženja i najčešće služe da odrade neki konkretan, često manji, obim posla. U tom smislu vrlo su atraktivni i pogodni za učenje i ne zahtevaju inicijalne veštine i znanja koje korisnik mora da ima, u odnosu na klasične programske jezike.

Tako je i *JavaScript* skript jezik i kao takav se piše spojeno sa rečju *Script*, iako se i u našem govornom jeziku Java tumači kao vlastita imenica a *script* jezik više kao pridev koji opisuje imenicu, pa se ove reči pišu odvojeno. Međutim, ime jezika je definisano kao jedna reč tj. **JavaScript**.

6.1. O JavaScript-u

JavaScript se definiše na puno načina ali najčešće se kaže da je najpopularniji skript jezik za web stranice koji je objektno orijentisani programski jezik, koji zahteva vrlo malo memorije na korisničkom računaru i nema složenu sintaksu jezika (pa ga neki autori nazivaju „lakim jezikom“). Pored ovoga, JavaScript može da radi na velikom broju najrazličitijih platformi i sistema, što ga dodatno čini vrlo rasprostranjenim. Iako je JavaScript inicijalno povezan sa web stranicama, on potpuno uspešno može da radi i u okruženjima koja nemaju veze sa web-om. JavaScript inicijalno podržava tzv. funkcionalno programiranje ali je već duži niz godina i objektno orijentisan i podržava rad sa objektima. Stručno govoreći, može se reći da je JavaScript *prototype-based*, *multi-paradigm* i *dynamic language*.

Do pre nekoliko godina, bazna podela web orijentisanih programskih jezika na klijentske i serverske programske jezike je bila vrlo pogodna i za definisanje JavaScript-a. U tom periodu, JavaScript je bio klasičan skript jezik koji je bio sastavni deo klijentskih programskih jezika, i stvar je bila potpuno jasna.

Međutim, poslednjih desetak godina, JavaScript je doživeo možda najveći obim promena od svih drugih jezika, i iz jednog relativno skromnog skript jezika počeo da se razvija u vrlo ozbiljan i respektabilan, ne samo jezik ili okruženje, nego tehnologiju. Za vrlo kratko vreme pojavilo se puno JavaScript biblioteka koje su inicijalno bile zamišljene da korisnicima još više olakšaju upotrebu ovog jezika, time što će se sve klasične grupe linija koda, koji rešava neki često korišćeni slučaj, grupisati u ime jedne funkcije ili metoda, i samo pozvati po tom imenu, i dobiti efekat bez mnogo muke. Tako je prvo nastao *jQuery*, kada se videlo da se kod animacija, efekata, tranzicija, slider-a, galerija i sl. prirodno može grupisati u posebnu biblioteku. Ova biblioteka je za vrlo kratko vreme postala toliko popularna da danas skoro da ne postoji sajt koji je ne primenjuje na više mesta. Mlađi programeri, danas, čak i preskaču klasičnu primenu JavaScript-a jer su oduševljeni pojednostavljivanjem i lakim načinom manipulacija *jQuery*-jem.

Paralelno sa razvojem WEB 2.0, pojavljuje se i AJAX, koji je baziran na JavaScript-u a koji je danas konceptualno zastupljen u svakom boljem sajtu. U istom periodu, pojavljuje se i JSON (*JavaScript Object Notation*), koji se za razliku od ranije korišćenog XML-a lakše i brže uči i koji je intuitivan. Pokazuje se da JavaScript vrlo lako komunicira sa ovim tipom podataka, tako da se danas najveći broj podataka između servisa i klijenta razmenjuje baš primenom JSON-a i parsiranjem JavaScript-om. Pojavom HTML 5 i CSS 3, veliki broj API-ja koji su doneli revoluciju u web-u je baziran na JavaScript-u, pa se formalno kaže da su HTML 5 i CSS 3 spoj ranijih verzija HTML-a i CSS-a sa JavaScript-om. Sve ovo i dalje je bilo usklađeno sa definicijom da je JavaScript klijentski jezik.

Klijentski jezik je onaj koji se u originalu dobija od web servera i tek na klijentskoj strani, žargonski kažemo kod klijenta, u browser-u interpretira i realizuje tako da njegov efekat vidi korisnik. Iz tog razloga, kao što je moguće videti originalni HTML i CSS kod, koji se dobije od web servera, moguće je videti i originalni JavaScript kod koji je napisao programer, a koji će se, u zavisnosti od potreba ili trenutka, interpretirati i prikazati efekat svoga delovanja u browser-u.

Kako je ovaj napredak išao vrlo brzo, svi su shvatili da je JavaScript postao „ozbiljan igrač“ ali je i dalje bio „nižeg ranga“ jer je klijentski jezik. Međutim, tada dolazi do vrlo velike promene i pojavljuje se *Node.js*, koji je serverska verzija JavaScript-a, sa svim karakteristikama koje klasični serverski jezici imaju u tom trenutku. Ove aktivnosti se intenzivno dešavaju krajem 2009. i početkom 2010. godine.

Iako smo svi mislili da je tog trenutka dostignut maksimum, i da smo dobili jezik koji je postao i klijentski i serverski, i to sa zastupljenošću koju drugi klijentski jezici ni približno nemaju, JavaScript je doživeo novi talas inovacija i „procvata“.

Naime, u vrlo kratkom roku počinju da se pojavljuju nove JavaScript biblioteke, ali ono što je privuklo mnogo više pažnje je pojava JavaScript okruženja, tj. framework-a. Pojavljuje se prvo framework *Backbone.js* koji je doneo novu

paradigmu u dizajniranju koda aplikacije u strukturi *model–view–presenter* (MVP). Nedugo zatim, kada je ovaj framework već počeo da zauzima bitnu ulogu i ima veliki broj programera koji ga koriste, pojavljuje se potpuno novi framework *Angular.js*, koji donosi još kvalitetniju organizaciju u formi *model–view–controller* (MVC) i *model–view–viewmodel* (MVVM) arhitektura. Angular izuzetno brzo zauzima vrlo bitnu ulogu i postaje tražen u svim većim kompanijama i softverskim rešenjima, i dobija potpuno nove verzije. Počevši od verzije 1, verzija 2 je donela toliko novina da pojedini autori smatraju da je to potpuno novi framework, pre nego nova verzija starog. Sada, Angular već ima verziju 4 i uveliko se priča o verziji 5. Čak i tada kada smo smatrali da je to stiglo do neke vrste maksimuma za jedan jezik, pojavljuje se potpuno novi framework JavaScript-a pod imenom *React.js*. U ovom trenutku React, po procenama, prevazilazi sve druge konkurente, što znači da je svaki novi framework JavaScript-a funkcionalno bolji, a ne samo marketinški proizvod koji je noviji po datumu proizvodnje.

Taman kada se pomisli da je ovoj priči kraj, pojavljuje se potpuno novi pravac, koji takođe osvaja JavaScript, a to je *WebGL*, kojim se ovaj jezik nameće kao vrlo jaka konkurencija klasičnim desktop alatima i programskim jezicima koji se bave grafikom, igricama i interaktivnim multimedijalnim sadržajima.

U ovom trenutku postoji izuzetno veliki broj biblioteka i ogromna količina koda koja je dostupna i napisana u JavaScript-u, za najrazličitije namene, tako da kada danas govorimo o JavaScript-u, skoro da niko ne počne sa opisom to je skript jezik, obzirom da je postao lider u klijentskom programiranju, serverskom kodu, razvojnom okruženju i da je nemoguće predvideti šta će sve od novina da se desi u narednih godinu dana.

Sada je podela na klijentske i serverske programske jezike postala vrlo „tesna“ kada se priča o JavaScript-u, obzirom šta su sve postali derivati ovog jezika. Žargonski, kada kažemo JavaScript mi mislimo na klijentski domen, dok kada kažemo Node.js mislimo na serverski. Slično ovome, termini Angular i React ukazuju na aktuelne framework-e.

Prema podacima sa sajta <http://w3techs.com/>, zaključno sa junom 2017. godine, zastupljenost klijentskih jezika u sajtovima je prikazana na slici 6.1. Iako ima puno podataka na Internetu koji definišu ove parametre, i koji se međusobno razlikuju, odnos JavaScript-a i drugih jezika je toliko veliki da nema potrebe da se posebno komentariše.

1. JavaScript	94.6%
2. Flash	6.3%
3. Silverlight	0.1%

Slika 6.1. Zastupljenost klijentskih jezika u sajtovima. (izvor <http://w3techs.com/>)

Već je napomenuto da postoji veliki broj različitih JavaScript biblioteka koje imaju vrlo veliki obim primena. Prema podacima sa sajta <http://w3techs.com/>,

zaključno sa junom 2017. godine, prvih pet najkorišćenijih biblioteka su prikazane na slici 6.2.

1. jQuery	72.6%
2. Bootstrap	15.5%
3. Modernizr	10.8%
4. MooTools	2.9%
5. ASP.NET Ajax	2.1%

Slika 6.2. Zastupljenost najpopularnijih JavaScript biblioteka. (izvor <http://w3techs.com/>)

Pored ovih najpopularnijih postoji puno drugih koje se vrlo često koriste u najširem spektru web sajtova. Ovom prilikom pobrojaće se samo neke biblioteke i okruženja koje treba zapamtiti obzirom na njihov obim primena:

- ✓ AngularJS
- ✓ Backbone.js
- ✓ Google Web Toolkit
- ✓ jQuery
- ✓ Knockout.js
- ✓ Modernizr
- ✓ Mojito
- ✓ Node.js
- ✓ React.js
- ✓ Socket.IO
- ✓ Underscore.js
- ✓ Vue.js

Bez obzira šta se od svih navedenih jezika, okruženja ili primena želi postići, u korenu svih njih je bazni jezik JavaScript koji se mora dobro razumeti i naučiti, pa je učenje svakog višeg nivoa ovog jezika tada mnogo lakše.

6.2. O Front-End-u i Back-End-u

Na tržištu rada podela na klijentske i serverske programske jezike skoro da ne postoji jer poslodavce zanima najčešće paket jezika i tehnologija koje rešavaju neki posao, a ne pojedinačni jezici i njihova interne podele. U tom smislu, uobičajena je podela poslova na *front-end* i *back-end* programere tj. poslove.

Front-End programer je neko ko poznaje ceo set jezika i tehnologija namenjenih za programiranje klijentskih aplikacija. S tim u vezi, najčešće se inicijalno podrazumeva poznavanje HTML-a, CSS-a, JavaScript-a, jQuery-ja, AJAX-a, XML-a i JSON-a. Sve navedeno zajedno se integriše u cilju kreiranja klijentske aplikacije.

Sa druge strane, serverske strane, Back-End programer je neko ko piše kod namenjen realizaciji na web serveru i koji podrazumeva veće i kompleksnije funkcionalnosti, rad sa sesijama, servisima, bazama podataka itd. Za ove poslove očekuje se poznavanje serverskih programskih jezika, od kojih su najpopularniji PHP, Node.js, ASP.NET, ... i rad sa bazama podataka tj. jezikom SQL.

Kako se sav serverski kod izvršava i prevodi na serveru tj. na serveru se u krajnjoj fazi prevodi u klijentske jezike, jer browser samo klijentske jezike može da „razume“ i interpretira, dobar back-end programer mora da poznaje i front-end kod. Time se poslovi front-end-a smatraju ulaznim poslovima kako za programere koji žele da ih dalje usavršavaju u klijentskom tako i za programere serverskih aplikacija.

U poslednjih nekoliko godina, serverske aplikacije se sve više pišu u formi web servisa. Na ovaj način back-end programer može da piše kod, ne samo u serverskim web orijentisanim jezicima, nego može da koristi i bilo koji drugi desktop programski jezik (npr. C#, Javu,...). Tada je odgovor servisa, onome ko ga sa klijentske strane „prozove“, u formi XML-a ili sada češće JSON-a, i tada back-end programer ne mora da poznaje ništa od front-end jezika. Ovo je postalo vrlo popularan oblik realizacije koda, jer se kod može bolje zaštititi, veća grupa ljudi u više jezika ga može pisati i nije ograničen na web tehnologije. Sa druge strane, ovo je povećalo očekivanja od front-end programera u smislu manipulacije dobijenim podacima iz XML-a tj. JSON-a i njihove interaktivne implementacije primenom jQuery-ja što je dodatno povećalo i važnost JavaScript-a i njegovih derivata.

Tako, iako je JavaScript inicijalno korišćen da doprinese povećanju kvaliteta sajta i interaktivnosti, došlo se do faze kada bez njega podaci iz web servisa ne mogu nikako da se prikažu i sajt više nije operativan. Zato je danas znanje JavaScript-a izuzetno važno za sve programere koji žele da se bave web-om, i to ne više samo poželjno nego je postalo obavezno.

6.3. Čemu služi i šta radi JavaScript

U prvom delu knjige, dosada je obrađena sintaksa i primena HTML-a i CSS-a kao jezika koji nisu programski. Dodavanjem JavaScript-a, i njegovih biblioteka, uz dodatak JSON-a, zatvara se inicijalni skup materije koja podrazumeva bazne poslove front-end programera.

Do sada je naučeno da se HTML koristi za stvaranje sadržaja koji se prikazuje u browser-u, dok se CSS koristi za stilizovanje i prezentaciju istog. Postavlja se pitanje gde je tu uloga JavaScript-a, obzirom da se sajt sa HTML-om i CSS-om može kompletno napraviti? Odgovor na to je vrlo opširan i krije se iza osnovne podele sajtova na statičke i dinamičke, uz mogućnost da i jedni i drugi budu interaktivni. Primenom samo HTML-a i CSS-a u potpunosti, i vrlo kvalitetno, se mogu napraviti statički web sajtovi. Međutim, ukoliko želimo da sajt pored toga bude i interaktivan, za to će se koristiti JavaScript.

Interaktivnost dalje podrazumeva širok spektar mogućnosti, ali se i ona grubo može posmatrati kroz: manipulaciju elemenata DOM-a i rad sa događajima.

Manipulacija elementima DOM-a daje mogućnost JavaScript-u da u pojedinim trenucima vremena, na željenim lokacijama web strane ili u zavisnosti od akcije korisnika, pristupi bilo kom elementu DOM-a i da mu promeni neko svojstvo ili vrednost. Praktično ovo znači da JavaScript može u bilo kom trenutku da pristupi bilo kom postojećem HTML tagu ili elementu, da mu promeni sadržaj, doda atribut, promeni vrednost atributa, da kreira element, da ga sakrije ili bilo šta drugo, kao da to programer radi ručno. Na taj način dolazi do potpune i trenutne promene sadržaja, obzirom da je HTML zadužen za generisanje sadržaja web strane. Na vrlo sličan način, JavaScript može da pristupi CSS-u, doda određenu klasu, svojstvo, promeni vrednost svojstva i sl. i na taj način potpuno ili delimično promeni vizuelni identitet sajta.

Kako je JavaScript u mogućnosti da ovo realizuje delimično ili da kompletno modifikuje inicijalni HTML tj. CSS kod, žargonski se kaže da je JavaScript njima nadređen.

Rad sa događajima obuhvata veliki skup isprogramiranih aktivnosti samog browser-a koji detektuje određene akcije i aktivnosti korisnika unutar web browser-a i dozvoljava JavaScript-u da za svaku od tih aktivnosti primeni neki kod. Tako browser svakim pokretom miša, detektuje da li je korisnik ušao u oblast nekog elementa ili iz nje izašao, da li je u fokusu npr. tekstualnog polja, da li kuca karakter sa tastature, da li je kliknuo na taster ili checkbox itd. Svaki put kada se neka od ovih aktivnosti desi, browser se trigeruje i „zapita“ da li u tom slučaju treba da se primeni neki kod. Ukoliko je taj kod definisan JavaScript-om kažemo da je taj događaj isprogramiran i da browser ima jasne instrukcije šta u tom trenutku da uradi.

Praktični primeri rada sa događajima su da se prelaskom miša preko drop-down menija, automatski otvara podmeni sa skupom linkova koji njemu pripadaju. Kako se ovo dešava? Ceo meni, sa svim svojim stavkama, kako glavnog tako i nižih hijerarhijskih nivoa, se inicijalno dobije od servera u formi HTML-a i CSS-a. Ukoliko programer želi da se svi podmeniji inicijalno sakriju, iskoristiće prvo događaj koji se aktivira neposredno po učitavanju celog sajta, i isprogramirati da u tom trenutku bude realizovana manipulacija nad svim podmenijima, i u CSS-u će proglasiti njihovo svojstvo takvim da se oni sakriju od korisnika. To znači da

će se ceo podmeni učitati, ali toliko brzo sakriti da korisnik to ne može da detektuje. Dalje će se čekati sledeći događaj, koji je prelazak mišem preko neke stavke u glavnom meniju. Kada se on desi, kao akcija korisnika, browser će kao reakciju na tu akciju pozvati JavaScript kod koji će ponovo da izvrši manipulaciju DOM-om i da prozove svojstvo CSS-a koje će sada da učini vidljivim samo onaj podmeni, na čiju se glavnu stavku prešlo mišem, i tako korisnik vidi pod meni. Ovaj postupak se dalje ponavlja za svaki drugi podmeni. Na ovom primeru se vidi velika koordinacija browser-a, korisnika i JavaScript-a nad inicijalno učitanim HTML-om i CSS-om.

Slični primeri su za klik na link ili taster *Pročitaj više*, kada se otvori dodatni deo sadržaja sajta, kao i realizacija slider-a kako sa slikama tako i sa tekstom u formi najnovijih vesti, galerije slika sa animacijama itd. Žargonski se kaže da je JavaScript zadužen za sve što se „mrda, stvara, nestaje ili menja“ kako u vremenu, tako u zavisnosti od aktivnosti korisnika, datuma, doba dana itd.

Ove inicijalne poslove JavaScript-a, manipulaciju DOM elementima i rad sa događajima, vrlo brzo su proširili i novi, sada već standardni, poslovi. Danas se rad sa formama tj. podacima koje korisnik unosi u formu, smatra jednim od prioriteta JavaScript-a. Ovi poslovi se odnose na proveru da li je neko obavezno polje popunjeno, pa ako nije da se ispiše informacija o grešci korisniku. Nakon toga, ako se nešto i unese, da se detaljno proveriti da li je to u odgovarajućem formatu, dužini, sa dozvoljenim skupom karaktera itd. Na primer, može se proveriti da li je *re-password* isti kao *password*, da li je mail u dobrom formatu, da li je izabran bračni status, da li je ime sa velikim slovom, da li JMBG broj ima tačno 13 cifara itd. Na ovaj način se sve provere korisničkih podataka „prebacuju“ na klijentski računar, i serveru šalju samo ako su zadovoljile sve definisane kriterijume. Da li ovo znači da se provere neće opet vršiti na serveru? Odgovor je: Ne, jer se uvek na klijentu mogu maliciozno promeniti ili blokirati kodovi, ali to radi jako mali broj korisnika, dok većina mora da ispravno unese sve što se traži u formi, i tako rastereti server velikog broja zahteva za kontrolu i odbijanje pogrešno popunjenih podataka.

Pored ovih aktivnosti, posebno mesto dobija jQuery koji je inicijalno bio namenjen za sve vrste animacija, tranzicija, atraktivnih efekata i svega što sajt čini vizuelno lepšim i kvalitetnijim. Danas se jQuery podjednako puno koristi i za manipulacije DOM-om, dohvatanje i prikazivanje podataka iz XML-a i JSON-a, rad sa AJAX-om itd.

Neki autori navode da se front-end tehnologije baziraju na tri ključna igrača: HTML, CSS i JavaScript, i porede to sa govornim jezikom. Tako se kaže da je HTML imenica koja kaže ko nešto radi, CSS pridev, koji kaže kako to radi, a JavaScript je glagol koji definiše šta radi. Na ovoj način se želi ukazati da je JavaScript namenjen za akciju i delovanje i da je njegova uloga u domenu konkretnih promena i akcija nad „pasivnim“ tj. statičkim HTML-om i CSS-om.

6.4. Šta je potrebno za pisanje JavaScript-a

JavaScript, kao programski jezik, nema neko unapred definisano razvojno okruženje, nego može da se piše u bilo kom, obzirom da ga skoro sva podržavaju. Skoro svi autori se slažu da je za početnike koji uče JavaScript najbolje da se krene sa nekim što prostijim okruženjem i često se preporučuje osnovi NotePad. Svakako je za početnike najbolje koristiti okruženje koje nema aktiviran tzv. *IntelliSense*, koji omogućava programeru da nakon nekoliko unetih karaktera automatski dobije ispisanu kompletnu sintaksu nekog metoda ili petlje, automatski završi pojedine delove koda ili inicijalno dobije generisane funkcije, klase itd. Na ovaj način početnici se sve više udaljavaju od sintakse i opterećuju velikim brojem dodatnih parametara, argumenata, svojstava itd. koja ne razumeju, a koji im u početku nisu ni potrebna. Svakako je dobro koristiti neko okruženje koje je namenjeno programerima i koje ume da ukaže na promene ili nepravilnu osnovnu sintaksu, mogućnost da se prati kod po numerisanim linijama koda, postavljaju *break-point*-i, lako učitavaju fajlovi različitih ekstenzija i sl.

Na tržištu postoji dosta ovakvih alata, od kojih je najveći broj onih kvalitetnijih komercijalnih pa različite firme koriste različita rešenja. Međutim, za učenje nije neophodno da se kupuje bilo koje okruženje i savetujem da izbor bude besplatni softver *Notepad++*, koji je namenski pisan za programere i koji ima veliki broj besplatnih dodataka kojima se može automatizovati pisanje koda, ubrzati ili okruženje lako prilagoditi samom programeru u smislu pisanja sopstvenih skraćenica, dizajna i sl.

Bez obzira koje okruženje se izabere, ono neće uticati na kasniju realizaciju koda. Mi ćemo u ovom kursu da se bavimo sa onim delom JavaScript-a koji se koristi za potrebe web aplikacija, pa će se rezultat delovanja pratiti isključivo u browser-u. Sam browser može da utiče na realizaciju tj. interpretaciju koda, i to je realan problem koji se novijim browser-ima sve brže i lakše rešava.

Međutim, činjenica je da web programer često mora da piše dodatne kodove koji će omogućiti da se isti kod na isti ili sličan način interpretira u svakom browser-u. Ne treba zaboraviti da prilikom pisanja koda nije jedino bitno da programer na svom računaru, u svom browser-u, potvrdi da kod radi, jer je taj kod namenjen širokom broju krajnjih korisnika. Oni, u svojim kućama, imaju različite konfiguracije računara, brzine Interneta, različite browser-e, verzije browser-a, antivirusne programe i sl. što često može uticati na brzinu, način i mogućnost prikaza JavaScript koda.

Zato je jako bitno da programer, na početku, svoj kod u postupku pisanja testira bar na dva ili tri browser-a na sopstvenom računaru i to onim koji su najzastupljeniji kod korisnika za koje piše kod. Takvim testiranjem smanjiće se kasniji broj detektovanih problema i bagova, koje utiču na lošiji stav korisnika o sajtu ili programeru.

6.5. Kako i gde se piše JavaScript

Obzirom da je fokus ove knjige na JavaScript koji je namenjen web sajtovima, JavaScript kod se mora, na neki način, integrisati u ranije definisan HTML i CSS kod web strane. Za tu potrebu može se iskoristiti:

1. interno napisan kod,
2. eksterno napisan kod ili
3. kombinacija interno i eksterno napisanog koda.

Interno napisani kod je kod koji je sastavni deo koda web strane i piše se zajedno sa HTML tj. CSS kodom. Eksterni kod se piše u potpuno odvojenom fajlu, koji ima ekstenziju *js*, i koji je potrebno integrisati u neku web stranu.

Kako se JavaScript integriše u web stranu, tj. HTML kod, a HTML ima tag kao svoju baznu strukturu, tako je sintaksom HTML-a definisan tag `<script>` za potrebe uključivanja svih skript jezika u web stranu. Tag `<script>` je u osnovi upareni tag, ali je u HTML 5 sintaksi dozvoljeno da se piše i kao neupareni kod upotrebe eksternih kodova. Kako ovo još uvek nije podržano u svim verzijama browser-a savetujem da se koristi kao upareni.

Iako je tag `<script>` dovoljna informacija za browser, da zna da tu nastupa neki skript jezik, a ne klasičan HTML kod, vrlo je bitno atributima ovog taga browser-u naglasiti koji interpretator treba da aktivira da bi razumeo sintaksu skript jezika, obzirom da JavaScript nije jedini skript jezik, a da browser-i podržavaju nekoliko njih. U ranijem periodu programeri su koristili atribut *language* i to u formi *language="JavaScript"* čime se browser-u jasno davalo do znanja koji je jezik predstojećeg koda. Međutim, u HTML 5 i novijim DTD-ovima, ovaj atribut je prevaziđen, i sada je obavezan atribut *type* i to u obliku *type="text/javascript"* ili striktnije kod aplikacija kao *type="application/javascript"*. Trenutno se češće koristi *type="text/javascript"* što vrlo podseća i na obavezan atribut *type* kod definisanja eksternog CSS koda. Pored atributa *type*, savremenim DTD-om, dozvoljeni su i atributi *charset*, *src*, *defer*, *event* i *for*.

Na ovaj način HTML kod, kojim se definiše prostor za pisanje internog JavaScript-a, izgleda ovako:

```
<script type="text/javascript">
  //neki JavaScript kod
</script>
```

Ključno pitanje je pozicija tj. mesto ovog taga unutar HTML strukture stranice. Ovaj kod se može postaviti unutar *head* i unutar *body* taga, teorijski na proizvoljnom mestu. Kada browser naiđe na tag `<script>` on zna da je sve između ovog taga i taga `</script>` kod koji pripada i tumači kao sintaksu JavaScript-a. Pre i posle ovog taga, sve pripada sintaksi i interpretaciji HTML-a.

Nailaskom na tag `<script>` browser analizira i izvršava dati JavaScript kod, i nakon tog izvršavanja se vraća u HTML i nastavlja sa interpretacijom HTML koda. Tako kažemo da se JavaScript, u ovom slučaju, izvršava na mestu na kome je napisan, u odnosu na ostatak koda strane. Primer integracije dva JavaScript koda na dva mesta unutar web strane je dat u primeru.

```

<!DOCTYPE html>
<html>
<head>
<title>Primer</title>
    <script type="text/javascript">
        //neki JavaScript kod 1
    </script>
</head>
<body>
<!-- neki html kod -->
    <script type="text/javascript">
        //neki JavaScript kod 2
    </script>
<!-- neki html kod -->
</body>
</html>

```

Drugi, i češći oblik integracije JavaScript-a je kroz eksterni JavaScript. Za tu potrebu koristi se takođe tag `<script>` sa svojim atributom *type*, ali se dodaje i atribut *src*, koji treba da ukaže na mesto gde se nalazi posebno napisan JavaScript fajl, koji je sa ekstenzijom **.js*. Na ovaj fajl može da ukazuje i apsolutna i relativna URL putanja i ovaj kod može biti negde na web serveru, gde je hostovan i sajt, ali može biti i preuzet sa neke poznate lokacije, sa drugog web servera. Kao i kod internog pisanog koda, pozicija taga `<script>` definiše mesto tj. trenutak kada se browser obraća eksternom fajlu, preuzima njegov sadržaj, i taj kod izvršava, na način kao da je taj kod napisan interno u kodu, na tom mestu. U primeru koji sledi pokazana je integracija dva eksterna fajla, od kojih je prvi na web serveru gde je hostovan i sajt, a drugi se preuzima sa drugog web servera.

```

<!DOCTYPE html>
<html>
<head>
    <title>Primer</title>
    <script type="text/javascript" src="kodovi/interniFajl.js"></script>
</head>
<body>
<script type="text/javascript" src="http://nekisajt.com/js/fajl2.js">
</script>
    <!-- neki html kod -->
</body>
</html>

```

U realnim sajtovima se praktikuje upotreba eksternih JavaScript fajlova jer se oni nalaze na jednom mestu, i samim tim sve promene, u smislu dopisivanja novog koda, korekcija ili održavanja postojećeg, se rade na jednom mestu. Pored toga, proizvoljan broj stranica se može obratiti ovakvom fajlu, i kroz eksterni pristup, preuzeti sadržaj ovog fajla, i integrisati ga u web stranu.

Međutim, treba biti vrlo oprezan, pogotovu zbog pravila SEO-a i samih korisnika. Pojedini eksterni fajlovi su često relativno veliki i potrebno je neko vreme da se uspostavi veza između browser-a i željenog fajla, da se njegov sadržaj preuzme i interpretira. Često se u fajlovima nalazi velika količina koda koja će biti potrebna tek u nekom trenutku kada korisnik izazove neku akciju, ili kada se sajt učitava, pa se preporučuje da se učitavanje svih eksternih fajlova, koji su potrebni tek nakon učitavanja sajta, radi neposredno na kraju *body* taga, kada se sav HTML kod dohvati i prikaže korisniku, i kada ima vremena da se učitaju svi drugi JavaScript fajlovi koji nisu bili neophodni za inicijalni prikaz web strane.

U tom slučaju, prethodni primer sa dva eksterna JavaScript fajla bi izgledao ovako

```

<!DOCTYPE html>
<html>
<head>
    <title>Primer</title>
</head>
<body>
<!--
    neki html kod
-->

<script type="text/javascript" src="kodovi/interniFajl.js"></script>
<script type="text/javascript" src="http://nekisajt.com/js/fajl2.js">
</script>
</body>
</html>

```

Iako se često kaže da je integracija JavaScript-a u web stranicu vezana za tag `<script>` postoji još jedan način da se kod integriše. Ova mogućnost je vrlo specifična i vezana je sa pojedine specifične događaje tj. aktivnosti korisnika na nekim elementima web stranice, ali je ipak regularna. Iako će o ovome biti više reči kada se budu obrađivali događaji, ovde će se to pomenuti na nivou sintakse i razumevanja integracije JavaScript-a sa web stranom.

Za pojedine tagove, postoje definisani atributi, opisani standardom HTML-a, koji su vrlo specifični i počinju sa rečicom *on*. Npr, ako posmatrano tag *input*, koji je tipa *button*, što znači da nema inicijalno definisanu logiku šta treba da uradi, moguće je definisati događaj, u ovom slučaju *onClick*, kao HTML atribut tog taga. Ovaj atribut je vrlo specifičan jer predstavlja događaj tj. definiše da će u

trenutku klika na ovaj taster da se pozove neki JavaScript kod i izvrši neka isprogramirana akcija. Kako je ovo klasičan atribut HTML-a, on ima svoju vrednost iza znaka = i unutar znakova navoda. Međutim, vrednost ovih atributa nije klasičan podatak nego se unutar tih znakova navoda direktno ulazi u „svet JavaScript-a” i unutar njih važi kompletna sintaksa i pravila JavaScript-a, iako on pre toga nigde nije integrisan, otvoren ili sl. Zatvaranjem znakova navoda, izlazi se iz JavaScript-a i vraća u HTML kod. U ovom primeru, pozvan je i izvršen metod JavaScript-a *alert()*.

```
<!DOCTYPE html>
<html>
<body>
<form action="proba.php">
    <input type="button" value="Klik" onClick="alert('Proba');" />
</form>
<p>Neki HTML sadržaj</p>
</body>
</html>
```

■ **Rezime:** JavaScript se realizuje u browser-u, i potrebno je da se integriše u web stranu. Integracija može biti kroz interno napisani kod ili povezivanje sa ranije napisanim eksternim fajlom u kome se nalazi JavaScript kod. Mesto na kome se piše interni ili poziva eksterni kod definiše mesto i trenutak kada se dati kod realizuje i postaje dostupan stranici. Pored ovoga kod je moguće pozvati primenom specijalnih atributa HTML tagova i čiji trenutak realizacije zavisi od događaja, tj. trenutka ili akcije koje korisnik treba da izvrši nad nekim HTML elementom.

Na kraju uvodnih napomena o JavaScript-u treba naglasiti i neke negativne aspekte koji se mogu proizvesti sa ovim jezikom. Naime, upotrebom JavaScript-a je omogućeno da se učitavanjem web strane automatski pokrene određeni JavaScript kod. Ovaj kod ima mogućnost da se obrati bilo kojoj udaljenoj web lokaciji i preuzme bilo koji drugi JavaScript kod, koji se automatski preuzima i pokreće u browser-u korisnika. Sve vreme dok se ovo dešava, korisnik o tome nema nikakva saznanja niti se od njega traži bilo kakva saglasnost.

Na ovaj način, na žalost, može se dohvatiti neki maliciozni kod koji može da nanese manje ili veće probleme ili štete krajnjem korisniku. Ovakvim malicioznim pristupima, pojedini sajtovi postaju vrlo rizični za upotrebu, i dolaskom korisnika na takve sajtove, u prvih nekoliko sekundi kompletan maliciozni kod se može realizovati kod korisnika, i nastaviti da pravi velike probleme, čak iako korisnik odmah napusti takav tip sajta.

Maliciozne uloge skripta mogu biti vrlo vidljive, u smislu da se u kodu prave beskonačne petlje koje maksimalno angažuju resurse procesora i memorije i dovode do zagušenja računara. Iako deluje da je ovo vrlo loše za krajnjeg korisnika, pokazuje se da je ovo najbolja od loših mogućnosti kojima se možemo izložiti. Naime, kada se ovakav problem uoči, jasno je da je do problema došlo i traži se neko od rešenja. Mnogo opasnije su opcije kada se maliciozni kod realizuje u pozadini, bez trenutno vidljivih aktivnosti, pa korisnik i ne zna da problem postoji. Samim tim problem se i ne rešava, a kod u pozadini može izazvati velike probleme, ne samo korisnikovom računaru, već i drugim računarima kroz DDOS napade i druge oblike distribuiranih ili pojedinačnih malicioznih tehnika.

U tom smislu, treba biti vrlo obazriv prilikom posete sajtovima koji nemaju proverenu reputaciju i garanciju kvaliteta kroz provere putem antivirusnih alata i posebnih dodataka za browser-e, koji mogu da detektuju ovakve maliciozne kodove.

VII OSNOVE JAVASCRIPT-A

Kako se programira u JS-u?

SADA kada je objašnjena potreba i uloga JavaScript-a u web tehnologijama, dolazi na red da se počne učenje konkretnih pravila jezika. Pokazano je kako se JavaScript integriše u web stranicu i sada se polako kreće u pisanje operativnog koda. Pisanje koda je inicijalno bazirano na sintaksnim pravilima jezika, a zatim na logici, koja ta pravila koristi u cilju dobijanja željenih rezultata, efekata, prikaza itd. JavaScript je nasledio svu opštu sintaksu jezika C, pa je samim tim sličan sa svim drugim jezicima koji su nastali iz jezika C. Kako ovo nije prvi nivo programiranja, i očekuje se da korisnik ove knjige poznaje osnovne termine o pisanju koda, algoritama i osnovnim pojmovima u programiranju. Stoga se ti bazni termini neće objašnjavati, ali će se kroz sve elemente jezika i sintakсна pravila ponovo proći da bi se korisnik podsetio ili prilagodio sintaksi ovog jezika. Prema tome, nikakvo prethodno predznanje iz JavaScript-a nije potrebno i krećemo od samog početka. Svako, ranije znanje u domenu programiranja je svakako dobrodošlo, ali se ipak preporučuje da se poglavlja ne preskaču, da bi se osiguralo da su svi predviđeni delovi savladani.

Na samom početku definisanja pravila jezika JavaScript, treba naglasiti da je ovaj jezik *case-sensitive*, što znači da reč *proba* i *Proba* ili *PROBA* nemaju isto značenje, tj. da se mora voditi računa o upotrebi veliki i malih slova. Ovo se odnosi i na imena promenljivih, imena funkcija, metoda ili posebnih operatora. Iako je programeru ostavljeno da sam definiše imena promenljivih, funkcija itd. preporučujem da se koristi „kamila stil“ (*camel case*) tj. način kojim se promenljive pišu početnim malim slovom, i ukoliko ima potreba da se u imenu nađe više reči, reči pišu sastavljeno, pri čemu je svako početno slovo tih dodatnih reči napisano kao veliko. Tako je promenljiva *proba*, *ime*, *user*, sasvim korektna, kao i *imeKorisnika*, *adresaStanovanjaIBrojStana*, *prikazKorisnikaPoPolu* itd.

7.1. Identifikatori u JavaScript-u

Identifikatori predstavljaju jedinstvena imena promenljivih, funkcija, svojstava, argumenata, metoda itd. i mogu sadržati jedan ili više karaktera. Prilikom definisanja identifikatora potrebno je voditi računa o nekoliko pravila:

1. Prvi karakter mora biti slovo, simbol `_` ili `$`
2. Ostali karakteri mogu biti slova, simbol `_` ili `$` i brojevi.
3. Ne preporučuje se upotreba slova koji su karakteristični za pojedina pisma kao što su: č, ħ, љ, Å itd.
4. Identifikator ne sme sadržati praznine tj. *space*.

Kako je već naglašeno, nazivi identifikatora su *case-sensitive*, i toplo preporučujem upotrebu opisanog „kamila stila“ (*camel case*).

Standardi propisuju reči koji imaju specifične upotrebe kao i rezervisane reči, koje nemaju specifične upotrebe, ali nisu na raspolaganju korisnicima da ih definišu i koriste kao identifikatore. U tom smislu potrebno je pratiti standard jezika, npr. ECMA-262, i biti u toku sa ovim grupama reči. U sledeća dva pasusa dati su skupovi reči koji su rezervisani i ne mogu se koristiti kao identifikatori:

1. *break, do, instanceof, typeof, case, else, new, var, catch, finally, return, void, continue, for, switch, while, debugger, function, this, with, default, if, throw, delete, in, try.*
2. *abstract, enum, int, short, boolean, export, interface, static, byte, extends, long, super, char, final, native, synchronized, class, float, package, throws, const, goto, private, transient, debugger, implements, protected, volatile, double, import, public.*

7.2. Komentari u JavaScript-u

Kao i u drugim programskim jezicima i JavaScript daje mogućnost rada sa komentarima. Komentari su reči ili rečenice koje se ne prevode tj. ne interpretiraju, i služe programerima da opišu delove koda, dileme, napomene i sl. kako bi se lakše snašli u kodu. Komentari se korisnicima ne prikazuju.

JavaScript ima mogućnost rada sa jednorednim i višerednim komentarima.

1. Jednoredni komentar je sadržaj koji je posle simbola //
2. Višeredni komentar je sve između simbola `/*` i `*/`

Primeri upotrebe komentara dati su u listingu.

```
<script type="text/javascript">
//primer jednorednog komentara
/*    Primer
      višerednog
      komentara
    */
</script>
```

Obzirom da starije verzije browser-a nisu uvek imale podršku za rad sa JavaScript-om, kao i da su inicijalno dolazile u verzijama koje su blokirale skript, dok ih korisnik ne odobri, programeri su cele delove JavaScript koda pisali unutar HTML komentara. Na taj način su sprečili neprofesionalne prikaze originalnog JavaScript koda, korisnicima unutar browser-a, u slučaju kada browser nije mogao taj kod da interpretira tj. izvrši. Ovo je danas takođe ostala praksa, ali daleko ređa, jer skoro svi browser-i dolaze sa podrškom JavaScript kodu i inicijalno imaju aktiviranu mogućnost rada sa skriptom. U slučaju da se kod ipak želi preventivno zaštititi od neželjenog prikaza korisniku, koriste se HTML komentari kao u datom primeru.

```
<!DOCTYPE html>
<html>
<head>
    <title>Primer</title>
    <script type="text/javascript">
    <!--
        function primer() {
            //neki JavaScript kod
        }
    // -->
    </script>
</head>
<body> </body>
</html>
```

7.3. Promenljive u JavaScript-u

Svaka promenljiva može da sadrži i *ime* i *sadržaj*. Ime promenljive se definiše u skladu sa opisanim pravilima za identifikatore i mora se koristiti operator *var* ispred imena promenljive. Tako se definisanje promenljive sa imenom *korisnickoIme* vrši na sledeći način:

```
<script type="text/javascript">
    var korisnickoIme;
</script>
```

Na kraju svake linije koda, kao i u većini drugih programskih jezika, stavlja se `;`.

Ukoliko se definiše promenljiva kao u datom primeru, njen sadržaj je nedefinisan, tj. *undefined*. Ukoliko se toj promenljivoj kasnije želi dodeliti sadržaj, potrebno je u posebnoj liniji koda imenu promenljive sa simbolom `=` dodeliti željenu vrednost. Ako se promenljivoj iz primera, *korisnickoIme*, želi dodeliti sadržaj *Petar Petrovic*, to se realizuje sa:

```
<script type="text/javascript">
    var korisnickoIme;
    korisnickoIme = "Petar Petrovic";
</script>
```

Sadržaj promenljive može se definisati i u liniji koda kada se ona i kreira, na sledeći način:

```
<script type="text/javascript">
    var korisnickoIme = "Petar Petrovic";
</script>
```

Treba primetiti da je u JavaScript-u, moguće definisati promenljivu bez eksplicitnog navođenja tipa promenljive, što nije karakteristično za klasične programske jezike. JavaScript, po automatizmu dodeljuje tip promenljivoj na osnovu njenog sadržaja.

Tako je moguće definisati dve promenljive, na sličan način, ali sa aspekta jezika one imaju potpuno različite tipove podataka.

```
<script type="text/javascript">
    var korisnikIme = "Petar Petrovic";
    var korisnikGodine = 28;
</script>
```

Treba primetiti da kod promenljive *korisnikGodine*, njen sadržaj nije stavljen unutar znakova navoda, i time je ukazano da se ne radi o stringu nego o broju.

JavaScript podržava rad sa pet „prostih“ tipova podataka i jednim složenim. Prosti tipovi podataka su String, Number, Boolean, Null i Undefined dok je

složen tip podataka Object. Opšta pravila za rad sa ovim tipovima podataka su ista kao i za većinu drugih programskih jezika u smislu definisanja njihovog sadržaja.

JavaScript podržava i kreiranje više promenljivih u jednoj liniji koda, čak i ako one imaju različite tipove podataka. Primer kreiranje tri promenljive je dat sa:

```
<script type="text/javascript">
var korisnikIme = "Petar Petrovic", korisnikGodine = 28, aktivan = false;
</script>
```

7.4. Operatori u JavaScript-u

Operatori u JavaScript-u se koriste na isti način kao i u jeziku C ili većini drugih programskih jezika.

Tako se definišu osnovni operatori za sabiranje +, oduzimanje -, množenje * i deljenje /.

```
<html>
<head>
<title>Primer</title>
<script type="text/javascript">
    var rezultat1 = 20 + 5; //rezultat je 25
    var rezultat2 = 20 - 5; //rezultat je 15
    var rezultat3 = 20 * 5; //rezultat je 100
    var rezultat4 = 20 / 5; //rezultat je 4
</script>
</head>
<body> </body>
</html>
```

Treba obratiti posebnu pažnju kada se koriste operatori sa različitim tipovima podataka. Tako na primer, ako se definiše zbir stringa i broja, JavaScript će uraditi automatsku konverziju jednog, od operatora, u onaj drugi, da bi se operacija realizovala. U slučaju kada postoji bar jedan operand kao string, konverzija onog drugog se takođe vrši u string. Preporuka je da se ovakve kombinacije izbegavaju, ili ako su nužna, da se dobro obrati pažnja na rezultat. Ovo je ilustrovanom sledećim primerom

```
<script type="text/javascript">
    var rezultat1 = 5 + 5; //rezultat je 10
    var rezultat2 = 5 + "5"; //rezultat je 55
</script>
```

Pored ove četiri bazne operacije, JavaScript omogućuje i upotrebu *pre* tj. *post* inkrementa odnosno dekrementa, sa logikom kao u svim programskim jezicima, datu u sledećem primeru:

```
<script type="text/javascript">
var broj = 10;
    var rezultat1 = --broj;
    var rezultat2 = broj--;
    var rezultat3 = ++broj;
    var rezultat4 = broj++;
</script>
```

Rad sa relacionim operatorima je takođe identičan sintaksi C-a i skoro svih drugih programskih jezika. Definisani su operatori za manje <, veće >, manje ili jednako <= i veće ili jednako >=.

Standardni operator za jednako je = dok je za različito !=. Ovde takođe treba biti vrlo pažljiv jer je operator jednako, sa jednim znakom jednako, operator za dodelu vrednosti. Uvek se dodeljuje vrednost koja je sa desne strane operatora = onome ko je sa leve strane. Tako linija koda:

```
x=5;
```

dodeljuje vrednost 5 promenljivoj x. Ukoliko se pak koriste dva operatora =, tj. ==, to ima potpuno drugo značenje i ukazuje na poređenje po sadržaju onoga što je sa leve i desne strane operatora.

Tako, ako napišemo:

```
5=="5"
```

dobićemo *true*, kao znak da se sa leve i desne strane nalazi ista vrednost. Iako je u ovom slučaju sa leve strane broj, a sa desne string, JavaScript, konvertuje broj u string i dolazi do zaključka da su ove vrednosti iste.

Treba naglasiti da postoji i operator koji se zapisuje kao trostruko =, tj. kao ===. Ovaj operator pored toga što poredi vrednost, onoga što je sa leve i desne strane, poredi i da li su tipovi podataka sa leve i desne strane isti. Tako u slučaju poređenja:

```
"5"=== "5" //će se dobiti true
```

Dok u slučaju od malo pre

```
5 === "5" //će se dobiti false, jer tipovi podataka nisu identični.
```

Slično je i kod upotrebe operatora različito, tj. != ili !==.

Pored ovih često korišćenijih operatora, JavaScript radi i sa unarnim operatorom + tj. – kao i sa velikom paletom operatora koji se primenjuju na binarnom nivou, i koji su takođe isti kao i u sintaksi C-a.

Operator za računanje modula % takođe se primenjuje kao u C-u, i ima formu primene kao u primeru:

```
<script type="text/javascript">
var broj = 10;
    var rezultat1 = broj %5; //rezultat je 0
    var rezultat2 = broj %3; //rezultat je 1
</script>
```

U JavaScript-u se često koriste kombinovani ili složeni operatori, koji su posledica delovanja ranije definisanih osnovnih operatora, i imaju istu primenu kao u jeziku C. Među njima se mogu izdvojiti +=, -=, *=, /=, %=.

Standardni logički operatori su: logičko I(&&) tj. konjunkcija, logičko ILI(||), tj. disjunkcija i logičko kao i operator NE(!) tj. negacija.

Često se u grupi operatora navodi i definicija za *kondicioni* ili *uslovni operator*. Njegovom primenom, omogućeno je da se u skraćenom obliku definiše odgovor koji će kod vratiti u zavisnosti da li je neki logički uslov ispunjen ili ne. Opšta sintaksa uslovnih operatora je:

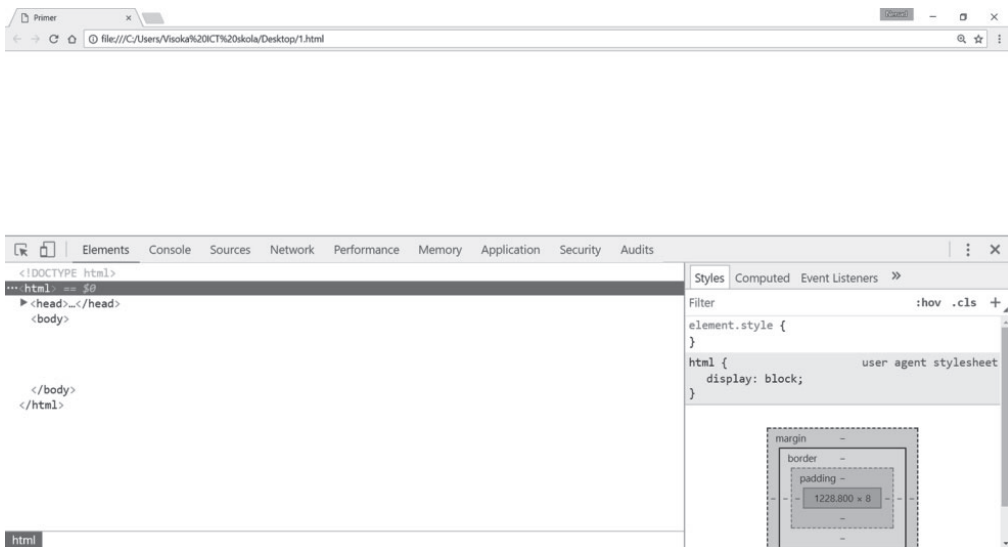
```
logički izraz ? vrednost ako je tačan : vrednost ako nije tačan;
```

ili u primeru:

```
<script type="text/javascript">
    var rezultat1 = 5<3 ? "prvi broj je manji" : "prvi broj je veci";
    //rezultat je "prvi broj je veci"
</script>
```

7.5. Ispis sadržaja van vidljivog dela sajta

U JavaScript-u se ispis vrednosti promenljivih može realizovati unutar vidljivog dela browser-a, kada je namenjen korisniku, ali i u inicijalno „nevidljivom delu“ browser-a, koji najčešće koriste programeri da vide neke međurezultate ili vrednosti pojedinih promenljivih u toku razvoja i testiranja koda. U ovom poglavlju obradiće se deo koji je van vidljivog dela sajta i koji se nalazi unutar *Inspect* dela browser-a. U ovaj deo browser-a se dolazi tako što se klikne desnim tasterom miša, bilo gde unutar web strane, i izabere opcija *Inspect*, ili u zavisnosti od browser-a prečicom sa tastature (na primer, kod Google Chrome-a je to *Ctrl+Shift+I*). Tada se dobija prozor, koji je unutar browser-a, kao na slici 7.1.



Slika 7.1. Upotreba *Inspect* dela browser-a u Chrome-u

Neposredno pored opcije *Elements*, nalazi se opcija *Console*, koja se koristi kao mesto gde će se prikazivati vrednosti promenljivih i aktivnosti koda koja je programer predvideo da se tu prikažu. Da bi se u tom delu videla vrednost promenljive *x*, koristi se sledeći kod:

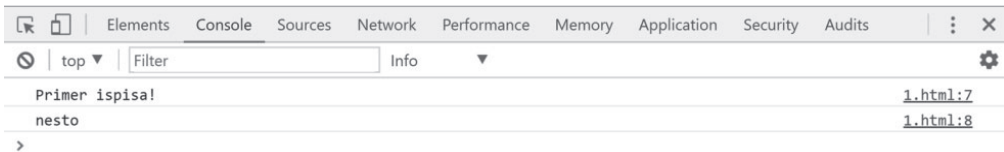
```
console.log(x);
```

dok, ako se želi samo prikazati fiksni string, moguće je umesto promenljive definisati direktan sadržaj koji se želi ispisati, kao:

```
console.log("nesto");
```

U slučaju da je kod definisan sa ove dve linije koda, i da se posmatra opcija *Console*, u ovom slučaju bi se prikazalo sledeće:

```
<html>
<head>
<script type="text/javascript">
  var x="Primer ispisa!";
  console.log(x);
  console.log("nesto");
</script>
</head>
<body> </body>
</html>
```



Slika 7.2. Ispis sadržaja pomoću JavaScript-a unutar Console-e sekcije

Kao što se vidi na slici 7.2, svaka linija ispisa se prikazuje za sebe, i desno od rezultata koji je prikazan se ukazuje u kojoj liniji koda se nalazi definisan ispis. U daljem tekstu će se *console.log()* koristiti za prikaz vrednosti različitih promenljivih sa ciljem da se korisnik privikne na njegovu upotrebu i kasnije koristi za kontrolu svih među stanja i rezultata u postupku pisanja koda.

```
<html>
<head>
<script type="text/javascript">
    var rezultat1 = 20 + 5;
    console.log(rezultat1); // ispisuje 25
    var rezultat2 = 20 - 5;
    console.log(rezultat2); // ispisuje 15
    var broj = 10;
    var rezultat3 = --broj;
    console.log(rezultat3); // ispisuje 29
    var rezultat4 = 5 < 3 ? "prvi broj je manji" : "prvi broj je veci";
    console.log(rezultat4); // ispisuje „prvi broj je veci“
</script>
</head>
<body> </body>
</html>
```

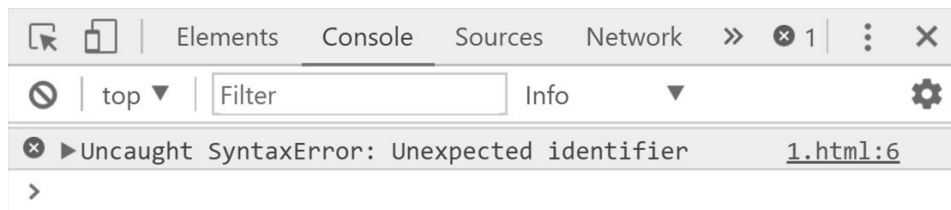
7.6. Specijalni stringovi

Pored rezervisanih imena identifikatora, u JavaScript-u, postoje i tzv. specijalni stringovi, koji predstavljaju jedan ili više karaktera, i koji imaju posebna značenja. Oni se često koriste u realnim situacijama jer predstavljaju pomoć programerima i često su jedino rešenje za izlaz iz pojedinih situacija.

Jedan od najkorišćenijih je karakter \ (*escape character*). Žargonski kažemo da ovaj karakter poništava upravljački efekat prvog narednog karaktera iza njega. Tako ako definišemo promenljivu tipa string kao:

```
<script type="text/javascript">
    var odlomak="Tako on dolazeći reče:"Dobro jutro prijatelji!", ali brzo ode
    svojim putem.";
    console.log(odlomak);
</script>
```

JavaScript će prijaviti grešku:



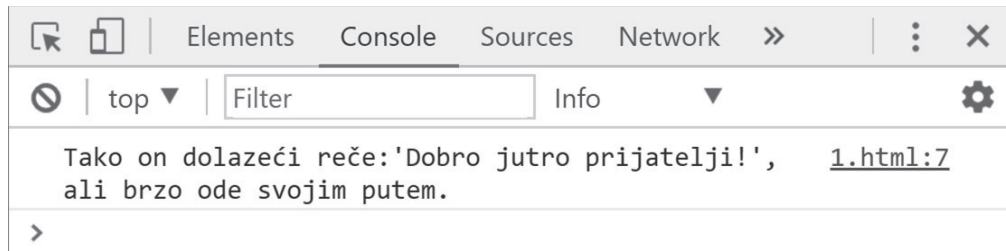
Slika 7.3. Primer pogrešne upotrebe znakova navoda

Ova greška, slika 7.3, je sintaksna greška jer su otvoreni znaci navoda ispred reči *Tako* protumačeni kao zatvoreni ispred reči *Dobro*, i dalji sadržaj nije u skladu sa sintaksom JavaScript-a, jer nije pod navodnicima kao string, a ne predstavlja neko ime funkcije, metoda ili sl.

U ovakvim situacijama, moguće je naći rešenje u promeni navodnika, pa recimo upotrebiti ovakvu kombinaciju navodnika:

```
<script type="text/javascript">
  var odlomak="Tako on dolazeći reče:'Dobro jutro prijatelji!', ali brzo ode
svojim putem.";
  console.log(odlomak);
</script>
```

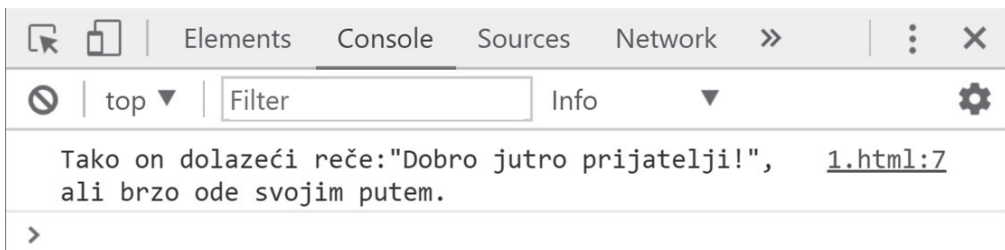
Dobija se željeni efekat, što je prikazana na slici 7.4.



Slika 7.4. Primer kombinovanja jednostrukih i dvostrukih znakova navoda

Međutim, ako je broj navodnika i njihovih kombinacija veći od postojeća dva, ili ako programer želi da sve navodnike uniformiše, tada se koristi pomenuti specijalni karakter `\`, neposredno ispred znaka navoda koji želimo da se „poništi“. Tada ga JavaScript „doživi“ kao „običan“ karakter a ne kao neki koji ima neko značenje, i može narušiti sintaksno pravilo, kao u prethodnom slučaju. Tako početni tekst može da se modifikuje na sledeći način:

```
<script type="text/javascript">
  var odlomak="Tako on dolazeći reče:\"Dobro jutro prijatelji!\", ali brzo
ode svojim putem.";
  console.log(odlomak);
</script>
```

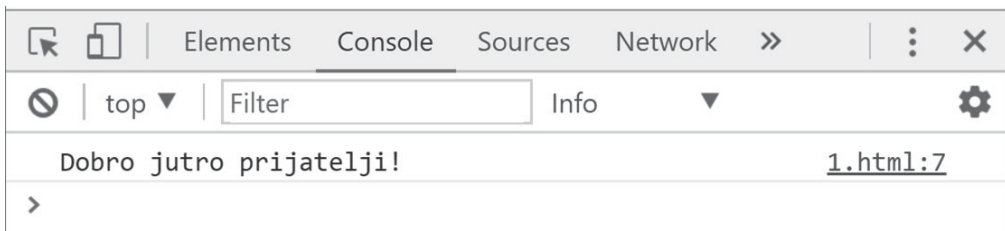


Slika 7.5. Upotreba simbola `\` za neutralisanje upravljačkog efekta znaka navoda

Kao i kod upotrebe za dvostruke znake navoda, `\`, koristimo `'` za jednostruke.

Ukoliko se *escape* simbol stavi ispred nekog “neupravljačkog” karaktera, on neće uticati na njegov prikaz, ali se *escape* simbol neće prikazati.

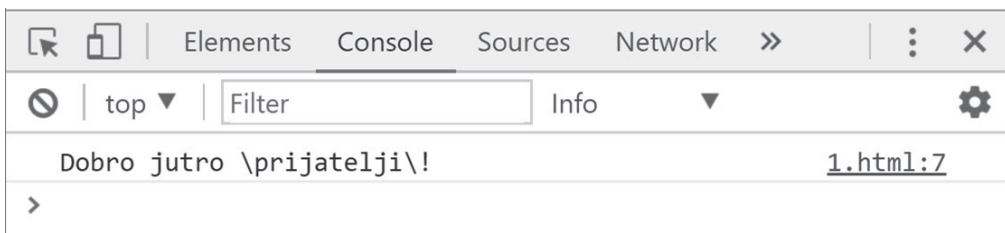
```
<script type="text/javascript">
var odlomak="Dobro jutro \prijatelji!";
console.log(odlomak);
</script>
```



Slika 7.6. Upotreba simbola `\` za neupravljački karakter

U tom slučaju, potrebno je neutralisati ulogu samog *escape* simbola, i staviti drugi *escape* simbol, koji tada postaje `\\`.

```
<script type="text/javascript">
var odlomak="Dobro jutro \\prijatelji\\!";
console.log(odlomak);
</script>
```

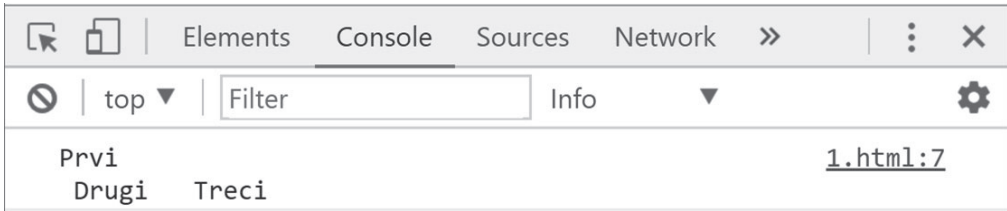


Slika 7.7. Upotreba simbola `\` za neutralisanje upravljačkog efekta simbola `\`

Kod programskog prelaska u novi red, tj. označavanje kraja tekućeg reda, koristimo `\n`. Na ovaj način sve što je u stringu iza `\n` prikazuje se u novom redu.

Često korišćen je i, `\t`, koji prikazuje razmak u širini tabulatora, a ne klasičnog *space*-a.

```
<script type="text/javascript">
var x="Prvi \n Drugi \t Treci";
console.log(x);
</script>
```



Slika 7.8. Upotreba simbola `\n` za prelazak u novi red

Pored ovih postoje i drugi specijalni simboli ali su za potrebe ovog kursa manje bitni i neće se sada navoditi, ali se preporučuje da se o tome ima svest i vodi računa kod njihove upotrebe.

7.7. Operator `typeof`

Kako se prilikom definisanja promenljive ne definiše njen tip, nego je dovoljno samo definisati ime tj. i vrednost, koristi se operator *typeof*. Pomoću njega može da se sazna tip sadržaja promenljive, u posmatranom trenutku. Ovaj operator uvek vraća string, kao odgovor, i to jedan od sledećih:

“*undefined*” ako je sadržaj nedefinisan tj. nikada nije dodeljen,

“*boolean*” ako je sadržaj promenljive tipa *boolean*,

“*string*” ako je sadržaj promenljive tipa *string*,

“*number*” ako je sadržaj promenljive tipa *number*,

“*object*” ako je sadržaj *object* ili *null* i

“*function*” ako je sadržaj funkcija.

Upotreba operatora *typeof* je vrlo jednostavna i promenljivu čiji tip sadržaja želimo da proverimo pišemo iza operatora, unutar, ili bez, običnih zagrada:

```
typeof (promenljiva);
```

U sledećem primeru pokazaće se kreiranje dve promenljive *x* i *y*, jedne sa vrednošću 5, a druge sa vrednošću *Web programiranje*. U slučaju prve promenljive, napravljena je još jedna promenljiva koja definiše njen tip, i onda je

ona odštampana kroz `console.log()`, dok je u drugom slučaju unutar `console.log()` direktno dohvaćen njen tip pomoću operatora `typeof`.

```
<html>
<head>
<script type="text/javascript">
    var x = 5;
    var tip = typeof (x);
    console.log(tip); // ispisuje number
    var y = "Web programiranje";
    console.log(typeof y); // ispisuje string
</script>
</head>
<body> </body>
</html>
```

Očekivano, rezultat za prvu je *number* a za drugu *string*. Ukoliko se promenljiva samo definiše, bez dodele vrednosti, kao u sledećem kodu promenljiva `x`, tip sadržaja za ovakvu promenljivu je *undefined*.

```
<script type="text/javascript">
    var x;
    console.log(typeof x);
</script>
```

undefined	<u>1.html:7</u>
>	

Slika 7.9. Prikaz tipa sadržaja promenljive `x` koja nema vrednost

Ukoliko se promenljiva definiše kao *number*, pa joj se doda npr. *string*, promenljiva će zbog automatske konverzije promeniti svoj tip. U sledećem primeru tip promenljive je odštampan dva puta, tj. pre i posle primene operatora sabiranje, sa ciljem da se pokaže da se mora voditi računa o tome da se tip sadržaja može promeniti primenom različitih vrsta operatora.

```
<html>
<head>
<script type="text/javascript">
    var x = 10;
    console.log(typeof x); // ispisuje number
    x = x + "5";
    console.log(x); // ispisuje 105
    console.log(typeof x); // ispisuje string
</script>
</head><body> </body>
</html>
```


U slučaju da se traži određivanje tipa sadržaja promenljive koja nije definisana, neće se prijaviti greška, kao u narednom primeru za promenljivu *z*, nego će i ona imati tip *undefined*.

```
<script type="text/javascript">
    var x = 10;
    console.log(typeof z);
</script>
```

undefined	1.html:7
>	

Slika 7.10. Prikaz tipa sadržaja nedefinisane promenljive

Oдавде se zaključuje da treba biti vrlo obazriv i da samo operatorom *typeof* nije moguće saznati da li je promenljiva ranije definisana ili ne. Zato se uvek preporučuje da se promenljive deklariraju (definiše se njeno ime na početku koda), dok se inicijalizacija (dodela prve vrednosti nekoj promenljivoj) može uraditi i kasnije, pa će *typeof* ukazivati da ta vrednost nije dodeljena deklarisanom promenljivoj.

Definisanjem, tj. deklaracijom, promenljive sa specifičnom vrednošću *null*, ukazuje na postojanje objekta koji nema definisan sadržaj. Iz tog razloga će tip takve promenljive takođe biti *object*.

```
<script type="text/javascript">
    var x = null;
    console.log(typeof x);
</script>
```

object	1.html:7
>	

Slika 7.11. Prikaz tipa sadržaja promenljive sa vrednošću *null*

Tip objekat će se puno koristiti kada se dođe do rada sa JSON-om, tako da će *null* ukazivati da nema definisanog sadržaja unutar objekta, što će najčešće ukazivati na problem sa serverom, jer će obično serverski kod isporučivati JSON klijentu.

Pored svega navedenog, treba naglasiti da postoji i specifična numerička vrednost koja se skraćeno zove NaN tj. *Not a Number*. Ova vrednost se koristi u situacijama kada se kao rezultat tj. tip sadržaja očekuje broj. Ukoliko očekivana vrednost nije broj, to ukazuje na neki problem. Ovo se često dešava kod deljenja sa nulom, ili deljenja sa promenljivima čiji sadržaji nisu dobro definisani ili se ne mogu konvertovati u broj. Za provere da li je neki sadržaj NaN koristi se funkcija *isNaN()*, koja vraća *true* ako testirani sadržaj nije broj tj. *false* ako jeste. I ovde treba biti obazriv jer će se kao *false* vratiti i kod sadržaja koji se **moгу**

konvertovati u broj. Ovo je slučaj sa logičkim promenljivima koje se konvertuju u 1 (*true*) tj. 0 (*false*), kao i u slučaju stringa u kojima su samo cifre (npr. „555“ se može konvertovati u 555, dok „pera“ ne može). U primeru koji sledi, pokazane su situacije kada će se kao *false* vratiti i sadržaji koji inicijalno nisu number, ali se u njega mogu konvertovati.

```
<script type="text/javascript">
  var x = isNaN(5);
  var y = isNaN("5");
  var z = isNaN(true);
  console.log(x); // ispisuje false
  console.log(y); // ispisuje false
  console.log(z); // ispisuje false
</script>
```

7.8. Konverzija

Već je pokazano da promenljiva koja ima jedan tip sadržaja, nakon primene neke operacije, može dobiti drugi tip sadržaja (sabiranje tipa number sa string daje string). Kažemo da je došlo do automatske konverzije jednog tipa u drugi tip podatka i to stručno zovemo *implicitnom* konverzijom. Suprotno tome, je *eksplicitna* konverzija, kada programer želi da jedan tip sadržaja konvertuje u drugi. Pomoću *typeof* se može utvrditi koji je u tom trenutku tip sadržaja neke promenljive i po potrebi on se može konvertovati u neki drugi.

Za eksplicitnu konverziju postoji nekoliko načina za realizaciju:

1. Konverzija nenumeričke vrednosti u tip *number*
2. Konverzija bilo kog tipa podatka u *string*

Za konverzije u *number*, postoje tri funkcije za ovu namenu: *Number()*, *parseInt()* i *parseFloat()*, dok kod konverzije u string najčešće koristimo metod *toString()* i funkciju *String()*.

Funkcija *Number()* radi po sledećim pravilima:

- a. vrednosti *true* i *false* konvertuje u 1 tj. 0,
- b. vrednost *undefined* konvertuje u NaN
- c. vrednost *null* konvertuje u 0
- d. prazan string konvertuje u 0
- e. string koji sadrži cifre, uz mogućnost simbola + ili -, kao i tačke, konvertuje se u broj (npr. „555“ se konvertuje u 555)
- f. string koji sadrži druge karaktere sem onih opisanih u tački e. konvertuje u NaN

Primeri upotrebe funkcije *Number()* su:

```
<script type="text/javascript">
    var vrednost1 = Number("Proba");
    var vrednost2 = Number("-555");
    var vrednost3 = Number("0002.20");
    var vrednost4 = Number("");
    var vrednost5 = Number(true);
    console.log(vrednost1); // ispisuje NaN
    console.log(vrednost2); // ispisuje -555
    console.log(vrednost3); // ispisuje 2.2
    console.log(vrednost4); // ispisuje 0
    console.log(vrednost5); // ispisuje 1
</script>
```

Druga funkcija za konverziju u tip *number* je *parseInt()*. Ovo je češće korišćena funkcija od *Number()* kada se zna da se želi dobiti samo celobrojna numerička vrednost. Primeri upotrebe funkcije *parseInt()* su dati sa:

```
<script type="text/javascript">
    var vrednost1 = parseInt("Proba543");
    var vrednost2 = parseInt("-555.5");
    var vrednost3 = parseInt("0002.20");
    var vrednost4 = parseInt("30");
    var vrednost5 = parseInt("543Proba");
    console.log(vrednost1); // ispisuje NaN
    console.log(vrednost2); // ispisuje -555
    console.log(vrednost3); // ispisuje 2
    console.log(vrednost4); // ispisuje 30
    console.log(vrednost5); // ispisuje 543
</script>
```

Obratiti pažnju na prvi i poslednji primer kod konverzije cifara u kombinaciju sa slovima, u prethodnom primeru. Ukoliko string počinje sa ciframa konverzija je moguća dok u suprotnom nije. Tako obratiti pažnju na treći primer kada je string konvertovan u celobrojnu vrednost 2.

Poslednja funkcija za konverziju u tip *number* je *parseFloat()*. Ova funkcija radi vrlo slično kao i *parseInt()* s tim što omogućava da pored celobrojnog dela broja imamo i decimalni. Primeri upotrebe funkcije *parseFloat()* su dati u primeru:

```
<script type="text/javascript">
    var vrednost1 = parseFloat("1.2e5");
    var vrednost2 = parseFloat("-555.5");
    var vrednost3 = parseFloat("0002.20");
    var vrednost4 = parseFloat("30.25.20");
    var vrednost5 = parseFloat("543Proba");
    console.log(vrednost1); // ispisuje 120000
```

```

console.log(vrednost2); // ispisuje -555.5
console.log(vrednost3); // ispisuje 2.2
console.log(vrednost4); // ispisuje 30.25
console.log(vrednost5); // ispisuje 543
</script>

```

Konverzija u string je moguća na dva načina. Prvi je metod *toString()*. Kako je ovo metod, on se primenjuje na prethodno definisanu promenljivu, tako što se stavi `.` i ovaj metod. Na primer, ako se promenljiva *x* sa sadržajem 555 želi konvertovati u string, to se zapisuje kao:

```

var x = 555;
var konvertovana = x.toString();

```

Kod ovog metoda treba obratiti pažnju na vrednosti koje vraća u slučaju vrednosti *null* i *undefined*. Kada se definišu promenljive sa tim vrednostima, kod prijavljuje grešku. U listingu koji je dat, te linije su zakomentarisane, ali ako se komentar na bilo kojoj skloni, prijavljuje se greška jer metod *toString()* ne može da konvertuje taj tip sadržaja.

```

<script type="text/javascript">
//var vrednost1;
    var vrednost2 = 543;
//var vrednost3 = null;
    var vrednost4 = -321.54;
//var x1 = vrednost1.toString();
    var x2 = vrednost2.toString();
//var x3 = vrednost3.toString();
    var x4 = vrednost4.toString();
//console.log(x1);
    console.log(x2); // ispisuje 543
//console.log(x3);
    console.log(x4); // ispisuje -321.54
</script>

```

Problemi koji nastaju sa metodom *toString()* se lako rešavaju upotrebom funkcije *String()*. Na istom početnom primeru od malo pre, ali bez komentara, funkcija *String()* uspešno konvertuje različite sadržaje u sadržaj tipa string.

```

<script type="text/javascript">
var vrednost1;
var vrednost2 = 543;
var vrednost3 = null;
var vrednost4 = -321.54;
    var x1 = String(vrednost1);
    var x2 = String(vrednost2);
    var x3 = String(vrednost3);
    var x4 = String(vrednost4);

```

```

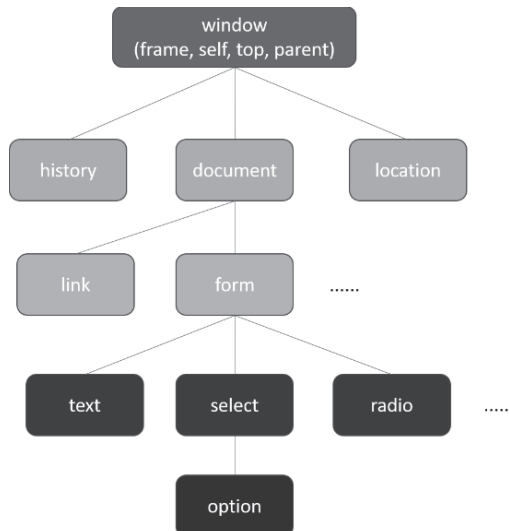
console.log(x1); // ispisuje undefined
console.log(x2); // ispisuje 543
console.log(x3); // ispisuje null
console.log(x4); // ispisuje -321.54
</script>

```

7.9. Rad sa DOM-om

DOM predstavlja skraćenicu od *Document Object Model* i predstavlja jednu vrstu plana organizacije svih objekata koji se nalaze na web strani. Stručno posmatrano, DOM je API (*Application Programming Interface*) za XML koji je proširen i za korišćenje HTML-a.

DOM se kreira od strane browser-a i omogućava browser-u, ali i programerima, da razumeju, povežu, prozovu i kasnije pristupe elementima stranice sa ciljem da pročitaju ili modifikuju neko njegovo svojstvo ili vrednost. Na ovaj način, DOM omogućava kompletan i intuitivan pristup elementima strane, po redosledu njihove pripadnosti nadređenim elementima tj. roditeljskim tagovima. Tako kažemo da kod pristupa i rada sa DOM-om primenjujemo hijerarhijski pristup objektima stranice. Na početku primene DOM-a različiti browser-i su imali različite pristupe u njegovoj organizaciji, ali se brzo došlo do oblika koji su primenjivali manje više svi aktuelni browser-i. Jedna od opštijih šema DOM-a data je na slici 7.12.



Slika 7.12. Struktura DOM-a i povezanost njegovih objekata

Kao što je prikazano na slici, na vrhu hijerarhije nalazi se objekat *window*. Ovaj objekat reprezentuje kompletan sadržaj prozor-a, tj. browser-a, u kome se prikazuje sadržaj web strane. Ovde su podržani i rad sa više frame-ova, pri čemu svaki *frame* takođe predstavlja *window* za sebe.

Svaki HTML dokument koji se učitava u window postaje *document object*. Hijerarhijski, ovaj objekat je najbitniji jer će on u sebi sadržati veliki broj drugih objekata koji pripadaju HTML kodu. Ovaj objekat će biti najkorišćeniji kod primena script-a jer će se skriptom prozivati pojedini objekti unutar *document object*-a i dalje nad njima primenjivati određene akcije. Iz tog razloga veliki broj narednih linija JavaScript koda će početi sa rečju *document*.

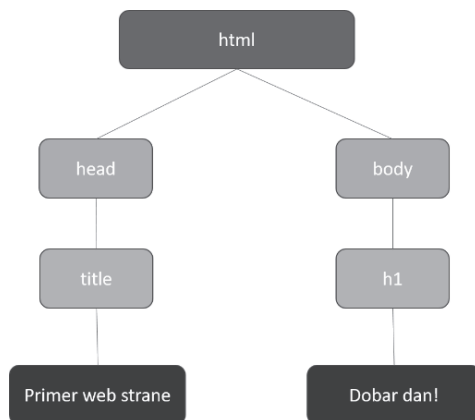
Form object ukazuje na sve što se nalazi unutar taga `<form>` i `</form>`. Ove tagove korisnik ne vidi, već samo elemente forme, ali forma povezuje svoje elemente forme i definiše celinu za sebe, pa se tako više različitih formi može naći unutar iste web strane a da međusobno ne dođe do kolizije. Unutar ovog objekta nalaze se svi elementi forme koji su dalje objekti za sebe i koji imaju svoje unutrašnje elemente, sadržaje, svojstva... kojima se pomoću DOM-a može pristupiti.

Na ovaj način, sve što se napiše unutar web strane dobija svoje mesto u DOM-u, i unutar hijerarhije se njegova pozicija jednoznačno može locirati. Na taj način skriptu će biti omogućeno da pristupi svakom objektu unutar DOM-a, i dalje ga modifikuje u skladu sa željama programera i metodama korišćenog skript jezika.

Programeri često DOM vizuelizuju na sledeći način: Ako se posmatra jednostavan HTML kod dat sa:

```
<html>
<head>
  <title>Primer web strane</title>
</head>
<body>
  <h1>Dobar dan!</h1>
</body>
</html>
```

on se može grafički prikazati kao skup objekata (node-ova), kao na slici 7.13.



Slika 7.13. Struktura DOM-a za datu web stranicu

Sada se jasno vidi da svaki element strane, kao što je tag, sadržaj taga, ugnježđenost tagova i njihov međusobni odnos postaju vrlo opisni i intuitivno povezani, da su vrlo pogodni za kasnije primene skript-a.

7.10. Sistemski dijalozi

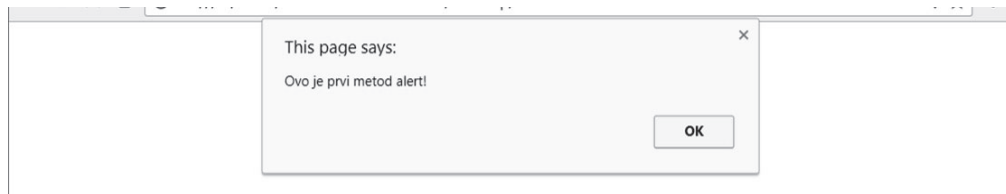
Sistemski dijalog (*system dialogs*) je mogućnost browser-a da korisniku prikaže određeni sadržaj pomoću tri metoda: *alert()*, *prompt()* i *confirm()*. Ovo su ugrađene stavke browser-a i ne može se mnogo uticati na način i dizajn prikaza, već samo na sadržaj. Ovi dijalozi se realizuju u formi posebnih prozora, sa međusobnim razlikama, pružajući i programeru i korisniku specifične mogućnosti.

Metod *alert()* – omogućava programeru da se definisani sadržaj prikaže korisniku u formu posebnog prozora kao oblik obaveštenja. Kada browser naiđe na ovu liniju koda, korisniku prikazuje poseban grafički prozor, i obustavlja svu dalju realizaciju koda, koji se nalazi u nastavku listinga. Ova obustava se završava kada korisnik klikne na taster OK, koji se automatski generiše ili na X čime zatvara taj prozor. Sadržaj koji se korisniku prikazuje je argument metoda *alert()*.

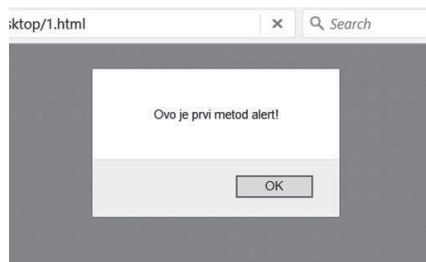
U sledećem primeru su kreirana dva metoda *alert()* da bi se pokazalo da se po realizaciji prvog, dalji kod obustavlja, i da se klikom na OK kod prvog, kod nastavlja i aktivira drugi metod.

```
<html>
<head>
<script type="text/javascript">
    alert("Ovo je prvi metod alert!");
    alert("DRUGI");
</script>
</head>
<body> </body>
</html>
```

Kao što je napomenuto, grafika prozora se razlikuje od browser-a do browser-a. Na slikama 7.14 i 7.15 je prikazano kako dati kod izgleda u Google Chrome-u i Firefox-u.



Slika 7.14. Prikaz upotrebe dijaloga primenom metoda *alert()* u Chrome-u



Slika 7.15. Prikaz upotrebe dijaloga primenom metoda `alert()` u Firefox-u

Metod `prompt()` – Slično kao metod `alert()`, i metod `prompt()` se vizuelno prikazuje u formi posebnog prozora, ali sada pored inicijalnog teksta, koji je imao i metod `alert()`, nudi i tekstualno polje za unos korisničkog sadržaja. Kako je u ovom tekstualnom polju moguće uneti inicijalno ispisan tekst, on se definiše kao drugi argument, po redosledu pisanja, unutar metoda `prompt()`. Tako je sintaksa ovog metoda:

```
prompt("argument1","argument2")
```

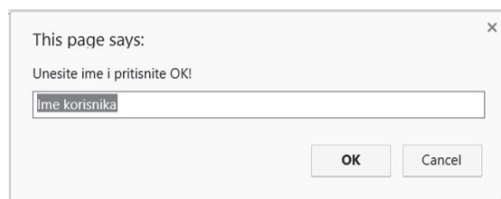
gde

argument1 - predstavlja poruku koja se prikazuje korisniku a

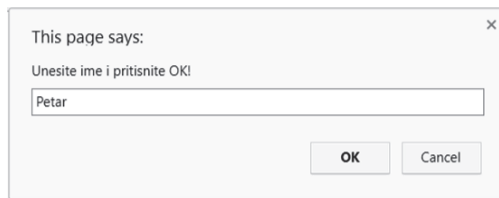
argument2 - predstavlja tekst koji se korisniku inicijalno pojavljuje u *text* polju (opciono korišćenje).

Za razliku od metoda `alert()`, sada se očekuje da korisnik unese određeni sadržaj koji treba da se „prihvati“ u neku promenljivu JavaScript-a, i tako postane promenljiva ovog jezika, sa sadržajem koji je definisao korisnik. U suprotnom, sadržaj koji korisnik unese će biti neupotrebljiv. U primeru koji sledi, korišćen je `console.log()` da prikaže vrednost promenljive nakon unosa sadržaja od strane korisnika, slike 7.16 i 7.17.

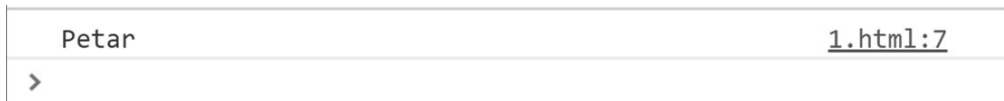
```
<script type="text/javascript">
var unetitekst = prompt("Unesite ime i pritisnite OK!", "Ime korisnika");
console.log(unetitekst);
</script>
```



Slika 7.16. Prikaz upotrebe dijaloga primenom metoda `prompt()` u inicijalnom prikazu



Slika 7.17. Prikaz upotrebe dijaloga primenom metoda `prompt()` nakon unosa sadržaja

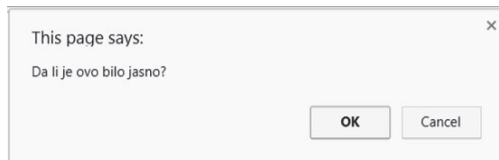


Slika 7.18. Prikaz unetog korisničkog sadržaja iz JavaScript-a

Metod `confirm()` – Kao i metod `alert()`, i metod `confirm()` se vizuelno prikazuje u formi posebnog prozora, ali pored inicijalnog teksta koji je imao i metod `alert()` i tastera `OK`, nudi i taster `Cancel`. Ovaj metod se koristi za postavljanje pitanja i vraća logičku vrednost `true` ili `false` u zavisnosti na koji taster je korisnik kliknuo. Ako se klikne na `OK`, vraća se `true`, dok se klikom na `Cancel`, vraća `false`. U slučaju da se prozor samo zatvori vratiće se takođe `false`.

Metod `confirm()` ima samo jedan argument, koji se prikazuje kao inicijalni tekst kao i metod `alert()`.

```
<script type="text/javascript">
    var unetitekst = confirm("Da li je ovo bilo jasno?");
    console.log(unetitekst);
</script>
```



Slika 7.19. Prikaz upotrebe dijaloga primenom metoda `confirm()`

Kako je rezultat metoda `confirm()` logička promenljiva, na bazi koje se kasnije nešto zaključuje, nije potrebno praviti posebnu promenljivu u koju se smešta njen rezultat, pa dalje pitati da li je ta promenljiva sa vrednošću `true` ili `false`. Dovoljno je metod `confirm()` staviti direktno u `if` uslov, bez potrebe da se poredi sa `true` ili `false`, jer će on već vratiti `true` ili `false`, i definisati šta u slučaju da je izabrano `OK` ili `Cancel`. Ovo je dato u primeru koji sledi.

```
<script type="text/javascript">
    if(confirm("Da li je ovo bilo jasno?"))
        {alert("Odlično!");}
    else    {alert("Moramo ponovo!");}
</script>
```

7.11. Kontrole toka i petlje

JavaScript podržava rad sa skoro svim klasičnim kontrolama toka i petljama. Kako ova materija pripada predmetu Osnove programiranja, ovde će se samo ukratko ponoviti primena kontroli tokova i petlji da bi se korisnik podsetio i prilagodio sintaksi JavaScript-a.

- ✓ If-else kontrola toka

Opšta sintaksa:

```
if (uslov)
    { šta se radi; ... }
else
    { šta se radi; ... }
```

Primer upotrebe:

```
<script type="text/javascript">
var unos = prompt("Koji je dan nakon ponedeljka?");
if(unos=="utorak")
    {alert("Tačno!");}
else
    {alert("Netačno!");}
</script>
```

- ✓ If-else if-else kontrola toka

Opšta sintaksa:

```
if (uslov)
    { šta se radi; ... }
else if
    { šta se radi; ... }
else
    { šta se radi; ... }
```

Primer upotrebe:

```
<script type="text/javascript">
var unos = prompt("Unesite ceo broj veći od 10 a manji od 15?");
var broj = parseInt(unos);
if(unos<=10)
    {alert("Unet je manji broj!");}
else if (unos>=15)
    {alert("Unet je veći broj!");}
else
    {alert("Unet je ispravan broj!");}
</script>
```

- ✓ switch kontrola toka

Opšta sintaksa:

```
switch(x){
  case a: šta se radi; break;
  case b: šta se radi; break;
  default: šta se radi;
}
```

Primer upotrebe:

```
<script type="text/javascript">
  var boja= prompt("Navedi jednu od boja koja se nalazi na srpskoj
  zastavi?");
  switch (boja){
    case "crvena":
      alert("Tačno!");
      break;
    case "plava":
      alert("Tačno!");
      break;
    case "bela":
      alert("Tačno!");
      break;
    default: alert("Što pre to naučite!"); }
</script>
```

- ✓ for petlja

Opšta sintaksa:

```
for (inicijalizacija; uslov; izraz nakon ciklusa petlje)
{šta se radi;}
```

Primer upotrebe:

```
<script type="text/javascript">
  for (var i = 1; i <= 8; i++){
    alert("Vrednost promenljive: i = " + i);
  }
</script>
```

- ✓ for-in petlja

Opšta sintaksa:

```
for (indeks in iskaz) {
  šta se radi...;
}
```

Primer upotrebe:

```
<script type="text/javascript">
var niz = ["a", "b", "c", "d", "e"];
for (var element in niz){
    console.log(element);
    console.log(niz[element]);
}
</script>
```

✓ while petlja

Opšta sintaksa:

```
while (uslov){
    šta se radi...;
}
```

Primer upotrebe:

```
<script type="text/javascript">
var i=1;
while(i<=8){
    alert("Vrednost promenljive: i = " + i);
    i=i+1;
}
</script>
```

✓ do-while petlja

Opšta sintaksa:

```
do {
    šta se radi...;
}
while (uslov);
```

Primer upotrebe:

```
<script type="text/javascript">
var i=1;
do{
    alert("Vrednost promenljive: i = " + i);
    i=i+1;
}
while(i<=8)
</script>
```

✓ break i continue

Kod početnika, *break* i *continue*, često izazivaju veliki problem u razumevanju, pa se često nepravedno zaobilaze, iako su vrlo korisne.

I *break* i *continue* služe za kontrolu izvršavanja broja iteracija neke petlje. *Break* forsira trenutni prekid tj. izlaz iz petlje, dok *continue* izlazi iz trenutne iteracije petlje, ali se dalje iteracije nastavljaju, onoliko puta koliko petlja to predviđa.

Najbolje je njihovu razliku pokazati na primeru. U sledećem kodu, definisana je *for* petlja koja se maksimalno može izvršiti devet puta. U svakoj iteraciji, unutar *if* uslova se proverava da li je trenutna vrednost promenljive *i* deljiva bez ostatka sa 3. Na taj način, *i* će imati vrednost 1, pa vrednost 2 i kod vrednosti 3 će proći uslov *if*-a i zbog *break* trenutno napustiti petlju i sve dalje iteracije. Van petlje nalazi se *alert()* koji ispisuje vrednost promenljive *i* pre ulaska u *if*, tj. vrednost 2, jer kako se ulaskom u *if* izašlo iz petlje, promenljiva *broj* nije stigla da poveća svoju vrednost u posmatranoj trećoj iteraciji *for* petlje. Na taj način, posmatrani kod će ispisati vrednost promenljive *broj* kao 2.

Primer upotrebe:

```
<script type="text/javascript">
var broj = 0;
for (var i=1; i < 10; i++) {
    if (i % 3 == 0) { break; }
    broj++;
}
alert(broj);
</script>
```

U slučaju kada se koristi *continue*, za istu petlju od maksimalno 9 iteracija, za *i*=1, promenljiva *broj* će biti 1. Za *i*=2 i *broj* će biti 2. Međutim, kada *i* bude 3, proćiće se kroz uslov *if*-a i zbog *continue*, odmah završiti ova iteracija *for* petlje i *broj* se neće promeniti tj. ostaće 2. Kada *i* bude 4, *broj* se povećava za 1 i dobija vrednost 3. Za *i*=5, *broj* postaje 4. Za *i*=6, aktivira se *continue*, i *broj* ostaje na staroj vrednosti 4. Kada *i* bude 7, *broj* se povećava i postaje 5. Za *i*=8, *broj* je 6. Za *i*=9 aktivira se *continue*, i *broj* ostaje na staroj vrednosti 6, i tu se svakako završava i *for* petlja, pa se kao vrednost promenljive ispisuje broj 6.

```
<script type="text/javascript">
var broj = 0;
for (var i=1; i < 10; i++) {
    if (i % 3 == 0) { continue; }
    broj++;
}
alert(broj);
</script>
```

Ovaj postupak mogao je i drugačije da se posmatra. Od ukupno devet iteracija, za vrednosti i kao 3, 6 i 9, promenjiva broj se ne povećava, tj. povećaće se preostalih 6 puta i imati vrednost 6.

7.12. Rad sa funkcijama

Rad sa funkcijama u JavaScript-u je identičan kao i u drugim programskim jezicima koji su nastali iz jezika C. Funkcije predstavljaju vrlo pogodan oblik pisanja koda, svaki put kada je potrebno iste ili slične linije koda napisati više puta. Tada se te grupe linija koda napišu unutar jedne funkcije, i svaki put kada nam treba njihov rezultat, potrebno je samo pozvati funkciju koja u sebi ima te linije koda, i dobija se rezultat.

Svaka funkcija mora imati *ime*, koje se kasnije poziva, i započinje rezervisanom rečju *function*. Ukoliko funkcija nema ulazne podatke, tj. argumente, pored imena funkcije piše se (). Ukoliko ih ima, onda se oni navode međusobno razdvojeni zarezom unutar (). Nakon toga ide *telu* funkcije, koje se piše unutar zagrada { }, i sadrži sve linije koda i logiku koju funkcija treba da realizuje. Ukoliko funkcija treba da vrati neki sadržaj tj. rezultat svog rada, koristi se rezervisana reč *return* iza koje ide promenljiva, čija vrednost se vraća onome ko je pozvao funkciju.

Tako se funkcija, koja se zove *obavestenje*, i koja u telu ima metod *alert()* definiše kao:

```
function obavestenje()
{
    alert ("Tekst na prozoru za obavestenje");
}
```

Ova funkcija nema ulaznih argumenata, i kako ne vraća neku vrednost, nema upotrebe reči *return*. Kreirana funkcija se može pozvati na dva načina. Jedan, tako što se bilo gde, i bilo koliko puta, unutar JavaScript koda funkcija pozove po njenom imenu, tj.

```
obaveštenje();
```

i tada se izvršava na mestu gde je pozvana. Drugi način je da se pozove pomoću događaja, u ovom slučaju klikom na taster, kada se poziva i izvršava svaki put kada se klikne na taster. Primer pozivanja funkcije, na dva opisana načina, je dat sa:

```
<html>
  <head>
    <script type="text/javascript">
      <!--
```

```

                                function obavestenje(){
                                alert ("Tekst na prozoru za obavestenje");
                                }
                                //-->
                                </script>
                                </head>
                                <body>
                                <script type="text/javascript"> obavestenje(); </script>
                                <input type="button" value="Pozovi f-ju" onclick="obavestenje();" />
                                </body>
                                </html>

```

Ukoliko funkcija ima ulazne argumente, onda se prilikom njenog pozivanja moraju navesti vrednosti predviđenih argumenata, koji se redom dodeljuju imenima argumenata definisanih prilikom definisanja funkcije. U narednom primeru, kreirana je funkcija *srednjaVrednost()* koja ima tri argumenta, koja treba definisati prilikom pozivanja funkcije.

```

<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
        <!--
        function srednjaVrednost(a,b,c)
        {
            var total;
            total = (a+b+c)/3;
            alert(total); }
        //-->
        </script>
    </head>
    <body>
    <script type="text/javascript"> srednjaVrednost(3, 5, 7); </script>
    <input type="button" value="Pozovi f-ju" onclick="srednjaVrednost(3, 5,
7);" />
    </body>
    </html>

```

Kada se funkcija poziva sa ciljem da vrati rezultat njenog rada, potrebno je pre njenog pozivanja napraviti promenljivu u koju će se vrednost, koju funkcija vrati, skladištiti. U primeru koji sledi, kreirana je promenljiva *broj* kojoj je dodeljena vrednost koju vraća funkcija. Da bi se ovo realizovalo, u funkciji je pomoću *return* naznačeno koja vrednost, u ovom slučaju promenljive *total*, će se vratiti onome ko funkciju pozove.

```

<html>
    <head>
        <script type="text/javascript">
        <!--

```

```

        function srednjaVrednost(a,b,c)
        {
            var total;
            total = (a+b+c)/3;
            return total;
        }
        //-->
    </script>

</head>
<body>
<script type="text/javascript">
    var broj= srednjaVrednost(3, 5, 7);
    alert(broj);
</script>
</body>
</html>

```

U JavaScript-u je dozvoljeno i korišćenje funkcije koja može imati proizvoljan i nedefinisan broj argumenata. Tada se funkcija, prilikom definisanja, piše bez argumenata, ali se prilikom njenog pozivanja, argumenti navode. Da bi se u telu funkcije znalo koji i koliko argumenata je definisano prilikom pozivanja funkcije, koristi se svojstvo *length* i pristup nizu *arguments*, da bi se pristupilo svim argumentima. Ovo je pokazano u primeru koji sledi kroz funkciju koja se poziva sa jednim, dva i tri argumenta i svaki put računa sumu argumenata uvećanu za 1.

```

<html>
    <head>
        <script type="text/javascript">
function proba() {
if(arguments.length == 1) {
    alert(arguments[0] + 1);
}
else if (arguments.length == 2) {
    alert(arguments[0] + arguments[1] + 1);
}
else if (arguments.length == 3) {
    alert(arguments[0] + arguments[1] + arguments[2] + 1);
}
}
        </script>
    </head>
    <body>
        <script type="text/javascript">
            proba(2);
            proba(2,3);
            proba(2,3,4);
        </script>
    </body></html>

```

7.13. Rad sa nizovima

Do sada je pokazano kako se kreira promenljiva i kako joj se dodeljuje sadržaj određenog tipa. U ovakvim, klasičnim, promenljivima moguće je smestiti samo jedan sadržaj. Tako na primer, u promenljivu osoba smeštamo ime osobe kao:

```
var osoba = "Petar";
```

Niz je struktura podataka koja omogućava da se u jednoj promenljivoj smesti više različitih sadržaja. Tako jedan niz može da zameni više promenljivih i da se pri tome svakoj pojedinačnoj vrednosti može pristupiti, promeniti i izbrisati, kao i sa klasičnim promenljivima. Tako umesto tri promenljive koje opisuju jednu osobu

```
var osobaIme = "Petar";
var osobaPrezime = "Petar";
var osobaGrad = "Kragujevac";
```

možemo da kreiramo niz osoba, i u njega stavimo iste ove podatke:

```
var osoba = ["Petar", "Petrovic", "Kragujevac"];
```

Upotreba uglastih zagrada, da se definiše skup elemenata niza, je jedan način da se kreira niz. Drugi način je upotrebom *new Array()*

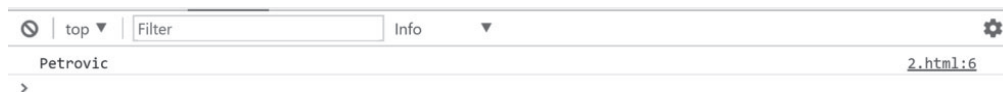
```
var osoba = new Array("Petar", "Petrovic", "Kragujevac");
```

U oba slučaja kreirana je jedna promenljiva, *osoba*, koja je u formi niza i ima po tri podatka tipa string kao svoje elemente. Pozicija prvog elementa niza, tj. njegov indeks je *0*, drugog *1* i tako redom. Elementu niza, npr. drugom elementu, se može pristupiti preko njegovog indeksa, u ovom slučaju indeks je *1*, na sledeći način

```
osoba[1];
```

Pristup elementu niza na ovaj način je dohvaćanje njegove vrednosti, pa ako se ta vrednost odštampa, dobija se

```
<script type="text/javascript">
var osoba = new Array("Petar", "Petrovic", "Kragujevac");
console.log(osoba[1]);
</script>
```

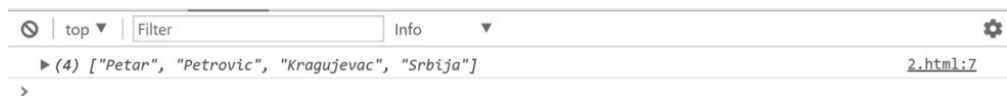


Slika 7.20. Prikaz vrednosti drugog elementa niza

Ukoliko želite upisati neku novu vrednost u niz, to je na sličan način moguće uraditi prozivanjem *imena* niza i *indeksa* niza, u zavisnosti od pozicije gde se želi upisati sadržaj. Ako u postojeći niz osoba želimo da dodamo novi podatak, obzirom da već tri postoje, na pozicijama, 0, 1 i 2, onda ćemo novi sadržaj smestiti na poziciju 3 kao:

```
<script type="text/javascript">
var osoba = new Array("Petar", "Petrovic", "Kragujevac");
osoba[3]="Srbija";
console.log(osoba);
</script>
```

Pa se štampanjem svih elemenata niza vidi i da je nova vrednost unutar niza



Slika 7.21. Prikaz vrednosti elementa niza nakon dodavanja novog elementa

Na ovaj način je dodat novi element niza, ali se na isti način i postojećem elementu može zameniti sadržaj. Na primer, ako se prvom elementu menja sadržaj iz *Petar* u *Pera*, to se radi na sledeći način

```
<script type="text/javascript">
var osoba = new Array("Petar", "Petrovic", "Kragujevac");
    osoba[3]="Srbija";
    osoba[0]="Pera";
console.log(osoba);
</script>
```



Slika 7.22. Prikaz vrednosti elementa niza nakon promene sadržaja prvom elementu

U JavaScript-u, niz može imati sadržaje elemenata koji su različitih tipova. Tako se dodavanjem novih elemenata, koji su tipa number i boolean, neće prijaviti greška.

```
<script type="text/javascript">
var osoba = new Array("Petar", "Petrovic", "Kragujevac");
    osoba[3]=1990;
    osoba[4]=true;
console.log(osoba);
</script>
```



Slika 7.23. Prikaz vrednosti elementa niza koji ima različite tipove sadržaja

U sledećem primeru prikazano je kreiranje dva niza i dodela vrednosti njihovim elementima na različite načine. Kod prvog niza, dodela je urađena nakon kreiranja niza, a kod drugog, prilikom kreiranja niza. Dodatno, pokazano je da se kao vrednosti nekog elementa niza mogu naknadno dodati vrednosti proizvoljnog tipa podatka, pri čemu vrednost može biti i element nekog drugog niza (u ovom slučaju se iz prvog niza uzima element sa indeksom 6 i stavlja u drugi niz na poziciju indeksa 3.)

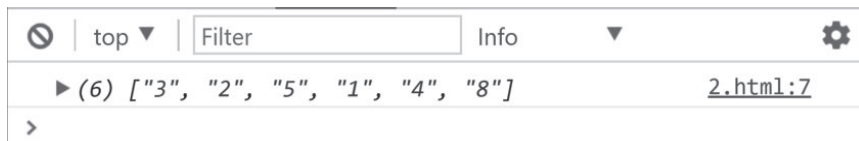
```
<script type="text/javascript">
var niz1 = new Array(7);
    niz1[0] = "Ponedeljak";
    niz1[1] = "Utorak";
    niz1[2] = "Sreda";
    niz1[3] = "Četvrtak";
    niz1[4] = "Petak";
    niz1[5] = "Subota";
    niz1[6] = "Nedelja";
var niz2 = new Array("3","2","5","1","4");
    niz2[0]= 99;
    niz2[3]= niz1[6];
    niz2[5]="Novi element niza 2";
console.log(niz2);
</script>
```

Nizovi dodatno olakšavaju rad programeru jer svaki programski jezik ima veći broj gotovih funkcija ili metoda kojima se neke stvari automatski rešavaju. Tako na primer, izračunavanje dužine niza, tj. broja elemenata niza, lako se rešava primenom svojstva *length*, pa tako imamo

```
<script type="text/javascript">
var niz1 = new Array(7);
niz1[0] = "Ponedeljak";
niz1[1] = "Utorak";
niz1[2] = "Sreda";
niz1[3] = "Četvrtak";
niz1[4] = "Petak";
niz1[5] = "Subota";
niz1[6] = "Nedelja";
console.log(niz1.length); // ispisuje 7
</script>
```

Dodavanje novog elementa niza, tako je moguće uraditi i metodom *push()* bez potrebe da se prethodno računa dužina niza pa odredi pozicija prvog sledećeg slobodnog elementa niza.

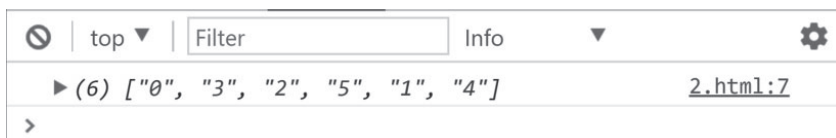
```
<script type="text/javascript">
var niz2 = new Array("3","2","5","1","4");
niz2.push("8");
console.log(niz2);
</script>
```



Slika 7.24. Upotreba metoda *push()* za dodavanje elementa niza

Kako *push()* dodaje novi element niza na poslednje mesto, tako metod *unshift()* dodaje element niza na prvo mesto:

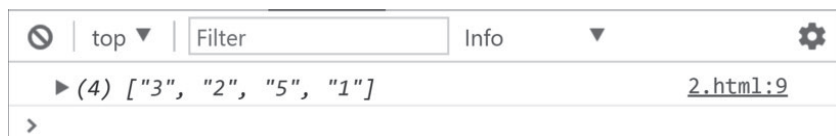
```
<script type="text/javascript">
var niz2 = new Array("3","2","5","1","4");
niz2.unshift("0");
console.log(niz2);
</script>
```



Slika 7.25. Upotreba metoda *unshift()* za dodavanje elementa niza

Brisanje elemenata niza je moguće primenom metoda *shift()*, za brisanje prvog elementa, i *pop()*, za brisanje poslednjeg elementa.

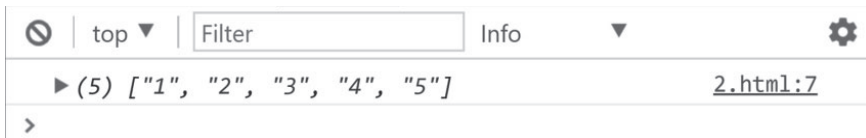
```
<script type="text/javascript">
var niz2 = new Array("3","2","5","1","4");
niz2.unshift("0");
niz2.shift();
niz2.pop();
console.log(niz2);
</script>
```



Slika 7.26. Upotreba metoda *shift()* i *pop()* za brisanje elemenata niza

Vrlo koristan je i metod `sort()` koji vrši sortiranje svih elemenata niza u odnosu na njihov sadržaj, od manje ka većoj vrednosti.

```
<script type="text/javascript">
var niz2 = new Array("3","2","5","1","4");
var sortirani = niz2.sort();
console.log(sortirani);
</script>
```



Slika 7.27. Upotreba metoda `sort()` za sortiranje elementa niza

Kod rada sa nizovima, često je potrebno programski proći kroz ceo niz, dohvatiti svaki element niza i sa njim nešto dalje raditi. Da bi se ovo uradilo, potrebna je petlja, koja će se vrteti onoliko puta koliko ima elemenata niza da bi u svakoj iteraciji rada pristupila novom elementu niza.

Ranije se najviše koristila *for* petlja. Međutim, da bi se znalo koliko ima elemenata niza, potrebno je svojstvom *length* prvo taj broj izračunati, i tada će *for* petlja imati informaciju koliko puta treba da se obrati nizu. U primeru koji sledi, *for* petljom će se pristupiti svakom elementu niza, i u svakoj iteraciji ispisati pozicija elementa kome se pristupilo, preko promenljive *i*, i sadržaj tog elementa niza, preko `niz2[i]`.

```
<script type="text/javascript">
var niz2 = new Array("3","2","5","1","4");
for(var i=0; i<niz2.length; i++)
{
    console.log("Element na poziciji " + i + " ima sadržaj " + niz2[i]);
}
</script>
```

Element na poziciji 0 ima sadržaj 3	2.html:8
Element na poziciji 1 ima sadržaj 2	2.html:8
Element na poziciji 2 ima sadržaj 5	2.html:8
Element na poziciji 3 ima sadržaj 1	2.html:8
Element na poziciji 4 ima sadržaj 4	2.html:8

Slika 7.28. Upotreba *for* petlje za dohvaćanje vrednosti svih elemenata niza

Da bi se rad sa nizovima bolje razumeo, pogledajmo jedan kompleksniji primer. U sledećem listingu definisana su dva niza, *USDzrave* i *datumPristupanja*. Ova dva niza imaju istu dužinu, i u prvom se nalaze države u USA a u drugom godine

njihovog pristupanja. Pozicija države u prvom nizu odgovara poziciji godini pristupanja za tu državu u drugom.

Od korisnika se pomoću *prompt()* traži da unese ime države za koju želi da dobije godinu pristupanja. Kada korisnik unese sadržaj, to se skladišti u promenljivu *unetaDrzava*. Zatim se pomoću *for* petlje, prolazi kroz sve elemente prvog niza, u kome je spisak država, i pomoću *if* se pita da li trenutni element niza ima vrednost kao uneti korisnički sadržaj. Ako se pronađe takvo ime, sa *break* se izlazi iz petlje i pri tome se automatski pamti vrednost promenljive *i*, u kojoj je zapamćena pozicija nađene države unutar niza *USDrzave*. Kako je na istoj poziciji, samo u drugom nizu *datumPristupanja*, godina pristupa te države, dovoljno je samo pročitati vrednost elementa niza na toj poziciji u drugom nizu. Ovaj primer bi bio identičan kada bi u prvom nizu bili gradovi Srbije, a u drugom pozivni brojevi za iste, pa se unosom grada očekivalo pronalaženje pozivnog broja.

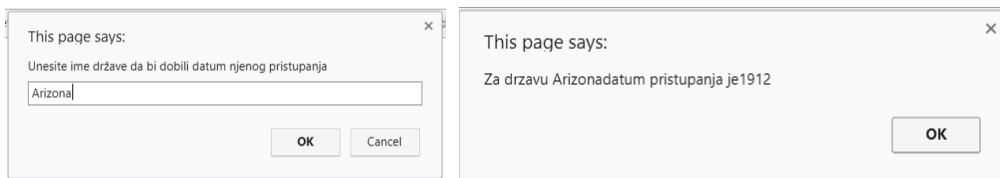
```

<html>
  <head>
    <script type="text/javascript">
      var USDrzave = new Array(51);
      USDrzave[0] = "Alabama";
      USDrzave[1] = "Alaska";
      USDrzave[2] = "Arizona";
      USDrzave[3] = "Arkansas";
      USDrzave[50] = "Wyoming";

      var datumPristupanja = new Array(51);
      datumPristupanja[0] = 1819;
      datumPristupanja[1] = 1959;
      datumPristupanja[2] = 1912;
      datumPristupanja[3] = 1836;
      datumPristupanja[50] = 1890;

      var unetaDrzava = prompt("Unesite ime države da bi dobili datum njenog
pristupanja");
      for ( var i = 0; i < USDrzave.length; i++)
      {
        if (USDrzave[i] == unetaDrzava) {break;}
      }
      alert("Za drzavu " + unetaDrzava + "datum pristupanja je" +
datumPristupanja[i] );
    </script>
  </head>
  <body>
  </body>
</html>

```



Slika 7.29. Pretraga niza i dohvaćanje vrednosti drugog elementa niza

Pored *for* petlje, za potrebe prolaska kroz elemente niza može se koristiti i bilo koja druga. Međutim, u novije vreme, programeri najčešće koriste *for-in* petlju, jer se za razliku od *for* petlje, ne mora računati dužina niza, pomoću svojstva *length*, nego sama petlja prolazi kroz sve elemente koje nađe u nizu, dok ne dođe do kraja.

U narednom primeru, pomoću petlje *for-in* prikazani su svi elementi niza imena. Promenljiva *indeks*, koja može imati proizvoljno ime, sadrži u sebi tekući indeks niza, a sadržaj tog elementa se dohvata pomoću *imena[indeks]*.

```
<script type="text/javascript">
var imena = new Array("Milena", "Marko", "Nenad", "Ana");
for(indeks in imena){
    console.log(imena[indeks]);
}
</script>
```

Milena	2.html:7
Marko	2.html:7
Nenad	2.html:7
Ana	2.html:7

>

Slika 7.30. Upotreba *for-in* petlje za dohvaćanje vrednosti svih elemenata niza

7.14. Rad sa objektima

Nakon rada sa klasičnim promenljivim, videli smo da struktura niza deluje dosta naprednije, jer može više vrednosti staviti u istu promenljivu, a da se i dalje svakoj ponaosob može pristupiti. Pored toga, rad sa nizovima je dosta olakšan upotrebom gotovih metoda i funkcija, koje rešavaju standardne i najčešće korišćene zahteve. Međutim, problem u radu sa nizovima je što su podaci međusobno neuređeni i nisu opisni. Tako element koji je na prvom mestu ne mora da ukazuje ninašta posebno, kao ni što ne mora da postoji unapred definisana veza između petog i šestog elementa. Ako bi hteli da nađemo ime korisnika, ono može biti bilo gde u nizu tj. u bilo kom elementu, pa kažemo da podaci nisu opisani na bilo koji način koji bi pomogao traženju, razumevanju ili radu sa podacima, sem da se oni pročitaju od strane čoveka i tek tada razumeju.

Intencija je da to na neki način bude definisano, pa da pored čoveka i mašina može bolje i brže da te podatke razume, a samim tim pretraži, sortira, uredi itd.

Za tu potrebu kreirani su objekti. Kao i drugi programski jezici i JavaScript podržava rad sa objektima, što je vrlo bitno jer je to sve češće korišćen oblik skladištenja podataka.

Niz smo kreirali pomoću uglastih zagrada, [], a objekat kreiramo pomoću vitičastih, tj. { }.

U nizu smo elemente razdvajali zarezom, i tako će se odvojiti i u objektu. Ključna razlika je što je kod kreiranja niza element niza bio konkretan sadržaj, npr. *string* ili *number*, dok kod objekta definišemo par (*key:value* tj. *ime:vrednost*). Kao što se vidi sada pored vrednosti, definišemo i *ime* tog podatka, da bi se kasnije tom elementu pristupalo po imenu, gde god on bio, a ne kao kod niza po indeksu.

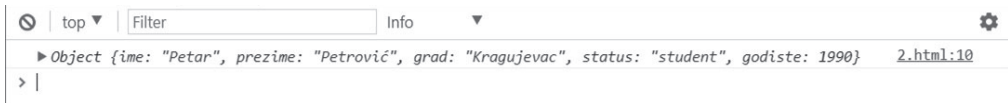
Primer kreiranja niza osoba tako je:

```
var osobaNiz= new Array("Petar", "Petrović", "Kragujevac", "student");
```

dok je objekat sa istim sadržajem:

```
<script type="text/javascript">
  var osobaObjekat = { ime:"Petar", prezime:"Petrović", grad: "Kragujevac",
    status:"student", godiste:1990};
  console.log(osobaObjekat);
</script>
```

Prilikom ispisa, sadržaj objekat se prikazuje na sledeći način:



Slika 7.31. Prikaz elemenata objekta u Console-i

Često se u primerima dobre prakse nalazi da se parovi *key:value* pišu jedan ispod drugog, samo zbog preglednosti, kao i svojstva kod CSS-a. Tako isti ovaj objekat možemo zapisati na ovaj način:

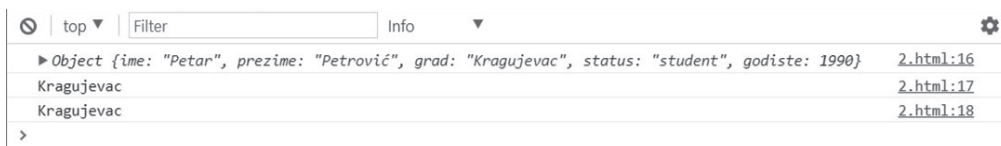
```
var osobaObjekat = {
  ime:"Petar",
  prezime:"Petrović",
  grad: "Kragujevac",
  status:"student",
  godiste:1990
};
```

Početnici često kažu da je objekat isto kao i skup više promenljivih, gde svaka ima svoje ime i svoju vrednost, i za početak se to tako može i prihvatiti. Kasnije

će se pokazati koje sve prednosti nudi rad sa objektima i videti da to prevazilazi definiciju skupa objedinjavanja podataka.

Pristup elementima objekta je dvojak: pomoću . notacije i pomoću *key*-a. Tako ako se želi pročitati grad odakle je korisnik to se može uraditi na sledeće načine:

```
<script type="text/javascript">
var osobaNiz = new Array("Petar", "Petrović", "Kragujevac", "student",
1990);
var osobaObjekat = {
  ime:"Petar",
  prezime:"Petrović",
  grad: "Kragujevac",
  status:"student",
  godiste:1990
};
      console.log(osobaObjekat);
      console.log(osobaObjekat.grad);
      console.log(osobaObjekat['grad']);
</script>
```

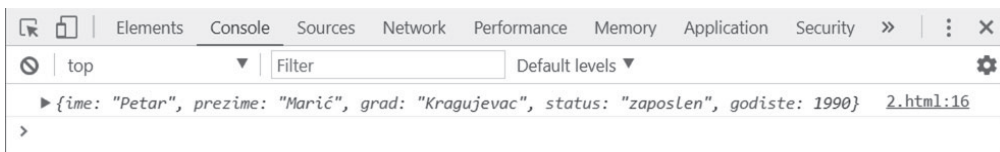


► Object {ime: "Petar", prezime: "Petrović", grad: "Kragujevac", status: "student", godiste: 1990}	2.html:16
Kragujevac	2.html:17
Kragujevac	2.html:18

Slika 7.32. Prikaz različitih načina za čitanje vrednosti svojstva objekta

Na sličan, dvojni način, može se izvršiti promena vrednosti. Tako ako želimo da promenimo *prezime* u *Marić* a *status* u *zaposlen*, to možemo uraditi pomoću:

```
<script type="text/javascript">
var osobaNiz = new Array("Petar", "Petrović", "Kragujevac", "student",
1990);
var osobaObjekat = {
  ime:"Petar",
  prezime:"Petrović",
  grad: "Kragujevac",
  status:"student",
  godiste:1990
};
osobaObjekat.prezime = "Marić";
osobaObjekat['status'] = "zaposlen";
      console.log(osobaObjekat);
</script>
```



Slika 7.33. Promena vrednosti svojstva unutar objekta

Kao što je niz mogao da se kreira na dva načina, pomoću uglastih zagrada i *new Array()*, na isti način i objekat se može kreirati pomoću ranije opisanih vitičastih zagrada ali i pomoću *new Object()*. Ako se ova druga opcija iskoristi za kreiranje istog objekta, kao u ranijem primeru, pa mu se naknadno dodeljuju vrednosti, dobija se isti rezultat, tj. kao na prethodnoj slici.

```
<script type="text/javascript">
var osobaObjekat2 = new Object();

osobaObjekat2.ime = "Petar";
osobaObjekat2.prezime = "Petrović";
osobaObjekat2.grad = "Kragujevac";
osobaObjekat2.status = "student";
osobaObjekat2.godiste = 1990;
    console.log(osobaObjekat2);
</script>
```

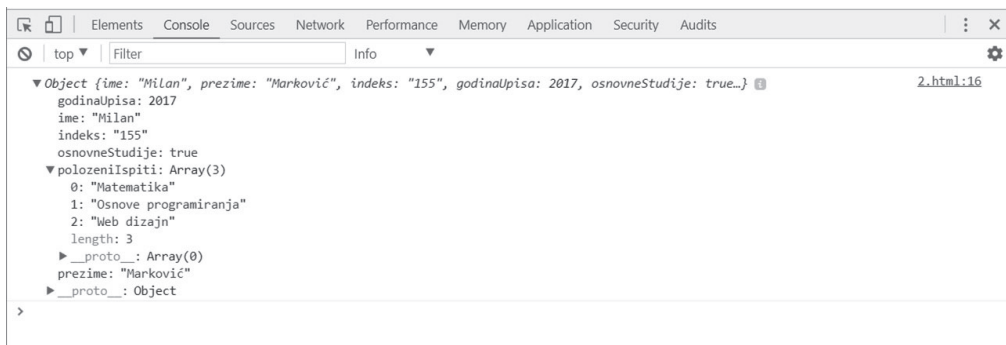
7.15. Napredna upotreba objekata

Pokazano je kako se kreira objekat, kako se objektu dodaju podaci i kako se modifikuju. Podaci u objektu su definisani kao par *key:value*. Key se najčešće naziva *svojstvom* objekta, a value *vrednošću svojstva* objekta. Vrednost svojstva objekta može biti bilo koji do sada definisani tip, uključujući i nizove i funkcije.

Tako možemo da definišemo objekat student sa sledećim tipovima podataka:

```
<script type="text/javascript">
var student = new Object();
student.ime = "Milan";
student.prezime = "Marković";
student.indeks = "155";
student.godinaUpisa = 2017;
student.osnovneStudije = true;
student.polozeniIspiti = ["Matematika", "Osnove programiranja", "Web
dizajn"];
    console.log(student);
</script>
```

Kada se sada pogleda ispis u konzoli, vidi se ceo skup podataka, ali zbog prisustva niza kao sadržaja jednog svojstva, potrebno je dodatno kliknuti na strelicu da se i ti podaci prikažu.



Slika 7.34. Dodavanje niza kao vrednosti svojstva unutar objekta

Ukoliko se želi pristup vrednosti svojstva *ime*, to se definiše kao

```
var ime = student.ime;
```

međutim ako se pristupa svojstvu *polozeniIspiti*:

```
console.log(student.polozeniIspiti);
```

dobija se baš ono što je i kreirano tj. niz. To znači da je kod *student.polozeniIspiti* direktan pristup nizu.



Slika 7.35. Pristup vrednosti svojstva objekta koji ima strukturu niza

Da bi se pročitao prvi polozeni ispit, potrebno je, kao i u slučaju klasičnog rada sa nizovima, pristupiti prvom elementu niza u sintaksi koja je korišćena kod nizova.

```
var ispit=student.polozeniIspiti[0];
console.log(ispit);
```

Ovako se dobija *Matematika*, tj. pročitao je prvi element niza *polozeniIspiti*. Slično bi se pristupilo bilo kom drugom elementu ovog niza.

Pored nizova, i funkcije mogu biti vrednosti dodeljene svojstvima objekata. Ukoliko se prethodno kreirani objekat *student* proširi dodavanjem novog svojstva *godineStudiranja*, sadržajem koji je *number* to bi se pisalo kao

```
student.godineStudiranja = 3;
```

Međutim, ako je ovaj podatak potrebno dinamički računati, a ne ručno unositi, tada se definiše funkcija kao sadržaj. U sledećem primeru, definisana je funkcija, sa ulaznim argumentom *godinaKadaJeUpisan*, koja će rezultat svog rada dodeliti svojstvu *godineStudiranja*. U telu funkcije nalazi se računanje razlike u odnosu na 2018. godinu i lokalnu promenljivu *godinaKadaJeUpisan*, što će dati broj godina studiranja u odnosu na 2018. godinu.

```
student.godineStudiranja = function(godinaKadaJeUpisan)
{
    return 2018 - godinaKadaJeUpisan;
}
var brojGodina = student.godineStudiranja(2016);
console.log(brojGodina);
```

Da bi se pristupilo novom svojstvu *godineStudiranja*, obzirom da je to sada funkcija a ne prost tip sadržaja, potrebno je samo staviti () i unutar njih proslediti vrednost koja je funkciji potrebna za rad. Na primer,

```
var brojGodina = student.godineStudiranja(2016);
```

kada se ispisuje vrednost 2. Na ovaj način bilo kakav složen kod se može definisati unutar funkcije i rezultat njegovog rada biti vrednost jednog od svojstava posmatranog objekta.

U datom primeru, definisano je da se funkcija poziva sa argumentom *godinaKadaJeUpisan*, koji je potreban za rad funkcije, i to predstavlja lokalnu promenljivu funkcije. Kako je ovo podatak koji se već nalazi u objektu, kao svojstvo *godinaUpisa*, postavlja se pitanje da li je to moguće direktno pročitati iz objekta, a ne posebno prosleđivati prilikom poziva funkcije. Odgovor je: Da. Za ove potrebe vrlo često će se koristiti promenljiva *this*, koji je pokazivač na tekući objekat. To znači da promenljiva *this* zna koji je objekat, unutar koga je pozvan i direktno ukazuje na njega, pa se direktno preko njega može pozvati bilo koje svojstvo tog objekta. U ovom slučaju *this*, unutar objekta *student*, znači *student*. Pa tako kao što i *student.ime* pristupa sadržaju *Milan*, tako i *this.ime* pristupa istom sadržaju. Tako, ako bi modifikovali funkciju na sledeći način

```
student.godineStudiranja = function()
{ return 2018 - this.godinaUpisa; }
var brojGodina = student.godineStudiranja();
console.log(brojGodina);
```

onda funkciji ne bi morao da se prosleđuje nikakav ulazni argument, jer u telu funkcije dohvatamo podatak iz objekta (*godinaUpisa*). Ako bi se sada pozvala funkcija *godineStudiranja()* koja je unutar objekta *student*, to bi se zapisalo kao

```
var brojGodina = student.godineStudiranja();
```

i dobila bi se vrednost 1.

Ovako definisane funkcije, unutar objekta se često nazivaju metodima. Kažemo da je metod funkcija koju pozivamo unutar objekta sa ciljem izvršavanja neke konkretne radnje nad objektom. Verujem da sada, ranije pomenuti metodi za rad sa nizovima postaju jasniji u smislu sintakse i logike rada.

```
niz2.sort();
niz2.shift();
niz2.push("8");
```

Do sada je opisano kako se objektu dodaje novo svojstvo i vrednost. Podsetimo se, da u slučaju da želimo da dodamo svojstvo *godine*, sa vrednoću 5, pisali bi

```
student.godine = 5;
```

Međutim, ako ovaj podatak treba da se automatski izračuna, na bazi napisanog metoda *godineStudiranja()*, ovo bi modifikovali tako da se sa desne strane jednakosti pozove metod, umesto upisivanje broja 5, kao:

```
student.godine = student.godineStudiranja();
```

Tako bi ceo kod postao:

```
<script type="text/javascript">
var student = new Object();
student.ime = "Milan";
student.prezime = "Marković";
student.indeks = "155";
student.godinaUpisa = 2017;
student.osnovneStudije = true;
student.polozeniIspiti = ["Matematika", "Osnove programiranja", "Web
dizajn"];
student.godineStudiranja = function()
{

    //alert(this.godinaUpisa);
    return 2018 - this.godinaUpisa;
}
student.godine = student.godineStudiranja();
console.log(student);
</script>
```

Ako bi se otišlo još jedan korak dalje, linija koda

```
student.godine = student.godineStudiranja();
```

bi mogla da se izbegne, jer je pozivanjem metoda, rezultat njegovog rada već u objektu, pa je dobro direktno iz metode tu vrednost upisati u pomenuto svojstvo *godine*.

To znači da bi telo metoda moglo da postane:

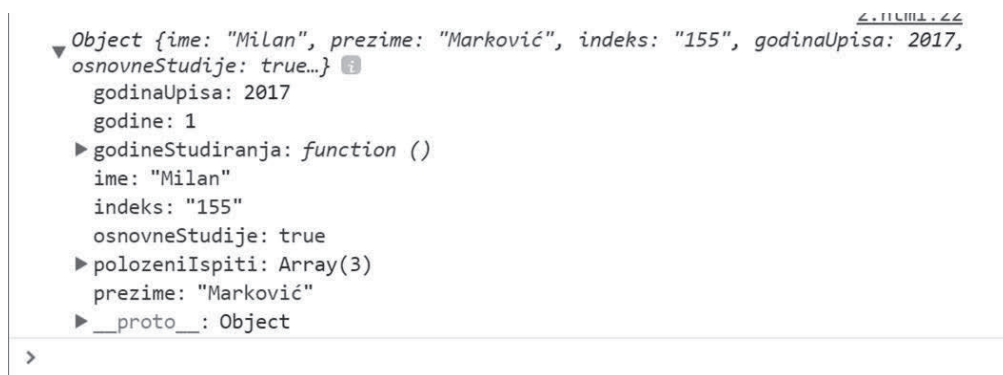
```
student.godineStudiranja = function(){
    this.godine = 2018 - this.godinaUpisa;
}
```

što znači da će se njegovim pozivanjem, u tekući objekat (*student*) i svojstvo *godine*, upisati vrednost koja se izračuna kao razlika godina.

Ostaje samo da se metod pozove, i njegovo delovanje će kreirati svojstvo *godine* sa vrednošću 1, u ovom slučaju. Konačno, kod postaje:

```
<script type="text/javascript">
var student = new Object();
student.ime = "Milan";
student.prezime = "Marković";
student.indeks = "155";
student.godinaUpisa = 2017;
student.osnovneStudije = true;
student.polozeniIspiti = ["Matematika", "Osnove programiranja", "Web
dizajn"];
student.godineStudiranja = function(){
    this.godine = 2018 - this.godinaUpisa;
}
student.godineStudiranja();
console.log(student);
</script>
```

a u konzoli se prikazuje:



Slika 7.36. Definisanje vrednosti svojstva objekta na bazi funkcije

Na ovaj način u objekte se mogu smestiti različite metode sa različitim poslovima koje treba da odrade. U zavisnosti koji od tih poslova je potreban, treba pozvati određenu metodu, a ona će rezultat svog rada ili vratiti programeru ili upisati kao svojstvo objekta u kome se nalazi.

7.16. Ispisivanje u dokumentu

JavaScript, pomoću DOM objekta `document`, može da ispisuje sadržaj unutar vidljivog dela web stranica. Ova mogućnost je podržana kroz četiri metoda: `write()`, `writeln()`, `open()` i `close()` od kojih se najčešće koristi `write()`. Upotreba ovog metoda, obzirom da pripada objektu *document* je u formi

```
document.write("argument");
```

gde `argument` podrazumeva sadržaj koji će JavaScript proslediti browser-u da ga on prikaže korisniku. Ovaj sadržaj se korisniku prikazuje na mestu na kome se nalazi linija koda `document.write()` u odnosu na prethodne ili naredne linije koda.

Tako, ako se u HTML-u ispiše jedan sadržaj, pa skriptom naredni, pa opet u HTML-u definiše neki drugi, takvim redom će se prikazati i u browser-u.

```
<html>
<head> </head>
<body>
<h1>Tekst ispisan u HTML-u</h1>
<script type="text/javascript">
    document.write("Tekst ispisan u JavaScript-u!");
</script>
<p>Drugi tekst ispisan u HTML-u</p>
</body>
</html>
```

Tekst ispisan u HTML-u

Tekst ispisan u JavaScript-u!

Drugi tekst ispisan u HTML-u

Slika 7.37. Prikaz ispisa teksta pomoću `document.write()`

Metod `write()` kao argument može imati string ili promenljivu čiji sadržaj treba ispisati u web stranici. U sledećem primeru, definisana je promenljiva *ime*, i njen sadržaj ispisan pomoću metoda `write()`.

```
<body>
<h1>Tekst ispisan u HTML-u</h1>
<script type="text/javascript">
    var ime = "Petar";
    document.write(ime);
</script>
<p>Drugi tekst ispisan u HTML-u</p>
</body>
```

Tekst ispisan u HTML-u

Petar

Drugi tekst ispisan u HTML-u

Slika 7.38. Prikaz ispisa vrednosti promenljive u web strani pomoću `document.write()`

Pored jedne promenljive, moguće je koristiti i više promenljivih sa nekim operatorom. Najčešće je to `+`, koji predstavlja konkatanaciju, nadovezivanje, stringova, ili neki drugi operator koji se može primeniti nad definisanim promenljivim. U sledećem primeru, kreirane su dve promenljive tipa string, kao i dve tipa number, i primenjen je operator `+`, unutar metoda `write()`.

```
<script type="text/javascript">
var ime = "Petar";
var prezime = "Petrović";
document.write(ime + " " + prezime);

var broj1 = 5;
var broj2 = 12;
document.write(broj1 + broj2);
</script>
```

Petar Petrović17

Slika 7.39. Prikaz ispisa rezultata matematičke operacije

JavaScript ne analizira sadržaj stringa koji prosledi browser-u na ispisivanje, tako da programeri često umesto klasičnog stringa prosleđuju HTML kod. Ovaj kod je sa aspekta JavaScript-a običan string, ali njegovim ispisivanjem u browser-u, browser prepoznaje HTML tagove i interpretira ih u skladu sa standardnim pravilima.

Tako ako prosledimo string `"<b>Petar Petrović </b>"`

```
<script type="text/javascript">
    document.write("<b>Petar Petrović </b>");
</script>
```

JavaScript će to u originalu proslediti browser-u na prikaz, ali će browser dobijanjem koda `<b>Petar Petrović </b>`, tekst **Petar Petrović** prikazati boldovano, kao da je to programer inicijalno napisao kao sastavni deo web stranice u HTML kodu.

Petar Petrović

Slika 7.40. Prikaz ispisa HTML koda pomoću `document.write()`

Koristeći konkatanaciju stringova, ovo je moglo da se napiše i kao:

```
<script type="text/javascript">
    var ime = "Petar Petrović";
    document.write("<b>" + ime + "</b>");
</script>
```

Ako se sada mogućnost metoda `write()` iskoristi za naprednije namene, npr. dinamičko pravljenje drop-down liste, zaključuje se da JavaScript može u mnogome da automatizuje ručno pisanje HTML koda.

Pretpostavimo da želimo da kreiramo listu gradova. HTML kod, drop-down liste, bi u slučaju tri grada izgledao ovako:

```
<select name="gradovi">
    <option>Beograd</option>
    <option>Kragujevac</option>
    <option>Niš</option>
</select>
```

Do sada smo ovo pisali ručno, i to nam nije predstavljalo problem. Međutim, u budućim primenama ovo će se ispisivati dinamički, što znači da će ovo pisati kod. Zašto? Razlog se krije u tome što će najveći broj podataka, u ovom slučaju gradova, biti promenljiv, u smislu da se vremenom dodaje još neki grad, i da programer ne želi da se ručno vraća u kod i dodaje novi tag *option* sa novim gradom. U realnom životu, ovi podaci se nalaze u bazi podataka, i te podatke ili unosi administrator, ili korisnici nekog sajta (npr. artikli koje prodaju, pesme koje uploaduju, komentari koje ostavljaju itd.). U svim tim situacijama želimo da se kod sam prilagodi broju podataka koje treba prikazati, što je u ovom slučaju u formi drop-down liste.

Ako se analizira HTML kod drop-down liste, zaključuje se da prva i poslednja linija koda, tj. otvaranje i zatvaranje taga `select` su vrlo jednostveni, i njih možemo da stavimo da se ispišu u posebnim metodima `write()`, na početku i kraju našeg koda. Međutim, otvoren tag *option*, *sadržaj* i zatvoren tag *option*, se ponavljaju onoliko puta koliko ima gradova. Kako smo pokazali rad sa petljama, kreiraćemo petlju koja će u svakoj iteraciji rada ispisivati jedan *option* element, i unutar njega upisati podatak iz niza.

U sledećem primeru, kreiran je niz *gradovi*. Pomoću metoda `write()` kreiran je string "<select name='gradovi'>" koji browser razume kao početak jedne drop-down liste. Sa aspekta HTML-a, sada bi trebalo napisati određeni broj tagova *option* i unutar njih upisati *sadržaj* koji će prikazati u listi. Za ovo će se iskoristiti

for-in petlja, koja u sebi ima samo jednu liniju koda, za ispis elementa *option*. Svakom iteracijom petlje, ispisaće se po jedan element *option*, sa sadržajem iz niza. Nakon petlje, potrebno je ispisati u jednoj liniji koda zatvoren tag *select*, i tako dobijamo JavaScript kodom ispisan kompletan kod HTML drop-down liste, sa podacima iz niza. Ovaj JavaScript kod bi ostao nepromenjen, bez obzira na broj elemenata u nizu tj. broj elemenata liste, što pokazuje da dinamički kreiran kod umnogome štedi i vreme i način pisanja koda, slika 7.41.

```
<script type="text/javascript">
var gradovi = new Array("Beograd", "Kragujevac", "Niš");
document.write("<select name='gradovi'>");
for(indeks in gradovi){
    document.write("<option>" + gradovi[indeks] + "</option>");
}
document.write("</select>");
</script>
```



Slika 7.41. Prikaz ispisa dinamički kreirana dropdown liste

7.17. Princip funkcionisanja JavaScript-a

JavaScript koji je namenjen za realizaciju web aplikacija, kao svoje izvršno okruženje ima ili browser (za klijentski skript) ili server (Node.js). Kako se mi sada bavimo klijentskim delom, to znači da je softver koji interpretira JavaScript korisnički browser. Kada JavaScript dođe od web servera do klijenta, pomoću *http_request*-a, browser taj kod prihvata i u prvoj fazi ga parsira. To znači, da browser poznaje svu sintaksu i pravila upotrebe ovog jezika, i u ovoj fazi ga proverava. Ova provera znači da se kontrolišu sva pravila jezika, upotrebe rezervisanih reči, promenljive, tipovi podataka, definisanja funkcija, pozivanja, upotrebe zagrada, zareza, tačka-zareza itd. Ukoliko postoji bilo koji problem, kod se ne izvršava i prijavljuje se greška. Ako su sva sintaksna pravila dobra, onda se taj kod prevodi u mašinski jezik koji je potreban računaru da bi taj kod izvršio i korisnik video neki konkretan efekat ili rezultat rada koda.

Prilikom izvršavanja koda, sistem kreira okruženje u kome se kod realizuje. To okruženje definiše oblast delovanja i prostor u kome su pojedine promenljive dostupne za pojedine delove koda. Tako, ako se kreira neki kod, bilo gde van funkcije, taj kod pripada DOM objektu *window*, i tretira se kao globalni objekat. Tako sve promenljive, tj. sada svojstva, globalnog objekta, postaju globalne promenljive, i one su dostupne svim „nižim“ elementima DOM-a tj. objektima

koji su unutar globalnog objekta *window*. Kako je ovo priča između objekta i njegovog svojstva, koja je prethodno objašnjena, definisanjem promenljive:

```
var ime = "Milan";
```

definisano je isto kao i da je kreirano svojstvo objekta *window* tj.

```
window.ime = "Milan";
```

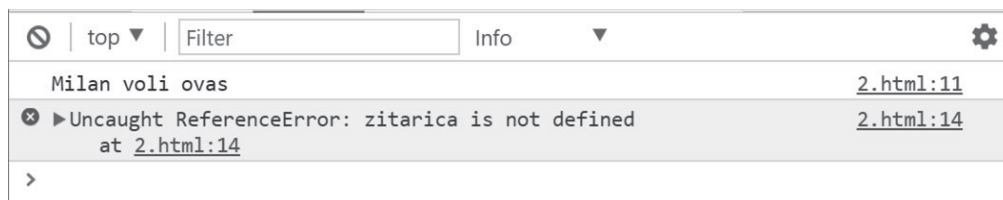
Naravno da je programerima lakše i brže da koriste prvu notaciju, ali je bitno da se razume prostor kome neka promenljiva pripada. Ovaj prostor sada je prostor globalnog objekta *window* pa je njegovo svojstvo, tj. promenljiva, globalna promenljiva i dostupna unutar celog koda.

Ukoliko je kod napisan unutar funkcije, onda se za tu funkciju pravi poseban prostor u kome se skladište njene promenljive, tako da njene promenljive nisu vidljive u globalnom prostoru, i njih nazivamo lokalnim promenljivima.

To je najbolje pokazati kroz primer. Ako se definiše promenljiva *ime*, van funkcije, tj. u globalnom objektu *window*, onda je ona globalna i dostupna unutar celog koda. Ako se definiše funkcija *primer()*, ona dobija poseban prostor za svoje izvršavanje i svoje promenljive, pa promenljiva koja je definisana unutar ove funkcije *zitarica*, postaje lokalna promenljiva, tj. dostupna samo ovoj funkciji unutar njenog okruženja. Ako se unutar funkcije ispiše sadržaj globalne i lokalne promenljive, to će biti regularno jer je globalna promenljiva *ime*, dostupna i unutar lokalnog prostora funkcije.

Međutim, ako se izvan funkcije ispisuje sadržaj obe pomenute promenljive, doći će do greške, jer će promenljiva *ime* biti dostupna, ali promenljiva *zitarica* nije dostupna u globalnom prostoru, jer je van tela funkcije u kojoj je kreirana.

```
<script type="text/javascript">
var ime = "Milan";
function primer(){
    var zitarica = "ovas";
    console.log(ime + " voli " + zitarica);
}
primer();
console.log(ime + " voli " + zitarica);
</script>
```



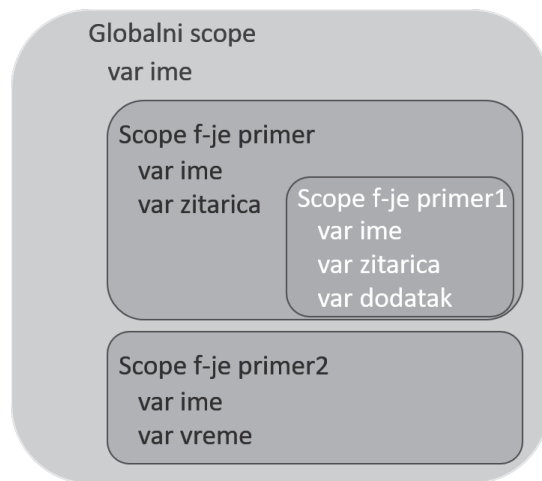
Slika 7.42. Upotreba lokalne i globalne promenljive

Stručno rečeno, termin *Scope*, ukazuje na prostor unutar koga su kreirane promenljive dostupne. U JavaScript-u samo funkcija kreira novi *Scope*, dok u nekim drugim jezicima to mogu da urade: *for*, *while*, *if*... Ukoliko funkcija, unutar svog koda, kreira novu funkciju, i ta nova funkcija kreira *Scope*. Ta nova funkcija ima roditeljsku funkciju unutar koje je kreirana, i to je vrlo bitno za *Scope*. Ugnježdavanje funkcija kreira *Scope chain*, tj. niz *Scope*-ova. Ovaj niz je jako bitan jer definiše prava pristupa promenljivima unutar niza. Ovo pravilo je jednostavno: uvek „dete“ funkcija, tj. dete *Scope* može da pristupi svojim promenljivima i promenljivima svojih roditelja, dok roditeljski *Scope* ne vidi promenljive „svoje dece“.

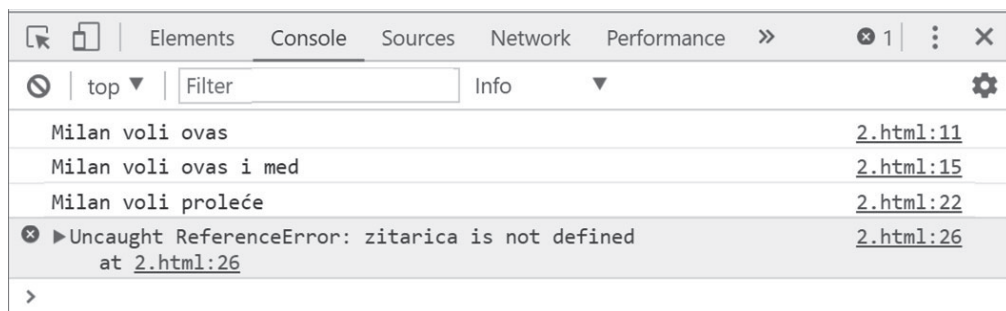
Ako pogledamo sledeći kod, za njega možemo da napravimo grafički prikaz *Scope*-ova u odnosu na kreirane funkcije i tako definišemo koja funkcija ima pristup kom *Scope*-u tj. promenljivima tog prostora.

```
<script type="text/javascript">
var ime = "Milan";
function primer(){
    var zitarica = "ovas";
    console.log(ime + " voli " + zitarica);
    primer1();
    function primer1(){
        var dodatak = "med";
        console.log(ime + " voli " + zitarica + " i " + dodatak); }
}
primer();
function primer2(){
    var vreme = "proleće";
    console.log(ime + " voli " + vreme); }
primer2();
console.log(ime + " voli " + zitarica + " u " + vreme);
</script>
```

Na slici 7.43 su grafički prikazani prostori tj. *scope*-ovi za globalni *scope* i za svaku od kreiranih funkcija, vodeći računa da ukoliko je funkcija kreirana unutar funkcije, kao u slučaju funkcije *primer1()*, ona ima roditeljsku funkciju, u ovom slučaju funkciju *primer()*. Na slici 7.43 je takođe definisano koji prostor vidi koje promenljive, pa se još jednom naglašava da globalni prostor vidi samo promenljivu *ime*, koja je u njemu i definisana, dok *scope* funkcije *primer1()*, vidi promenljivu *ime*, kao globalnu, promenljivu *zitarica*, kao promenljivu njegovog roditelja i svoju lokalnu promenljivu *dodatak*. Ovakav pristup je opis termina *Scope chain*, kojim *scope*-ovi imaju pristup promenljivima koji su definisani unutar njih i njihovih roditelja, dok obrnuto ne važi. Tako u listingu koda koji je dat, možemo da ispišemo sve promenljive sem *zitarice* u liniji 26, jer je to promenljiva koja ne pripada globalnom *scope*-u.



Slika 7.43. Šematski prikaz prostora promenljivih tj. *scope*-a



Slika 7.44. Prikaz ispisa vrednosti lokalnih promenljivih i njihova dostupnost

Najvažnije za početnike je da dobro razumeju pojam globalne i lokalne promenljive tj. pojam *scope*-a, da bi znali kada koju promenljivu mogu koristiti. Ako pođemo od toga da je ovo sada razjašnjeno, sledeći korak je objašnjenje promenljivih sa istim imenom u različitim *scope*-ovima. Ako se prethodni kod modifikuje sa jednom linijom koda

```
var ime = "Ana";
```

kojom se u funkciji *primer* dodaje nova promenljiva *ime*, postavlja se pitanje gde će se njen uticaj pojaviti. Ova dilema kod početnika nastaje jer je kreirana promenljiva sa istim imenom a drugom vrednošću. Ovo treba posmatrati jedino kroz aspekt *scope*-ova. Njen uticaj će biti samo tamo gde je njen *scope*.

```

<script type="text/javascript">
var ime = "Milan";
function primer(){
    var ime = "Ana";
    var zitarica = "ovas";
    console.log(ime + " voli " + zitarica);
  
```

```

    primer1();
    function primer1(){
        var dodatak = "med";
        console.log(ime + " voli " + zitarica + " i " + dodatak);
    }
}
primer();
function primer2(){
    var vreme = "proleće";
    console.log(ime + " voli " + vreme);
}
primer2();
</script>

```

To znači da će prilikom ispisa, unutar funkcije *primer* i *primer1*, promenljiva *ime* imati vrednost *Ana*, jer je tu oblast njenog delovanja. Ovaj prostor je dobio novu promenljivu *ime*, pomoću *var*, i vrednost *Ana*, i sada je ova nova lokalna promenljiva. Van *scope*-a u kome deluje funkcija *primer*, sadržaj *Ana* ne postoji i to je slučaj sa ispisom u funkciji *primer2*, gde se ispisuje promenljiva *ime*, koja pripada globalnom *scope*-u, sa svojim sadržajem *Milan*.

Ana voli ovas	2.html:12
Ana voli ovas i med	2.html:16
Milan voli proleće	2.html:23
>	

Slika 7.45. Prikaz ispisa vrednosti globalnih promenljivih i njihova dostupnost

Da je na istom mestu napisana ova linija koda:

```
ime = "Ana";
```

tada bi situacija bila potpuno drugačija. U ovom slučaju, nema reči *var*, pa nema kreiranja nove promenljive, nego se pristupa postojećoj promenljivoj *ime*, koja je globalna, i sada se u nju upisuje nova vrednost tj. vrednost *Ana*. Kako je promenljiva *ime* globalna, ona se nalazi u *scope*-u funkcije *primer*, i ona ima pravo da joj pristupi i modifikuje. Modifikacija se odnosi na globalnu promenljivu pa je njena promena izvršila uticaj na sva mesta gde se ova promenljiva poziva, a to je u ovom slučaju i *scope* funkcije *primer2*. Tako će sada kod ispisa biti prikazano:

Ana voli ovas	2.html:12
Ana voli ovas i med	2.html:16
Ana voli proleće	2.html:23
>	

Slika 7.46. Prikaz ispisa vrednosti promenljivih i njihova dostupnost

Na osnovu ovih primera treba uvek pažljivo analizirati *scope*-ove i to će vrlo precizno i lako rešiti dilemu ko ima pravo pristupa kojim promenljivima tj. koja promenljiva je u kom delu koda lokalna ili globalna.

7.18. Metode za rad sa stringovima

JavaScript ima veći broj metoda za rad sa stringovima, od kojih će se najbitnije obraditi u ovom poglavlju. Ove metode su definisane u samom jeziku pa programer ne mora da ih piše, nego samo da ih primeni. Ranije je opisan način da se za dati objekat, stavljanjem tačke i imena metoda, „aktivira“ taj metod. Ključna stvar je znati imena metoda, obzirom da ih mi ne pišemo nego samo koristimo. Zato će se u sledećim primerima pokazati primene nekih od metoda za rad sa stringovima, koje se najčešće koriste u praksi.

Konverzija u velika ili mala slova: Za konverziju koristimo dva metoda *toUpperCase()* i *toLowerCase()*. Sama imena metoda ukazuju da će konverzija stringa u velika slova biti pomoću metoda *toUpperCase()* dok će u mala biti pomoću *toLowerCase()*. Ovi metodi mogu se primeniti na promenljivoj koja je ranije definisana u kodu (*ime*) ili direktno nad stringom (*MIKA*).

```
<script type="text/javascript">
var ime = "Pera";
    var x = "MIKA".toLowerCase() ;
    var y = ime.toLowerCase();
    var z = ime.toUpperCase();
console.log(x); // ispisuje mika
console.log(y); //ispisuje pera
console.log(z); // ispisuje PERA
</script>
```

Određivanje dužine stringa: Svojstvo *length* se koristi za određivanje dužine stringa. Dužina je realan broj karaktera koji se u stringu nalazi, pa se broje počev od jedan. Tako će u sledećem kodu dužina stringa *Pera* biti 4 a *Mika Mikić* 10, jer se broji i *space* kao karakter u stringu.

```
<script type="text/javascript">
var a="Pera";
var b="Mika Mikić";
    var x = a.length ;
    var y = b.length ;
console.log(x);
console.log(y);
</script>
```

Određivanje karaktera unutar stringa na osnovu pozicije: Ukoliko se želi saznati tj. dohvatiti karakter koji se nalazi na određenoj poziciji u stringu, koristi se metod *charAt()*, što treba da ukaže na termin „karakter na poziciji...“. Elementi niza se numerišu počev od vrednosti 0, tako da se vizuelno string *Pera Perić* može predstaviti kao:

P	e	r	a		P	e	r	i	ć
0	1	2	3	4	5	6	7	8	9

pa kažemo da se npr. na poziciji 5 nalazi karakter *P*. U kodu se to zapisuje:

```
<script type="text/javascript">
var a="Pera Perić";
    var x = a.charAt(2);
    var y = a.charAt(9);
console.log(x); // ispisuje r
console.log(y); // ispisuje ć
</script>
```

Određivanje pozicije datog stringa u većem stringu: Kod metoda *charAt()* kod na bazi pozicije pronađe karakter, dok kada je potrebno obrnuto tj. da se na bazi karaktera pronađe njegova pozicija, koristi se *indexOf()*. Ovaj metod će pronaći prvo pojavljivanje datog stringa, unutar većeg stringa, i vratiti poziciju prvog karaktera za traženi string. Ukoliko traženog stringa ili karaktera nema vratiće se vrednost -1. U datom kodu, karakter *z* ne postoji i zato će se ispisati -1. Karakter *e* se prvi put pojavljuje na poziciji 1, dok se string *ra*, prvi put pojavljuje u reči *Pera* i to počev od druge pozicije.

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
    var x = a.indexOf("z");
    var y = a.indexOf("e");
    var z = a.indexOf("ra");
console.log(x); // ispisuje -1
console.log(y); // ispisuje 1
console.log(z); // ispisuje 2
</script>
```

Metod *indexOf()* traži poziciju počev od leve ka desnoj strani. Metod *lastIndexOf()* radi pretragu na isti način, ali počev od kraja stringa pa prema početku. Tako će za isti početni string i karakter *e*, biti pronađeno *e* u reči *ide*, pa će prikazati poziciju 7.

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
var y = a.indexOf("e");
console.log(y); </script>
```

I metod *indexOf()* i *lastIndexOf()* pronalaskom prvog traženog karaktera ili stringa prekidaju svoj rad i ispisuju rezultat. Ukoliko se ne želi ograničiti samo na prvo tj. poslednje pojavljivanje stringa u reči, moguće je, u oba metoda, staviti drugi argument koji će definisati od koje pozicije se kreće u traženje zadanog stringa. Tako, ako se želi traženje pozicije karaktera *a*, ali počev od petog karaktera pa na desno, ili karaktera *o* počev od 19-og, pa na levo, to zapisujemo kao:

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
var x = a.indexOf("a", 5);
var y = a.lastIndexOf("o", 19);
console.log(x); //ispisuje se 10
console.log(y); //ispisuje se 18
</script>
```

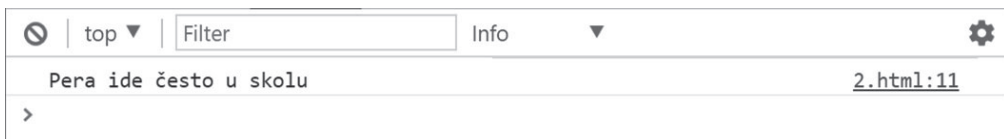
Isecanje dela stringa iz većeg stringa: U programiranju se često koriste metodi koji „isecaju“ deo stringa. JavaScript ima tri metoda za ove potrebe: *substring*, *slice* i *substr*. Metodi *substring* i *slice* su skoro identični, dok se razlikuju od metoda *substr*. Svi ovi metodi imaju dva argumenta, ali se njihova značenje razlikuje u zavisnosti od metoda, pa treba biti vrlo obazriv.

Ukoliko se želi isecanje grupe karaktera od pozicije *x* do pozicije *y* koristimo metod *substring(x,y)*. Za isecanje grupe karaktera, počev od pozicije *x* a u dužini od *y* karaktera, koristimo metod *substr(x,y)*. Oba metoda isecanje rade po principu matematičkih intervala definisanih zagradama [). To znači da u konačni skup ulazi karakter sa prve pozicije, ali ne i poslednji. Tako u sledećem primeru, za *substring(5,7)* ulaze karakteri na poziciji 5 i 6, ali ne i na poziciji 7.

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
var x = a.substring(5,7);
var y = a.substr(5,2);
console.log(x); // ispisuje id
console.log(y); // ispisuje id
</script>
```

Zamena stringa sa drugim stringom: Zamene stringa se realizuje metodom *replace()*. Ovaj metod ima dva argumenta, prvi koji definiše šta treba pronaći, i drugi koji definiše čime pronađeno treba zameniti.

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
var x = a.replace("rado", "često");
console.log(x);
</script>
```



Slika 7.47. Upotreba metoda `replace()`

Podela stringa u formu niza: Konverzija stringa u niz je vrlo česta u programiranju i realizuje se metodom `split()`. Ovaj metod originalni string „secka“ u odnosu na neki definisani separator, pri čemu se separator „gubi“ a svi dobijeni delići stringa se smeštaju u posebne elemente niza. Tako ako se posmatra string

```
var proba= "Pera ide rado u skolu";
```

i kao separator stavi space, tj.

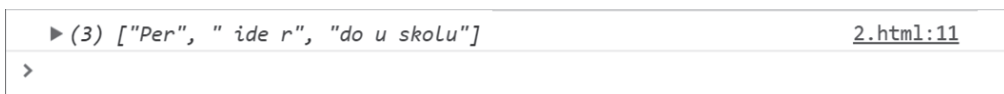
```
proba.split(" ");
```

dobiće se niz u čijim elementima su sledeće reči

Pera	ide	rado	u	školu
0	1	2	3	4

Ukoliko se kao separator stavi npr. karakter `a`, dobija se:

```
<script type="text/javascript">
var a = "Pera ide rado u skolu";
var x = a.split("a");
console.log(x);
</script>
```



Slika 7.48. Upotreba metoda `split()`

U realnim situacijama, kod metoda `split()`, se koriste separatori koji su namenski kreirani i postavljeni primenom drugih metoda koji su stringove spajali u jedan string, pa se u odnosu na tako izabrani separator vrši „cepanje“ da bi se ponovo dobili originalni delovi. Iz tog razloga to obično budu neki karakteristični simboli ili grupe karaktera, koje se neće naći kao originalni sadržaj stringa. Npr. u radu sa tekstualnim fajlovima tako se koristi separator `\r` ili `\n`.

7.19. Metode za rad sa brojevima

Kao i kod rada sa stringovima, JavaScript ima veći broj metoda za rad sa brojevima. Najčešće se koriste karakteristične matematičke funkcije ili funkcije koje su pogodne za obradu u računarskom sistemu. Zato se često koristi objekat *Math*, i njemu pripadajući metodi, za lako izračunavanje različitih podataka. U sledećim primerima navedene su neke od najkorišćenijih metoda, sa kratkim opisom šta rade i kako izgleda praktična primena tih metoda.

Generisanje slučajnog broja: Metod *random()* služi za slučajno generisanje broja u intervalu [0-1).

```
<script type="text/javascript">
var a = Math.random();
console.log(a); // npr. 0.6351390991512158
</script>
```

Određivanje maksimalne i minimalne vrednosti skupa: Metode *min()* i *max()* pronalaze minimalnu i maksimalnu vrednost u skupu argumenata.

```
<script type="text/javascript">
var x = Math.min(0, 97, 8, 34, 44, -14, -1);
var y = Math.max(0, 97, 8, 34, 44, -14, -1);
console.log(x); // ispisuje -14
console.log(y); // ispisuje 97
</script>
```

Zaokruživanje decimalnog broja na najbliži ceo broj: Ovo se realizuje metodom *round()*:

```
<script type="text/javascript">
var x = Math.round(4.7);
var y = Math.round(4.4);
console.log(x); // ispisuje 5
console.log(y); // ispisuje 4
</script>
```

Zaokruživanje decimalnog broja na prvi viši ceo broj: Ovo se realizuje metodom *ceil()*:

```
<script type="text/javascript">
var x = Math.ceil(4.7);
console.log(x); // ispisuje 5
</script>
```

Zaokruživanje decimalnog broja na prvi niži ceo broj: Ovo se realizuje metodom *floor()*:

```
<script type="text/javascript">
var x = Math.floor(4.7);
console.log(x); // ispisuje 4
</script>
```

Primenom ovih metoda može se dobiti veliki spektar realno potrebnih funkcionalnosti. Npr, za generisanje slučajnog broja u intervalu [1-10] može se primeniti:

```
<script type="text/javascript">
var x = Math.floor(Math.random() * 11);
console.log(x); // ispisuje broj u intervalu od [0-10]
</script>
```

7.20. Metode za rad sa datumom i vremenom

Svaki programski jezik ima metode koje omogućavaju rad sa datumom i vremenom. Klijentski jezici, podatak o tekućem datumu i vremenu dobijaju sa klijentskog računara, dok serverski jezici od web servera. Pogrešno setovano sistemsko vreme na klijentskom računaru uzrokuje rad sa takvim pogrešnim podacima, pa je sigurnije koristiti serversko vreme. Sa druge strane, serversko vreme dolazi od web servera, koji može biti bilo gde u svetu, pa i u drugoj vremenskoj zoni, pa ni ono nije uvek pogodno za pojedine namene. Realno, koriste se oba vremena, sa ciljem da sve što može da se odradi kod klijenta tu bude odradeno i da se server što manje opterećuje. Kako JavaScript, kao klijentski jezik, može da radi isključivo sa klijentskim sistemskim datumom i vremenom, sada ćemo se fokusirati na metode koje to omogućavaju.

U JavaScript-u postoji mogućnost dohvaćanja informacije o svim detaljima vezanim za sistemski datum i vreme (dan, sat, minut, godina, mesec, ime dana, vremenska zona, PM/AM, itd.). Rad sa ovim podacima može se posmatrati kroz dva pravca delovanja: njihovo dohvaćanje (*get*) i postavljanje (*set*).

Pravljenjem objekta klase, pomoću

```
new Date();
```

dohvata se celokupan zapis, sa svim detaljima o sistemskom datumu i vremenu.

Smeštanjem svih ovih podataka u lokalnu promenljivu, i primenom posebnih metoda, izdvajaju se pojedinačne informacije od značaja (broj sati, dan, godina, broj minuta, itd.).

Na primer, koliko je trenutno sati dobijamo pomoću:

```
var datum = new Date();
var sat = datum.getHours();
```

Kako ovo funkcioniše. Kreiranjem objekta `datum`, u njemu se nalaze sve informacije o trenutnom datumu i vremenu dobijene od klijentskog sistemskog vremena. Primenom metode `getHours()`, iz svih podataka dobija se samo informacija o broju sati u tom trenutku, i ona se smešta u promenljivu `sat`. Za dobijanje svih drugih dostupnih podataka koriste se druge metode koje imaju vrlo intuitivna imena i svojim imenom logično ukazuju na ono što rade.

Tako za rad sa vremenom najčešće koristimo:

<code>getHours()</code>	Vraća broj sati (0-23)
<code>getMinutes()</code>	Vraća broj minuta (0-59)
<code>getSeconds()</code>	Vraća broj sekundi (0-59)
<code>getMilliseconds()</code>	Vraća broj milisekundi (0-999)
<code>now()</code>	Vraća broj milisekundi počev od Januara 1970. do sada
<code>getTime()</code>	Vraća broj milisekundi od Januara 1970. do zadatog datuma

Za rad sa datumima i danima, najčešće koristimo:

<code>getDate()</code>	Vraća broj dana u mesecu (1-31)
<code>getMonth()</code>	Vraća broj meseci (0-11)
<code>getFullYear()</code>	Vraća tekuću godinu
<code>getDay()</code>	Vraća broj dan u nedelji (0-6)

Posebnu pažnju treba obratiti na broj meseci u godini, jer se vraća broj u intervalu 0-11, što je drugačije od naše svakodnevne komunikacije koja je bazirana na intervalu 1-12, pa dobijenu vrednost treba uvećati za jedan da bi se dobila naša notacija meseca. Sa druge strane, broj dana u nedelji je u intervalu 0-6, gde 0 ukazuje na nedelju, 1 na ponedeljak dok 6 ukazuje na subotu.

Za skoro sve definisane metode postoje i njihovi “dvojnici” za dobijanje informacija u skladu sa tzv. *Universal Time (iniverzalno kompjutersko vreme)*. Imena ovih metoda imaju UTC između reči `get` i onoga što se traži, pa tako imamo: `getUTCDate()`, `getUTCDay()`, `getUTCHours()` itd.

Ako pretpostavimo da se naredni kod pokreće u nedelju 6.8.2017. u 19:21:05 tada imamo sledeći rezultat:

```
<script type="text/javascript">
var datum = new Date();

var datum_dan = datum.getDate();
document.write("Sada je datum: " + datum_dan + "<br/>");
```

```

var dan = datum.getDay();
document.write("Sada je redni broj dana u nedelji: " + dan + "<br/>");

var mesec = datum.getMonth();
document.write("Sada je redni broj meseca: " + mesec + "<br/>");

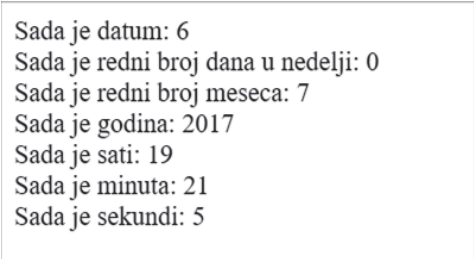
var godina = datum.getFullYear();
document.write("Sada je godina: " + godina + "<br/>");

var sat = datum.getHours();
document.write("Sada je sati: " + sat + "<br/>");

var minut = datum.getMinutes();
document.write("Sada je minuta: " + minut + "<br/>");

var sekund = datum.getSeconds();
document.write("Sada je sekundi: " + sekund + "<br/>");
</script>

```



```

Sada je datum: 6
Sada je redni broj dana u nedelji: 0
Sada je redni broj meseca: 7
Sada je godina: 2017
Sada je sati: 19
Sada je minuta: 21
Sada je sekundi: 5

```

Slika 7.49. Rad sa datumom i vremenom u JavaScript-u

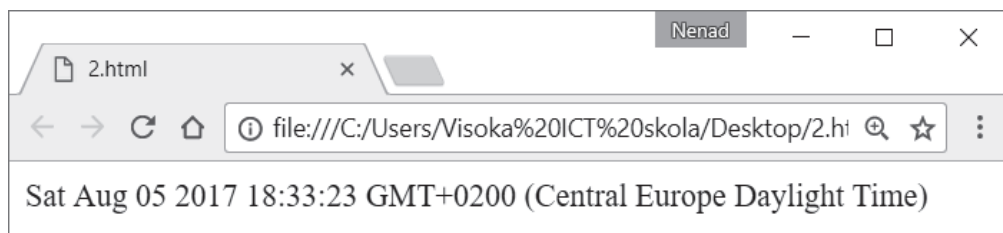
Prefiks *get* u svim navedenim metodama ukazuje da se radi o dohvatanju podataka sa klijentskog računara, onakvim kako se kod klijenta nalaze. Za skoro sve navedene metode može se koristiti prefiks *set* umesto *get* sa ciljem da se vreme koje je dobijeno promeni na željenu vrednost tj. setuje na željenu vrednost koja je na nivou koda. Tako imamo: *setDate()*, *setMonth()*, *setFullYear()*, *setHours()*, *setMinutes()*, *setSeconds()*, *setTime()*, *setYear()*, *setUTCHours()*, *setUTCMinutes()*, *setUTCMonth()*, *setUTCFullYear()*...

Ako pretpostavimo da se dati kod pokreće u nedelju 6.8.2017. u 19:33:23 tada imamo sledeći rezultat:

```

<script type="text/javascript">
    var datum = new Date();
    datum.setDate(5);
    datum.setHours(18);
    document.write(datum);
</script>

```



Slika 7.50. Prikaz mogućnosti setovanja pojedinih vrednosti u radu sa datumom i vremenom

U ovom poglavlju su prikazana osnovna pravila pisanja JavaScript-a, elementi jezika i njihova implementacija. U narednim poglavljima će se izučavati specifičnosti jezika u kontekstu primene u web sajtovima. Bez obzira na neki od viših nivoa primene ovog jezika, opšta pravila ostaju da važe za svaku njihovu implementaciju, pa su posebno bitna da se dobro razumeju i usvoje na pravi način.

VIII UPRAVLJANJE DOM ELEMENTIMA

Interakcija JS-a sa web stranom

PRETHODNO poglavlje obrađuje bazna pravila rada sa JavaScript-om koja će se koristiti u svim kasnijim primenama. Ključna uloga svakog skript jezika je da upravlja tj. manipuliše elementima web strane. To znači da se skript jezikom može promeniti, kreirati ili izbrisati, tj. poništiti uticaj, bilo čega što je inicijalno napisano u HTML i CSS kodu.

Upravljanje elementima web strane, vrši se pomoću DOM-a. Svakom elementu web strane, pomoću DOM-a, može se pristupiti JavaScript-om i dalje sa njim postupati kao sa objektom. Kao i svaki drugi objekat, i ovako dobijeni objekat može promeniti svoja stanja, tj. svojstva, primenom odgovarajućih metoda. Sa druge strane, ako u HTML kodu napišemo

```

```

i ovom HTML elementu pristupimo preko DOM-a, svi atributi HTML-a (*src*, *alt*, *id*, *align*, ...) postaju svojstva tako dobijenog objekta, pa se promenom vrednosti svojstva realno utiče na promenu sadržaja atributa, čime se kompletno može uticati na sve što je web dizajner napisao u HTML kodu. Isto tako, ako za dati HTML element imamo definisan CSS stil kao

```
#slika{ width:30%;
        margin:20px;
        padding:5px;
        display: block; }
```

i pristupimo preko DOM-a CSS-u, sva svojstva CSS-a (*width*, *margin*, *padding*, *display*,...) postaju svojstva dobijenog objekta i onda se promenom njihove vrednosti utiče na promenu vrednosti svojstva u CSS-u i na promenu načina prikaza u browser-u. Na ovaj način JavaScript, nakon učitavanja inicijalno napisanog HTML i CSS koda, može promeniti skoro sve što se u tom kodu nalazi i pored toga kreirati potpuno nove sadržaje, blokove, svojstva i sl. čime u velikoj meri može da promeni inicijalni sadržaj. Iz tog razloga kažemo da je JavaScript nadređen HTML-u i CSS-u.

8.1. Metod `getElementById()`

Jedan od najkorišćenijih metoda za manipulaciju elementima DOM-a je `getElementById()`. Ovaj metod se primenjuje nad objektom *document*, što praktično znači nad celom web stranom. Ovaj metod pronalazi element HTML-a koji se nalazi kao argument metoda `getElementById()` po njegovom atributu *id*. Tako ako u HTML imamo tag *div* čiji je atribut *id*= "*primer*"

```
<div id="primer" width="1000px">
    <!-- Neki HTML kod -->
</div>
```

pomoću metoda `getElementById("primer")` direktno pristupamo ovom delu HTML koda i on u JavaScript-u postaje objekat. Ukoliko se ne pronađe element sa takvim *id*-jem vratiće se *null*. Ovaj metod pronalazi prvi po redosledu pisanja element sa ovim ID-jem. U praksi, ovaj objekat smeštamo u promenljivu koju dalje pozivamo kada želimo da uradimo neku akciju nad ovim objektom.

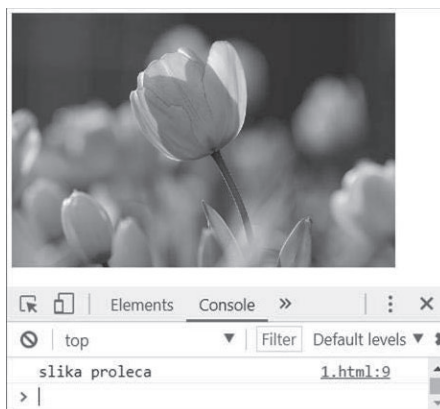
```
var elementPrimer = document.getElementById("primer");
```

Nakon pristupa elementu, najbitnije je znati koje metode i svojstva su programeru na raspolaganju da bi pomoću njih preduzeo neku konkretnu akciju. Primena odgovarajućeg svojstva može biti sa aspekta čitanja njegove vrednosti ili postavljanja njegove nove vrednosti. Posmatrajmo sledeći kod i sliku *prolece.jpg*.

```

<html>
<head> </head>
<body>
<div id="primer" width="1000px">
    
</div>
<script type="text/javascript">
    var elementSlika = document.getElementById("slika").alt;
    console.log(elementSlika);
</script>
</body>
</html>

```



Slika 8.1. Dohvatanje i prikaz vrednosti *alt* atributa elementa *img*

HTML kodom je učitana slika proleće u element *div*. JavaScript-om je pomoću

```
document.getElementById("slika")
```

pristupljeno elementu *img*, i sada je to postalo objekat. Jedno od svojstava ovog objekta je i HTML atribut *alt*. Ukoliko objektu traži ovo svojstvo, kod će vratiti njegovu vrednost, na isti način kao kada smo objašnjavajući rad sa objektima iskoristili *student.ime*. U ovom slučaju to zapisujemo kao:

```
document.getElementById("slika").alt;
```

Na ovaj način, kod će vratiti vrednost ovog svojstva, tj. u ovom slučaju tekst *slika proleca*.

Ukoliko se vrednost ovog svojstva želi promeniti tada se pomoću znaka `=` definiše nova vrednost:

```
document.getElementById("slika").alt = "nova slika";
```

Ako bi se nakon ove linije koda ponovila linija koda:

```
document.getElementById("slika").alt;
```

sada bi se dobila vrednost *nova slika*. Na ova dva načina se dohvata ili postavlja vrednost svojstva, što je za programera, u ovom slučaju, promena sadržaja vrednosti HTML atributa *alt*. Ovu promenu uradio je JavaScript, nakon dobijanja originalnog koda od web servera, i ona je vidljiva od trenutka kada browser primeni napisani JavaScript kod, na klijentskom računaru. Veći vizuelni uticaj za korisnika bi bila promena slike. Na sličan način inicijalna slika tj. vrednost atributa *src* bi mogla da se promeni na sledeći način:

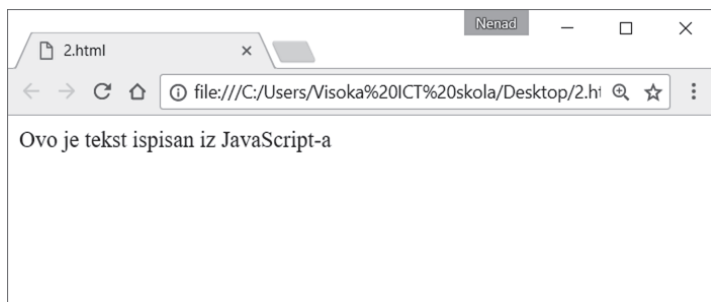
```
document.getElementById("slika").src = "slika2.jpg";
```

Ova promena bi se videla neposredno nakon izvršavanja JavaScript koda, što bi u ovom slučaju bilo skoro trenutno i korisnik ne bi ni mogao da detektuje inicijalnu sliku definisanu HTML kodom, sem slike koju je promenio JavaScript.

Na ovaj način se pomoću svojstava DOM objekta utiče na promenu inicijalnog prikaza koda u browser-u. Drugi način je primena specifičnih svojstava ili metoda, koje nisu atributi HTML-a.

Primenom svojstva *textContent*, selektovani DOM element dobija novi sadržaj koji je tipa tekst. Sadržaj elementa je tekst koji se nalazi između otvorenog i zatvorenog HTML taga, što je u sledećem primeru element *div*. Iako element *div* ima inicijalni sadržaj sliku, JavaScript će taj sadržaj „pregaziti“ novim koji je tekst Ovo je tekst ispisan iz JavaScript-a, koji je jedino što korisnik i vidi u browser-u, nakon učitavanja web strane.

```
<html>
<head></head>
<body>
<div id="primer" width="1000px">
    
</div>
<script type="text/javascript">
    document.getElementById("primer").textContent = "Ovo je tekst
ispisan iz JavaScript-a";
</script>
</body>
</html>
```



Slika 8.2. Upotreba svojstva *textContent* za definisanje sadržaja elementa sa id="primer"

Svojstvo *textContent* ima mogućnost da postavi samo tekst kao sadržaj, dok svojstvo *innerHTML* ima mogućnost da se pored teksta definiše i bilo koji HTML kod. Ovo svojstvo je jedno od vrlo bitnih i često korišćenih svojstava jer time što omogućava da se kreira HTML kod, skoro da se sve dalje može raditi iz JavaScript-a, bez inicijalnog sadržaja u HTML kodu. Ali idemo redom. Primena svojstva *innerHTML* je skoro ista kao i *textContent*. Na selektovani DOM element, treba primeniti svojstva i dati mu vrednost iz znaka jednakosti. Ako se na prethodnom primeru samo umesto svojstva *textContent* stavi *innerHTML*, dobija se isti rezultat kao na prethodnoj slici, jer *innerHTML* može da uradi isto što i *textContent*, ali može i dodatni sadržaj da definiše.

Ako sada u istom primeru promenimo samo liniju koda gde je *innerHTML* i u sadržaj pored teksta stavimo i HTML tag *b*, tj.

```
document.getElementById("primer").innerHTML = "<b>Ovo je tekst  
ispisan iz JavaScript-a </b>";
```

dobijamo prikan boldovan tekst:

Ovo je tekst ispisan iz JavaScript-a

Slika 8.3. Upotreba svojstva *innerHTML* za definisanje sadržaja elementa sa *id="primer"*

Ako isti sadržaj dalje proširimo sa novim tagovima kao

```
<script type="text/javascript">  
document.getElementById("primer").innerHTML = "<b>Ovo je tekst  
ispisan iz JavaScript-a </b> <h1> Ovo je tabela </h1> <div id='proba'> <p  
class='prikaz'> Neki tekst u tagu </p></div>";  
</script>
```

Dobijamo prikaz kao da je ovaj HTML kod inicijalno ispisan u kodu web strane. Sa aspekta browser-a sasvim je svejedno da li je to inicijalni i naknadno kreirani HTML kod, što programeru daje veliki prostor za manipulaciju sadržajem strane.

Ovo je tekst ispisan iz JavaScript-a

Ovo je tabela

Neki tekst u tagu

Slika 8.4. Upotreba svojstva *innerHTML* za ispis više elemenata sa pripadajućim CSS klasama

U ovom primeru stavljeni su i atributi *id*, kod taga *div*, i *class* kod taga *p*. Njihov efekat sada ne postoji jer ne postoji nikakav inicijalno definisan CSS kod, koji bi

se na ove elemente primenio. Međutim, ako se definiše CSS kod, kao eksterni kod web strane, sa sadržajem:

```
body{ background-color:yellow; }
#proba{width:500px;
        height:300px;
        background-color:green; }
.prikaz{ color:white;
        font-size:30px;
        text-align:center; }
```

Dobija se izgled browser-a kao na slici 8.5.



Slika 8.5. Prikaz web stranice nakon dodavanja HTML koda pomoću svojstva `innerHTML`

Ovim se pokazuje da u CSS kodu, koji je učitao na početku strane, i u kome se inicijalno nalaze stilovi za HTML elemente kojih nema u tom trenutku u HTML kodu (`<div id='proba'>` i `<p class='prikaz'>`), ne pravi problem i da „čeka“ da se ti elementi pojave (u ovom slučaju delovanjem JavaScript-a). Kada se ti elementi pojave kao deo web strane, browser potpuno regularno realizuje i dobijeni HTML kod i za njega napisani CSS kod, na isti način kao i da je sav ovaj kod bio sastavni deo inicijalne web stranice. Ovakvim mogućnostima, programer inicijalno definiše sav HTML kod, koji će se u svim kasnijim fazama dodati ili promeniti uticajem JavaScript-a, i za njega unapred definiše i napiše sav potreban CSS kod, čime se stiče utisak da svaka nova promena u web strani deluje potpuno profesionalno i sa punim grafičkim identitetom.

JavaScript je u mogućnosti da naknadno kreira ili modifikuje i CSS kod kao i HTML kod, ali će o tome biti reči kasnije.

8.2. Metod `getElementsByClassName()`

Prethodno opisani metod `getElementById` je jedan od najbržih u kontekstu realizacije jer je vrednost atributa `id` jedinstvena i brzo se pronalazi. Međutim,

kao što je moguće element pronaći po vrednosti atributa *id*, moguće je i vrednosti atributa *class*. Ovo se realizuje pomoću metoda *getElementsByClassName()*, koji vraća sve elemente koji imaju definisanu vrednost atributa *class*. Nakon selekcije DOM elementa, nad njim se dalje primenjuje bilo koje svojstvo JavaScript-a i tako vrši uticaj na element.

U sledećem primeru definisana su elementi *p* sa istom klasom, *pasus*, i prvo su nakon selekcije JavaScript-om prebrojani, svojstvo *length*, a nakon toga drugom i trećem je promenjen inicijalni sadržaj. Kako sada imamo više elemenata koji se vraćaju kao odgovor JavaScript-a, oni kreiraju strukturu niza, i svakom se ponaosob pristupa kao elementu niza (npr. *document.getElementsByClassName("pasus")[2]*)

```
<html>
<head> </head>
<body>
<div id="primer" width="1000px">
    <p class="pasus">Ovo je prvi pasus</p>
    <p class="pasus">Ovo je drugi pasus</p>
    <p class="pasus">Ovo je treći pasus</p>
</div>
<script type="text/javascript">
    alert(document.getElementsByClassName("pasus").length);
    document.getElementsByClassName("pasus")[1].innerHTML =
"Morate se ulogovati da bi videli ovaj sadržaj";
    document.getElementsByClassName("pasus")[2].innerHTML =
"Morate se ulogovati da bi videli ovaj sadržaj";
</script>
</body>
</html>
```

Ovo je prvi pasus

Morate se ulogovati da bi videli ovaj sadržaj

Morate se ulogovati da bi videli ovaj sadržaj

Slika 8.6. Prikaz selekcije elemenata DOM-a pomoću metoda *getElementsByClassName()*

8.3. Metod *getElementsByTagName()*

Kao i u slučaju metoda *getElementsByClassName()*, JavaScript metod kojim se mogu selektovati svi elementi DOM-a po imenu taga je *getElementsByTagName()*. Ovaj metod takođe vraća sve elemente sa definisanim imenom taga u formi niza. Ako se ovaj metod primeni na logiku prethodnog koda, tada bi on izgledao ovako:

```

<html>
<head>
</head>
<body>
<div id="primer" width="1000px">
    <p class="pasus">Ovo je prvi pasus</p>
    <p class="pasus">Ovo je drugi pasus</p>
    <p class="pasus">Ovo je treći pasus</p>
</div>
<script type="text/javascript">
    alert(document.getElementsByTagName("p").length);
    document.getElementsByTagName("p")[1].innerHTML =
"Morate se ulogovati da bi videli ovaj sadržaj";
</script>
</body>
</html>

```

8.4. Metod `querySelector()`

Do sada su objašnjeni metodi koji su po striktnom imenu taga ili atributa selektovali određeni DOM element. Metod `querySelector()` se može posmatrati kao objedinjena varijanta metoda `getElementById()` i `getElementsByClassName()`. Stručno, kažemo da metod `querySelector()` kao argument, koristi CSS selektor elementa. Obzirom da u imenu metoda nije definisano da li se radi sa selektorom *id* ili *class*, u atributu se mora navesti # ili . kao prefiks i tako definisati da li je reč o atributu *id* ili *class*. Metod `querySelector()` vraća samo jedan element i to prvi element sa definisanim CSS selektorom.

U narednom primeru pozvan je metod `querySelector(".pasus")`, da promeni sadržaj elementa *p*, koji ima definisan atribut *class*, kao i `document.querySelector("#naslov")`, da promeni sadržaj elementa *h2* koji ima definisan atribut *id*.

```

<html>
<head> </head>
<body>
<div id="primer" width="1000px">
    <p class="pasus">Ovo je prvi pasus</p>
    <p class="pasus">Ovo je drugi pasus</p>
    <h2 id="naslov"></h2>
</div>
<script type="text/javascript">
    document.querySelector(".pasus").innerHTML = "Ovo je definisano
metodom querySelector()";

```

```
document.querySelector("#naslov").innerHTML = "Ovo je naslov
definisan metodom querySelector()";
</script>
</body>
</html>
```

Ovo je definisano metodom querySelector()

Ovo je drugi pasus

Ovo je naslov definisan metodom querySelector()

Slika 8.7. Prikaz selekcije elemenata DOM-a pomoću metoda querySelector()

Upotreba ovog metoda nije ograničena samo na upotrebu „osnovnih“ CSS selektora kao atributa. Dozvoljena je upotreba svih selektora koji se mogu koristiti u CSS-u. Tako se u sledećem primeru koristi selektor `document.querySelector("#primer > .pasus")`, pa se neće delovati na sve `p` elemente sa atributom `class="pasus"`, ili selektor za element `h2` kao `document.querySelector("h2")`.

```
<html>
<head> </head>
<body>
<div id="primer" width="1000px">
    <p class="pasus">Ovo je prvi pasus</p>
    <h2 id="naslov"></h2>
</div>
    <p class="pasus">Ovo je drugi pasus</p>
<script type="text/javascript">
    document.querySelector("#primer > .pasus").innerHTML = "Ovo je
definisano metodom querySelector()";
    document.querySelector("h2").innerHTML = "Ovo je naslov definisan
metodom querySelector()";
</script>
</body>
</html>
```

Ovo je definisano metodom querySelector()

Ovo je naslov definisan metodom querySelector()

Ovo je drugi pasus

Slika 8.8. Prikaz složenije selekcije elemenata DOM-a pomoću metoda querySelector()

8.5. Metod `querySelectorAll()`

Metod `querySelector()` vraća samo prvi element za definisani selektor, dok metod `querySelectorAll()` vraća sve elemente za definisani selektor. Njegova upotreba je identična upotrebi metoda `querySelector()` sem što se pojedinačnim elementima mora pristupati kao elementima niza, jer ih može biti više od jednog.

```
<html>
<head> </head>
<body>
<div id="primer" width="1000px">
  <p class="pasus">Ovo je prvi pasus</p>
  <h2 id="naslov"></h2>
</div>
  <p class="pasus">Ovo je drugi pasus</p>
<script type="text/javascript">
  document.querySelectorAll(".pasus")[1].innerHTML = "Ovo je
definisano metodom querySelector()";
</script>
</body>
</html>
```

Ovo je prvi pasus

Ovo je definisano metodom `querySelector()`

Slika 8.9. Prikaz selekcije elemenata DOM-a pomoću metoda `querySelectorAll()`

8.6. Svojstvo `attributes` i druga slična svojstva

Prethodno objašnjeni metodi su imali za cilj da izvrše selekciju nekog DOM elementa, da bi se nakon toga sa njim nešto radilo. Pokazano je kako se utiče na sadržaj elementa primenom svojstava `textContent` i `innerHTML`. Međutim, sastavni deo DOM elementa su i atributi selektovanog HTML taga. Ovim atributima se može pristupiti pomoću svojstva `attributes`. Ovo svojstvo se definiše iza nekog od metoda koji selektuju DOM element i vraća skup svih atributa i njihovih vrednosti za selektovani element. Kako atributa može biti više, vraćeni podatak je u formi niza, pa se svojstvom `length` može izračunati dužina niza tj. indirektno broj definisanih atributa selektovanog elementa. Pošto se vraća objekat, koji sadrži ime i vrednost svakog od atributa, za tako dobijen objekat (npr. `attributes[0]`), koristimo njegova svojstva `name` i `value` da bi pristupili imenu tj. vrednosti dohvaćenog atributa. Tako pomoću linije koda:

```
document.querySelector("#primer").attributes[0].name
```

dohvatamo ime prvog atributa unutar elementa čiji je *id*=“*primer*“. U sledećem primeru data je primena svojstava *attributes*, *length*, *name* i *value*.

```
<html>
<head> </head>
<body>
<div id="primer" width="1000px" class="blok1">
    <p class="pasus">Ovo je prvi pasus</p>
</div>
<script type="text/javascript">
var brojAtributa = document.querySelector("#primer").attributes.length;
    for (var i=0; i<brojAtributa; i++){
document.write("Ime atributa: " +
    document.querySelector("#primer").attributes[i].name);
document.write("Vrednost atributa: " +
    document.querySelector("#primer").attributes[i].value + "<br/>");
    }
</script>
</body>
</html>
```

Ovo je prvi pasus

Ime atributa: id Vrednost atributa: primer
 Ime atributa: width Vrednost atributa: 1000px
 Ime atributa: class Vrednost atributa: blok1

Slika 8.10. Prikaz selektovanih atributa pomoću svojstva *attributes*

Pored svojstva *attributes* postoji još nekoliko svojstava i metoda koje služe za dohvaćanje imena ili vrednosti atributa selektovanog elementa. Jedan od njih je i metod *getAttribute("class")* koji vraća vrednost definisanog atributa u selektovanom elementu. Primer njegove upotrebe bi bio:

```
document.getElementsByTagName("p")[0].getAttribute("class")
```

kada bi se u prethodnom kodu vratila vrednost *pasus*.

Pored ovog, na sličan način se koriste i sledeći metodi:

- ✓ *element.hasAttribute()* // vraća *true* ili *false* ukoliko ima ili nema definisani atribut
- ✓ *element.removeAttribute()* // uklanja definisani atribut iz elementa
- ✓ *element.setAttribute()* // postavlja ili menja vrednost definisanom atributu

Takođe, naglašavam da postoji veliki broj metoda i svojstava koji se mogu primeniti za najrazličitije namene, i da se njihov broj stalno povećava pa treba pratiti sve ove promene, kao i podrške od strane različitih verzija browser-a, kroz dokumentaciju jezika.

Pored navedenih metoda, izdvojio bih i nekoliko dodatnih svojstava za rad sa atributima:

- ✓ `element.className` // dohvata i setuje vrednosti atributa `class`, selektovanog elementa
- ✓ `element.title` // dohvata i setuje vrednosti atributa `title`, selektovanog elementa
- ✓ `element.id` // dohvata i setuje vrednosti atributa `id`, selektovanog elementa
- ✓ `element.lang` // dohvata i setuje vrednosti atributa `lang`, selektovanog elementa

8.7. Svojstva za rad sa node-ovima

DOM je organizovan po principu stabla i predstavlja kompletan HTML kod web stranice. Elementi tog stabla sa *node*-ovi i oni predstavljaju svaki HTML element. Za početak, ceo dokument je jedan *node* i to *document node*. Dalje, svaki HTML element stranice je *node* i to *element node*. Ovo se odnosi na inicijalne, roditeljske (*parent*), elemente tj. tagove, i na sve koji su unutar njih, tj. decu (*child*) tagove. Svaki atribut unutar HTML elementa je takođe *node* i to *attribute node* i na kraju, svaki tekst koji je sadržaj HTML elementa je *text node*. Na ovaj način, kada se kuca HTML kod, tj. sve njegove komponente jezika, pomoću DOM-a sve postaje uređena struktura kroz koju se programski može „kretati“ a samim tim pristupati, menjati, brisati i kreirati novi sadržaj.

U JavaScript-u postoji veliki broj metoda i svojstava kojima se može selektovati *node*, „kretati“ po strukturi stabla DOM-a, kreirati novi *node*, setovati ili modifikovati njegov sadržaj i na kraju brisati *node*. Ovaj skup metoda se intenzivno povećava i omogućava programerima sve lagodniji rad sa *node*-ovima i DOM-om, pa treba aktivno pratiti novine u jeziku ali i podrške u različitim browser-ima za novije metode i svojstva. Neki od sajtova na kojima se mogu detaljnije pročitati opisi metoda i svojstva su:

- ✓ https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- ✓ <https://api.jquery.com/category/manipulation/>,
- ✓ <http://www.javascriptkit.com/domref/elementmethods.shtml>

Do sada opisani metodi za selekciju elemenata DOM su bili na principu poznavanja HTML koda, tj. poznavanja tačne strukture elemenata, atributa tih elemenata, sadržaja atributa *id* ili *class* itd. Na bazi tog poznavanja, pojedini elementi su „prozivani“ i nakon selekcije sa njima se dalje nešto radilo.

Iako deluje potpuno logično da će programer poznavati svoj kod do najsitnijeg detalja, i da uvek može da primenom ranije opisanih metoda selektuje bilo koji element DOM-a, to u praksi nije uvek slučaj. Naime, ako pođemo od toga da se JavaScript-om želi kreirati novi node, tj. HTML element, potrebni su nam metoda koji će to uraditi na nivou skript-a. Kreiranje novih elemenata može biti u neograničenom broju, pa čak i do nivoa da se cela HTML strana kreira iz JavaScript-a. Isto se odnosi na promene sadržaja elemenata. Na neku web stranu može da deluje veći broj skript fajlova, od kojih je neke pisao programer, a neke preuzeo kao gotove kodove.

Kreiranje i dodavanje novih elemenata obično se vrši u situacijama kada se sadržaj kreira dinamički na bazi podataka od servera tj. baze podataka. Pretpostavimo da želimo da prikazemo imenik ili kontakt listu korisnikovih prijatelja sa neke socijalne mreže. Ako bi sadržaj takve liste bio dostupan u nekom formatu, a najčešće je to JSON ili XML, i ako bi se JavaScript obratio sa zahtevom da dobije takav fajl, i u njemu kontakte, onda bi kod nakon dobijanja tih kontakata, te kontakte trebalo da prikaže. Prikaz podataka bi se realizovao kroz neku petlju, koja bi u svakoj iteraciji generisala kod za jednog korisnika, kreirajući neki HTML element (npr. u formatu neuređene liste).

Samo dohvaćanje, kreiranje i prikazivanje se dešava kod korisnika u trenutku kada on aktivira tu opciju, i programer u tom trenutku nema pristup tim podacima niti može da zna unapred koliko će tih podataka biti, jer on te podatke u statičkom obliku HTML koda nije ni pisao.

Tako dolazimo do trenutka kada kod nije potpuno poznat programeru i kada on treba da napiše JavaScript kod koji će se prilagoditi i količini podataka i strukturi koju treba obratiti i prikazati. Pretpostavimo još dalje scenario, da je korisniku nakon dinamičkog prikaza takvog skupa podataka potrebno, posebnim stilom, obeležiti članove porodice ili prijatelje sa posla. Da bi se ovo uradilo kod dodatno treba da ima isprogramiranu logiku koji od novokreiranih elemenata treba pronaći po nekom kriterijumu, modifikovati u smislu CSS koda, zbog drugačijeg prikaza itd. Svo opisano kreiranje elemenata, da bi se oni prikazali, kretanje kroz njih u smislu pretrage, uticaja na njih u smislu promene načina prikaza i sl. se realizuje metodama i svojstvima JavaScript-a za rad sa elementima DOM-a.

U takvim situacijama, ređe će se koristiti klasični metodi za selekciju, kao u slučaju direktnog pristupa `document.getElementsByClassName("kontakt")[22]`, jer je broj elemenata inicijalno nepoznat ili promenljiv pa je potrebno dinamički urediti pristup, selekciju i prepoznavanje svih elemenata DOM-a.

8.7.1. Kreiranje novog elementa DOM-a sa tekstualnim sadržajem

Ukoliko se želi JavaScript-om kreirati novi element DOM-a, koristi se metod *createElement()* i kao njegov argument bilo koji tag HTML-a. Nakon delovanja ovog metoda, u memoriji browser-a se kreira novi objekat, koji je HTML element. Ovo se još uvek ne prikazuje korisniku. Da bi se novi element prikazao, potrebno je proći kroz sledeće korake:

1. Kreiranje novog elementa
2. Kreiranje sadržaja novog elementa
3. Dodela sadržaja novom elementu
4. Selekcija nekog postojećeg elementa HTML strane u odnosu na koji se želi postaviti novi element
5. Postavljanje novog elementa u odnosu na selektovani

Prvi korak se realizuje kao

```
var noviElement= document.createElement("h1");
```

Na ovaj način kreiran je element `h1`, i možemo zamisliti da je u pozadini kreirana struktura, tj. *node*

```
<h1> </h1>
```

koji nema sadržaj niti pripadnost nekom delu HTML strane.

Drugi korak se može realizovati kao

```
var sadrzajElementa = document.createTextNode("Ovo je sadržaj iz  
JavaScript-a!");
```

Na ovaj način kreira se *text node*, tj. sadržaj za neki *node* koji je tekstualnog tipa. Da bi ovaj sadržaj bio dodeljen nekom *node-u*, koristimo neki od metoda za dodelu sadržaja *node-u*, kao što je *appendChild()*. Ovaj metod jedan *node*, u ovom slučaju *sadrzajElementa*, kreira kao dete drugog *node-a*, u ovom slučaju *noviElement*. Ovaj treći korak se može tako definisati na sledeći način:

```
noviElement.appendChild(sadrzajElementa);
```

Nakon ovog koraka, možemo da zamislimo da se u memoriji browser-a dodelio sadržaj nekom *node-u*, tj. kreiralo sledeće:

```
<h1> Ovo je sadržaj iz JavaScript-a! </h1>
```

Da bi ovako dobijeni sadržaj bio vidljiv i za korisnika, on se mora pozicionirati negde unutar postojeće HTML strane. Da bi se ovo realizovalo potrebno je selektovati neki postojeći element stranice, kao npr.

```
var deo_strane= document.getElementById("kontejner");
```

Ovako selektovani element se koristi kao orijentir gde treba postaviti novo kreirani element iz memorije browser-a. Postoji puno metoda za postavljanje novog elementa, u odnosu na selektovani (unutar njega na početak, unutar njega na dno, pre njega, iza njega, ...) ali ako u ovom slučaju iskoristimo isti metod kao u tački 3, tj. metod *appendChild()*, sa ciljem da novokreirani sadržaj bude umetnut kao dete selektovanog elementa, to se zapisuje kao

```
deo_strane.appendChild(noviElement);
```

Na ovaj način kod iz memorije postaje sastavni deo DOM-a, i HTML koda, pa ga browser u tom trenutku prikazuje korisniku. Ceo kod se zapisuje na sledeći način:

```
<html>
<head> </head>
<body>
<div id="kontejner">
    <p class="pasus">Ovo je prvi pasus</p>
</div>
<script type="text/javascript">
    var deo_strane= document.getElementById("kontejner");
    var noviElement= document.createElement("h1");
    var sadrzajElementa = document.createTextNode("Ovo je sadržaj iz
JavaScript-a!");
    //var sadrzajElementa = "<img src='1.jpg' alt='slika' />";
    //noviElement.innerHTML = dodatak;
    noviElement.appendChild(sadrzajElementa);
    deo_strane.appendChild(noviElement);
</script>
</body>
</html>
```

Kao rezultat koda korisniku se pored inicijalnog sadržaja u elementu *p*, sada prikazuje i sadržaj elementa *h1* koji je kreiran JavaScript-om.

Ovo je prvi pasus

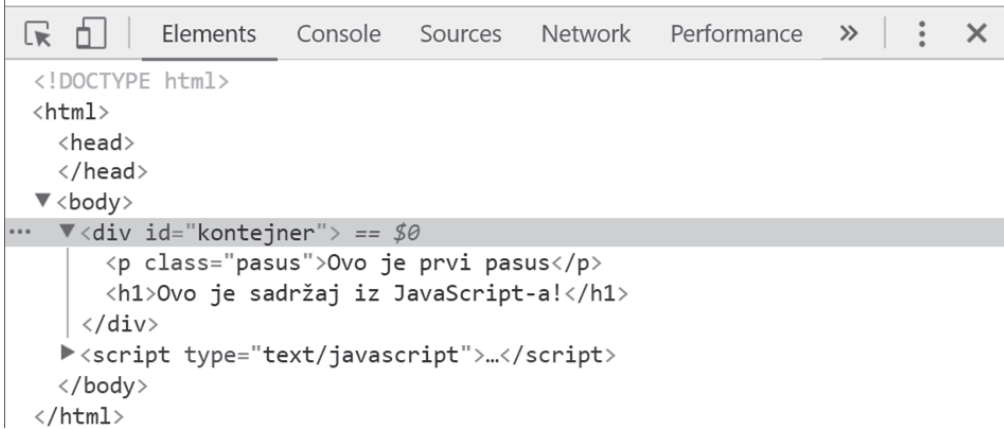
Ovo je sadržaj iz JavaScript-a!

*Slika 8.11. Upotreba metoda *appendChild()* za dodavanje novog elementa DOM-a*

Ukoliko se pogleda u sadržaj stranice u Inspect-u, videćemo da je kreirani element *h1* sastavni deo koda, kao da ga je programer inicijalno napisao sa elementom *p*.

Ovo je prvi pasus

Ovo je sadržaj iz JavaScript-a!



Slika 8.12. Prikaz HTML koda web strane nakon delovanja metoda `appendChild()`

8.7.2. Kreiranje novog elementa DOM-a sa proizvoljnim sadržajem

U prethodnom primeru korišćen je metod `createTextNode()` za kreiranje tekstualnog sadržaja. Ukoliko se želi definisati bilo koji drugi sadržaj, to je moguće uraditi, u tački 2, kao kreiranje klasične promenljive u JavaScript-u, npr.

```
var sadrzajElementa = "<img src='prolece.jpg' alt='slika' />";
```

Kako ovo sada nije sadržaj tipa *node*, a njega je potrebno dodeliti nekom kreiranom elementu u tački 1, može se iskoristiti svojstvo *innerHTML* kao:

```
noviElement.innerHTML = sadrzajElementa;
```

Ako sav ostali kod ostane isti, kao u prethodnom primeru, i samo se kreira element *div* umesto *h1*, imamo:

```

<html>
<head> </head>
<body>
<div id="kontejner">
  <p class="pasus">Ovo je prvi pasus</p>
</div>
<script type="text/javascript">
  var deo_strane= document.getElementById("kontejner");
  var noviElement= document.createElement("div");
  var sadrzajElementa = "<img src='prolece.jpg' alt='slika' />";

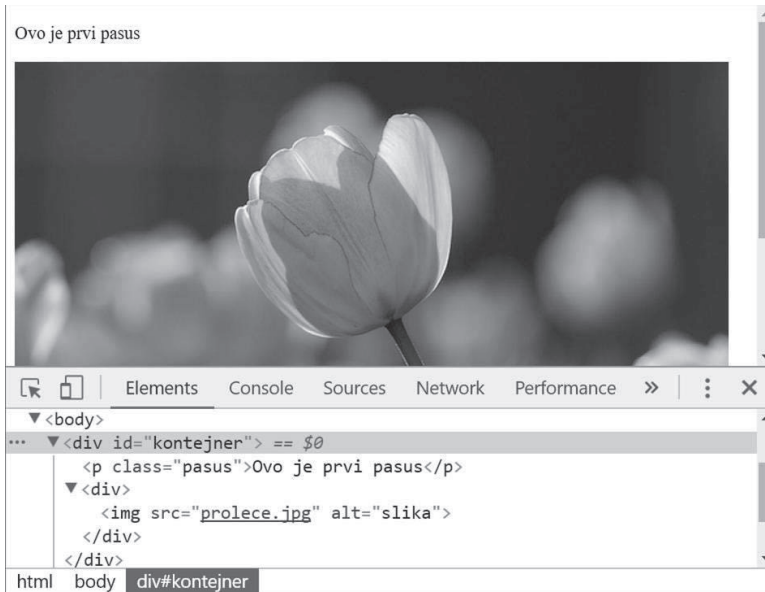
```

```

    noviElement.innerHTML = sadrzajElementa;
    deo_strane.appendChild(noviElement);
</script>
</body>
</html>

```

Ovakav kod korisniku prikazuje sliku, kao sadržaj novokreiranog elementa *div*, na sledeći način:



Slika 8.13. Prikaz HTML koda web strane nakon delovanja elemenata *img*

Vrlo često se kod kreiranja elemenata DOM mora voditi računa o tome da kreirani elementi moraju biti dovoljno različiti da bi im ili JavaScript ili neki drugi jezik (klijentski ili serverski) mogao jednoznačno pristupiti. Ako pretpostavimo da se unutar *for* petlje kreira deset *span* elemenata u kojima su *p* elementi, to bi se zapisalo:

```

<script type="text/javascript">
  var deo_strane= document.getElementById("kontejner");
  for( var i=0; i<10; i++){
    var noviElement= document.createElement("span");
    var sadrzajElementa= "<p class='primer'> proba </p>";
    noviElement.innerHTML = sadrzajElementa;
    deo_strane.insertBefore(noviElement, null);
  }
</script>

```

Svi kreirani *p* tagovi imaju istu klasu i isti sadržaj. Međutim, ako bi umesto *p* elemenata tu bili elementi forme, oni bi morali međusobno da se razlikuju da bi serverski kod mogao da ih razlikuje. U sledećem primeru kreirana je funkcija

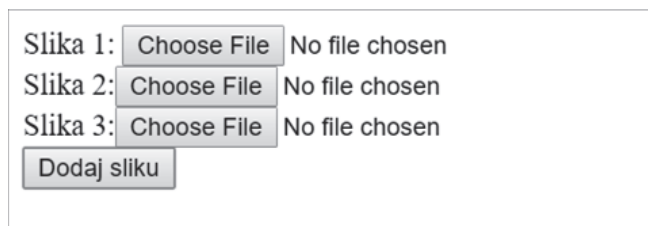
dodaj() , koja se poziva svakim klikom na taster, i unutar koje je kod kojim se kreira jedan element forme za upload fajlova. Svaki sledeći element forme treba da ima drugačiji *name*, i to *file1*, *file2* itd., a da se ispred elementa forme ispisuje *Slika 1*, *Slika 2* itd. Za te potrebe kreirana je lokalna promenljiva *broj*, koja se inkrementira svakim pozivom funkcije, i koja se koristi da se u konkatanaciji stringa *file* i vrednosti ove promenljive kreiraju stringovi *file1*, *file2*,... tj. *Slika 1*, *Slika 2*, ...

```

<html>
<head> </head>
<body>
  <div id="kontejner">
    Slika 1: <input type="file" name="fajl1" /><br/>
  </div>
  <input type="button" value="Dodaj sliku" onclick="dodaj();" />
  <script type="text/javascript">
    var broj= 1;
    function dodaj(){
      var deo_strane= document.getElementById("kontejner");
      var noviElement= document.createElement("div");
      broj++;
      var sadrzajElementa= "Slika " + broj + ":<input type='file'
name='fajl" + broj + "' /><br/>";
      noviElement.innerHTML = sadrzajElementa;
      deo_strane.insertBefore(noviElement, null);
    }
  </script>
</body>
</html>

```

Na ovaj način, nakon dva klika na taster korisniku se prikazuje jedan inicijalni i dva dodatna polja za upload, kao na slici.



Slika 8.14. Prikaz dinamičkog dodavanja novih DOM elemenata u web stranu

Pored navedenog, postoji puno drugih metoda i svojstava za rad sa elementima DOM-a, koji najčešće svojim imenom jednoznačno ukazuju na to šta je efekat njihovog delovanja.

Od velikog broja metoda ukazujem na neke koji mogu biti vrlo korisni:

- ✓ *element.hasChildNodes()* // vraća true ako element ima child node
- ✓ *element.childElementCount* // vraća broj child node-ova
- ✓ *element.childNodes* // vraća broj child node-ova
- ✓ *element.contains()* // vraća true ako node ima potomka tj. sadrži
- ✓ *element.firstChild* // vraća prvi child node-a
- ✓ *element.lastChild* // vraća poslednji child node-a
- ✓ *element.nextSibling* // vraća sledeći node na istom nivou DOM stabla
- ✓ *element.nodeName* // vraća ime (naziv) node-a
- ✓ *element.nodeType* // vraća tip node-a
- ✓ *element.nodeValue* // kreira ili dohvata sadržaj node-a
- ✓ *element.ownerDocument* // vraća root element za dati element
- ✓ *element.parentNode* // vraća roditeljski node elementa
- ✓ *element.previousSibling* // vraća prethodni node na istom nivou DOM-a
- ✓ *element.removeChild()* // briše child node definisanog elementa
- ✓ *element.replaceChild()* // menja child node definisang elementa
- ✓ *element.cloneNode()* // kopira selektovani element
- ✓ *element.appendChild()* // dodaje novi child node, unutar elementa, kao njegov poslednji child node
- ✓ *element.insertBefore()* // dodaje novi child node, unutar elementa, pre definisanog child node

8.8. Svojstva za rad sa dimenzijama i pozicijama elemenata

JavaScript nudi nekoliko svojstava i metoda kojim se dohvata visina, širina ili rastojanje (*offset*) od neke pozicije. Programeri koriste ove mogućnosti da mogu kodom da lociraju poziciju nekog elementa koji je recimo relativno pozicioniran, ili za elemente u responsive dizajnu itd. Često se ova svojstva, u naprednim primenama, koriste za poziciju pop-up prozora, slider-a koji se prilagođavaju različitim dimenzijama slika, skrolovanje stranica u *single-page* aplikacijama itd. Kod primene ovih svojstava treba voditi računa koji sve delovi tj. elementi stranice ulaze u konačno vraćenu cifru.

Ovde su navedena neka od svojstava i metoda koja se često koriste:

<i>element.offsetTop</i>	// vraća visinu elementa, uključujući padding, border i scrollbar
<i>element.offsetWidth</i>	// vraća širinu elementa, uključujući padding, border i scrollbar
<i>element.offsetLeft</i>	// vraća horizontalno rastojanje elementa, poziciju od leve ivice browser-a
<i>element.offsetParent</i>	// vraća poziciju, dimenzije, okvira elementa
<i>element.offsetTop</i>	// vraća horizontalno rastojanje elementa, poziciju od gornje ivice browser-a
<i>element.scrollHeight</i>	// vraća ukupnu visinu elementa, uključujući i padding
<i>element.scrollIntoView()</i>	// pomeranje selektovanog elementa unutar vidljive zone browser-a
<i>element.scrollLeft</i>	// postavljanje ili dohvaćanje broja piksela sadržaja elementa koji se pomera horizontalno
<i>element.scrollTop</i>	// postavljanje ili dohvaćanje broja piksela sadržaja elementa koji se pomera vertikalno
<i>element.scrollWidth</i>	// vraća ukupnu širinu elementa uključujući i padding

Na primer, u datom kodu će se ispisati vrednost 220.

```

<html>
<head>
<style>
#primer{margin: 20px;
padding: 20px;
width: 400px;
height: 400px;
position: relative;
top: 200px;
left: 200px;
border: 1px solid red; }
</style>
</head>
<body>
<div id="primer">
    Ovo je primer div-a!
</div>
<script type="text/javascript">
    var x = document.getElementById("primer").offsetTop;
    alert(x);
</script> </body> </html>

```

8.9. HTML DOM objekti, kolekcije i svojstva

U sve tri verzije HTML DOM-a definisani su različiti objekti, kolekcije i svojstva koje omogućavaju bolju interakciju sa HTML kodom. Za neka od njih moguće je naći alternativne načine u JavaScript-u dok su neki unikatni u poslovima koje obavljaju. Iako je broj ovih objekata i svojstava u svakoj verziji DOM-a sve veći i kako se većina iz prethodnih verzija zadržava i u narednim, to pokazuje da zajednica koristi i smatra ove mogućnosti vrlo korisnim za pojedine situacije. U ovom delu će se pobrojati samo neki od pomenutih objekata, kolekcija i svojstava da bi se stekao uvid u to kako oni izgledaju i za koje potrebe se mogu koristiti. Karakteristično je da sva ova svojstva inicijalno pripadaju objektu *document*, a ne nekom proizvoljno selektovanom elementu. Neka od njih će se naknadno kasnije obrađivati kroz različite oblike implementacija.

```
document.baseURI      // vraća apsolutni URI stranice tj. dokumenta
document.documentURI // vraća URI stranice tj. dokumenta
document.URL          // vraća kompletni URL stranice tj. dokumenta
document.domain       // vraća domain name servera
document.documentElement // vraća kompletan sadržaj <html> elementa
document.head         // vraća kompletan sadržaj <head> elementa
document.body         // vraća kompletan sadržaj <body> elementa
document.cookie       // vraća kompletan sadržaj kolačića
document.doctype      // vraća doctype dokumenta
document.anchors      // vraća sve <a> elemente sa atributom name
document.embeds       // vraća kompletnu listu <embed> elemenata
document.forms        // vraća kompletan sadržaj svih <form> elemenata
document.images       // vraća kompletan sadržaj svih <img> elemenata
document.lastModified // vraća podatak o datumu i vremenu kada je stranica
postavljena ili modifikovana
document.links        // vraća kompletan sadržaj svih <a> i <area> elemenata
koji imaju href attribute
document.scripts      // vraća kompletan sadržaj svih <script> elemenata
```

8.10. Svojstva objekta style

Svi do sada opisani metodi i svojstva su se odnosila na selekciju, modifikaciju, kreiranje i brisanje HTML elemenata. Pokazano je da JavaScript na opisane načine može imati punu kontrolu nad web stranom, u domenu HTML koda. Međutim, postavlja se pitanje da li je tako nešto moguće uraditi i sa CSS-om, obzirom da bi se onda omogućio kompletan uticaj na statičku web stranicu. Odgovor je: Da. Ovo je omogućeno kroz svojstvo *style*, koje je svojstvo selektovanog elementa DOM-a. Na primer, za element koji ima *id*= "primer":

```
document.getElementById("primer").style
```

Primenom svojstva *style*, možemo zamisliti da je u pozadini, kod selektovanog elementa, kreiran HTML atribut *style*, čime se podstiče efekat inline CSS-a. Sadržaj atributa *style*, u klasičnom HTML-u, bi bio klasična sintaksa CSS-a, tj. *svojstvo:vrednost*. Međutim, ako se ovo realizuje JavaScript-om, i svojstvom *style*, onda se on ponaša kao objekat i ima svoja svojstva. Ta svojstva treba da zamene, ili su najčešće identična, svojstvima originalnog CSS-a. Tako postoji svojstvo *color*, objekta *style*, i zapisujemo ga kao:

```
document.getElementById("primer").style.color
```

Ukoliko svojstvu *color* želimo da dodelimo vrednost red, to bi se zapisalo kao

```
document.getElementById("primer").style.color="red";
```

Na ovaj način, selektovanom elementu, u ovom slučaju elementu sa *id*= "primer" definisan je inline CSS kojim se, na postojeći CSS kod, dodaje svojstvo *color* sa vrednošću *red*, na isti način kao da je inicijalno napisano u CSS kodu. Ukoliko bi za isti selektovani element u CSS kodu bilo definisano isto svojstvo sa drugom vrednošću, a nakon toga to isto svojstvo dobilo drugu vrednost pomoću JavaScript-a, bila bi primenjena ta druga vrednost, jer je ona novija po redosledu pisanja i samim tim poništava prethodno definisanu vrednost. Po ovom pravilu, sve što dođe kroz delovanje JavaScript-a će uvek imati prioritet i moći će da menja inicijalno definisano stanje.

Najvažniji korak kod upotrebe svojstva *style* je poznavati koja njegova svojstva se mogu upotrebiti i na taj način delovati na CSS kod. Spisak ovih svojstava je prilično dugačak, i ovde se neće davati u potpunosti, ali se može pogledati u standardu jezika, ili sajtovima koji se bave ovom tematikom (<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style> ili https://www.w3schools.com/jsref/dom_obj_style.asp). Kako se URL adrese sajtova često menjaju najbolje je pretraživanjem u web pretraživačima pronaći aktuelne web strane na kojima se nalaze ovi i drugi slični podaci.

Od najkorišćenijih svojstava izdvojiće se sledeća:

<i>alignItems</i>	<i>content</i>	<i>margin</i>
<i>animation</i>	<i>direction</i>	<i>maxHeight</i>
<i>background</i>	<i>display</i>	<i>maxWidth</i>
<i>backgroundAttachment</i>	<i>filter</i>	<i>minHeight</i>
<i>backgroundColor</i>	<i>flex</i>	<i>minWidth</i>
<i>backgroundImage</i>	<i>flexDirection</i>	<i>opacity</i>
<i>backgroundPosition</i>	<i>flexFlow</i>	<i>padding</i>
<i>backgroundRepeat</i>	<i>font</i>	<i>position</i>
<i>backgroundSize</i>	<i>fontFamily</i>	<i>right</i>
<i>border</i>	<i>fontSize</i>	<i>tableLayout</i>
<i>borderColor</i>	<i>fontStyle</i>	<i>textAlign</i>
<i>borderImage</i>	<i>fontWeight</i>	<i>textDecoration</i>
<i>bottom</i>	<i>height</i>	<i>textJustify</i>
<i>boxShadow</i>	<i>icon</i>	<i>textTransform</i>
<i>clear</i>	<i>justifyContent</i>	<i>transition</i>
<i>color</i>	<i>left</i>	<i>verticalAlign</i>
<i>columnCount</i>	<i>letterSpacing</i>	<i>visibility</i>
<i>columnFill</i>	<i>listStyle</i>	<i>width</i>
<i>columnWidth</i>	<i>listStyleType</i>	<i>zIndex</i>

Kao primer upotrebe svojstva style pokazaće se primena nekoliko svojstava kojima se definišu promene u CSS-u, pomoću JavaScript-a.

```

<html>
<head>
</head>
<body>
<div id="primer">
    <p id="pasus">
        Ovo je primer div-a!
    </p>
</div>
<div id="skriveni">
    Ovo je drugi div-a!
</div>

<script type="text/javascript">
var element = document.getElementById("primer");
    element.style.color="red";
    element.style.backgroundColor="yellow";
    element.style.border = "10px solid green";
    element.style.fontFamily = "Verdana";
    element.style.fontSize = "36px";
    element.style.height = "120px";
    element.style.padding = "40px 20px 30px 10px";
    document.getElementById("pasus").style.textAlign = "right";

```

```
document.getElementById("pasus").style.textDecoration=
    "underline";
document.getElementById("skriveni").style.display = "none";
</script>
</body>
</html>
```



Slika 8.15. Prikaz web strane nakon dodavanja CSS stilova kroz JavaScript

8.11. Svojstvo `classList` i njegove metode

Svojstvo `classList` vraća vrednosti atributa `class` i može se koristiti za dodavanje ili brisanje novih imena klasa selektovanog elementa, primenom metoda `add()` tj. `remove()`.

Pored dodavanja i brisanja postoji podrška i za efekat *toggle*, primenom metoda `toggle()`, kojim se određeni stil dodaje, ako ne postoji, ili briše, ako postoji. Svojstvo `classList` se primenjuje na selektovani HTML element u opštoj sintaksi:

```
document.getElementById("imIdElementa").classList.add("stilNovi");
```

Primenom svojstva `classList` i njegovih metoda, može se zamisliti da se postiže efekat proširivanja spiska CSS klasa koji se primenjuju na određeni element HTML-a. Tako ako inicijalno ne postoji atribut `class`, metodom `add` se postiže efekat njegovog dodavanja kao i dodavanja njegovog sadržaja. Ukoliko atribut `class` postoji kao npr.

```
<div id="primer" class="artikal"> </div>
```

i vrednost `artikal`, dohvati primenom svojstva `classList`, a zatim metodom `add()` doda nova klasa *akcija*, efekat bi bio kao da je inicijalno pisalo:

```
<div id="primer" class="artikal akcija"> </div>
```

Na ovaj način se unapred pripremljeni delovi CSS koda mogu dinamički ili interaktivno primeniti na neki element i tako promeniti način prikaza određenog dela web strane. Na isti način se mogu ukloniti i vizuelni identitet učiniti drugačijim.

Npr. ovo bi moglo da se primeni na stil kod korisnika koji nisu autorizovani, dok se nakon uspešnog logovanja način prikaza podataka menja.

U sledećem primeru pokazana je primena svojstva *classList* i metoda *add()*.

```

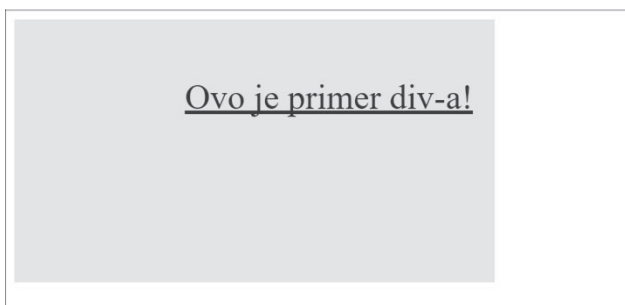
<html>
<head>
<style>
.styleDodatni {
  text-align: right;
  text-decoration:underline;
  background-color: yellow;
  color: red;
  padding:20px;
  font-size: 32px;
  width: 400px;
  height: 200px; }
</style>
</head>

<body>
<div id="primer">
  <p id="pasus">
    Ovo je primer div-a!
  </p>
</div>

<script type="text/javascript">
  document.getElementById("primer").classList.add("styleDodatni");
</script>

</body>
</html>

```



Slika 8.16. Prikaz delovanja svojstva *classList* i njegovih metoda

Opisani metodi predstavljaju trenutno aktuelne načine pristupa i manipulacije DOM elementima. Bez obzira koji način za selekciju elemenata se koristi sada, ili koji će se novi metodi koristiti u budućnosti, selekcija konkretnog elementa ili grupe elemenata je ključni segment rada JavaScript-a. Ranije je objašnjeno da različiti metodi JavaScript-a mogu umnogome promeniti kompletan izgled web strane. Međutim da bi se to desilo, neophodna je selekcija elementa DOM-a i tek

tada primena konkretnih metoda. Iz tog razloga, metode za selekciju i upravljanje DOM-om zauzimaju vrlo bitno mesto u učenju JavaScript-a.

IX RAD SA DOGAĐAJIMA

Interakcija korisnika sa web stranom

JEDAN od baznih stubova na kojima JavaScript gradi interaktivnost između korisnika i web strane je mogućnost rada sa događajima (*events*). Događaji zato predstavljaju vrlo bitan segment u izučavanju skript jezika i treba mu posvetiti posebnu pažnju. Rad sa događajima je relativno lak i nakon razumevanja koncepta rada i pojma događaja, potrebno je proučiti koji sve događaji postoje, i šta nam nude, da bi se dobio najbolji efekat u konkretnim situacijama.

Događaji su reakcije sistema na akcije korisnika i predstavljaju stanje pojedinih DOM elemenata. Kažemo da browser detektuje veliki broj akcija korisnika i stanja web strane i da programeru daje mogućnost da u tim specifičnim trenucima realizuje neki unapred napisani kod. To praktično znači da svaki prelazak mišem preko nekog HTML elementa, svaki klik na taster, izbor elementa padajuće liste, selekcija nekog radio tastera, detekcija greške prilikom učitavanja slike, detekcija trenutka učitavanja cele web strane, izlazak iz tekstualnog polja formulara, klik sa tastature na neki taster prilikom unosa username-a itd. Za browser predstavlja vrlo specifične aktivnosti korisnika. Za svaku od ovih aktivnosti, browser se

trigueruje (okida) i pita se da li nešto treba da uradi nad elementom gde se ta aktivnost događa. Ukoliko programer ne koristi događaje, tj. ne piše kod koji definiše kada će se i u kojim uslovima desiti, browser će svaki put da „okine“ i zapita se da li treba nešto da uradi, i kako nema nikakvog koda, neće ništa ni preduzeti. Na ovaj način dobijamo klasičan statički web sajt.

Ukoliko programer isprogramira kod koji će se dešavati u ovim specifičnim trenucima, ili kao rezultat akcije korisnika na web strani, kažemo da imamo isprogramirane događaje i statički sajt postaje interaktivan sajt. Termin interaktivan sajt treba da ukaže da postoji vrlo jasna i unapred definisana logika koja definiše šta je reakcija web strane na neku akciju korisnika tj. da imamo neki oblik interakcije korisnika sa sajtom tj. web stranom.

Postoji velika lista događaja, što znači da postoji veliki broj karakterističnih situacija koje browser detektuje i za svaku od njih pruža mogućnost programeru da napiše kod koji će se u tom slučaju desiti. Pogledajmo sledeći primer koji je deo forme za registraciju korisnika za Gmail.

The image shows three sequential screenshots of the Gmail registration form, illustrating the focus and blur events. Each form has fields for Name (First and Last), Choose your username (with a placeholder @gmail.com), Create a password, and Confirm your password.

- Left screenshot:** The 'First' name field is in focus, indicated by a thick border.
- Middle screenshot:** The 'First' name field is blurred, and the 'Choose your username' field is in focus.
- Right screenshot:** The 'Choose your username' field is blurred, and the 'Create a password' field is in focus. A red error message 'You can't leave this empty.' is visible below the 'Choose your username' field.

Slika 9.1. Primer implementacije događaja i interakcije JavaScript-a sa web stranom (inicijalni prikaz, ulaz u fokus polja i napuštanje fokusa sa praznim sadržajem) izvor <https://mail.google.com/mail/signup>

Nakon inicijalne forme, klikom miša u polje za Name, detektuje se događaj *onFocus()* i browser automatski prikazuje treptuću vertikalnu crtu kao oznaku da je korisnik u fokusu polja. Ukoliko korisnik ne unese nikakav sadržaj, kod se još uvek neće „buniti“, jer čeka da se neki sadržaj upiše. Ako korisnik odluči da ne unese ništa, i da klikom van tog polja izađe iz tog polja, aktivira se događaj *OnBlur()*, kojim vertikalna linija u tekstualnom polju prestaje da treptće kada se kontroliše uneti sadržaj. Kako se tada detektuje da sadržaj nije unet, JavaScript kodom se menja CSS kojim se boji border tekstualnog polja za Name u crvenu boju i ispisuje tekst upozorenja neposredno ispod polja (*You can't leave this empty.*). Ukoliko korisnik nikada ne uđe u ovo polje, i pokuša da klikne na taster *Next step*, na dnu forme, aktivira se događaj *onClik()* koji proverava sva polja, i za svako koje nije regularno popunjeno prikazuje grešku na sličan način.

Ovaj primer treba okvirno da pokaže interakciju korisnika i stranice, i inicijalno razumevanje događaja, koji će se detaljnije objašnjavati u nastavku. Kao drugi primer rada sa događajima, posmatrajmo klasični drop-down meni. Meni ima glavne stavke i pod stavke, koje su definisane HTML kodom. Po učitavanju kod

strane, aktivira se događaj *onLoad()*, u kome se može isprogramirati kod za sakrivanje svih postavki menija, dok glavne stavke ostaju vidljive. Prelaskom mišem preko neke od glavnih stavki menija, aktivira se događaj *onMouseOver()*, kod koga je isprogramirano da napravi vidljivim samo stavke podmenija, koje pripadaju toj glavnoj stavci menija. Prelaskom miša na drugu glavnu stavku, prvo se aktivirao događaj *onMouseOut()*, nakon izlaska iz prethodne stavke menija, koji je ponovo sakrio prethodne stavke podmenija, pa zatim ponovo događaj *onMouseOver()* za drugu stavku u glavnom meniju koji otkriva njen podmeni itd.

Iako događaji još uvek nisu formalno objašnjeni, vidi se da su imena događaja vrlo intuitivna i vrlo jednoznačno ukazuju na trenutak ili akciju od strane korisnika koja se može programirati. Isti događaj se može isprogramirati na proizvoljno mnogo elemenata HTML strane a da pri tome na svakom od njih može da ima drugačiji vizuelni efekat za korisnika. Tako događaj postaje „dobar sluga“ za sve što je programeru potrebno i umnogome povećava kvalitet i vizuelni identitet web strane.

9.1. Načini aktivacije događaja

Događaji se neformalno mogu podeliti po mestu, trenutku ili akciji korisnika tj. sistema. Tako kažemo da događaji mogu da budu triggerovani na jedan od sledećih načina:

- ✓ Klikom korisnika na miš, ili prelaskom miša, preko nekog konkretnog HTML elementa,
- ✓ Pritiskom na neki od tastera na tastaturi,
- ✓ Promenu dimenzija browser-a ili zatvaranje prozora browser-a,
- ✓ Trenutak kada se web strana učita, tj. kada se učitaju svi kodovi i fajlovi te web strane,
- ✓ Trenutak kada se submit-uje forma,
- ✓ Trenutak kada se video fajl pusti, pauzira ili završi emitovanje ili
- ✓ Trenutak kada se desi greška na web strani.

Pored ovih osnovnih podela, mogu se izvršiti i detaljnije koje uključuju događaje za: Network, Websocket, Session History, CSS Transition, Printing, Text Composition, Clipboard, Drag & Drop, Progress... Detaljniji spisak podela na ovaj način može se pogledati na sajtu <https://developer.mozilla.org/en-US/docs/Web/Events>.

Bez obzira na koji način se događaji delili njihov skup je konačan i striktno definisan, kao i način upotrebe. Lista događaja se vremenom menja i proširuje

kako se pojavljuju nove potrebe korisnika i programera, pa i ovu oblast treba aktivno pratiti.

9.2. Vrste događaja

Imena događaja prilično jasno ukazuju akciju koja događaj aktivira. Međutim, ne može se svaki događaj primeniti na svaki element HTML-a, što u početku može delovati kao klasična teorija, ali se vremenom to pokaže kao krajnje logičan skup pravila. Recimo, događaj kojim se detektuje trenutak klika miša od strane korisnika, se zove *onClick*, i može se primeniti na veliki broj elemenata, jer se na najveći broj njih može kliknuti. Sa druge strane, događaj kojim se detektuje promena neke vrednosti, *onChange*, je nelogično primenjivati na npr. fiksnu sliku, ili događaj *onFocus*, kojim se detektuje ulazak u tekstualno polje, primenjivati za klasičan tekstualni naslov. Imena događaja se pojavljuju u dva oblika: sa prefiksom *on* i bez njega. Tako imamo *onClick* i *click*, *onFocus* i *focus* itd. Efekat ovih događaja je isti i sa *on* i bez *on*, ali se za različite vrste implementacije koriste ili jedni ili drugi i to sa vrlo egzaktnim pravilom. U narednom poglavlju detaljno se objašnjavaju načini implementacije, dok će se u ovom akcenat staviti na upoznavanje događaja, i to samo nekih od najkorišćenijih.

Kompletni listinzi događaja se mogu naći u standardu jezika ili na velikom broju sajtova, od kojih su neki:

1. <https://developer.mozilla.org/en-US/docs/Web/Events>,
2. <http://help.dottoro.com/larrqqck.php>,
3. https://www.w3schools.com/jsref/dom_obj_event.asp

Događaji sa prefiksom *on* se koriste kod direktne integracije događaja u HTML i kod primene metoda *attachEvent()*, *detachEvent()* i *fireEvent()*, dok se imena događaja bez prefiksa *on* koriste kod primene metoda *addEventListener()* i *removeEventListener()*.

U narednoj tabeli navešće se događaji, koji u većini slučajeva imaju prefiks *on*, jer je to češći način navođenja, iako se u zavisnosti od implementacija mora voditi računa koja verzija imena koristi. U tabeli su navedeni samo neki od događaja, koji se češće koriste.

Događaj	Opis
animationend	Događaj se realizuje kada se CSS animacija završi
animationstart	Događaj se realizuje kada se CSS animacija pokrene
onabort	Događaj se realizuje kada se prekine učitavanje resursa
onafterprint	Događaj se realizuje kada se započne štampanje stranice

onblur	Događaj se realizuje kada se napusti fokus elementa
onchange	Događaj se realizuje kada se promeni vrednost elementa forme
onclick	Događaj se realizuje kada se klikne na element
oncopy	Događaj se realizuje kada se kopira sadržaj elementa
oncut	Događaj se realizuje kada se realizuje cut opcija sadržaja elementa
ondblclick	Događaj se realizuje kada se dvoklikne na element
ondragend	Događaj se realizuje kada korisnik završi prevlačenje elementa
ondrop	Događaj se realizuje kada se prevučeni element postavi na željeno mesto
onerror	Događaj se realizuje kada se detektuje greška u učitavanju eksternog fajla
onfocus	Događaj se realizuje kada se uđe u fokus (tekstualnog) polja
oninput	Događaj se realizuje kada element dobije korisnički unet sadržaj
onkeypress	Događaj se realizuje kada korisnik pritisne taster na tastaturi
onkeyup	Događaj se realizuje kada korisnik pusti taster na tastaturi
onload	Događaj se realizuje kada se objekat učita
onloadeddata	Događaj se realizuje kada se objekat (media data) učita
onmouseenter	Događaj se realizuje kada korisnik pređe mišem preko nekog elementa
onmouseleave	Događaj se realizuje kada korisnik mišem izađe iz zone nekog elementa
onmouseout	Događaj se realizuje kada korisnik mišem izađe iz zone nekog elementa ili njegovih child-ova
onmouseover	Događaj se realizuje kada korisnik pređe mišem preko nekog elementa ili njegovih child-ova
onoffline	Događaj se realizuje kada browser počne da radi u offline modu
ononline	Događaj se realizuje kada browser počne da radi u online modu
onpaste	Događaj se realizuje kada korisnik iskopira (paste) sadržaj u neki element
onplay	Događaj se realizuje kada se startuje media fajl ili nije u statusu pauze
onplaying	Događaj se realizuje kada se startuje media fajl nakon pauze ili procesa baferovanja

onprogress	Događaj se realizuje kada je browser u procesu preuzimanja (download-ovanja) media fajla
onreset	Događaj se realizuje kada se forma resetuje
onresize	Događaj se realizuje kada se promeni prikaz dokumenta (resize)
onscroll	Događaj se realizuje kada se skroling element skroluje
onsearch	Događaj se realizuje kada korisnik unese sadržaj u polje za pretragu
onselect	Događaj se realizuje nakon što korisnik selektuje neki tekstualni sadržaj u elementu forme
onstorage	Događaj se realizuje kada se Web Storage update-uje
onsubmit	Događaj se realizuje kada se forma submit-uje
ontoggle	Događaj se realizuje kada korisnik otvori i zatvori element details
ontouchend	Događaj se realizuje kada se skloni prst sa touch screen
ontouchstart	Događaj se realizuje kada se prstom dotakne touch screen
preventDefault()	Prekidanjem događaja, radnja koja je dodeljena događaju se neće desiti
stopPropagation()	Sprečava dalju realizaciju događaja dok on traje
transitionend	Događaj se realizuje kada se CSS tranzicija završi

9.3. Implementacija događaja

Događaji se mogu primeniti na nekoliko različitih načina, iako je efekat delovanja skoro uvek identičan. Gruba podela načina implementacije događaja je kroz primenu u

- ✓ HTML-u ili
- ✓ JavaScript-u.

9.3.1. Implementacija događaja u HTML-u

Sintaksom jezika HTML je predviđena upotreba specifičnih atributa, kod pojedinih tagova, koji počinju sa *on*. Tako se događaj klika na neki taster definiše sa *onClick*, ulaskom u fokus elementa sa *onFocus* itd. Prefiks *on* treba da žargonski ukaže na pojavu šta se dešava po kliku na taster, po ulasku u fokus polja itd. Kako su ovo atributi HTML-a, primenjujemo standardnu sintaksu za kreiranje HTML atributa tj. operator = za dodelu vrednosti i znake navoda, unutar kojih se definiše sadržaj atributa. Karakteristično za primenu događaja u formi

HTML-a je što je sadržaj ovih atributa, „direktan ulazak u JavaScript“, i u njemu važi klasična sintaksa skripta.

Ako posmatramo taster, koji je tipa *button*, i koji nema inicijalno definisanu logiku, i njemu dodelimo događaj *onClick()*, imamo mogućnost da sami definišemo šta će se desiti klikom na taster. Ovo se realizuje dodavanjem događaja u taster kao

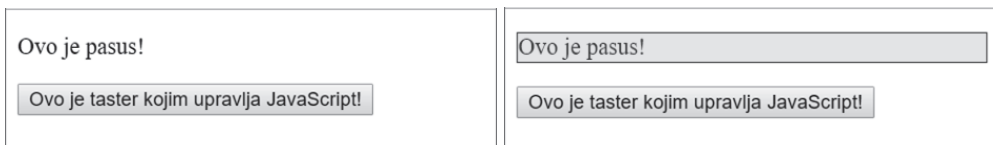
```
onClick="alert('Kliknuli ste na taster');"
```

tj. ako pogledamo ceo kod imamo:

```
<body>
<div id="primer">
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja JavaScript!"
      onClick="alert('Kliknuli ste na taster');"/>
  </form>
</div>
</body>
```

Na ovaj način klikom na taster će se aktivirati metod *alert* i prikazati u posebnom prozoru tekst “Kliknuli ste na taster“. Obim koda koji se može definisati kao vrednost atributa *onClick* nije ograničen, pa u njemu može pisati bilo šta što je sintaksno ispravno sa aspekta JavaScript-a. U sledećem primeru klikom na taster dodaje se stil elementu *p*:

```
<html>
<head>
<style>
.styleDodatni {
  background-color: yellow;
  color: red;
  border: 1px solid red; }
</style>
</head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja JavaScript!"
      onClick="document.getElementById('pasus').classList.add('styleDodatni');"/>
  </form>
</div>
</body>
</html>
```



Slika 9.2. Implementacija događaja click kroz HTML element

Kako obim koda može biti relativno veliki, praktikuje se da se taj kod ipak ne piše unutar HTML-a nego kao eksterni JavaScript kod, koji će se napisati u formi funkcije i ta funkcija pozvati svaki put kada je potrebno da se izvrši. Kod iz prethodnog primera je sada smešten u formu funkcije u eksterni JS fajl a funkcija se poziva klikom na taster.

```
<html>
<head>
<style>
.styleDodatni {
    background-color: yellow;
    color: red;
    border: 1px solid red; }
</style>
<script type="text/javascript" src="eksterni.js">
</head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
<form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
    JavaScript!" onClick="primeni( );"/>
</form>
</div>
</body>
</html>
```

Gde je sadržaj fajla eksterni.js definisan kao:

```
function primeni() {
    document.getElementById('pasus').classList.add('styleDodatni');
}
```

Ukoliko bi se isti kod modifikovao dodavanjem samo jednog tekstualnog polja, i u njemu događaj *onFocus()*, kojim se poziva funkcija *unos()*, taj HTML kod bi trebalo neznatno modifikovati sa:

```
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
```

```

<form action="#">
<input type="text" id="ime" onClick="unos();"/>
    <input type="button" value="Ovo je taster kojim upravlja
        JavaScript!" onClick="primeni();"/>
</form>
</div>
</body>

```

dok se sav ostali kod piše u eksternom fajlu, tj. u ovom slučaju funkcija *unos()* kao:

```

function primeni(){
document.getElementById('pasus').classList.add('styleDodatni');
}
function unos(){
alert("Ovde unesite ime!");
}

```

Dati kod se poziva odmah po ulasku u tekstualno polje, i u ovom slučaju ispisuje tekst u posebnom prozoru.

Kao zaključak bi se moglo zaključiti sledeće: Ukoliko se događaji definišu kroz HTML kod, pišu se kao atributi određenih elemenata. Imena događaja su rezervisane reči i mogu se dodeliti tačno određenim elementima. Kada se definiše atribut koji predstavlja događaj, njegov sadržaj je direktno JavaScript kod koji će se pozvati i izvršiti u trenutku kada se aktivira događaj koji je definisan. Zbog velike količine koda i preglednosti, preporuka je da se kao vrednost atributa definiše funkcija a njen sadržaj piše u eksternom JavaScript kodu.

9.3.2. Implementacija događaja u JavaScript-u

Pored prvog načina integracije događaja kroz HTML, češća je forma integracije kroz JavaScript. Ovakva integracija se, za sada, može uraditi na tri načina, od kojih su dva podržana u velikom broju browser-a, dok treći polako zauzima svoje mesto. Ovi načini su u formi svojstva objekta (npr. za događaj *click*)

```
objekat.onclick = handler;
```

ili metoda selektovanog objekta, kao npr.

```
objekat.addEventListener("click", handler);
objekat.attachEvent("onclick", handler);
```

A) Integracija primenom svojstva: U prvom slučaju, kada se događaj koristi kao svojstvo objekta, imena događaja su sa prefiksom *on*. Selekcija objekta se može uraditi na sve do sada opisane načine, primenom nekog od metoda za selekciju,

ili skraćeno prozivanjem jedinstvenog identifikatora objekta, tj. po vrednosti atributa *id*.

U sledećem primeru, dat je kod gde je svojstvo *onclick*, tj. događaj, dodeljen tasteru, preko identifikatora tastera *id*, i na taj način je selektovan objekat. U komentarima je dat alternativni način selekcije.

```

<html>
<head> </head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
      JavaScript!" id="taster"/>
  </form>
</div>

<script type="text/javascript">
//var objekat = document.getElementById('taster');
// objekat.onclick = function() {
  taster.onclick = function() {
    document.getElementById('pasus').style.display="none";
  };
</script>
</body>
</html>

```

U datom kodu korišćena je takozvana “bezimena funkcija” tj. kao handler, iza znaka jednakosti, koristi se oznaka za funkciju *function() { }*. Ova funkcija nema ime jer se nigde u kodu eksplicitno ne poziva, nego se telo ove funkcije podrazumevano izvršava u liniji koda u kojoj se događaj realizuje. Na ovaj način se događaju može dodeliti proizvoljan broj linija koda, koji treba da se realizuju.

B) Integracija primenom metoda `addEventListener()` : Trenutno najkorišćeniji način je upotreba metoda *`addEventListener()`*. Prvi argument ovog metoda ukazuje na događaj, ali se piše bez prefiksa *on*, dok drugi ukazuje na radnju koju treba uraditi. Ovde treba biti pažljiv, u smislu šta i kako se definiše šta je željena radnja. Ako se kao radnja napiše kao linija koda, koja je u skladu sa sintaksom JavaScript-a,

```

taster.addEventListener("click",document.getElementById('pasus').style
.display="none");

```

ona će se jednokratno realizovati u trenutku učitavanja web strane, kao da nema događaja koji njome upravlja, i dalje će događaj, ili naredni klik na taster, biti neupotrebljiv.

Ovo se još bolje može pokazati kada se kao *handler* definiše metod *alert()*. Po učitavanju strane, on će se realizovati, bez klika na taster, i nakon toga svaki naredni klik neće reagovati.

```
<body>
<div id="primer">
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
      JavaScript!" id="taster"/>
  </form>
</div>
<script type="text/javascript">
  taster.addEventListener("click", alert("Proba"));
</script>
</body>
```

Iz tog razloga se kao *handler* najčešće poziva funkcija, koja može imati veći broj linija koda, ali i tu treba voditi računa. Ukoliko se funkcija ispravno definiše i pozove, kao *handler*, na regularan način pisanjem imena funkcije i (), ovo će se protumačiti kao pozivanje funkcije u kodu, bez obzira na događaj, i ona će se realizovati, jednokratno po učitavanju, i dalje će događaj biti neupotrebljiv.

Zato se kod poziva funkcije u ovakvim situacijama, funkcija poziva po imenu, bez upotrebe zagrada. Tako kod pišemo kao:

```
<!DOCTYPE html>
<html>
<head> </head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
JavaScript!" id="taster"/>
  </form>
</div>
<script type="text/javascript">
function izvrsi(){
  alert("ok");
  document.getElementById('pasus').style.display="none";
}
document.getElementById('taster').addEventListener("click", izvrsi);
// taster.addEventListener("click", izvrsi);
</script>
</body>
</html>
```

Selekcija objekta se i u ovom slučaju može raditi pomoću klasičnih metoda za selekciju, u ovom slučaju korišćen je *getElementById('taster')* ili samo po atributu *id*, tj. *taster*, u liniji koda koja je zakomentarisana.

Možda je za početnike najbolje da se i u ovom slučaju koristi “bezimena funkcija”, i da se problem zagrada i pozivanja funkcija uniformiše za oba načina primene kod realizacije događaja kroz JavaScript. Ako bi se koristila “bezimena funkcija”, prethodni kod bi se zapisao na sledeći način:

```
<html>
<head> </head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
  <form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
      JavaScript!" id="taster"/>
  </form>
</div>
<script type="text/javascript">
document.getElementById('taster').addEventListener("click",
function(){ alert("ok");
              document.getElementById('pasus').style.display="none";
});
</script> </body> </html>
```

ili kod skraćenog oblika selekcije, pozivanjem identifikatora *taster*:

```
<script type="text/javascript">
taster.addEventListener("click", function(){
  alert("ok");
  document.getElementById('pasus').style.display="none";
});
</script>
```

Jedna od bitnih pogodnosti upotrebe metoda *addEventListener()* je što se dinamički, JavaScript-om, nekom selektovanom elementu dodaje događaj i radnja koja treba da se realizuje. Često se u realnim primenama nameće potreba da se u nekom trenutku taj dodeljeni događaj ukloni i da se sve vrati u početno stanje. Ovo se realizuje metodom *removeEventListener()*. Primenom ovog metoda, moguće je bilo koji dodeljeni događaj ukloniti, nakon čega se isti ili drugi opet može dodati. Ako se prethodni kod modifikuje, tako da se pozivanjem funkcije *izvrsi()*, prikaže poruka *ok*, i ukloni *Listener*, narednim klikom na taster više se neće pozivati funkcija *izvrsi()*.

```
<script type="text/javascript">
function izvrsi(){
  alert("ok");
```

```

        taster.removeEventListener("click", izvrsi);
    }
    taster.addEventListener("click", izvrsi);
</script>

```

Opisana dva metoda se često koriste u varijanti sa objektom događaja, koji je argument bezimene funkcije. Ovaj objekat ima svoja svojstva koja mogu biti vrlo korisna u različitim situacijama. Ovde će se pokazati primer upotrebe objekta *event*, a može biti i drugo ime, za detekciju kliknutog tastera na mišu.

```

<html>
<head> </head>
<body>
<div id="primer">
<p id="pasus">Ovo je pasus!</p>
<form action="#">
    <input type="button" value="Ovo je taster kojim upravlja
        JavaScript!" id="taster" />
</form>
</div>
<script>
taster.addEventListener("mousedown", function(event) {
    if (event.which == 1)
        alert("Kliknut levi taster");
    else if (event.which == 2)
        alert("Kliknut srednji taster");
    else if (event.which == 3)
        alert("Kliknut desni taster");
    });
</script>
</body>
</html>

```

Upotreba metoda *addEventListener()* je podržana u svim browser-ima, i treba forsirati njegovu upotrebu kako zbog JavaScript-a tako i zbog vrlo slične sintakse koja će se kasnije koristiti u jQuery-ju.

C) *Integracija primenom metoda attachEvent()*: Ovaj metod je novijeg datuma i do sada ga podržava samo nekoliko browser-a, pa o tome treba voditi računa. Sintaksa primene ovog metoda je jako slična metodi *addEventListener()*. Prvi argument je događaj, ali sada sa prefiksom *on*, dok je drugi *handler*. U sledećem primeru dat je primer integracije metoda *attachEvent()*.

```

<script>
document.getElementById('taster').attachEvent("onclick", function() {
    alert("Novi tekst");
});
</script>

```

9.4. Primeri implementacija događaja

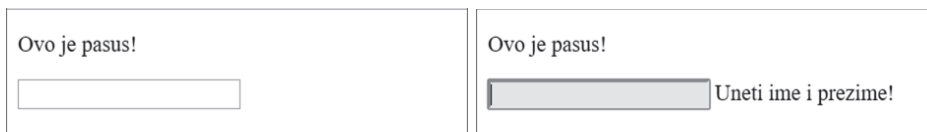
Primer 1. Realizacija promene slike prilikom prelaska miša preko iste, u formi realizacije događaja u HTML-u.

```
<script>
function promena(kojaSlika){
    document.getElementById("slikaSajt").src= kojaSlika + ".jpg";
}
</script>
</head>
<body>

</body>
```

Primer 2. Realizacija prikaza teksta i promene pozadinske boje text polja, ulaskom u fokus polja, i vraćanje na početno stanje izlaskom iz fokusa polja. Primena događaja *focus* i *blur* je realizovana metodom *addEventListener()*.

```
<html>
<head> </head>
<body>
<div id="okvir">
<p id="pasus">Ovo je pasus!</p>
<form action="#">
    <input type="text" id="primer" />
    <span id="opis"></span>
</form>
</div>
<script>
    document.getElementById("primer").addEventListener("focus",
function() {
    document.getElementById("primer").style.backgroundColor = "yellow";
    document.getElementById("opis").textContent = "Uneti ime i prezime!";
});
    document.getElementById("primer").addEventListener("blur",
function() {
    document.getElementById("primer").style.backgroundColor = "white";
    document.getElementById("opis").textContent = "";
});
</script>
</body>
</html>
```



Slika 9.3. Primer programiranja interakcije sa elementom forme

Primer 3. Realizacija učitavanja druge stranice pomoću događaja *click*, realizovanog kroz HTML.

```
<body>
<div id="okvir">
    <button onclick="window.location='3.html';">Link ka stranici
    3.html</button>
</div>
</body>
```

Primer 4. Realizacija detekcije promene izbora stavke u drop-down meniju, pomoću događaja *change*, realizovanog događajem *addEventListener()*.

```
<body>
<div id="okvir">
<p id="pasus">Ovo je pasus!</p>
    <form action="#">
        <select id="lista">
            <option>Stavka 1</option>
            <option>Stavka 2</option>
        </select>
        <span id="opis"></span>
    </form>
</div>
<script>
    document.getElementById("lista").addEventListener("change",
function() { document.getElementById("opis").textContent = "Promena"; });
</script>
</body>
```

Primer 5. Realizacija pozdravne poruke pomoću događaja *onLoad* u tagu *body*.

```
<body onLoad = "pozdrav();">
<div>
    <p id="pasus">Ovo je pasus!</p>
</div>
<script>
    function pozdrav(){ alert("Dobrodosli na sajt!"); }
</script>
</body>
```

Primer 6. Grafički prikaz pozicije prilikom skrolovanja stranice.

```

<html>
<head>
<style>
  #okvir {border: 1px solid red;
          width: 400px;
          position: fixed;
          top: 0px; right: 0px; }
  #prikaz {height: 15px;
           background: green;
           width: 0%; }
  body { height: 2500px; }
</style>
</head>
<body>
<div id="okvir">
  <div id="prikaz"></div>
</div>
<h1>Skrolovanje stranice</h1>
<script>
  var bar = document.getElementById("prikaz");
  addEventListener("scroll", function() {
    var max = document.body.scrollHeight - innerHeight;
    var percent = (pageYOffset / max) * 100;
    bar.style.width = percent + "%";
  });
</script>
</body>
</html>

```

Primer 7. Primer kreiranja novog prozora i popune sadržaja istog, sadržajem definisane web strane, klikom na taster, uz mogućnost da se klikom na drugi taster otvoreni prozor zatvori.

```

<html>
<head> </head>
<body>
<div id="okvir">
  <div id="prikaz"></div>
</div>
<button onClick="otvaranjeNovogProzora();">Klik za otvaranje novog
  prozora</button> <br/>
<button onClick="zatvaranjeNovogProzora();">Klik za zatvaranje novog
  prozora</button>
<script type="text/javascript">
  var prozor;
  function otvaranjeNovogProzora(){

```

```

    prozor=window.open("novi.html", "height=300px, width=200px")
  }
  function zatvaranjeNovogProzora(){
    prozor.close()
  }
</script>
</body>
</html>

```

9.5. Rad sa tajmerima

Pored događaja, JavaScript nudi mogućnost rada sa tajmerima. Tajmer omogućava odloženo izvršavanje neke funkcije za definisano vreme. Vreme se definiše u milisekundama. Nakon isteka definisanog vremena, funkcija unutar tajmera se realizuje i korisnik vidi efekat tog koda. U JavaScript-u se najčešće koriste dva metoda, objekta window, koji predstavljaju tajmere:

1. `setTimeout()` / `clearTimeout()`
2. `setInterval()` / `clearInterval()`

Razlika između ova dva metoda je što se metod *setTimeout()* izvršava samo jednom, i nakon toga se automatski zaustavlja, dok se *setInterval()* izvršava neograničeno mnogo puta, sve dok se ne zaustavi. Metode `clearTimeout()` i `clearInterval()` koristimo za brisanje tj. zaustavljanje rada tajmera. Sintaksa upotrebe ovih metoda je:

```
setTimeout( handler, time)
```

Na isti način kao i u prethodnim slučajevima, handler je radnja koja se izvršava, ali tek nakon što prođe definisano vreme od trenutka pokretanja ove linije koda. Handler može biti JavaScript kod koji se direktno piše u metodu `setTimeout`, kao:

```
setTimeout("alert('Dobar dan!')", 3000);
```

kada će se korisniku prikazati prozor sa tekstom, ali nakon 3 sekunde (3s = 3000ms). Takođe može se koristiti „bezimena funkcija“ kao

```
setTimeout(function() {alert('Dobar dan!')}, 3000);
```

ili se pozvati klasična funkcija koje je negde u kodu posebno definisana

```

function izvrsi(){
    alert('Dobar dan!');
}
setTimeout(function izvrsi(), 3000);

```

U sledećem primeru metod `setTimeout()` je iskorišćen za prikaz štoperice. Inicijalnim učitavanjem koda, ne dešava se ništa. Klikom na taster Broji, poziva se funkcija brojanje, koja ispisuje vrednost promenljive sekunde, u ovom slučaju 0, u tekstualno polje. Nakon toga, povećava vrednost promenljive sekunde na 1 i aktivira tajmer, koji će za 1s pozvati funkciju brojanje. Ovo je klasična rekurzija, gde funkcija poziva samu sebe, ali sa odloženim periodom od 1s. Ponovnim pozivanjem funkcije, sve se ponavlja iz početka. Ispisuje se vrednost promenljive sekunde, tj. sada 1, povećava promenljiva na vrednost 2 i ponovo poziva funkcija putem tajmera. Ovakvo pozivanje je neophodno jer se koristi metod `setTimeout()`, koji se nakon pozivanja i izvršavanja zaustavlja, pa se mora ponovo pozivati, i to najčešće na ovaj način rekurzijom. Ova štoperica će tako na svakih sekund pozivati funkciju koja će ispisati vrednost promenljive koja je za 1 veća nego prethodne sekunde i tako simulirati brojanje tj. tok vremena. Klikom na taster *Zaustavi*, poziva se funkcija `zaustavi()`, koja „zaustavlja“ tj. briše definisani tajmer. Iz tog razloga prilikom kreiranja tajmera, tj. pozivanjem metoda `setTimeout()` se on dodeli nekoj promenljivoj, u ovom slučaju promenljivoj tajmer, i kažemo da smo mu tako dodelili jednoznačno ime. Na ovaj način moguće je imati proizvoljan broj tajmera, jer će svaki imati svoje ime, i svaki po želji pokrenuti tj. zaustaviti kao u datom primeru pomoću

```
clearTimeout(tajmer)
```

Funkcija koja se poziva unutar tajmera, tj. obim njenog koda i njene funkcionalnosti, nisu ni na koji način ograničene upotrebom tajmera.

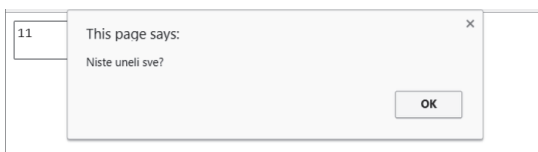
```
<html>
<head>
<script type="text/javascript">
var sekunde = 0;
var tajmer;
function brojanje(){
    document.getElementById('prikaz').value = sekunde;
    sekunde = sekunde + 1;
    tajmer = setTimeout("brojanje()",1000);
}
function zaustavi() {clearTimeout(tajmer);}
</script>
</head>
<body> <form>
    <input type="text" id="prikaz"> <br/>
    <input type="button" value="Broji" onClick="brojanje()" />
    <input type="button" value="Zaustavi" onClick="zaustavi()" />
</form> </body>
</html>
```

Na sličan način upotrebom metoda *setInterval()* kreirano je ispisivanje ocena kroz konzolu, počev od vrednosti 5 pa do 10. Da bi se broj 10 ispisao, proverava se da li se stiglo do broja 11 i tada se tajmer prekida tj. briše.

```
<html>
<head>
<script type="text/javascript">
var broj = 5;
var ocena = setInterval(function() {
    console.log("Ocena može biti: ", broj++);
    if (broj == 11) {
        clearInterval(ocena);
        console.log("Ne može bolje!");
    }
}, 2000);
</script>
</head>
<body> </body>
</html>
```

Message	Source
Ocena može biti: 5	2.html:6
Ocena može biti: 6	2.html:6
Ocena može biti: 7	2.html:6
Ocena može biti: 8	2.html:6
Ocena može biti: 9	2.html:6
Ocena može biti: 10	2.html:6
Ne može bolje!	2.html:9

Slika 9.4. Primer upotrebe metoda *clearInterval()*



Slika 9.5. Primer upotrebe metoda *setTimeout()*

Upotreba tajmera je najčešće u kombinaciji sa događajima, kada se na bazi neke akcije korisnika pokreće tajmer da bi se odradila neka druga isprogramirana radnja. U sledećem primeru, unosom karaktera unutar tekstualnog polja, poziva se funkcija koja aktivira tajmer unutar koga se kontroliše da li je uneti sadržaj bar 13 karaktera, i ako nije čeka se 1 sekund od poslednjeg unosa i korisniku ispisuje poruka, kao podsetnik da treba da nastavi dalji unos.

```
<body>
    <textarea>Unesite Vas JMBG</textarea>
<script>
var textarea = document.querySelector("textarea");
var tajmer;
textarea.addEventListener("keydown", function() {
    clearTimeout(tajmer);
    var sadrzaj = document.querySelector("textarea").value.length;
```

```
if (sadržaj<13){  
    tajmer = setTimeout(function() {alert('Niste uneli sve?');}, 1000); }  
});  
</script>  
</body>
```

U realnim uslovima, rad sa klijentskim jezicima se najčešće vezuje za rad sa događajima i mogućnosti browser-a da detektuju širok spektar korisničkih aktivnosti. U ovom delu su obrađeni najkorišćeniji događaji i akcenat je stavljen na njihovu upotrebu i mogućnosti. U zavisnosti koje konkretne aktivnosti se očekuju u pojedinim trenucima programerima je ostavljena mogućnost da ih koriste na najrazličitije načine. Primarni cilj ovog poglavlja je da korisnika upozna sa pojmom događaja i njihovom primenom, jer je implementacija svih događaja vrlo slična. Kod koji će se pozvati nakon aktiviranja nekog događaja je ono što se piše primenom klasičnih elemenata jezika JavaScript i može biti bilo koji kod sa bilo kojim funkcionalnostima, nevezano od događaja koji ih je pozvao. Iz tog razloga, događaje treba doživeti kao fleksibilan način unapređenja korisničke aplikacije, i ne treba se vezivati za događaje koji u ovom trenutku postoje, jer se i spisak događaja vremenom menja. Bitno je razumeti suštinu i način primene i uživati u svim mogućnostima koje nam događaji nude.

X RAD SA ELEMENTIMA FORME

Interakcija sa korisničkim sadržajem

FORMA predstavlja jedini način da korisnik unese i prosledi bilo kakav podatak web aplikaciji ili web serveru. Ovi podaci se mogu koristiti za trenutne obrade ili analize ali se češće upisuju u bazu podataka i trajno pamte. Svaka registracije, logovanje, ostavljanje komentara ili postova, izbor grada, pola ili sl. se realizuje kroz formu. Realizacija forme pripada HTML-u i ovde se neće posebno analizirati. Upotrebom samo HTML-a, elementi forme će se kreirati i korisnik može unositi podatke, čekirati, selektovati i sl. ali će svi ti podaci nestati zatvaranjem browsera. Da bi se podaci zapamtili potrebna je baza podataka a sa njom može da komunicira samo serverski jezik, npr. PHP. Podaci iz forme se mogu automatski poslati nekoj PHP stranici na serveru, tako što se u tagu *form*, definiše atribut *action* čija vrednost je PHP stranica kojoj se podaci automatski šalju nakon submit-ovanja forme. Submit-ovanje podrazumeva da postoji taster tipa *submit* i da korisnik klikne na taj taster. Tako ako imamo formu sa jednim tekstualnim poljem i jednim tasterom, kao u narednom kodu, i unesemo sadržaj *Milan*, taj podatak se direktno šalje stranici *obrada.php* putem URL adrese, slika10.1.

```

<html>
<head> </head>
<body>
<form method="GET" action="obrada.php">
    <input type="text" id="ime" name="ime"> <br/>
    <input type="submit" value="Slanje" name="taster"/>
</form>
</body>
</html>

```



The image shows a simple web form. It consists of a rectangular box containing a text input field and a submit button. The text input field has the word 'Milan' entered into it. Below the input field is a button with the text 'Slanje' on it.

Slika 10.1. Korisnički sadržaj u elementu forme pre slanja podataka prema serveru

U ovoj formi je izabran način slanja GET metodom, da bi se videlo šta se tačno prenosi kroz URL adresu. Kako stranica *obrada.php* ne postoji, prijaviće se greška, ali će podaci iz forme biti prosleđeni putem URL adrese na sledeći način:

```
file:///C:/Users/Desktop/obrada.php?ime=Milan&taster=Slanje
```

Iz ovoga se vidi da će browser, nakon submit-ovanja forme, sve elemente forme, u ovom slučaju text polje i taster, prepoznati po atributu *name*, i vrednost tog atributa proslediti serverskoj strani uz vrednost atributa *value*.

Tako se za tekstualno polje šalje *ime=Milan* a za taster *taster=Slanje*. Svi podaci se šalju iza simbola *?* koji odvaja ime stranice kojoj se podaci šalju od podataka, dok se podaci međusobno razdvajaju sa *&*. Nakon dobijanja ovih podataka server ove podatke treba da prihvati, i dalje sa njima nešto uradi, i na kraju ih eventualno upiše u bazu podataka.

Sve što je do sada opisano ne pominje JavaScript ni u jednoj fazi rada, pa se postavlja pitanje gde je njegova primena? Odgovor je: Na više mesta i to više nego preporučljivo.

Pre svega, primarni cilj je da se sve što treba, i što se može, uradi na strani klijenta, tj. na korisničkom računaru, a to je jedino moguće upotrebom JavaScript-a. To podrazumeva da se svi podaci iz forme, tj. browser-a, pre slanja serveru, prihvate u promenljivima JavaScript-a i dalje sa njima postupa kao da su tu inicijalno i kreirane. To dalje podrazumeva da se izvrše sve provere, u smislu da li su popunjena određena polja, ili kliknuti neki obavezni tasteri, da li je pored toga što je nešto upisano, uneti sadržaj u skladu sa definisanim pravilima (definisana dužina teksta, velika slova, upotreba cifara, ispravne mail adrese itd.). Zatim se ti podaci dodatno analiziraju, obrade i tek kada se sve uspešno proveriti i pripremi, to se pošalje serveru. Na taj način se server „štedi“ od svih ovih provera i vraćanja

podataka nazad klijentu, ako se ispostavi greška ili nepravilnost, i za sve to se koristi korisnički računar. Dodatno, pored toga što će se ispravni podaci poslati serveru, a loši vratiti korisniku sa instrukcijama kako da se poprave, JavaScript može pomoću AJAX-a da komunicira sa web serverom „u pozadini“ rada sajta, i na taj način šalje i prima podatke koje interaktivno prikazuje korisniku bez potrebe da se stranica ponovo učitava.

Na bazi ovih informacija, zaključujemo da se nakon napisanog HTML koda, sve provere, obrade informacija, priprema, slanje i dalja komunikacija sa serverom obavlja JavaScript-om. U daljem tekstu će se pokazati kako se uneti podaci u HTML formu dohvataju i postaju promenljive JavaScript-a da bi se nad njima primenjivale sve željene operacije, metodi ili svojstva ovog jezika.

Napredne provere unetog sadržaja regulisaće se primenom regularnih izraza, koji će se obraditi u drugom delu ovog poglavlja i koji predstavljaju izuzetno bitan segment rada u realnim kodovima.

10.1. Rad sa elementima forme

Svaki element forme je element DOM-a i pristupa mu se pomoću metoda koje su objašnjene u prethodnim poglavljima. Bez obzira kako se izvrši selekcija, programer pomoću svojstava tog objekta dobija željene informacije i podatke o onome što je korisnik uneo ili kliknuo u formi.

Najkorišćenija su svojstva kojima se čitaju vrednosti atributa (name, id, value, checked) kao i univerzalna svojstva kao što je length.

Bitno je naglasiti da će se pomoću linije koda:

```
var x=document.getElementById("element_id").value;
```

u promenljivu *x* smestiti vrednost atributa *value* iz selektovanog objekta dok će u slučaju:

```
document.getElementById("element_id").value=y
```

vrednost promenljive *y* biti dodeljena vrednosti atributa *value*.

Ako se iz HTML-a zna da se kod tekstualnog polja, u atributu *value*, nalazi ono što je uneo korisnik ili inicijalno programer, kažemo da je u *value* korisnikov sadržaj. Tako u prvoj liniji koda, u promenljivu *x*, u JavaScript-u, skladištimo ono što je korisnik uneo u formi u HTML-u, i time smo sve dalje prebacili na teren JavaScript-a. U drugoj liniji koda, neku vrednost iz JavaScript-a, promenljivu *y*, postavljamo kao vrednost atributa *value*, što znači da kodom popunjavamo sadržaj tog polja umesto korisnika.

10.1.1. Rad sa tekstualnim poljima

U HTML-u postoji nekoliko tipova tekstualnih polja (text, password, file, textarea) i sa svima njima se radi na skoro identičan način. Ako pretpostavimo da imamo formu sa samo jednim tekstualnim poljem i tasterom, klikom na koji se poziva funkcija JavaScript-a, tada imamo:

```
<form method="GET" action="#" name="formular1">
<input type="text" id="ime" name="ime" value="Uneti ime"> <br/>
<input type="button" value="Slanje" onClick="provera()"/>
</form>
```

Selekciju objekta DOM-a, u ovom slučaju elementa forme, možemo uraditi na nekoliko ranije opisanih načina, od kojih su neki:

```
var ime = document.getElementById("ime");
//var ime = document.querySelector("#ime");
```

ali i još jednim načinom kojim se direktno pristupa elementima po strukturi DOM-a, baziranih na vrednosti atributa name. Tako u dokumentu, pronalazimo element sa atributom name=formular1, što je element form, pa unutar njega element sa name=ime, što je tekstualno polje koje želimo da selektujemo.

```
var ime = document.formular1.ime;
```

Bez obzira koji od metoda za selekciju se odabere, selektovani objekat je smešten u promenljivu ime.

```
var ime = document.getElementById("ime");
//var ime = document.querySelector("#ime");
//var ime = document.formular1.ime;
```

U zavisnosti šta se od selektovanog objekta želi dohvatiti koristimo različita svojstva ovog objekta. Ukoliko želimo vrednost atributa name pišemo kao

```
var naziv = ime.name;
```

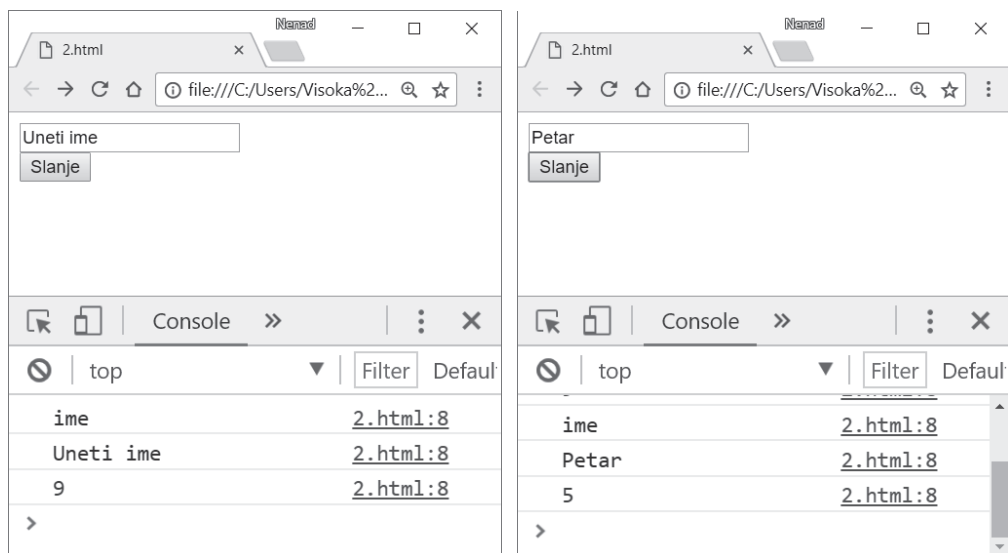
ako želimo konačni sadržaj tekstualnog polja pišemo kao

```
var sadrzaj = ime.value;
```

dok ako želimo dužinu unetog sadržaja pišemo kao

```
var duzina = sadrzaj.length;
```

Ako sve ove vrednosti ispišemo pre promene sadržaja od strane korisnika, klikom na taster, kao i nakon toga, imamo:



Slika 10.2. Prikaz rada sa različitim svojstvima tekstualnih polja

Vidi se da se kao sadržaj atributa *value* prosleđuje sadržaj text polja neposredno pre klika na taster, bez obzira da li je to inicijalni tekst iz HTML-a ili kasnije unet korisnički tekst.

10.1.2. Rad sa checkbox-ovima

Rad sa checkbox-ovima se realizuje na isti način i sa istim svojstvima kao i tekstualnim poljima, s tim što se u praksi najčešće koristi svojstvo *checked* koje ne postoji kod tekstualnih polja. Ovo svojstvo vraća true ili false u zavisnosti da li je checkbox čekiran ili ne. Svaki checkbox je zaseban jer se svaki ponaosob može čekirati ili odčekirati, i svaki se definiše svojim identifikatorom da bi mu se jednoznačno pristupilo. Ako se posmatra sledeći kod

```
<html>
<head> </head>
<body>
<form method="GET" action="#" name="formular1">
  Check box 1<input type="checkbox" name="ch1" value="nesto1"
  id="cBox1" /> <br/>
  Check box 2<input type="checkbox" name="ch2" value="nesto2"
  id="cBox2" checked="checked" />
  <input type="button" value="Slanje" onClick="provera();" />
</form>

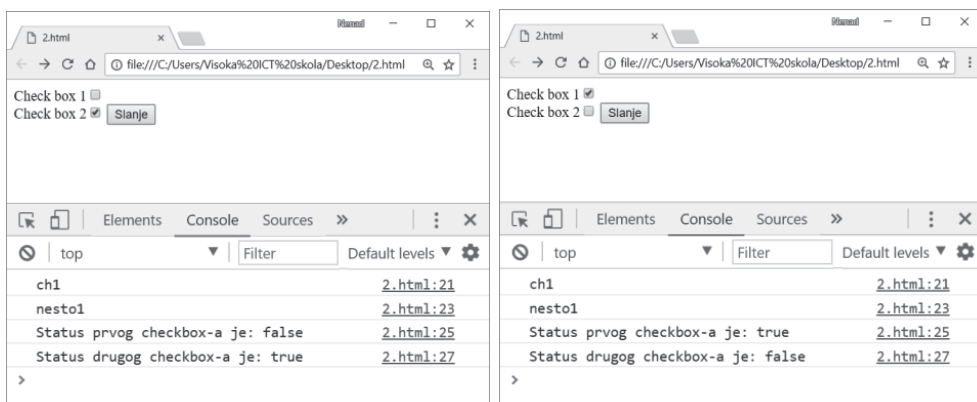
<script type="text/javascript">
function provera(){
var chBox1 = document.getElementById("cBox1");
var chBox2 = document.getElementById("cBox2");
```

```

//var chBox1 = document.querySelector("#cBox1");
//var chBox1 = document.formular1.ch1;
var naziv = chBox1.name;
console.log(naziv);
var sadrzaj = chBox1.value;
console.log(sadrzaj);
var status1 = chBox1.checked;
console.log("Status prvog checkbox-a je: " + status1);
var status2 = chBox2.checked;
console.log("Status drugog checkbox-a je: " + status2);
}
</script>
</body>
</html>

```

i u inicijalnom stanju klikne na taster i pročitaju vrednosti svojstava, videće se da je kod prvog tastera svojstvo `checked` sa vrednošću `false` a kod drugog `true`. Ukoliko se prvi čekira a drugi odčekira, dobija se obrnuta situacija u skladu sa čekiranim stanjem.



Slika 10.3. Prikaz rada sa različitim svojstvima checkbox elemenata forme

Često se u praksi sreće situacija da se checkbox-ovima daju iste vrednosti atributa `name`, i na taj način se napravi grupa elemenata u formi niza. Svaki od njih i dalje ima jedinstven `id`, i može mu se jedinstveno pristupiti ili stilizovati, ali se selekcija uprošćava na način da se jednom linijom koda dohvate svi checkbox-ovi iz željene grupe. Primer kada se dva checkbox-a kreiraju sa istim `name` atributom, a pristupa se svakom ponaosob, u strukturi niza je data u sledećem primeru:

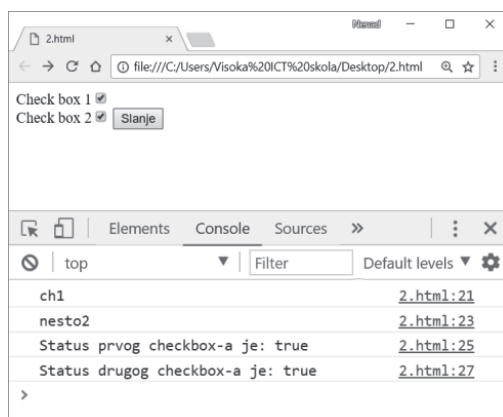
```

<body>
<form method="GET" action="#" name="formular1">
  Check box 1<input type="checkbox" name="ch1" value="nesto1"
id="cBox1" /> <br/>
  Check box 2<input type="checkbox" name="ch1" value="nesto2"
id="cBox2" checked="checked" />

```

```
<input type="button" value="Slanje" onClick="provera();" />
</form>
```

```
<script type="text/javascript">
function provera(){
//var chBox1 = document.getElementById("cBox1");
//var chBox2 = document.getElementById("cBox2");
//var chBox1 = document.querySelector("#cBox1");
var chBoxAll = document.formular1.ch1;
var naziv = chBoxAll[0].name;
console.log(naziv);
var sadrzaj = chBoxAll[1].value;
console.log(sadrzaj);
var status1 = chBoxAll[0].checked;
console.log("Status prvog checkbox-a je: " + status1);
var status2 = chBoxAll[1].checked;
console.log("Status drugog checkbox-a je: " + status2);
}
</script>
</body>
```



Slika 10.4. Prikaz rada sa različitim svojstvima checkbox polja kada im se pristupa kao grupi

10.1.3. Rad sa radio tasterima

Radio tasteri se od checkbox-ova razlikuju samo po tome što svaki checkbox može biti čekiran u svakom trenutku dok može biti samo jedan iz grupe radio tastera, u jednom trenutku. Da bi se formirala grupa radio tastera, svim elementima se definiše ista vrednost atributa name. Tako različiti elementi pripadaju istoj grupi i selekcija jednog automatski deselektuje selekciju prethodnog, iz te grupe.

Radio tasteri, kao HTML elementi, se selektuju na isti način kao i checkbox-ovi. To može biti pomoću pojedinačne selekcije preko nekog jedinstvenog atributa,

npr. *id*, ili preko atributa *name*. Ređe se praktikuje selekcija preko *id*, sem u situacijama kada se tačno zna broj radio tastera i njihovi identifikatori, npr. zaposlen/nezaposlen, oženjen/neoženjen i sl. U situacijama kada se tasteri dodaju programski tj. dinamički na bazi informacija iz baze podataka, i kada se njihov broj ne zna, češće se pristupa selekciji preko nekog zbirnog selektora, kao što je *name*. Ovo se realizuje kao pristup formi i *name* atributu grupe. Ako je *name* atribut svih elemenata u radio grupe *rb*, tada je selekcija svih:

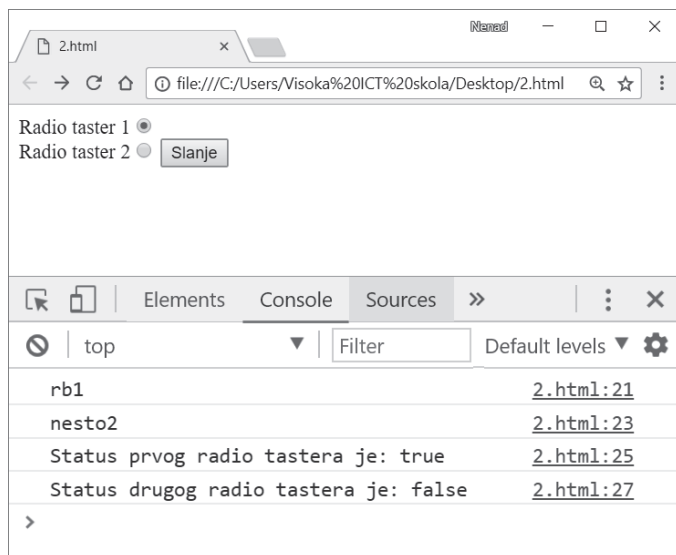
```
var rbAll = document.formular1.rb;
```

Kako se ovakvim selektorom selektuje veći broj elemenata struktura promenljive *rbAll* će biti niz, svi statusi radio tastera, unutar grupe, će biti automatski smešteni u niz. Pristup elementima niza je pristup statusu da li je taster čekiran ili ne. Redosled smeštanja informacija o tasterima u niz je redosled kako su radio tasteri pisani u HTML kodu. Posmatrajmo sledeći kod:

```
<html>
<head> </head>
<body>
<form method="GET" action="#" name="formular1">
Radio taster 1<input type="radio" name="rb" value="nesto1" id="rb1" />
<br/>
Radio taster 2<input type="radio" name="rb" value="nesto2" id="rb2"
checked="checked" />
<input type="button" value="Slanje" onClick="provera();" />
</form>

<script type="text/javascript">
function provera(){
//var rb1 = document.getElementById("rb1");
//var rb2 = document.getElementById("rb2");
var rbAll = document.formular1.rb;
var id = rbAll[0].id;
console.log(id);
var sadrzaj = rbAll[1].value;
console.log(sadrzaj);
var status1 = rbAll[0].checked;
console.log("Status prvog radio tastera je: " + status1);
var status2 = rbAll[1].checked;
console.log("Status drugog radio tastera je: " + status2);
}
</script>
</body>
</html>
```

Data su dva radio tastera i taster, klikom na koji se poziva JavaScript funkcija. Ukoliko se promeni inicijalna selekcija, i čekira prvi radio taster, dobija se sledeći rezultat:



Slika 10.5. Prikaz rada sa različitim svojstvima radio tastera

Ovakav način lako dohvata informaciju o onome šta je korisnik čekirao kod radio tastera, ali u većim grupama može doći do težeg razumevanja koji je čekiran, jer to npr. može biti 12 element niza, a nije potpuno jasno koji je to taster. Zato se često unutar if kontrole toka ispituje koji taster je čekiran i kada se pronade za njega se dohvata vrednost atributa value, koja je za korisnika nevidljiva. Tako se u atributu value piše neki logičan tekst koji je razumljiv korisniku i programeru, obzirom da tekst koji je ispred ili iza grafike radio tastera je vidljiv samo korisniku u trenutku gledanja forme.

```

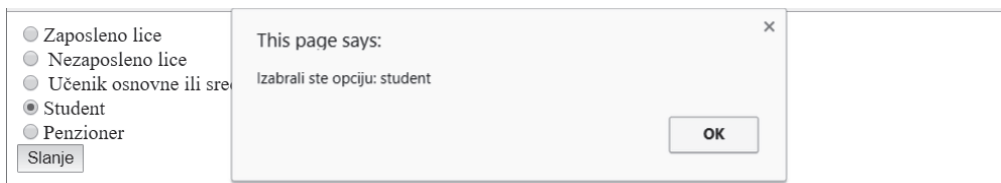
<body>
<form method="GET" action="#" name="formular1">
  <input type="radio" name="rb" value="zaposlen" id="rb1" />Zaposleno
  lice
  <br/>
  <input type="radio" name="rb" value="nezaposlen" id="rb2" />
  Nezaposleno lice<br/>
  <input type="radio" name="rb" value="ucenik" id="rb3" /> Učenik
  osnovne ili srednje škole <br/>
  <input type="radio" name="rb" value="student" id="rb4" />Student
  <br/>
  <input type="radio" name="rb" value="penzioner" id="rb5" />Penzioner
  <br/>
  <input type="button" value="Slanje" onClick="provera();" />
</form>

```

```

<script type="text/javascript">
function provera(){
    var rbAll = document.formular1.rb;
    for (var i=0; i<rbAll.length; i++){
        if (rbAll[i].checked){
            alert ("Izabrali ste opciju: " + rbAll[i].value);
            break;}
    }
}
</script>
</body>

```



Slika 10.6. Prikaz rada sa različitim svojstvima radio tastera kada im se pristupa u celini

10.1.4. Rad sa dropdown listama

Rad sa dropdown listama je malo drugačiji nego rad sa drugim elementima forme. Razlog se nalazi u tome što je lista definisana od najmanje dva elementa, *select* i *option*, i što dodatno *option*-a može biti više. Čim ih ima više, logično je da je rezultat njihove selekcije neki niz, i da pristupanju elementima niza pristupa stanjima elemenata *option*. U JavaScript-u se, po hijerarhiji, prvo pristupa roditeljskom elementu *select*, pa njegovoj deci *option*. Selekcija elementa *select* se može uraditi na bilo koji od opisanih načina, npr:

```
var ddlPlacanje = document.getElementById("lista");
```

Pristup svim *option* elementima selektovanog elementa *select* je pomoću *options*:

```
var x= ddlPlacanje.options[1].text;
var y= ddlPlacanje.options[1].value;
```

Kako se primenom *options* selektuju svi elementi *option*, potrebno je samo pristupiti nekom od njih u nizu. U prethodnom kodu se pristupa drugom elementu *option* u nizu, pomoću `ddlPlacanje.options[1]`. Primenom svojstva *value*, čita se vrednost atributa *value* koji se nalazi u selektovanom elementu *option*. Primenom svojstva *text* čita se vidljivi sadržaj za korisnika tj. sadržaj selektovanog elementa *option* (tekst između taga `<option>` i `</option>`).

Ako se posmatra sledeća lista:

```
<html>
<head> </head>
```

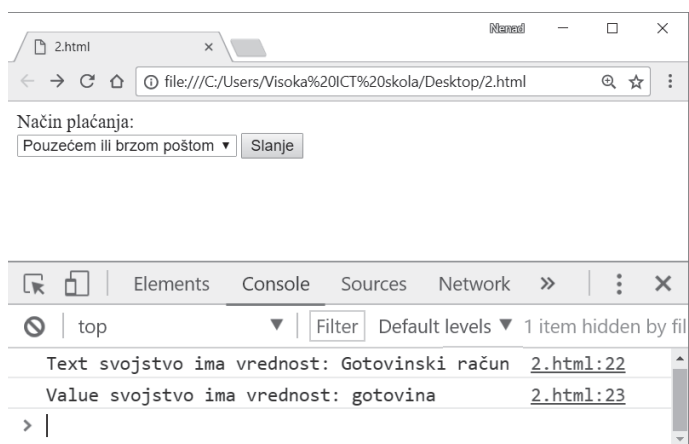
```

<body>
<form method="GET" action="#" name="formular1">
Način plaćanja:<br/>
<select name="placanje" id="lista">
    <option value="izaberite">Izaberite</option>
    <option value="gotovina">Gotovinski račun</option>
    <option value="kartica"> Kartica Komercijalne banke</option>
    <option value="pouzece"> Pouzećem ili brzom poštom</option>
</select>
<input type="button" value="Slanje" onClick="provera();" />
</form>

<script type="text/javascript">
function provera(){
    var ddlPlacanje = document.getElementById("lista");
    var x=ddlPlacanje.options[1].text;
    var y=ddlPlacanje.options[1].value;
    console.log("Text svojstvo ima vrednost: " + x);
    console.log("Value svojstvo ima vrednost: " + y);
}
</script>
</body>
</html>

```

i izabere opcija „Pouzećem ili brzom poštom“, a ispiše vrednost `options[1].text`; tj. `options[1].value`; dobiće se rezultat kao na slici.



Slika 10.7. Prikaz rada sa različitim svojstvima dropdown liste

Naime, kod ispisuje Gotovinski račun tj. gotovina, što je druga po redu stavka u listi, ali nije ono što je korisnik izabrao. Na ovaj način se fiksno pristupilo nekom elementu, ali bez vođenja računa šta je stvarno selektovano od strane korisnika. Da bi se detektovao korisnikov izbor, koristi se svojstvo *selectedIndex*. Ovo svojstvo vraća broj stavke tj. poziciju selektovanog elementa option. Tako ako korisnik izabere stavku Izaberite, vratiće se 0, dok ako izabere Kartica

Komercijalne banke vratiće se 2. Na ovaj način se dinamički dobija informacija o tome šta je korisnik izabrao, i taj podatak je potrebno staviti umesto ranije fiksno upisanog broja 1 u kodu

```
var x= ddlPlacanje.options[1].text;
var y= ddlPlacanje.options[1].value;
```

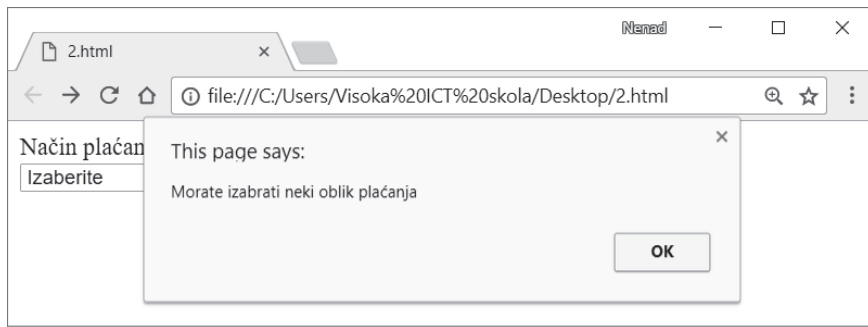
Sada taj kod postaje

```
var ddlPlacanje = document.getElementById("lista");
var x=ddlPlacanje.options[ddlPlacanje.selectedIndex].text;
var y=ddlPlacanje.options[ddlPlacanje.selectedIndex].value;
```

Na ovaj način od koda se traži da u promenljive *x* i *y* dohvati vrednost svojstva *text* i *value* od onog *option*-a koji je selektovao korisnik (podatak o tome šta je selektovao je u *ddlPlacanje.selectedIndex*, a informacija o *text* i *value* se dohvata iz selekcije elementa kao: *ddlPlacanje.options[ddlPlacanje.selectedIndex]*).

Često se korisnik uslovljava da mora da izabere neku vrednost drugačiju od Izaberite, i pomoću svojstva *selectedIndex* to se lako može uraditi ispitivanjem da li je korisnik izabrao prvu stavku po redosledu pojavljivanja tj. stavku kojoj je *selectedIndex=0*. U sledećem kodu u promenljivoj selektovani se dohvata redni broj selektovanog option-a, i ukoliko je 0, što znači stavka Izaberite, prijavljuje se greška, dok u suprotnom treba nešto drugo uraditi.

```
<script type="text/javascript">
function provera(){
    var ddlPlacanje = document.getElementById("lista");
    var selektovani=ddlPlacanje.selectedIndex;
    if (selektovani != 0){ // ispravno }
    else{ alert("Morate izabrati neki oblik plaćanja"); }
    }
</script>
```



Slika 10.8. Provera da li je korisnik ostavio inicijalnu vrednost u dropdown listi

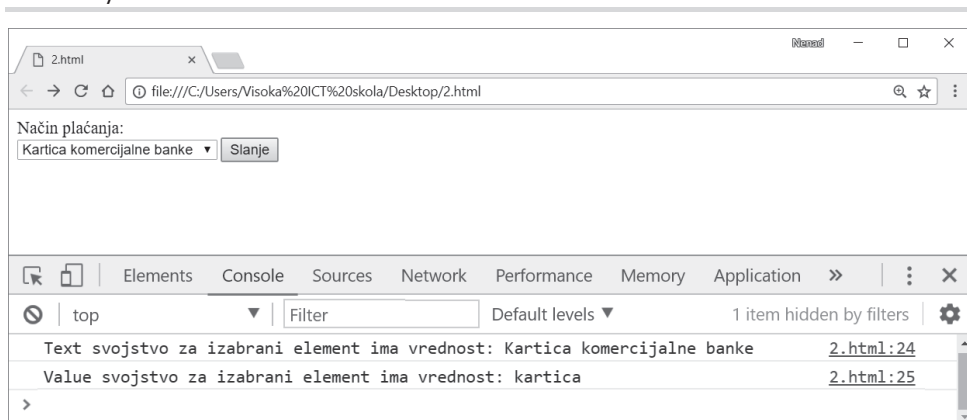
Ako se unutar *if* kontrole toka sada umesto komentara upiše kod kojim se dohvata i ispisuje *text* i *value* svojstvo selektovanog elementa imamo kompletan kod kao

```

<html>
<head> </head>
<body>
<form method="GET" action="#" name="formular1">
Način plaćanja:<br/>
<select name="placanje" id="lista">
    <option value="izaberite">Izaberite</option>
    <option value="gotovina">Gotovinski račun</option>
    <option value="kartica"> Kartica komercijalne banke</option>
    <option value="pouzece"> Pouzećem ili brzom poštom</option>
</select>
<input type="button" value="Slanje" onClick="provera();"/>
</form>

<script type="text/javascript">
function provera(){
    var ddlPlacanje = document.getElementById("lista");
    var selektovani=ddlPlacanje.selectedIndex;
    if (selektovani != 0){
        var x=ddlPlacanje.options[ddlPlacanje.selectedIndex].text;
        var y=ddlPlacanje.options[ddlPlacanje.selectedIndex].value;
        console.log("Text svojstvo za izabrani element ima vrednost: " + x);
        console.log("Value svojstvo za izabrani element ima vrednost: " + y);
    }
    else{ alert("Morate izabrati neki oblik plaćanja"); }
}
</script>
</body>
</html>

```



Slika 10.9. Dohvatanje dostupnih podataka iz izabranog elementa dropdown liste

U datom kodu selekcija liste je urađena pomoću atributa id, dok je bilo moguće da se uradi i na bilo koji drugi način. Ako bi se radila preko name atributa onda bi se modifikovala linija koda za selekciju kao:

```
var ddlPlacanje = document.formular1.placanje;
```

Pored ovog načina moguće je koristiti i alternativni, za rad sa atributom name kao:

```
var ddlPlacanje = document.forms["formular1"]["placanje"];
```

na ovaj način se koristi rad sa multidimenzionim nizom, gde je jednom elementu niza, ["formular1"], smešten novi niz, sa svom svojom decom elementima kao što je u ovom slučaju lista ["placanje"].

10.1.5. Rad sa tasterima

Tasteri su poslednja grupa elemenata forme, koja nije obrađena, a koja se obično previdi kada se pomisli na elemente forme. Postoji ukupno tri tipa tastera u HTML-u: submit, reset i button. Taster tipa submit automatski šalje sve podatke iz forme ka stranici koja je definisana u atributu action. Na ovaj način, ugrađena logika šalje podatke serveru, ali onakve kakve je uneo korisnik. Tj., ako korisnik ništa ne upiše i submit-uje formu, i to će se kao prazna polja proslediti serveru.

Programerima, sa druge strane, treba slanje podataka, ali sa nekim vidom odloženog dejstva, tj. tek nakon provere i zaključka da je sve što je bilo potrebno popunjeno kako treba. Tako nam treba taster tipa submit da bi poslao podatke, ili sa druge strane da ih baš ne pošalje odmah, što zvuči kao kontradikcija. Međutim, ovaj bitan zadatak odrađuje događaj onSubmit, koji se definiše u tagu form. Ovaj događaj se aktivira nakon submit-ovanja forme, i ako u njemu pozovemo funkciju u JavaScript-u, npr. provera(), prvo će se izvršiti kod funkcije, a nakon toga će se nastaviti sa procesom slanja. Ovo je prikazano u kodu koji sledi:

```
<html>
<head> </head>
<body>
<form method="GET" action="obrada.php" name="formular1"
onSubmit="provera();">
  <input type="text" id="ime" name="ime" value="Uneti ime" /> <br/>
  <input type="submit" value="Slanje"/>
</form>

<script type="text/javascript">
function provera(){
alert("Proba");
}
</script>
</body>
</html>
```

Na ovaj način se umeće neki JavaScript kod pre postupka slanja, ali je problem što ako se u tom kodu zaključi da postoji problem, to neće sprečiti nastavak slanja pogrešnih ili nepotpunih podataka ka serveru. Ovo se dodatno rešava postavljanjem rezervisane reči *return* ispred imena funkcije, prilikom njenog poziva, tj.

```
<form method="GET" action="obrada.php" name="formular1"
onSubmit="return provera();">
```

Na ovaj način slanje podataka će biti prekinuto pozivanjem funkcije *provera* u JavaScript-u, ali se nakon završetka JavaScript koda neće nastaviti slanje sve dok se od funkcije ne vrati *true*, kao oblik „zelenog svetla“ za nastavak rada. Time se postiže da ukoliko *provera* ustanovi problem sa korisničkim podacima, funkcija vrati *false*, i korisniku se slanje obustavlja. Vraćanjem *true*, slanje se nastavlja.

Ako u prethodnom kodu proverimo da li je korisnik upisao nešto u tekstualno polje, tj. da li je dužina unetog sadržaja veća od 0, slanje nastavljamo dok u suprotnom ispisujemo poruku o grešci.

The image shows a web form interface. It consists of a rectangular text input field. To the right of this field, the text "Morate uneti ime!" is displayed. Below the input field is a button labeled "Slanje". The entire form is enclosed in a thin black border.

Slika 10.10. Upotreba *onSubmit()* događaja za prekidanje slanja podataka web serveru

Primenom *return*, i presecanjem slanja podataka serveru, svi podaci se prvo preusmeravaju u JavaScript, pa se primenom klijentskih alata proveravaju, i tek kada sve bude zadovoljavajuće šalju se serveru. Na sličan način se radi sa tasterom tipa *reset* i događajem *onReset()*.

Svi tipovi tastera, kao i svi drugi elementi, prilikom slanja prenose podatak o *name* i *value* atributu. Tako ako u web strani ima više tastera, moguće je jednoznačno zaključiti na koji taster se kliknulo. Jedan od načina je da se klikom na svaki taster poziva ista funkcija, ali da se prilikom poziva funkcije prosledi *this*, tj. pokazivač na tekući objekat, tj. pokazivač na element unutar koga se kliknulo. Na taj način, iako puno tastera može postojati i svi pozivati istu funkciju, funkciji se prosleđuju podaci elementa iz koga je kliknuto. U ovom primeru ispisuje se *value* vrednost tastera na koji se kliknulo.

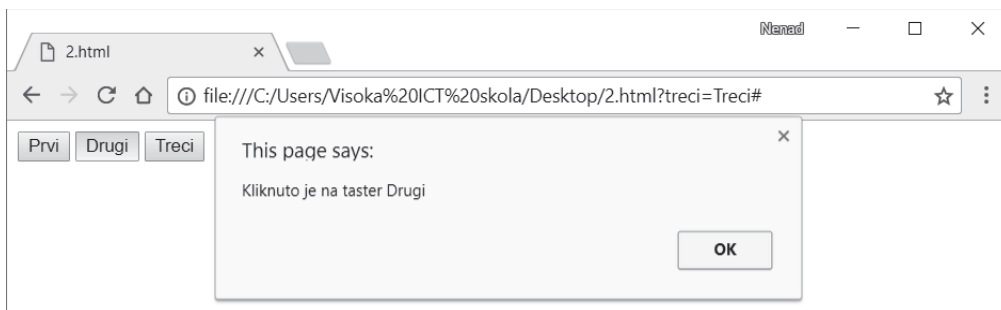
```
<html>
<head> </head>
<body>
  <form method="GET" action="#" name="formular1" >
    <input type="button" name="prvi" value="Prvi"
onClick="provera(this);" />
    <input type="button" name="drugi" value="Drugi"
onClick="provera(this);" />
```

```

        <input type="submit" name="treci" value="Treci" onClick=
"provera(this);"/>
    </form>

    <script type="text/javascript">
    function provera(taster){ alert("Kliknuto je na taster " + taster.value); }
    </script>
    </body>
    </html>

```



Slika 10.11. Prikaz rada sa svojstvima tastera

10.2. Regularni izrazi

Prilikom popunjavanja forme korisnik, koji unosi sadržaj sa tastature, može uneti proizvoljnu kombinaciju slova, brojeva, specijalnih karaktera i sl. Da li je korisnik uneo neki sadržaj, lako se može proveriti izračunavanjem dužine sadržaja nekog elementa forme kao:

```

var sadrzaj = document.getElementById('ime').value;
if (sadrzaj.length == 0){
    alert("Ništa nije uneto u polje za ime!");
}

```

Slično, ako se zna da je potrebno uneti tačno određeni broj karaktera, npr. 13 za JMBG, tada bi provera bila

```

if (sadrzaj.length == 13)

```

Međutim, svaki drugi oblik provera u smislu unetog sadržaja, postaje mnogo komplikovan. Baš takva provera, provera unetog sadržaja, je ključna za programera i celu aplikaciju. Tako je vrlo bitno da li je kod unetog sadržaja za JMBG, pored dužine od 13 karaktera, unet skup samo cifara. Slično, da li je kod unosa imena unet samo skup slova, da li je prvo slovo veliko a ostala mala, da li je mail adresa odgovarajućeg formata, da li je broj telefona ili registarskih tablica u dobrom obliku i sl. Sve ove provere se vrlo lako rešavaju primenom regularnih

izraza, kojima se može definisati i tip i dužina i struktura sadržaja koji se očekuje od strane korisnika.

Regularni izrazi se koriste nezavisno od jezika u kome se primenjuju, i mogu se koristiti i na klijentskoj i na serverskoj strani. Opšta pravila su ista za sve primene, ali se neki vrlo specifični nivo detalja kao i imena metoda vezuju za konkretne jezike u koje se ovaj kod piše.

Regularni izrazi (*regular expression*) predstavljaju objekte kojima se definiše skup pravila kojima programer želi da proveriti uneti korisnički sadržaj. Pravila mogu da definišu dužinu unetog sadržaja, upotrebu karaktera, kombinacije dozvoljenih karaktera i sl. Kada se definiše, jedan regularni izraz se može primeniti neograničeno mnogo puta za proveru unetog sadržaja. Provera podrazumeva da se nekom od metoda jezika, proveriti da li je uneti korisnički sadržaj u skladu sa definisanim pravilima u regularnom izrazu. Najčešće se kao rezultat dobija *true* ili *false*, kao znak da jeste ili nije u skladu sa definisanim pravilom. Svaki regularni izraz, se smešta u jednu promenljivu i definiše jedan skup pravila za proveru sadržaja. Najčešće se u kodu nalazi veći broj regularnih izraza, jer svaki od njih kontroliše neki drugi sadržaj (npr. jedan izraz za proveru imena, drugi za proveru telefona i treći za količinu). Regularni izraz se može sastojati od dva dela: *paterna* i *modifikatora*. *Patern* ima ključnu ulogu u radu regularnog izraza i definiše konkretan skup pravila za proveru. *Modifikator* je opciona varijanta kojom se može delimično prilagoditi tj. preciznije definisati način rada regularnog izraza.

Da bi se razlikovali od klasičnih promenljivih u JavaScript-u, *patern* regularnog izraza se piše između kosih crta (/). Tako se sintaksa može definisati kao:

```
var regularnilzraz = /patern/modifikator;
```

ili ako nema modifikatora kao

```
var regularnilzraz = /patern/;
```

Pored ovog zapisa, moguće je pozvati konstruktor objekta *RegExp* kao

```
var regularnilzraz = new RegExp('patern');
```

Ključni posao kod regularnih izraza je definisanje *paterna*. Da bi se on kreirao potrebno je znati pravila koja su dozvoljena za njegovo pisanje.

Patern treba da definiše koje karaktere korisnik može uneti, u kom redosledu i uolikoj dužini. Zato kažemo da je pisanje *paterna* definisano „klasičnim“ karakterima, karakterima sa posebnim značenjem i kvantifikatorima. Klasični karakteri predstavljaju one karaktere koji su dozvoljeni za unos, npr. 1, A, III, é, s itd. Karakteri sa posebnim značenjem definišu skraćene oblike pojedinih skupova, ili pravila po kojim se karakteri mogu grupisati, pisati i sl.

Kvantifikatori definišu broj i način ponavljanja određenog karaktera ili grupe karaktera.

Krenimo redom: ukoliko želimo da dozvolimo da korisnik unese neki od karaktera a, b, c, d ili e to zapisujemo upotrebom uglaste zagrade kao

[abcde]

Na ovaj način je dozvoljeno da korisnik unese bilo koji od karaktera koji su u zagradi, npr *b* je dozvoljen dok *f* nije. Ukoliko se želi definisati veći skup, mogu se nabrojati svi dozvoljeni karakteri, koji ne moraju biti samo slova, kao npr.

[abcdefghijklmn13459DSQšđž]

Kako se često koriste manje više standardni skupovi karaktera, npr. velika slova, ili mala slova, ili sve cifre, uvedene su i skraćeni oblici zapisa, da se ne bi svaki put navodila sva slova azbuke. Tako imamo zvanične skraćenice:

[0-9] - cifre od 0 do 9
 [A-Z] - velika slova od A do Z
 [a-z] - mala slova od a do z
 [A-Z] - velika i mala slova

Ovako definisani skupovi karaktera se uvek mogu proširiti sa još dozvoljenih karaktera kao u slučaju dodavanja srpskih slova, kada je dozvoljeno da se unese bilo koje malo slovo engleskog pisma ali i neko od slova č, ć, ž, š ili đ.

[a-zćčžšđ]

Treba naglasiti da upotrebom uglaste zagrade definišemo skup dozvoljenih karaktera i to samo jednog od ponuđenog skupa. Ako želimo da dozvolimo više njih iz tog skupa, potrebno je koristiti kvantifikator što će kasnije biti objašnjeno.

Upotreba uglaste zagrade je proširena i specijalnim simbolima. Jedan od najkorišćenijih je ^ (kapa). Ukoliko se ovaj simbol nađe na početku sadržaja uglaste zagrade, on označava negaciju. Tako patern

[abcd]

znači bilo koje od slova a, b, c ili d, tj. bilo koji karakter unutar uglastih zagrada, dok

[^abcd]

znači bilo šta što nije a, b, c ili d, tj. bilo koji karakter koji je **nije unutar** uglastih zagrada. Na sličan način, ako se želi zabraniti upotreba karaktera 8 i 9 pišemo

[^89]

Pored uglaste, i obična zagrada ima svoje značenje i koristi se za grupisanje skupova pravila od kojih se primenjuje samo jedno. Ovo je sinonim za logičku operaciju ili tj. disjunkciju i zapisuje se kao:

$(x|y)$

Ovim je dozvoljeno da se upotrebi ili x ili y. Naravno da umesto korišćenih x i y može biti bilo koji složeniji oblik. Tako ako dozvolimo da neko upiše bilo koje veliko ili malo slovo iz skupa od a do f, pišemo

$([a-f]|[A-F])$

Navođenjem željenih karaktera van uglaste zagrade, ne definiše se skup nego reč u celini. Tako ako želimo da definišemo dozvoljene formate za uploadovani multimedijalni fajl pišemo:

$(swf|mov|wma|mpg|mp3|wav)$

Na ovaj način dozvoljeno je uneti reč wav, ali ne i war. Efekat stavljanja u uglaste zagrade npr. $[wav]$ bi imao potpuno drugo značenje i ukazivao da je dozvoljeno koristiti karakter w, a ili v.

Kako se vremenom pokazala potreba da se što kraće zapisuju pojedini karakteristični skupovi pravila, definisani su dodatni specijalni karakteri, metakarakter, kojima se u još kraćem obliku definišu karakteristični skupovi karaktera.

$\backslash w$ - traženje samo slova

$\backslash W$ - traženje ne slovnih karaktera

$\backslash d$ - traženje cifara

$\backslash D$ - traženje ne-cifarskih karaktera

$\backslash s$ - traženje razmaka (space)

$\backslash S$ - traženje grupe karaktera bez razmaka

$\backslash b$ - traženje preklapanja na početku ili kraju reči

$\backslash B$ - traženje preklapanja bilo gde sem na početku ili kraju reči

$\backslash n$ - traženje karaktera za novi red

$\backslash r$ - traženje karaktera za povratak (return)

$\backslash t$ - traženje tab karaktera

Upotrebom ovih specijalnih skraćenica, ukoliko želimo da dozvolimo unos velikih slova, malih slova i cifara, pišemo:

$[A-Za-z\backslash d]$ ili još kraće $[\backslash w\backslash d]$.

Na do sada opisane načine, programer može definisati karakter koji želi da dozvoli prilikom provere unetog sadržaja. Da bi se pravilo, koje se definiše za jedan karakter, multipliciralo i da bi se omogućila provera većeg broja unetih karaktera potrebni su kvantifikatori. Oni omogućavaju da se neko pravilo primeni veći broj puta. Kvantifikatori se mogu definisati posebnim simbolima tj. pomoću +, * ili ?, ali i pomoću upotrebe vitičastih zagrada. Kvantifikator se piše iza karaktera, ili grupe karaktera, na koji se želi primeniti. U zavisnosti koji se kvantifikator koristi postiže se sledeći efekat:

+ *jedno ili više pojavljivanja*

* *nula ili više pojavljivanja*

? *nula ili jedno pojavljivanje*

Ukoliko se sada posmatra skup slova i cirafa, koji je dozvoljavao da se pojavi samo jedan od dozvoljenih karaktera, uz modifikator +

$[\backslash w \backslash d]^+$

Omogućeno je da se unese najmanje jedan ili proizvoljno mnogo karaktera koji moraju biti slova ili cifre. Kvantifikatori +, * ili ? mogu delovati vrlo grubo, ali se često koriste u situacijama kada se ne zna maksimalna dužina unetog sadržaja. Ukoliko je dužina poznata, koriste se vitičaste zagrade koje dosta preciznije mogu limitirati dužinu unetog sadržaja. Ove zagrade se mogu koristiti u tri oblika, u zavisnosti koliko argumenata imaju. Tako se pomoću zagrada definiše:

$\{n\}$ *n pojavljivanja*

$\{n, m\}$ *od n do m pojavljivanja*

$\{n, \}$ *najmanje n pojavljivanja*

Ukoliko bi se sada definisalo pravilo za unos JMBG-a, koji ima tačno 13 cifara, pisali bi:

$[\backslash d]\{13\}$

dok bi se unos dana u nedelji, koji može biti samo slovo, minimalne dužine 5 (sreda) i maksimalne 9 (ponedeljak) pisao kao:

$[\backslash w]\{5,9\}$

Ako pretpostavimo da je potrebno definisati patern za unos imena, koje mora početi velikim slovom, a iza njega imati samo mala slova, i ako pretpostavimo da je najkraće ime 2 a najduže 12 karaktera, takav patern bi bio:

$[A-Z\check{C}\acute{C}\check{Z}\check{S}\check{D}][a-z]\{1,11\}$

Ovim se iz prve uglaste zgrade omogućava da prvo slovo mora biti veliko, i to samo jedno, jer nema kvantifikatora iz uglaste zgrade, dok iz skupa malih slova

mora biti izabrano bar jedan a maksimalno jedanaest karaktera. Ukoliko se želi definisati jedan patern za ime i prezime, i uz pretpostavku da i ime i prezime moraju početi velikim slovom, da iza njega imaju samo slova, i da je to u opsegu od 2 do 12 karaktera, patern se može napisati na ovaj način:

[A-ZČĆŽŠĐ] [a-zčćžšđ] {1,11} (\s[A-ZČĆŽŠĐ] [a-zčćžšđ] {1,11})+

Prvi deo paternna omogućava unos imena, npr. Petar, dok drugi deo zbog kvantifikatora + mora da se realizuje bar jednom, pa se može uneti Petar Petrović. Na sličan način, + dozvoljava da se drugi deo paternna ponovi i više puta pa je i uneti sadržaj Ana Marija Ćirić, u skladu sa paternom.

Pored navedenih, često se koriste i karakteri ^ i \$, kojima se ukazuje da li provera pravila u paternu treba da se radi na početku, kraju ili na celokupno unetom korisničkom sadržaju.

abc\$ proverava pojavljivanje stringa abc na kraju posmatranog sadržaja

^abc proverava pojavljivanje stringa abc na početku posmatranog

^abc\$ proverava pojavljivanje stringa abc kao celokupnog sadržaja

Ovde treba dobro obratiti pažnju na simbol ^ koji je kod upotreba sa uglastim zagradama predstavljao negaciju, a sada oznaku za proveru od početka unetog sadržaja. Primenom ovih kvantifikatora, lako se može napisati patern koji kontroliše da li je uneto ime proizvoljno slike npr. jpg formata, tj. da li se završava sa .jpg

/[a-zA-Z0-9-_\.] + (.jpg)\$ /

Ukoliko bi se dozvolilo da se unese ime fajla, koji može biti jedan od više tipova:

/[a-zA-Z0-9-_\.] + \.(swf|mov|wma|mpg|mp3|wav)\$ /

Tada bi se taj skup definisao unutar obične zgrade sa karakterom |, kojim se dozvoljava samo jedan iz skupa, a sa kvantifikatorom \$ se postiže da ime može biti bilo šta, ali se mora završiti jednim od definisanih tipova. Kako se pre tipa mora pojaviti . i da se ne bi svaki put pisala ispred imena tipa, izdvojena je pre zgrade, što znači da je obavezna.

Tačka ima vrlo specifično značenje, tj. predstavlja džoker znak i može kao takav upravljački znak da zameni bilo koji karakter koji korisnik unese. Kako to nije željeno stanje u ovom slučaju, ispred tačke se stavlja, ranije objašnjen simbol \, kojim se neutrališe upravljačko svojstvo tačke i ona se tada tumači kao karakter koji se uneo sa tastature kao tačka.

Pored ovih postoje još dva bitna kvantifikatora,

? = abc pronalazi sve stringove iza kojih se nalazi abc

? ! = abc pronalazi sve stringove iza kojih se ne nalazi abc

Primenom ovih kvantifikatora moguće je locirati string *ne*, na mestima gde iza njega nije *space* /ne(?!)/

```
var recenica = "Nikada ne ostavljajte neku nerešenu situaciju za sutra!";
```

dok ako bi se tražila reč *ne* kao negacija, tj. da iz nje mora biti *space*, pisali bi kao

```
/ne(=?)/
```

Ako bi se želelo locirati *ne* u reči **nerešenu**, naveli bi da iza njega treba da bude karakter *r*, kao:

```
/ne(=?r)/
```

Nakon definisanja *paterna*, potrebno je primeniti neki od dozvoljenih metoda kojima se *patern* primenjuje na neki string, koji je najčešće dobijen od korisnika. Metodi su podeljeni prema tome da li se primenjuju na sintaksu definisanja *paterna* kao string ili kao *RegExp()* metod.

U slučaju kada se *patern* definiše kao string, mogu se koristiti:

match(x) - koristi jedan argument (*patern*) i vraća sve substring-ove (definisane *paternom*) koji se nalaze u analiziranom stringu ili vraća *null* ako ne može da se pronađe ni jedan,

search(x) - koristi jedan argument (*patern*) za pretragu sadržaja analiziranog stringa. Vraća poziciju prvog karaktera, ako je nađen sadržaj definisan *paternom*, ili -1 ako nije pronađen,

replace(x, y) - koristi dva argumenta, pri čemu pronalazi sadržaj definisan prvim argumentom (*patern*) i menja ga sadržajem drugog argumenta

split(x) - koristi jedan argument (*patern*) i deli originalni string na delove (format niza), u odnosu na definisani argument

U slučaju kada se *patern* definiše kao *RegExp* metod, kao i u slučaju definisanja *paterna* kao string, mogu se koristiti:

test(x) - koristi jedan argument (analizirani string) i proverava da li se taj argument poklapa sa pravilom u *paternu*. Kao rezultat vraća *true* ako je pronađeno preklapanje, odnosno *false*.

exec(x) - koristi jedan argument (analizirani string) i pronalazi preklapanja *paterna* sa argumentom. Kao rezultat vraća niz pronađenih stringova definisanih *paternom* ili *null* ako nema preklapanja.

Primer primene regularnog izraza dat je u sledećem primeru. U HTML delu definisana je forma sa jednim tekstualnim poljem, i u njemu događaj *onBlur()* kojim se poziva funkcija *provera*, i njoj prosleđuje uneti sadržaj u tekstualnom polju. Kada korisnik unese neki sadržaj, i izađe iz tekstualnog polja, poziva se funkcija i prosleđuje se uneti sadržaj. Unutar funkcije definisan je *patern* uzorak,

u kome je definisano da je dozvoljen unos velikih slova, malih slova i cifara, u minimalnoj dužini 6 i maksimalnoj 8 karaktera. Korišćeni metod *test()* kao argument koristi string koji treba da analizira u skladu sa pravilom definisanom u paternu. Ukoliko se potvrdi da je pravilo paterna zadovoljeno vraća se *true*, i zato je pogodno da se to postavi u if kontrolu toka. U slučaju da je sve dobro, u span elementu, neposredno iza tekstualnog polja, upisuje se tekst „Dobro je!“, dok se u suprotnom aktivira *else* deo i ispisuje tekst „Nije dobro!“.

```

<html>
<head> </head>
<body>
<form method="GET" action="#" name="formular1" >
    <input type="text" name="tekst" onBlur="provera(this.value);"/>
    <span id="greska"></span>
</form>
<script type="text/javascript">
function provera(x){
    var uzorak = /^[A-Za-z\d]{6,8}$/;
    if (uzorak.test(x)) {
        document.getElementById('greska').innerHTML = "Dobro je!";
    }
    else {
        document.getElementById('greska').innerHTML = "Nije dobro!";
    }
}
</script>
</body>
</html>

```

Ukoliko se sada unese sadržaj *Pera*, prijaviće se greška, jer je deo sadržaja dobar ali dužina nije bar 6 karaktera. Isto je slučaj i sa sadržajem *Pera Peric*, koji je predugačak.

Pera	Nije dobro!	Pera Peric	Nije dobro!
-----------------------------------	-------------	---	-------------

Slika 10.12. Prikaz primene regularnog izraza kada se unosi sadržaj pogrešne dužine

Ako se sada upiše *Pera123*, to će biti u skladu sa paternom.

Pera123	Dobro je!
--------------------------------------	-----------

Slika 10.13. Prikaz primene regularnog izraza kada se unose regularan sadržaj

U slučaju da korisnik unese nedozvoljeni sadržaj, čak i u ispravnoj dužini, prijaviće se greška jer nije u skladu sa paternom.

**/---

Nije dobro!

Slika 10.14. Prikaz primene regularnog izraza kada se unosi pogrešan sadržaj

Primer upotrebe metoda *exec()* se može posmatrati ako se u prethodnom kodu promeni deo JavaScript-a sa:

```
<script type="text/javascript">
function proverax(x){
    var uzorak = /proba/;
    document.getElementById('greska').innerHTML = uzorak.exec(x); }
</script>
```

Metod *exec()* pronalazi unutar njegovog argumenta sadržaj koji je definisan paternom. Ako ga pronađe u datom kodu ga ispisuje, dok ako ga nema, neće prikazati ništa. U prvom primeru se reč *proba* pronalazi i prikazuje dok u drugom, kada se izbriše slovo *a*, i ostane reč *prob*, neće doći do pronalaženja reči *proba* i neće se prikazati ništa.

Ovo je proba izraza!

proba

Ovo je prob izraza!

Slika 10.15. Prikaz primene regularnog izraza sa metodom *exec()*

Metod *search()* pronalazi početnu poziciju paterna unutar analiziranog stringa. U slučaju da je patern definisan kao *proba*, pronalazi se pozicija početka ove reči tj. slovo *p*, unutar analiziranog stringa. U sledećem primeru prikazuje se kod i rezultat delovanja metoda *search()* kroz prikaz pronađene pozicije, u polju neposredno pored tekstualnog polja za unos sadržaja.

```
<script type="text/javascript">
function proverax(x){
    var uzorak = /proba/;
    document.getElementById('greska').innerHTML = x.search(uzorak);
}
</script>
```

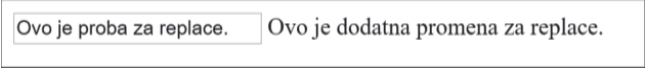
Ovo je proba za search.

7

Slika 10.16. Prikaz primene regularnog izraza sa metodom *search()*

Metod *replace()* je karakterističan za veliki broj programskih jezika i služi za zamenu jednog stringa drugim unutar nekog analiziranog stringa. U sledećem primeru paternom je definisan string *proba* koji primenom metoda *replace()* treba pronaći i zameniti sa stringom *dodatna promena*.

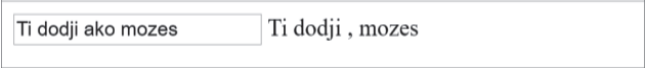
```
<script type="text/javascript">
function provera(x){
    var uzorak = /proba/;
    document.getElementById('greska').innerHTML = x.replace(uzorak,
"dodatna promena");
}
</script>
```



Slika 10.17. Prikaz primene regularnog izraza sa metodom `replace()`

Metod `split()` se vrlo često koristi u radu sa stringovima jer omogućava deljenje analiziranog stringa na podstringove koji se automatski smeštaju u niz. Ova podela se vrši u odnosu na argument metoda `split()`, pri čemu on ne ostaje u konačnom rezultatu, već samo delovi stringa ispred i iza definisanog argumenta.

```
<script type="text/javascript">
function provera(x){
    var uzorak = /ako/;
    document.getElementById('greska').innerHTML = x.split(uzorak);
}
</script>
```



Slika 10.18. Prikaz primene regularnog izraza sa metodom `split()`

Metod `match()` se najčešće koristi kada se inicijalno objašnjava rad sa regularnim izrazima. Ovaj metod pronalazi sve delove analiziranog stringa koji se poklapaju sa paternom. Ukoliko se patern definiše bez modifikatora, onda se pronalazi samo prvi string koji se poklapa sa paternom. Ukoliko se pak koriste modifikatori, na raspolaganju su:

i koji omogućava preklapanje bez obzira na velika i mala slova

g koji omogućava globalno preklapanje, tj. pronalaženje svih elemenata koji se preklapaju a ne samo prvog

m koji omogućava preklapanje unutar više redova.

Primena modifikatora je **iza** paterna regularnog izraza, npr. `var uzorak = /ne/g`; i u zavisnosti od korišćenog modifikatora definisani patern dodatno se koriguje u skladu sa efektom modifikatora. Primena modifikatora je posebno izražena kod metoda `match()`, jer on inicijalno pronalazi samo prvi string. U sledećem primeru, korišćen je patern sa i bez modifikatora g da bi se pokazala razlika u delovanju metoda `match()`.

```
<script type="text/javascript">
function provera(x){
    var uzorak = /ne/g;
    document.getElementById('greska').innerHTML = x.match(uzorak);
}
</script>
```

ne! ne zelim neposten rad. ne,ne,ne

Slika 10.19. Prikaz primene regularnog izraza sa metodom `match()` sa modifikatorom `g`

ne! ne zelim neposten rad ne

Slika 10.20. Prikaz primene regularnog izraza sa metodom `match()` bez modifikatora `g`

Metod `test()` se može koristiti i kod definisanja regularnog izraza u formi stringa ili primenom metoda `RegExp()`. Metod `test()` je jedan od najkorišćenijih kod početnika jer direktno testira da li je analizirani string u skladu sa definisanim paternom. U sledećem primeru kontroliše se da li je uneti sadržaj regularno upisani naziv slike sa ekstenzijom. Definisanim paternom dozvoljeno je unosenje svih karaktera sem space-a, jednog ili više, koji se mora završiti tačkom i jednom od definisanih ekstenzija slike.

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^S+\.(gif|jpg|jpeg|png)$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);
}
</script>
```

neka slika.jpg

false

neka-slika.jpg

true

Slika 10.21. Prikaz primene regularnog izraza sa metodom `test()`

Primer 1. Napisati regularni izraz za proveru unete mail adrese. Pretpostaviti da mail adresa može da sadrži slova, tačku i srednju crtu, obavezan simbol `@`, naziv mail servera i domen koji ima dva, tri ili četiri slova.

Rešenje:

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*\\.\\w{2,4}+$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);
}
</script>
```

pera.peric@gmail.com	true
----------------------	------

pera.pericgmail.com	false	pera.peric@gmail	false
---------------------	-------	------------------	-------

Slika 10.22. Prikaz primene regularnog izraza za proveru mail-a

Primer 2. Napisati regularni izraz za proveru unetog decimalnog broja. Dozvoljen je unos jedne ili više cifara, sa ili bez tačke. Ukoliko se tačka koristi, iza nje može ali ne mora biti proizvoljan broj cifara.

Rešenje:

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^\\d+(\\.\\d*)?$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);
}
</script>
```

1.235	true	22	true
-------	------	----	------

22.a	false
------	-------

Slika 10.23. Prikaz primene regularnog izraza za proveru decimalnog broja

Primer 3. Napisati regularni izraz za proveru unete heksadecimalne oznake boje. Obavezan je unos # i iza nje tri ili šest karaktera koji moraju biti mala ili velika slova tj. broj.

Rešenje:

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^#[a-f][A-F][0-9]{3}([a-f][A-F][0-9]{3})?$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);
}
</script>
```

#ccc	true	#e2e2e2	true
------	------	---------	------

#eee4	false
-------	-------

Slika 10.24. Prikaz primene regularnog izraza za proveru unete heksadecimalne oznake boje

Primer 4. Napisati regularni izraz za proveru unetog broja indeksa. Obavezan je unos broja koji je praćen kosom crtom i dvocifrenim brojem koji ukazuje na godinu upisa.

Rešenje:

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^(\d){1,4}\/(\d){2}$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);}
</script>
```

true

true

Slika 10.25. Prikaz primene regularnog izraza za proveru unetog broja indeksa

Predloženo rešenje se može dodatno unaprediti zabranom unosa prve cifre koja može biti nula. Ako se to uvede kao dodatni uslov kod se menja kao:

```
<script type="text/javascript">
function provera(x){
    var uzorak = /^[1-9](\d){1,3}\/(\d){2}$/;
    document.getElementById('greska').innerHTML = uzorak.test(x);
}
</script>
```

Iskusni programeri kažu da je svima u početku delovalo teško da usvoje i brzo pišu regularne izraze, ali da je recept, ne sintaksa, nego razlaganje problema koji se definiše paternom na situacije koje su dozvoljene i samo upotreba sintakse da se to zapiše. Najčešći problem je nedovoljna analiza problema koji se rešava jer se često previdi dosta mogućih zabrana koje kasnije generišu bagove u aplikacijama.

Najbolji način da se regularni izrazi savladaju je vežbanje i samostalno definisanje i rešavanje problema u smislu sadržaja koji treba dozvoliti ili zabraniti.

XI jQuery

Rad sa bibliotekom JavaScript-a

INTENZIVAN razvoj JavaScript-a i sve veći broj njegovih korisnika neminovno je nametnuo da veliki broj programera piše velike količine koda koji su manje više isti. Na primer, svima je potreban kod za kreiranje slajdera, galerija, menija itd. Kako je neracionalno da svako ovo radi za sebe, nekako prirodno se nameće potreba da se taj kod napiše jednom i na jednom mestu, a da ga svi drugi koriste i po potrebi minimalno modifikuju. Na ovaj način nastaju različite biblioteke nekog programskog jezika. Biblioteka je skup metoda ili klasa koje operativno rešavaju neki manji problem koji je jedna logička celina za sebe. Korišćenjem biblioteka kod se standardizuje, postaje dostupan svima, lakši za upotrebu kod programera i jednostavniji u odnosu na pisanje celokupnog koda od početka.

JavaScript je vremenom razvio veliki broj svojih biblioteka, koje su sve namenjene klijentskim primenama i koje su besplatne. Jedna od prvih i najkorišćenijih je jQuery. Ova biblioteka je zamišljena da bude lako primenljiva i da programeru omogući lakše manipulisanje DOM-om tj. HTML-om, CSS-om, događajima, različitim i animacijama.

Kako je jQuery u osnovi JavaScript kod, koji je samo neko napisao i grupisao u neki od metoda, to se i on realizuje kod klijenta i koristi sa svim do sada definisanim pravilima JavaScript-a. Zvaničan sajt ove biblioteke je <http://jquery.com/> i na njemu se nalaze sve njene verzije. Kao i svaka druga biblioteka i jQuery ima veliki broj verzija. Ove verzije ukazuju na novije biblioteke u smislu dodavanja novih metoda kojima se postižu novi efekti ili se lakše koriste ranije metode. Isto tako, verzije se odnose i na obim koda u njima, pa tako imamo za određenu verziju biblioteke njenu podverziju koja može biti obeležena sa min, što ukazuje ne njen manji obim. Manji obim je dobijen samo uklanjanjem nepotrebnih simbola, komentara, space-ova i sl. ali je isti po funkcionalnosti.

Prvi korak u primeni jQuery-ja je da se on kao biblioteka integriše u određenu web stranu. Kako je to JavaScript, onda se koristi klasičan način za integraciju eksternog JavaScript-a. Putanja do tog eksternog skripta može biti lokalna tj. relativna ili apsolutna. Ukoliko se biblioteka fizički preuzme sa sajta jquery.com, i ako je ime tog fajla jquery-3.2.1.min.js, i ako se snimi u naš projekat u folder *script*, tada se ona u web stranicu uključuje pomoću:

```
<script src="script/jquery-3.2.1.min.js"></script>
```

Nakon ove linije koda, sve metode ove biblioteke su dostupne za primenu u JavaScript kodu, kao da ih je programer sam napisao i sada želi da ih upotrebi za svoje namene. Programeri češće koriste apsolutnu putanju, kojoj se dohvata biblioteka sa udaljene lokacije koja je podržana od strane Google i poziva se sa:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
```

Bilo koji od opisanih načina da se koristi efekat je identičan i dalje ostaje da se ova biblioteka počne operativno koristiti.

11.1. Princip rada jQuery-ja

Kao i klasični kod u JavaScript-u, i kod upotrebe jQuery-ja će se prvo izvršiti selekcija određenog elementa DOM-a i zatim na njemu primeniti određeni metodi koji se nalaze u biblioteci. Praktikuje se da se kod koji se koristi za pisanje i primenu jQuery-ja piše u posebnom fajlu, sa ekstenzijom js, a ne u fajlu u kome se nalazi preuzeta biblioteka.

Tako će se u realnim situacijama biti najmanje dve eksterna JavaScript fajla, od kojih je jedan za učitavanje biblioteke a drugi za pisanje koda (u narednom listingu to je fajl *moj-kod.js*) kojim se metode dobijene u biblioteci koriste.

Kao i kod klasičnog JavaScript-a, eksterni fajlovi se mogu učitati u *head* sekciji, ako su potrebni za inicijalni prikaz strane, ili na kraju *body* sekcije.

```

<!DOCTYPE html>
<html>
<head>
<script      src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/
jquery.min.js"></script>
<script src="moj-kod.js"></script>
</head>
<body>

</body>
</html>

```

Sa istom logikom kao i kod JavaScript-a, eksterni kod se može učitati u head sekciji, ako je potrebe za inicijalni prikaz stranice, ili na dnu body elementa, ako je potreban nakon inicijalnog prikaza sajta, jer tako doprinosi bržem učitavanju web stranice.

Kao što je naglašeno, primena jQuery-ja se realizuje selekcijom određenog elementa, nad kojim se želi primeniti neki efekat, i primenom metoda. Uloga metoda je da operativno izvrši neku radnju nad selektovanim elementom. Selekcija se vrši pomoću simbola \$ iza koga se navodi oznaka elementa koji se želi selektovati. Kao i kod klasične primene skripta, selekcija može biti po imenu taga, vrednosti atributa id ili atributa class. Sledeći primeri pokazuju ove tri selekcije respektivno:

`$('.div')`—Pristupa svim div elementima unutar HTML fajla

`$('#desno')`—Pristupa svim HTML elementima sa id=desno

`$('.naslov')`—Pristupa svim HTML elementima sa class=naslov

Kada se izvrši selekcija, primenjuje se neki od metoda. Ako pretpostavimo da želimo metod *hide()*, koji je jedan on najkorišćenijih i služi sa sakrivanje selektovanog elementa, sintaksa bi bila sledeća:

```

$('.div').hide();
$('#desno').hide();
$('.naslov').hide();

```

Primena jQuery-ja treba da bude jednostavna i laka, ali se isto tako može pisati i proizvoljno veliki kod kojim se rešavaju najrazličitije situacije.

Jedna od prvih situacija koja se pojavljuje kao problem je što se kod definisanja jQuery koda, tj. selektora, ako je napisan u head elementu, prijavljuje greška. Ova greška je posledica toga što se definiše selektor za neki element, a ni početak body taga još uvek nije učitao u browser, pa se ne može selektovati nepostojeći element. Taj element će se tek naknadno učitati kada se učita cela strana. Iz tog razloga, potrebno je nekako odložiti izvršavanje jQuery koda dok se cela HTML strane (tj. objekat document) ne učita i tako svi elementi stranice ne budu

dostupni. Ovo se realizuje metodom *ready()*, unutar koga se piše ceo jQuery kod. Po učitavanju stranice metod se aktivira i sav jQuery kod se tada aktivira. Kako je po učitavanju stranice potrebno aktivirati neki veći kod koristiće se ranije opisana bezimena funkcija kao:

```
$(document).ready(function(){ });
```

Ovde je kao selektor definisan objekat *document*, tj. cela web stranica, i na nju je primenjen metod *ready()*. Kod koji se piše unutar funkcije je klasičan JavaScript kod sa svom svojom sintaksom i pravilima, ili kod kojim se koriste metodi iz jQuery biblioteke, npr:

```
$(document).ready(function(){
    $('#desno').hide();
});
```

11.2. Selektori u jQuery-ju

Pored imena elementa ili vrednosti atributa *id* tj. *class*, selektor može biti i složenije definisan skup pravila, kao i kod primena u klasičnom CSS-u. Tako na primer selektorom

```
$('#proizvodi p')
```

Selektujemo element čiji je id sa vrednošću *proizvodi*, i unutar njega sve elemente *p*. Selekcija je mogla da ide još više po dubini kao:

```
$('#proizvodi #akcija .opisi')
```

Treba primetiti da kod ovakve vrste selektora može selektovati element koji može biti veći ili manji u prikazu u browser-u. Tj. ako se selektuje *div*, koji predstavlja polovinu stranice, ili cela tabela ili taster, jQuery će raditi identično. Ovim se pruža mogućnost da se događaji dodele selektovanom elementu, pa da aktiviranje događaja ne mora biti ograničeno na tačno definisane elemente, kao u HTML-u. Ako se selektuje cela tabela, i dodeli događaj *click*, onda će se on aktivirati ako se klikne bilo gde na tabelu. Ovo nije bilo moguće u klasičnom HTML-u koji poziva JavaScript.

Pored opisane strukture „ufinjavanja“ selektora, dodatno je na raspolaganju i upotreba specijalnih dodataka kojima se selekcija dodatno profilise. Tako je na raspolaganju veći broj opcija, od kojih su najčešće korišćene *odd*, *even*, *eq*, *first* i *contains*. Sledeći primeri pokazuju njihove upotrebe.

Za pristup svim *span* elementima koji sadrže reč *leto*:

```
$('#span:contains(leto)')
```

Za pristup svim neparnim *div* elementima (1,3,5,...):

```
$('#div:odd')
```

Za pristup svim parnim *div* elementima (0,2,4,...):

```
$('#div:even')
```

Za pristup trećem *div* elementu:

```
$('#div:eq(2)')
```

Nakon selekcije, u pozadini koda se selektuje deo DOM-a tj. jedan ili više elemenata koji pripadaju DOM-u. Kao i kod klasičnog skripta, nakon selekcije je moguće primeniti i neki od standardnih svojstava koja su dozvoljena nad elementima DOM-a, a koja nisu posebno vezana za jQuery. Na primer, ako se napiše sledeći kod:

```
$(document).ready(function(){
    alert($('#artikli p').length + ' elemenata!');
});
```

Kod će kroz alert ispisati broj *p* elemenata koji se nalazi u elementu čiji je *id*= “*artikli*“. Ukoliko bi se kao selektor definisao:

```
$('#table tr:even').length
```

Kod bi vratio ukupan broj parnih redova, tag *tr*, unutar HTML tabele. Ukoliko je unutar strane više tabela, konkretna tabela se može selektovati stavljanjem atributa *id* u tag *table*. Ako je vrednost tog atributa *tabela1*, selektor bi bio

```
$('#tabela1 tr:even')
```

Ako se unutar tabele koriste tagovi *thead* i *tbody*, a žele se samo prebrojati redovi u *tbody*, selektor bi se konačno modifikovao kao

```
$('#tabela1 tbody tr:even')
```

U zavisnosti koje svojstvo ili metoda se napiše iza selektora dobiće se neki rezultat ili efekat. Cilj je razumeti načine njihove upotrebe dok se spisak dostupnih metoda vremenom povećava i mora se pratiti u dokumentaciji biblioteke.

11.3. Metode jQuery-ja

Postoji veoma veliki broj metoda koje su dostupne u različitim jQuery bibliotekama. Nemoguće je pobrojati sve metode, ali će se ovde pokazati njihova implementacija, upotreba i manipulacije, i to za neke od najkorišćenijih, dok se

očekuje da programer vremenom i po potrebi dalje primenjuje druge metode. Na Internetu postoji veliki broj sajtova koji prikazuju metode jQuery-ja, od koji je jedan <https://api.jquery.com/>.

Jedan od prvih metoda koja se uvode kod učenja jQuery-ja je metod `css()`. Ovaj metod se može koristiti za dve stvari: čitanje nekog CSS svojstva selektovanog elementa, ili njegovo setovanje. Ako se definiše

```
var boja = $('#tabela1 tbody tr:even').css('color');
```

u promenljivu `boja` će se dohvatiti vrednost CSS svojstva `color`, tj. boja teksta u parnim redovima tabele. Isto svojstvo se može upotrebiti za setovanje vrednosti nekog CSS svojstva. Tako na primer ako se definiše

```
$('#tabela1 tbody tr:even').css('background-color','#e2e2e2');
```

Svi parni redovi tabele će imati pozadinu sive boje, bez obzira kako je ona inicijalno definisana u CSS fajlu. Ovim se postiže jedan od najkorišćenijih efekata, tj. efekat zebre. Zebra efekat je vrlo pogodan za sve tabele gde ima više redova ili kolona i pruža mogućnost da se korisnik lakše snađe sa podacima.

Ukoliko se želi definisati vrednost većem broju svojstava, to se pojedinačno može uraditi kao:

```
$('#tabela1 tbody tr:even').css('background-color','#e2e2e2');
$('#tabela1 tbody tr:even').css('color','#cc0000');
```

ili objedinjeno kao:

```
$('#celebs tbody tr:even').css(
  {'background-color': '#e2e2e2', 'color': '#cc0000'}
);
```

Kod objedinjavanja treba biti vrlo pažljiv jer je operator dodele vrednosti `:` kao i u klasičnom CSS-u, dok je `,` razdvajanje svojstava. Kod upotrebe za samo jedno svojstvo karakter `,` se koristi između imena svojstva i njegove vrednosti.

Na ovaj način jQuery direktno pristupa određenom elementu i menja njegov CSS. Drugi način za promenu CSS-a je upotreba metoda `addClass()`. Ovim metodom se selektovanom elementu dodeljuje određena CSS klasa, koja je ranije napisana u CSS-u, i na taj način pomenuti CSS deluje na selektovani element. Na primer, ako se u CSS fajlu nalazi:

```
.mojstil {font-style: bold;
          background-color: #CCC222;}
```

a u jQuery-ju se definiše:

```
$('#blok1').addClass('mojstil');
```

Na element sa *id*='#blok1' primeniće se stil *mojstil*, i u ovom slučaju promeniće se pozadinska boja. Na sličan način kao što se metodom *addClass()* dodaje klasa, pomoću metoda *removeClass()* se klasa uklanja.

```
$('#blok1').removeClass('mojstil');
```

Pomoću opisanih metoda direktno se može pristupiti CSS kodu i promenom istog u web stranici postići bilo koji efekat. Pored promene CSS-a jQuery ima i metode kojima se neke od najčešćih akcija mogu uraditi direktno nekom od metoda. Na primer, najčešće se pojedini blokovi sakrivaju i prikazuju u pojedinim trenucima (dropdown meni, slajder, galerija slika, dodatne informacije, itd.). Ovo se može lako realizovati selekcijom elementa čiji sadržaj treba da se sakrije ili prikaže i primenom metoda *addClass()* ili *css()* tako što će se prozvati svojstvo *display* i setovati vrednost *none* ili *block*. Inicijalno se to i radilo na ovaj način, ali se ovako piše malo duži kod i postiže relativno grub efekat. Tj. definisane promene su trenutne, a to je malo drugačije od onoga što očekujemo u realnim sajtovima gde uvek postoje blage tranzicije ili efekti kod prikaza nekog sadržaja. Iz tog razloga, jQuery nudi dva metoda, *show()* i *hide()*. Ovi metodi služe za prikaz selektovanog elementa ili njegovo sakrivanje, i inicijalno rade isti posao kao i CSS, ali uz kraću sintaksu.

```
$('#dodatneInfo').hide();
$('#dodatneInfo').show();
```

Pored toga što su kraće u sintaksi, primenom ovih metoda moguće je postići i dodatne efekte, koji su dalje definisani kao argumenti metoda. To je prednost upotrebe metoda, jer ako je ona isprogramirana da nudi dodatne opcije, onda se one vrlo lako integrišu i efekat se lako dobija. Na primer, metode *show()* i *hide()* imaju mogućnost definisana brzine njihove realizacije u browser-u. Ovo se postiže argumentima "slow", "fast", ili definisanjem broja milisekundi za koliko želimo da se efekat prikaza ili sakrivanja realizuje. Tako sintaksa postaje

```
$('#dodatneInfo').hide("fast");
$('#dodatneInfo').show(2000);
```

Ove metode nude mogućnost definisanja i drugog argumenta, koji je obično funkcija ili kod, a koji će se izvršiti nakon što se izvrši metod u definisanom trajanju. Tako opšta sintaksa postaje

```
$(selektor).show(speed,callback);
```

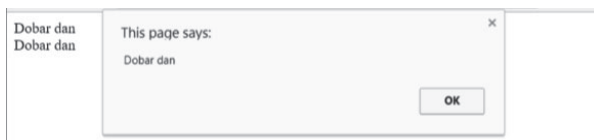
Često korišćeni su i metodi *text()* i *html()*. Oni se mogu posmatrati kao sinonimi za svojstva JavaScript-a: *textContent* i *innerHTML*. Naime metodom *text()* se može pročitati ili setovati tekstualni sadržaj selektovanog elementa, dok se metodom *html()* pored tekstualnog sadržaja može definisati i HTML kod koji će se ispravno interpretirati u browser-u. Ukoliko se HTML kod definiše kroz metod *text()* on će se prikazati kao običan tekst i neće se tumačiti kao kod. Ukoliko se

ovi metodi koriste da dohvatanje sadržaja selektovanog elementa, koristimo ih bez argumenata. U sledećem primeru data su dva diva sa inicijalnim sadržajem. U jQuery-ju se prvo metodom `text()` dohvata sadržaj prvog diva i prikazuje korisniku pomoću `alert`, a nakon toga se metodama `text()` i `html()` setuje novi sadržaj divova, koji u sebi ima i `html` tagove. Kod metoda `text()` tagovi se prikazuju korisniku a kod metoda `html()` oni se interpretiraju kao regularan HTML kod.

```

<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
<script type="text/javascript">
$(document).ready(function(){
    var prvi = $('#prvi').text();
    alert (prvi);
    $('#prvi').text("<h2>Ovo je novi tekst</h2>");
    $('#drugi').html("<h2>Ovo je novi tekst</h2>");
});
</script>
</head>
<body>
<div id="prvi"> Dobar dan</div>
<div id="drugi"> Dobar dan</div>
</body> </html>

```



Slika 11.1. Upotreba metoda `text()` i `html()`

Svi metodi se mogu kombinovati sa bilo kojim osnovnim ili dopunskim selektorima koji su do sada opisani i to treba posmatrati kao opšte pravilo, jer je nemoguće i nepotrebno prikazati sve primere ovih kombinacija. U ovom slučaju, ako bi hteli da selektujemo prvi div iz prethodnog koda, i dohvatimo njegov sadržaj, kao selektor se može definisati i

```
var prvi = $('div:first').text();
```

Veća grupa metoda je namenjena za realizaciju efekata kretanja tj. promena. U zavisnosti da li je promena definisana prilikom prikaza ili sakrivanja ili se efekat koristi za prikaz nadole ili nagore itd. postoji veći broj vrlo korisnih i atraktivnih metoda. Neka od njih su: `fadeIn()`, `fadeOut()`, `fadeTo()`, `fadeToggle()`, `slideDown()`, `slideUp()`, `slideToggle()`... Recimo ako bi smo želeli da u

prethodnom primeru, po učitavanju stranice sakrijemo prvi div, uz diskretnu animaciju, pišemo

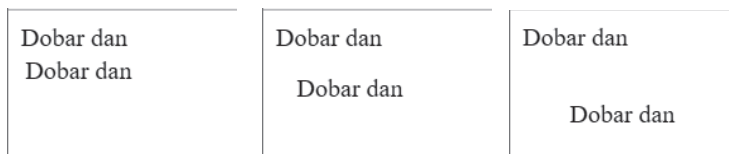
```
$('#div:first').fadeOut();
```

Svi ovi metodi se primenjuju na isti način kao i svi drugi, i svakom od njih se može kao argument definisati vreme trajanje njegove animacije. Ovi metodi su vrlo česti kod izrada menija, slajdera, galerija, prikaza vesti itd. Pored navedenih metoda, koji imaju unapred isprogramiran način rada, vrlo popularan je i metod *animate()* koji programeru daje mogućnost da sam definiše način promena i vreme za koje će se one desiti. Metod *animate()* ima dva argumenta. Prvi je krajnje stanje definisano CSS-pravilima a drugi je vreme za koje kod treba da iz početnog dođe u to definisano krajnje stanje.

Tako na primer, ako želimo da se u prethodnom primeru drugi div animira u trajanju od 3 sekunde, na način da mu se dodaju sva četiri *padding*-a, to se može uraditi na sledeći način:

```
<script type="text/javascript">
$(document).ready(function(){
$('#drugi').animate({ padding: '30px' }, 3000);
});
</script>
```

Dati kod će u svih tri sekunde automatski izračunavati koliko brzo treba da se pomera sadržaj, da bi od početnih vrednosti *padding*-a, za tačno 3 sekunde stigao do zadatih. Korisnik ovo vidi kao kontinuirano kretanja i promenu sadržaja.



Slika 11.2. Prikaz realizacije metoda *animate()* u različitim trenucima vremena

Kom primene metoda *animate*, u prvom argumentu, može se staviti proizvoljan broj CSS svojstava i njihovih vrednosti koje želimo da kod nakon vremena animacije dostigne.

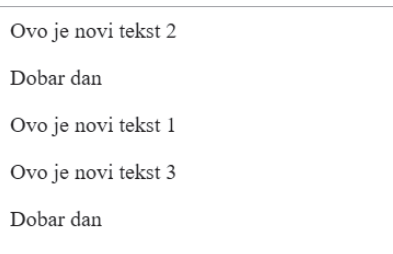
Vrlo slično kao u JavaScript-u, jQuery ima veći broj metoda kojima se selektovani element “umeće” u postojeću strukturu stranice. Na taj način se u stranicu može dinamički i interaktivno dodavati proizvoljno mnogo novi sadržaja do tačke da se kompletan inicijalni sadržaj promeni. Metodi za dodavanje novih elemenata su različiti jer omogućavaju dodavanje na početku željenog elementa, ili na kraju, ili ispred tj. iza njega itd. Najkorišćeniji metodi su: *after()*, *append()*, *appendTo()*, *before()*, *insertAfter()*, *insertBefore()*, *prepend()*, *prependTo()*, ...

Primeri primene nekih od ovih metoda su dati u sledećem kodu.

```

<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
<script type="text/javascript">
$(document).ready(function(){
$('<p> Ovo je novi tekst 1 </p>').insertAfter('#prvi');
$('<p> Ovo je novi tekst 2 </p>').insertBefore('#prvi');
$('<p> Ovo je novi tekst 3 </p>').prependTo('#drugi'); });
</script>
</head>
<body>
<div id="prvi"> Dobar dan</div> <div id="drugi"> Dobar dan</div>
</body>
</html>

```



Slika 11.3. Prikaz mogućnosti metoda za dodavanje novih elemenata DOM-a

Kako je “ubacivanje” novih elemenata omogućeno sa opisanim metodama, logično je da postoji i način da se određeni elementi uklone iz strane. Za te potrebe koristimo metod *remove()*. Ovaj metod se može primeniti na dva načina tj. bez argumenata kada uklanja samo selektovani element (onaj kome je id=”dodatniOpis”):

```
$('#dodatniOpis').remove();
```

ili sa argumentima kada je moguće ukloniti više elemenata po nekom kriterijumu. U ovom primeru uklanjaju se svi p elementi koji kao sadržaj imaju reč “napad”:

```
$('p').remove(':contains("napad")');
```

Metodi *children()* i *parent()* se koriste u situacijama kada u odnosu na selektovani element želimo da pristupimo njegovoj deci elementima ili nadređenom roditeljskom elementu. U jQuery-ju se može definisati klasična promenljiva, kao u dosadašnjim primerima, pomoću *var*. Ukoliko je sadržaj promenljive “prosti tip podataka”, kao string, broj, boolean i sl. koristi se sintaksa kao i u JavaScript-u.

```
var sadrzaj = $("#prvi").text();
```

Međutim ako promenljiva ukazuje na objekat DOM-a, koji u sebi može imati manje ili više složenu strukturu, tada se ispred imena promenljive koristi prefiks \$. Tako ako imamo HTML kod kao:

```
<div id="root">
    <div>Prvi</div>
    <div>Drugi</div>
    <div>Treci</div>
</div>
```

svu decu elemente koji se nalaze u `<div id="root">` selektujemo kao

```
var $deca = $('#root').children();
```

Sada promenljiva *\$deca* predstavlja složenu promenljivu, jer ima više elemenata tj. objekata u sebi. Iz tog razloga nad njim se mogu dalje primeniti svi ranije opisani metodi i svojstva. Recimo prebrojavanje broja selektovane dece bi bilo

```
alert("Broj dece elemenata je" + $deca.length);
```

Pravljanjem klasične promenljive tipa string bi se kreiralo pomoću *var sadrzaj=""*; kao i upotreba svih klasičnih petlji i kontrole tokova, koji su obrađeni u JavaScript-u. Tako ako i želimo da pristupimo svoj deci elementima, bez obzira koliko ih ima, i da iz njih dohvatimo sve njihove sadržaje (metodom `text()`), nakon selekcije sve dece elemenata, potreban je neka od petlji (npr. *each()*) kojom se prolazi kroz sve elemente, i u svakoj iteraciji pristupa tekućem elementu (*\$(this)*) i čita njegov sadržaj (*\$(this).text()*).

```
$(document).ready(function() {
    var $deca = $('#root').children();
    var sadrzaj="";
    $('#root').children().each( function() { sadrzaj += $(this).text(); });
    alert(sadrzaj); });
```

Za „kretanje“ kroz DOM može se koristiti i metod *next()* kojim se prelazi na naredni element u odnosu na tekući. Ako imamo div (*id="slike"*) unutar koga se nalazi veći broj slika, i od kojih jedna ima klasu (*class="aktivna"*) pišemo:

```
<div id="slike">
    
    
    
    
    
</div>
```

Selektovanje one slike koja ima klasu aktivna je tada:

```
var trenutni = $('#slike .aktivna');
```

dok je selekcija sledeće slike:

```
var sledeci = trenutni.next()
```

Jedna od „caka“ za određivanje ima li sledećeg elementa je da se za sledeći element izračuna njegova dužina.

```
trenutni.next().length;
```

Ukoliko element postoji, ta dužina će biti 1 a ukoliko ne postoji biće 0. Ovo se često koristi za kreiranje slajdera. Slajder treba da naizmenično prikazuje slike, jednu po jednu, nakon nekog vremena. Kada se sve slike prikažu, najčešće se kreće sa ponovnim prikazom prve slike i tako redom. Postizanje efekta da se na određeni period nešto desi je pomoću ranije opisanog metoda *setTimeout()*. Ako se kao njegov argument definiše funkcija *slideShow* i vreme 1500, to znači da će se funkcija *slideShow* pozvati za 1.5s.

Ako se u funkciji selektuje aktivna, slika, što će u ovom slučaju biti prva u nizu, potrebno je proveriti da li iza nje postoji sledeća slika. Ako postoji, potrebno je preći na nju i nju prikazati, dok u suprotnom treba ponovo prikazati prvu u nizu slika. Provera da li postoji sledeća radi se sa *trenutni.next().length*. Ako postoji, prelazak na sledeću je pomoću *trenutni.next()*. Ako ne postoji, a to znači da se došlo do poslednju u skupu slika, potrebno je prvo vratiti se na roditeljski element, *trenutni.parent()*, pa onda selektovati svu njegovu decu, *trenutni.parent().children()*, ali među njima uzeti baš prvi element *trenutni.parent().children(':first')*. Na ovaj način se od poslednje slike, vraćamo na početnu. Da bi se naredna, tj. sledeća, slika prikazala, potrebno je trenutnu sliku sakriti i tu narednu prikazati. Ako CSS klasa aktivna definiše ko je vidljiv a ko ne, onda je dovoljno tekućoj slici ukloniti klasu aktivna, i time je učiniti nevidljivom, a sledećoj dodati klasu aktivna, da bi ona postala vidljiva.

Ovo je realizovano pomoću: *trenutni.removeClass('aktivna');* i *sledeci.addClass('aktivna');*. Na kraju da bi se ceo ovaj proces ponavljao, iz funkcije *slideShow()* se pomoću *setTimeout(slideShow, 1500);* ponovo poziva ista funkcija na svakih 1.5s. Celokupan kod ovakvog slajdera je dat u listingu:

```
$(document).ready(function(){
    slideShow();
});
function slideShow() {
    var trenutni = $('#slike.aktivna');
    var sledeci = trenutni.next().length ? trenutni.next() :
trenutni.parent().children(':first');
    trenutni.removeClass('aktivna');
    sledeci.addClass('aktivna');
    setTimeout(slideShow, 1500);
}
```

U datom kod CSS kod je definisan kao:

```
#slike img {
  display: none;}
#slike .aktivna {
  display: block;}
```

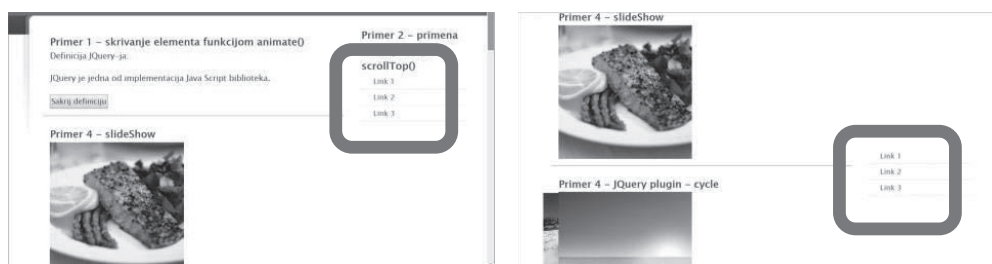
Najveći broj metoda se primenjuje na objekat *document*, ali to nije uvek slučaj. Primenom jQuery-ja moguće je selektovati bilo koji objekat i na njega primeniti neki metod. Ovo će se prikazati na primeru fluktuirajućeg menija. Ovakav meni treba da se pomera i stalno bude na istom rastojanju od gornje ivice browser-a, bez obzira koliko korisnik skrolovaio stranicu gore ili dole. Ako se u HTML-u kreira klasična lista za potrebe menija:

```
<ul id="linkovi">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
</ul>
```

I ako se definiše sledeći jQuery kod:

```
$(document).ready(function(){
  $(window).scroll(function () {
    $('#linkovi').css('top', $(document).scrollTop());
  });
});
```

Tada je kao selektor izabran objekat *window*, jer se u odnosu na njega posmatra neka akcija, i njemu dodeljen metod *scroll()* kojim se svakim skrolovanjem aktivira kod bezimene funkcije i koji pristupa listi (id="linkovi") i menja njeno CSS svojstvo *top*, tako da uvek bude onoliko koliko je do vrha prozora. Kako je rastojanje stalno isto, bez obzira na početak web strane, meni će se kretati gore dole, kako se stranica skroluje, ali tako da uvek bude isto odmaknuto od vrha prozora.



Slika 11.4. Prikaz upotrebe metoda *scroll()* za fiksiranje pozicije menija

Već je napomenuto da u jQuery-ju postoji veliki broj metoda. Međutim, do sada je korišćena samo bazna biblioteka. Pored nje, postoji veliki broj dodatnih biblioteka koje su napisane za specifične primene. U njima se nalaze dodatni

metodi koji su nam na raspolaganju. Ako se na primer, posmatra gore definisani HTML kod sa divom u kome je skup slika, i ako se u HTML stranu uključi dodatna biblioteka *cycle.all.min.js*, onda se dobija mogućnost korišćenja njenog metoda *cycle()* koji omogućava gotovu animaciju za galeriju slika.

```
<script type="text/javascript" src="jquery.cycle.all.min.js" ></script>
```

Upotrebom njegovog argumenta, može se definisati pravac i način sklanjanja prethodnih slika. Na primer, ako se slike pomeraju po *x* osi, tako što se umeću ispod prethodnih, koristi se *fx: 'shuffle'*. Tako ceo jQuery kod kojim se kreira galerija slika, za ranije dati HTML kod, postaje:

```
$(document).ready(function(){
    $('#slike').cycle({
        fx: 'shuffle'
    });
});
```



Slika 11.5. Prikaz animirane galerije slika realizovane primenom metoda *cycle()*

Prikaz slike sa plažom koja se pomera nalevo, do krajnje leve ivice slike, i podvlači pod narednu sliku sa čamcem.

11.4. Rad sa događajima u jQuery-ju

Sa istom namenom kao u JavaScript-u, koriste se događaji i u jQuery-ju. Događaji imaju ista imena i iste osobine kao što je prikazano u poglavlju IX. Jedino se razlikuje način primene.

11.4.1. Upotreba događaja kao metoda objekta

U jQuery-ju najčešće se koriste dva oblika za definisanje događaja. Stariji, gde se događaj tretira kao i svaki drugi metod, na selektovanim objektom ima opštu sintaksu:

```
$(selektor).dogadjaj( handler);
```

Na ovaj način se određeni događaj može dodeliti bilo kom selektovanom objektu, i kada se događaj desi da se pokrene neki kod definisan handler-om. Kao handler se može definisati neki drugi kod koji želimo da se desi, ali se češće taj kod smešta u bezimenu funkciju, i tada može biti proizvoljno veliki. Tako sintaksu možemo posmatrati kao:

```
$(selektor).dogadjaj(function(){
    // neki kod koji se realizuju po aktivaciji događaja
});
```

Ako posmatramo događaj *click*, i dodeljujemo ga tasteru, a kao rezultat očekujemo da se iznad tastera doda tekst “Opis” to bi zapisali kao:

```
<html>
<head>
<script      src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/
jquery.min.js"> </script>
<script type="text/javascript">
$(document).ready(function(){
$('#taster').click(function(){
    $('#<p> Opis </p>').insertBefore('#taster');
});
});
</script>
</head>
<body>
    <button id="taster">Klikni za prikaz</button>
</body>
</html>
```

Ukoliko bi želeli da postignemo isti efekat dodavanja teksta iznad tastera, ali ne klikom na taster, nego recimo prelaskom preko njega, ceo kod bi ostao isti samo bi događaj bio *mouseenter()* i to bi zapisali kao:

```
$('#taster').mouseenter(function(){
    $('#<p> Opis </p>').insertBefore('#taster');
});
```

Ukoliko se želi obojiti polje forme u koje se uđe, korišćemo događaj *focus()* i metod *css()* za definisanje boje ivica.

```
<html>
<head>
<script      src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/
jquery.min.js"></script>
<script>
$(document).ready(function(){
    $('#user').focus(function(){ $(this).css("border", "2px solid red"); });
```



```

});
</script>
</head>
<body>
    Username <input type="text" id="user"/>
</body>
</html>

```

Na slične načine se primenjuje bilo koji drugi od klasičnih JavaScript događaja. Vremenom se pojavljuje potreba da se neki kod, koji se može napisati upotrebom do tada definisanih metoda, objedini i postane metod, da bi se brže implementirao. Tako pored klasičnih, tj. inicijalnih događaja, imamo i malo naprednije koji predstavljaju kombinaciju inicijalnih. Posmatrajmo sledeći kod:

```

$(document).ready(function(){
    $('#taster').click(function(){
        if($('#opis').is(':visible')) {
            $('#opis').hide();
        } else {
            $('#opis').show();
        }
    });
});

```

Ovaj kod radi sledeće: nakon klika ne neki element (id="taster") pomoću *if*-a proverava da li je neki drugi element (id="opis") vidljiv ili ne. Ovo se proverava pomoću metoda *is()* i selektora *:visible*. Ako pretpostavimo da je prvi element taster a drugi div element, tada će klikom na taster da se pomoću *if*-a proveriti da li je div, sa svojim sadržajem, vidljiv ili ne. Ako je vidljiv, tada će taj div biti sakriven, pomoću metoda *hide()* dok ako nije vidljiv tada će se prikazati metodom *show()*. Na ovaj način korisnik može sve vreme da klikće po tasteru i sadržaj diva će se prikazivati i sakrivati naizmenično. Ovo će se kasnije upotrebiti kod dropdown menija kada klikom na link želimo da se sadržaj podmenija prikaže a novim klikom da se sakrije. Ovaj kod se regularno može koristiti i predstavlja naizmenično smenjivanje dve akcije (u ovom slučaju sakrivanje i prikaz) nekog elementa. Kažemo da se svaki parni put dešava jedna akcija (npr. prikaz) a svaki neparni klik uslovi drugu akciju (npr. sakrivanje). Kako je primena ovog koda veoma česta, u nekoj od verzija jQuery-ja kreiran je metod *toggle()*, koji u potpunosti menja gore prikazani kod. Inicijalno *toggle* u sebi sadrži implementiran metod *show()* i *hide()* naizmenično realizuje jedan ili drugi na selektovanim elementom. Sada isti kod postaje dosta kraći i pregledniji:

```

$('#taster').click(function(){
    $('#opis').toggle();
});

```

Ovakvih primera skraćivanja tj. objedinjavanja koda ima dosta i samo je bitno pratiti koje nove metode i mogućnosti donose nove verzije biblioteka. Tako na primer i metod *toggle()* može biti korišćen sa definisanjem brzine realizacije sakrivanja tj. prikaza selektovanog sadržaja:

```
$('#opis').toggle('slow');
```

ili da se iskoristi sa još novijom verzijom koja prikaz i sakrivanja radi ne pomoću *hide* tj. *show* nego *slideDown()* i *slideUp()*, i kada postaje novi metod *slideToggle()*.

```
$('#disclaimer').slideToggle('slow');
```

Već je napomenuto da se često u novijim verzijama kodovi sažimaju u nove metode i tako postaju lakši za korišćenje. Pored *toggle()* metoda, često korišćen je i slično dobijeni *hover()* metod. Ako posmatramo sledeći kod, videćemo da se nad istim selektorom, redom tabele, definišu dva događaja, tj. prelazak mišem preko reda i izlazak mišem van reda. U slučaju prelaska dodaje se klasa pozadina, koja boji tekući red nekom drugom pozadinskom bojom, i tako olakšava vizuelni identitet i lakše snalaženje. U slučaju kada se izađe van tog reda, ta CSS klasa se uklanja i red dobija inicijalnu pozadinsku boju.

```
$('#table tr').mouseover(function(){
    $(this).addClass('pozadina'); });
$('#table tr').mouseout(function(){
    $(this).removeClass('pozadina');
});
```

Ovo je vrlo česta situacija kada želimo da prelaskom miša preko nekog elementa (tastera, linka, vesti, slike, artikla,...) imamo neku akciju, dok sklanjanjem sa tog elementa vraćamo sve u inicijalno stanje. Ovaj kod to rešava, ali se može pisati i kraće od koda postoji metod *hover()*. Ovaj metod u sebi ima integrisanu logiku koja je identična prethodnom kodu i koja se za dati element aktivira prelaskom miša i aktiviranjem jedne akcije tj. izlaskom miša van elementa, kada se aktivira druga akcija. Tako bi prethodni kod mogao da se napiše kao:

```
$('#table tr').hover(
function(){$(this).addClass('pozadina');},
function(){$(this).removeClass('pozadina');}
);
```

Tj. mogli bi da to zapišemo kao opštu sintaksu:

```
$(selektor).hover(
function() { //sta se radi kad se predje misem},
function() { //sta se radi kad se izađe misem }
);
```

Metod *hover()* je vrlo korišćen kod menija i slajdera obzirom da funkciju koju nudi programerima. U kombinaciji sa metodama događaji postižu svoj pun efekat. Tako na primer neku animaciju možemo dodeliti događaju i na taj način će tek nakon klika korisnika na taster da se nekom animacijom sakrije tekst, ko u primeru:

```
$("#hideDefinition").click(function() {
    $('#definition').animate({
        opacity: 'hide',
        height: 'hide' }, 'slow');
});
```

Slično, kombinaciju događaja i metoda možemo da radimo i sa dodatnim jQuery bibliotekama. Npr. ako u projekt implementiramo biblioteku *jquery.scrollTo.js* i napišemo sledeći kod:

```
$(document).ready(function(){
    $('#scrollToTop a').click(function(){
        $.scrollTo(0,'slow');
        return false;
    });
});
```

Omogućićemo da se klikom na link (a) unutar nekog bloka na dnu strane (*#scrollToTop*), kompletan sadržaj web strane automatski sporo skroluje na vrh strane. Ovo je česta opcije u stranicama gde ima puno sadržaja i kada korisnik dođe do dna da ne mora on ručno da se skroluje na vrh, nego klikom a link ili taster to uradi kod uz diskretnu animaciju kretanja na gore.

11.4.2. Upotreba događaja pomoću metoda *on()*

Drugi način za upotrebu događaja je upotreba metoda *on()*. Ovaj metod kao svoj prvi argument ima događaj, dok je drugi kod koji treba da se realizuje kada se događaj desi. Ovaj metod omogućava da se na jedan selektovani element istovremenom doda više događaja i akcija koji svaki od njih treba da radi ako se događaj desi. Pored toga, ovaj metod može da pristupa dinamički kreiranim elementima dok prethodni način to ne može. To znači da ukoliko jQuery-jem kreirano npr. taster, i želimo da tom dinamički kreiranom tasteru definišemo događaj *click*, to će moći pomoću metoda *on*, jer će on moći da “vidi” i elemente koji nisu inicijalno u web strani neko su kasnije kodom generisane.

Opšta sintaksa metoda *on* bi mogla da se zapiše kao:

```
$(selektor).on("događaj", function(){ //neki kod });
```

Ako posmatramo raniji kod kojom smo tasteru definisali događaj *click* i neku akciju tj.

```
$('#taster').click(function(){
    $('<p> Opis </p>').insertBefore('#taster');
});
```

Onda bi isti efekat primenom metoda on postigli sa na sledeći način:

```
$('#taster').on("click", function(){
    $('<p> Opis </p>').insertBefore('#taster');
});
```

Ukoliko se objedinjuje više događaja na istom selektovanom elementu koristi se:

```
$(selektor).on({
    dogadjaj1 : function() { // sta se radi kada se desi dogadjaj 1},
    dogadjaj2 : function() { // sta se radi kada se desi dogadjaj 2},
    dogadjaj3 : function() { // sta se radi kada se desi dogadjaj 3}
});
```

Ako se u prethodnom kodu, selekcijom tastera, definišu dva događaja: *click()* kojim se ispisuje tekst ispred tastera, i *mouseover()* kojim se pozadina cele strane boji plavom bojom, to se zapisuje kao:

```
$(document).ready(function(){
    $('#taster').on({click : function(){
        $('<p> Opis </p>').insertBefore('#taster');},
    mouseover: function(){ $('body").css('background-color', "blue");}
    });
});
```

11.5. Izrada menija u jQuery-ju

Meniji spadaju u jedan od najčešćih elemenata web strane. Sa aspekta HTML-a meni treba da bude napisan kao neuređena lista, koja se proizvoljno može stilizovati upotrebom CSS-a. Ukoliko su meniji interaktivnog karaktera, onda se najčešće dele u ekspanirajuće i dropdown menije.

Bez obzira koji meni se realizuje pomoću jQuery-ja HTML kod je identičan. U sledećem primeru je jedna verzija liste za potrebe menija.

```
<ul id="kategorije">
    <li><a href="#">Obuća</a>
        <ul class="active">
            <li><a href="#">Cipele (200)</a></li>
            <li><a href="#">Sandale (24)</a></li>
            <li><a href="#">Patike (325)</a></li>
            <li><a href="#">Čizme (50)</a></li>
        </ul>
    </li>
</ul>
```

```

    </li>
    <li><a href="#">Odeća</a>
      <ul>
        <li><a href="#">Košulje (135)</a></li>
        <li><a href="#">Pantalone (89)</a></li>
        <li><a href="#">Farmerke (67)</a></li>
        <li><a href="#">Trenerke (41)</a></li>
        <li><a href="#">Duksevi (123)</a></li>
      </ul>
    </li>
  </ul>
</div>

```

11.5.1. Ekspandirajući meni

Ekspandirajući meni je najčešće definisan tako da su sve stavke liste jedna ispod druge. Ukoliko postoje podstavke, one se prikazuju neposredno ispod posmatranog elementa liste, pomerajući sve druge elemente liste na dole. CSS kod koji se u tom slučaju može primeniti za prethodno definisani HTML kod je:

```

#kategorije {
  width:200px;}
#kategorije, #kategorije ul{
  list-style-type: none;
  padding: 0;
  margin: 0;}
#kategorije li {
  cursor:pointer;
  background:#C60;
  border-bottom:3px double #FFF;
  font-size: 15px;}
#kategorije li a {
  padding-left: 10px;
  text-decoration: none;
  color: #fff;}
#kategorije li > ul > li{
  cursor:auto;
  border-bottom:1px solid #C60;
  padding:0 14px;
  background-color:#3a1d00;}
#kategorije li > ul > li > a:hover{
  text-decoration:underline;}

```

Obuća
Cipele (200)
Sandale (24)
Patike (325)
Čizme (50)
Odeća
Košulje (135)
Pantalone (89)
Farmerke (67)
Trenerke (41)
Duksevi (123)

Slika 11.6. Inicijalni prikaz stilizovane liste pomoću HTML-a i CSS-a

Na ovaj način dobija se statički prikaz menija kao na slici 11.6.

Da bi dati kod postao interaktivan, pišemo jQuery kod sa ciljem da se dobije ekspanadirajući meni. Da bi se ovo postiglo, potrebne su dve faze rada. U prvoj da se po učitavanju stranice, selektuju sve podliste, tj. podmeniji, i da se inicijalno sakriju. Selekciju svih podlisti lako možemo da uradimo pomoću `$( '#kategorije > li > ul' )` i da na njih primenim metod `hide()`. Ovakvim kodom dobijamo prikaz samo glavnih elemenata liste tj. prvog hijerarhijskog nivoa kao na slici 11.7.



Slika 11.7. Prikaz ekspanidirajućeg menija nakon sakrivanja podmenija

Ono što je dalje potrebo uraditi, u drugoj fazi, je napisati kod kojim će se klikom na neki od elemenata liste, koji su sada vidljivi, pronaći njena sakrivena podlista i prikazati. Ponovnim klikom potrebno je podlistu sakriti. Ovo se opet jednostavno može uraditi pomoću `toggle()` koji će se dodeliti glavnim elementima liste pomoću `$( '#kategorije > li' ).toggle(function(){} )`. Na ovaj način svakim neparnim klikom na određeni element liste, njemu će se desiti jedna akcija, a na svaki parni (drugi, četvrti, šesti... put) neke druga. Obzirom da želimo da se na neparni klik (prvi, treći,...) podlista prikaže, i to za onaj element na koji smo kliknuli, koristićemo `this`. `This` će znati na koji smo element kliknuli, i sada je potrebno pronaći njegovu podlistu i nekim metodom je prikazati. To se može uraditi pomoću `$( this ).find('ul').slideDown()`. U drugoj situaciji kada je potrebno da se podlista sakrije samo će se koristiti drugi metod tj. `$( this ).find('ul').slideUp()`. Tako ceo kod postaje:

```
$(document).ready(function(){
    $( '#kategorije > li > ul' )
        .hide()
        .click(function( e ){e.stopPropagation(); });
    $( '#kategorije > li' ).toggle(
function(){ $(this).find('ul').slideDown();
}, function(){ $( this ).find('ul').slideUp();
});
});
```

U objedinjenom kodu korišćen je i događaj `click()` koji se pomoću metoda `stopPropagation()` prekidaju sve zaostale animacije. Ovaj kod će i bez ovog dela ispravno raditi ali je poželjno da se i on postavi da se ne bi nagomilavale animacije koje ne mogu da se završe i da ceo meni “poskakuje”. Kada se sada klikne na prvu stavku u meniju, tj. `Obuća`, `toggle()` detektuje da je to prvi put, pronalazi podmeni, tj. element `ul`, i prikazuje ga sa animacijom. Korisnik tada vidi:

Obuća
Cipele (200)
Sandale (24)
Patike (325)
Čizme (50)
Odeća

Slika 11.8. Prikaz ekspanirajućeg menija nakon prvog klika na prvu stavku (Obuća)

Ukoliko bi korisnik opet kliknuo na Obuća, *toggle()* bi detektovao drugi klik, i pronašao podmeni i sakrio ga, čime bi se vratili u inicijalno stanje. Ako se ne bi ponovo kliknulo na *Obuća*, dok je otvoren podmeni, i kliknulo na *Odeća*, njegov *toggle()* bi za tu stavku detektovao prvi klik, našao njegov podmeni i prikazao ga. Tada bi dobili situaciju kao na narednoj slici i tako redom.

Obuća
Cipele (200)
Sandale (24)
Patike (325)
Čizme (50)
Odeća
Košulje (135)
Pantalone (89)
Farmerke (67)
Trenerke (41)
Duksevi (123)

Slika 11.9. Prikaz ekspanirajućeg menija nakon prvog klika na prve stavke (Obuća i Odeća)

Kako je za ekspanirajući meni potrebno ostaviti dovoljno prostora ispod njega, da više stavki može da bude istovremeno otvoreno, on u nekim organizacijama sajtova ume da bude teško uklopiv. U tim situacijama koristi se dropdown meni. Treba naglasiti da je u ovom primeru korišćen *toggle()* koji se aktivirao klikom na neku od stavki, ali je nepromenjen kod mogao da ostane ako se samo umesto *toggle()* stavi *hover()* čime bi se sve isto aktiviralo, ali ne klikom, nego prelaskom miša preko stavke menija.

11.5.2. Dropdown meni

Ovaj tip menija spada u najkorišćenije jer zauzima najmanje prostora i obično se postavlja na vrh strane. Često se pravi u varijantama da podmeniji prekrivaju sadržaj sajta, a ne pomeraju na dole, tako da nije potrebno predvideti prostor za njegovo otvaranje. Ovo je naročito bitno u situacijama kada je sadržaj menija dinamički i kada dolazi iz baze podataka pa inicijalno programer i ne zna koliko će stavki meni imati.

Ako za potrebe prikaza ovog tipa menija uzmemo ovu listu:

```

<ul id="meni">
  <li><a href="#">Korišćene funkcije</a>
    <ul class="active">
      <li><a href="#">animate()</a></li>
      <li><a href="#">css()</a></li>
      <li><a href="#">show()</a></li>
      <li><a href="#">hide()</a></li>
      <li><a href="#">Ostalo</a></li>
    </ul>
  </li>
  <li><a href="#">Korišćeni plugin-ovi</a>
    <ul>
      <li><a href="#">ScrollTo</a></li>
      <li><a href="#">Cycle</a></li>
    </ul>
  </li>
</ul>

```

Za nju je potrebno definisati i odgovarajući CSS kod. Primetiti da je ova lista pozicionirana apsolutno, i da je postavljena skroz gore levo. Da su svi elementi liste pozicionirani levo a da su podliste, pomoću *position: absolute;* i *left: -999em;* potpuno pomerene iz vidljivog dela browsera zbog upotrebe negativne vrednosti svojstva *left*.

```

#meni {
  position: absolute;
  top: 0; left: 0;}
#meni, #meni ul {
  padding: 0; margin: 0;
  list-style: none;}
#meni li { float: left;}
#meni a {
  padding: 5px;
  display: block;
  color: #FFF;
  width: 170px;
  font: bold 15px "Trebuchet MS", sans-serif;}
#meni li ul {position: absolute;
  left: -999em;
  width: 180px;
  background-color: #0072e5;}
#meni li: hover ul, #meni li ul: hover {
  left: auto;}

```

Ukoliko se ne napiše bilo kakav jQuery kod, sada je dobijen prikaz dve glavne stavke u meniju, i pomoću hover-a u CSS-u, realizovano da se prelaskom mišem

prekog nekog od elemenata liste svojstvo *left* setuje na *auto*, pa tako postaje vidljivo, obzirom da je pre toga imalo veliku negativnu vrednost.

Ukoliko se sada doda jQuery kod:

```
$(document).ready(function(){
    $('#meni li ul').css({
        display: "none",
        left: "auto"
    });
    $('#meni li').hover(function() {
        $(this)
            .find('ul')
            .stop(true, true)
            .slideDown('slow');
    }, function() {
        $(this)
            .find('ul')
            .stop(true, true)
            .fadeOut('slow');
    });
});
```

dobije se klasični dropdown meni. U prvom delu koda, po učitavanju stranice se pristupa podmenijima (*#meni li ul*) i oni se CSS-om sakrivaju (*display: "none"*). Prelaskom mišem preko neke stavke menija (*hover*) za tu stavku se pronalazi njen podmeni (*find('ul')*) i sa metodom *slideDown('slow')* prikazuje korisniku. Odmah po sklanjanju miša sa stavke *li*, *hover*-om se to detektuje i aktivira druga funkcija, koja pronalazi pomenuti podmeni i sada ga sa metodom *fadeOut('slow')* sakriva. Efekat rada datog koda, kada se mišem pređe preko stavke *Korišćene funkcije*, je prikazan na slici 11.10.



Slika 11.10. Prikaz realizacije dropdown menija

11.5.3. Realizacija tabova

Rad sa tabovima do skoro je bio isključivo povezan sa desktop aplikacijama. Međutim, sada je isti princip moguće realizovati i u web aplikacijama. Tabovi se često vezuju sa menijima jer se i oni mogu upotrebiti za strukturiran i limitiran prikaz podataka. U ovom delu objasniće se jedno rešenje realizacije tabova.

Ako se kreira inicijalni HTML kod na sledeći način:

```

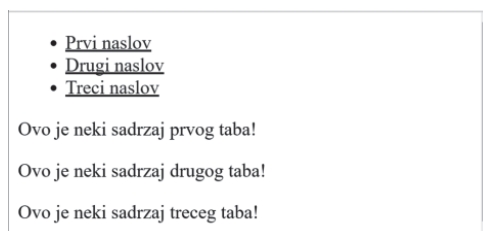
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css" />
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/
jquery.min.js"></script>

<script type="text/javascript">
    // neki kod
</script>

</head>
<body>
    <ul id="meni-tabs">
        <li class="aktivan"><a href="#prvi">Prvi naslov</a></li>
        <li><a href="#drugi">Drugi naslov</a></li>
        <li><a href="#treci">Treci naslov</a></li>
    </ul>
    <div id="sadrzaj">
        <p id="prvi">
            Ovo je neki sadrzaj prvog taba!
        </p>
        <p id="drugi">
            Ovo je neki sadrzaj drugog taba!
        </p>
        <p id="treci">
            Ovo je neki sadrzaj treceg taba!
        </p>
    </div>
</body>
</html>

```

Njegov sadržaj se inicijalno u browser-u prikazuje na sledeći način:

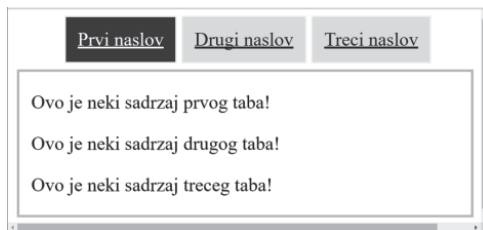


Slika 11.11. Inicijalni prikaz liste i p elemenata u browser-u

Ako se sada doda CSS kod u fajl *style.css* takav da elemente liste postavi u horizontalni redosled, definiše dužina i ivice, kao:

```
#sadrzaj{width: 360px;
  border: 2px solid #c2c2c2;
  padding: 0px 10px;}
#meni-tabs{width: 100%;
  list-style: none;}
#meni-tabs li{
  display: inline;
  border: 1px solid #f2f2f2;
  padding: 10px;
  background-color: #e2e2e2;}
#meni-tabs li.aktivan{ background: #333333; }
#meni-tabs li.aktivan a{ color: #ffffff; }
```

Dobije se uobičajeni statički prikaz kao:

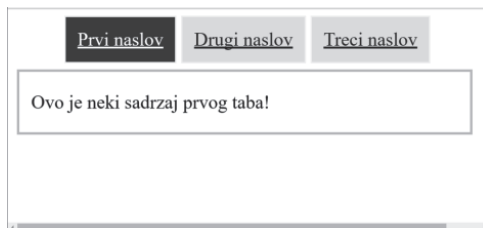


Slika 11.12. Stilizovani prikaz liste i *p* elementa pomoću CSS-a

Klik na bilo koji link u ovom trenutku ne izaziva bilo kakvu akciju, jer su linkovi interno referencirani pomoću #. Ako se sada umesto komentara u delu za interni JavaScript selektuju svi, sem prvog *p* elementa, i sakriju, to se zapisuje kao:

```
$(document).ready(function(){
  $('#sadrzaj p:not(:first)').hide();
});
```

Rezultat delovanja će biti sakrivena druga dva *p* elementa, koji su iz prvog.



Slika 11.13. Efekat sakrivanja svih *p* elementa sem prvog

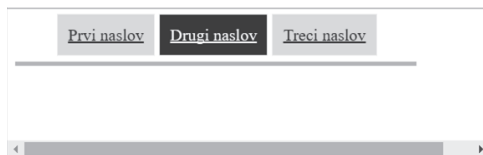
Upotreba klase *aktivan* služi da vizuelno pokaže korisniku na kome tabu se trenutno nalazi. Inicijalno to je prvi tab. Sada je potrebno kreirati kod kojim će se klikom na određeni *li* element baš tom elementu dodati klasa *aktivan* dok će kod drugih biti uklonjena. Pored toga, čim korisnik klikne na neki *li* element, on želi da vidi neki drugi sadržaj koji pripada tom tabu, pa sve do tada vidljive sadržaje treba sakriti. Ovo se može realizovati pomoću:

```

$('#meni-tabs li').click(function() {
    $('#sadrzaj p').hide();
    $('#meni-tabs .aktivan').removeClass("aktivan");
    $(this).addClass('aktivan');
});

```

Ovim kodom omogućeno je da se klikom na neki od tabova, on prikaže kao aktivan, prethodno aktivan poništi a svi sadržaji *p* elemenata uklone. Recimo klikom na drugi tab dobija se:



Slika 11.14. Realizacija promene elementa *p* koji ima definisanu klasu *aktivan*

Sada je potrebno omogućiti da se klikom na određeni tab njegov odgovarajući *p* element prikaže. Da bi se to uradilo, potrebno je saznati vrednost *href* atributa, elementa *a*, u *li* elementu na koji se kliknulo. Zato sa *this* pronalazimo koji *li* je izabran, dok pomoću *find('a:first')* pristupamo *a* elementu unutar *li* elementa na koji se kliknulo. Dalje pomoću *attr('href')* čitamo vrednost atributa *href*, za link na koji se kliknulo. Ova vrednost je sada dohvaćena i smeštena u lokalnu promenljivu *kliknuti*. Ovaj podatak je bitan jer je svaki *p* element identifikovan preko atributa *id* baš sa ovim sadržajem (prvi, drugi, treci). Tako ako korisnik klikne na stavku *Treci naslov*, dohvaćće se vrednost *href* atributa *#treci*. Sada je na bazi te informacije potrebno pronaći taj element unutar grupe *p* elemenata i prikazati ga metodom npr. *fadeIn()*. Selekcija tog metoda je vrlo jednostavna preko njemu nadređenog elementa *#sadrzaj*, pa se ovaj deo koda može zapisati kao:

```

var kliknuti = $(this).find('a:first').attr('href');
$('#sadrzaj ' + kliknuti).fadeIn();

```

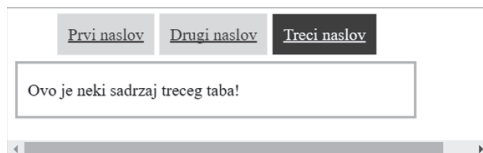
Celokupan kod sada izgleda ovako:

```

<script type="text/javascript">
$(document).ready(function(){
    $('#sadrzaj p:not(:first)').hide();
    $('#meni-tabs li').click(function() {
        $('#sadrzaj p').hide();
        $('#meni-tabs .aktivan').removeClass("aktivan");
        $(this).addClass('aktivan');
        var kliknuti = $(this).find('a:first').attr('href');
        $('#sadrzaj ' + kliknuti).fadeIn();
    });
});
</script>

```

i omogućava realizaciju tabova i njima pripadajućih sadržaja.



Slika 11.15. Prikaz finalne realizacije dela strane koji je organizovan u formi tab-ova

11.6. Rad sa elementima forme

Jedina specifičnost, u smislu sintakse, kod jQuery-ja kada se radi sa elementima forme je upotreba metoda *val()*. Ovaj metod služi za dohvaćanje vrednosti atributa *value*, i sinonim je za svojstvo *value* u JavaScript-u.

Iako nema posebnih metoda, rad sa formama u jQuery-ju je često dosta lakši nego u JavaScript-u zbog lake selekcije elemenata, grupisanja i dodeljivanja događaja. Kao specifičnost u selekciji kod elemenata forme treba istaći mogućnost da se koristi selektor `:` iza koga može da ide vrednost atributa *type*. Tako se selekcijom cele forme, npr. preko *id*=`"formValidation"` u *form* tagu, direktno može pristupiti tasteru koji je tipa *submit* kao `$('#formValidation :submit')`. Na sličan način mogu se selektovati sva tekstualna polja kao `$('#formValidation :text')`. Ovo je vrlo brz način da se selekcija uradi na proizvoljno puno elemenata forme, i da se prolaskom kroz petlju pristupa jednom po jednom elementu, iako se njihovi identifikatori ne moraju poznavati. Pomoću *this*, u jednoj iteraciji petlje pristupa se tačno jednom elementu i nad njim se može primeniti bilo koji metod. Ako je potrebno u svakoj iteraciji dohvatiti sadržaj koji je korisnik uneo u polje i proveriti da li je njegova dužina jednaka 0 (što znači da nije uneo sadržaj) to se zapisuje kao:

```
$('#formValidation :text').each(function() {
    if($(this).val().length == 0) { // neki kod }
});
```

Ako posmatramo praktičan primer forme, koja ima tri tekstualna polja i taster, kao u listingu:

```
<form action="#" method="post" id="formValidation">
  <div class="form-element">
    <label for="ime">Ime:</label>
    <input type="text" id="tbIme" name="tbIme"/>
  </div>
  <div class="form-element">
    <label for="ime">Prezime:</label>
    <input type="text" id="tbIme" name="tbPrezime"/>
  </div>
```

```

<div class="form-element">
    <label for="ime">Broj indeksa:</label>
    <input type="text" id="tbBrIndeksa" name="tbBrIndeksa"/>
</div>
<div class="form-element">
    <input type="submit" id="btnValidate" name="btnValidate"
value="Proveri"/> </div>
</form>

```

i ako se ovom fajlu doda jQuery kod, kojim se klikom na taster sva tekstualna polja selektuju, i proverava da li je ostao prazan sadržaj, pa u zavisnosti od toga iverice oboje u crvenu ili zelenu boju, to se može zapisati kao:

```

$(document).ready(function(){
    $('#formValidation :submit').click(function(e) {
        $('#formValidation :text').each(function() {
            if($(this).val().length == 0) {
                $(this).css('border', '2px solid red');
            }
            else{ $(this).css('border', '2px solid green'); }
        });
        e.preventDefault();
    });
});

```

Primer 7 – Validacija formulara

Ime:

Prezime:

Broj indeksa:

Slika 11.16. Primer validacije elemenata forme u jQuery-ju

Primenom jQuery-ja vrlo lako se može selektovati i grupa radio tastera. Ako se pretpostavi da su dati elementi forme kao:

```

<input type="radio" name="login" value="Google account"/> sa Google
nalogom
<input type="radio" name="login" value="Studentski nalog"/> sa
studentskim nalogom

```

Sledećim kodom se lako može dohvatiti value vrednost čekiranog radio tastera. Korišćena je selekcija `:radio`, što ukazuje na elemente koji imaju HTML atribut `type="radio"`, ali kako ih može biti više u jednoj strani, lokalizovani su na one koji imaju `name` atribut sa vrednošću `login`.

```
$(document).ready(function(){
    $('#formLogin :radio[name=login]').change(function(){
        var nacinLogovanja = $(this).val();
        alert("Izabrali ste: " + nacinLogovanja);
    });
});
```

Česta primena klijentskih jezika je ranije opisana kontrola unetog sadržaja. U slučaju da je, nakon sabmitovanja forme, podatke prvo potrebno proveriti na klijentskoj strani, pa tek onda poslati dalje ako su u redu tj. vratiti korisniku ako ima neka greška, može se koristiti metod *submit()*. Ovaj postupak je u klasičnom JavaScript-u bio definisan upotrebom događaja *onSubmit()* i korišćenjem reči *return*.

Na vrlo sličan način koristi se jQuery metod *submit()*. Ako pretpostavimo da je korisnik uneo neke podatke u tekstualna polja, i kliknuo na taster tipa submit, sledeći kod će proveriti da li postoji bar neki sadržaj u svim tekstualnim poljima i ako pronađe polje u kome nema sadržaja automatski postavi kursor u to polje, kao znak da očekuje unos sadržaja na to mesto.

U tom slučaju slanje podataka serveru se prekida, pomoću *return false*, dok ako je sve u redu vraća se *true* i slanje ka serveru se nastavlja.

```
$(document).ready(function(){
    $("#formWebSite").submit( function(){
        var error = false;
        $(this).find(":text").each(function(){
            if ($(this).val().length == 0) {
                alert("Sva tekstualna polja moraju biti popunjena!");
                $(this).focus();
                error = true;
                return false; // prekida "each" petlju
            }
        });
        if (error) { return false; }
        return true;
    });
});
```

Ukoliko se u jQuery-ju želi koristiti regularni izrazi, najčešće se koriste upotrebom pomenutog metoda *RegExp()*. Pravila definisanja paterna, upotreba metoda i sl. je identična kao i klasičnom JavaScript-u, sem što se patern ne definiše kao klasičan string nego pomoću metoda *RegExp()*. U sledećem primeru, klikom na taster dohvata se uneti sadržaj tekstualnog polja i „šalje“ na proveru funkciji *proveraDatuma()*. Ukoliko je uneti sadržaj ispravan, funkcija vraća *true* nakon čega se poruka o grešci sakriva tj. u suprotnom prikazuje korisniku.

```

$(document).ready(function() {
    $('#greska').hide();

    $('.submit').click(function(event){
        var sadrzaj=$('#datum').val();
        if(proveraDatuma(sadrzaj)){ $('#greska').hide(); }
        else{
            $('#greska').show();
            event.preventDefault();
        }
    });
});

function proveraDatuma(sadrzaj)
{
    var uzorak= new RegExp(/\\b\\d{1,2}[\\/-]\\d{1,2}[\\/-]\\d{4}\\b/);
    return uzorak.test(date);
}

```

Primena jQuery-ja mogu biti vrlo raznolike, ali se pored onih koje se odnose na animacije, efekte i grafiku, ističu one koje se odnose na laku selekciju DOM elemenata. Ovo je posebno izraženo u situacijama kada se kod dobija od servera dinamički i kada se ne može posmatrati *hardcode*-ovano. Na primer, ako se posmatra grupa checkbox-ova, koji mogu imati proizvoljno puno elemenata, često se nameće potreba da se izvrši automatska selekcija cele grupe.

```

<form>
    <input name="prilog" type="checkbox" />Prilog 1<br/>
    <input name="prilog" type="checkbox" />Prilog 2<br/>
    <input name="prilog" type="checkbox" />Prilog 3<br/>
    <input name="prilog" type="checkbox" />Prilog 4<br/>
    <input id="grupa" name="prilog" type="checkbox"/>Svi prilozi<br/>
</form>

```

Ovo je moguće uraditi prozivanjem svakog od elemenata forme, ali je to nepraktično u uslovima kada je taj broj veliki ili inicijalno nedefinisan. U tim situacijama, selekcija, a u ovom slučaju čekiranje svih checkbox-ova, moguća je pomoću tri linije koda.

```

$(document).ready(function(){
    $('.grupa:checkbox').change(function() {
        var svi = ':checkbox[name=' + $(this).attr('name') + ']';
        $(svi).attr('checked', $(this).attr('checked'));
    });
});

```

Dati kod se aktivira klikom na poseban *checkbox*, opisan sa *id*="grupa", kada se selektuju svi drugi checkbox-ovi koji imaju istu vrednost atributa *name*, kao i on. Njihovom selekcijom, pristupa se njihovom atributu *checked* i ono se setuje sa

vrednošću *checked*. Na taj način, sva polja se automatski čekiraju. Na sličan način može se realizovati i njihovo vraćanje u početno stanje. Ovakve i slične primene se mogu primeniti kod selekcije više artikala, prijave za više oglasa, pregleda više brendova i sl.

jQuery programerima pruža izuzetno veliki spektar mogućnosti za programiranje interakcije web sajtova. Biblioteka se širi velikom brzinom nudeći programerima sve lakši i brži rad, a sa druge strane omogućavajući izuzetno kvalitetne i atraktivne grafičke mogućnosti. Ovde su prikazane neke od osnovnih i najkorišćenijih implementacija jQuery-ja i neophodno je da se stalno prate noviteti i mogućnosti novih verzija i načini njihovih primena. Najveći broj programera uživa u radu sa jQuery-jem i smatra to najizazovnijim i najkreativnijim poslom, pa se iskreno nadam da ćete i Vi na isti način doživeti ovu JavaScript biblioteku.

Literatura

- 1 A. Clarke, Search engine optimization 2015: Learn SEO with smart internet marketing strategies, CreateSpace Independent Publishing Platform, 2014.
- 2 A. Freeman, The Definitive Guide to HTML5, Apress, 2011.
- 3 B. Lawson, R. Sharp, *Uvod u HTML 5 za programere*, Mikro knjiga, 2012.
- 4 B. Marchal, *XML by example*, Que, Indiana, 2000.
- 5 J. Sterne, Web Metrics-Proven Methods for Measuring Web Site Success, Wiley Publishing, 2002.
- 6 C. Jones, SEO Step-by-Step - The Complete Beginner's Guide to Getting Traffic from Google, CreateSpace Independent Publishing Platform, 2014.
- 7 Chris Bates, *XML in theory and practice*, John Wiley and Sons, 2003.
- 8 P. Crowder i D. A. Crowder, *Creating Web Sites Bible*, Wiley Publishing, 2008.
- 9 D. Hunter, K. Cagle, D. Gibbons, N. Ozu, J. Pinnock i P. Spencer, *XML od početka*, CET, Beograd, 2001.
- 10 R. Mansfield, *CSS Web Design For Dummies*, Wiley Publishing, 2005.
- 11 D. M. Woods, *HTML5 and CSS*, Cengage Learning, 2012.
- 12 Erik T. Ray, *Learning XML*, O'Reilly & Associates, Inc., 2001.
- 13 W. Suh, Web engineering: Principles and Techniques, Idea Group Publishing, 2005.
- 14 Hakon Wium Lie i Bert Bos, *Cascading Style Sheets: Designing for the Web*, Third Edition, Addison Wesley Professional, 2005.
- 15 J. A. Brannan, *Brilliant HTML & CSS*, Pearson Education Limited, 2009.
- 16 <http://www.w3.org>
- 17 J. Duckett, *HTML and CSS: Design and Build Websites*, Wiley 2011.
- 18 K. Jamsa MBA, K. King i A. Anderson, *HTML i Web dizajn*, Mikro knjiga, Beograd, 2003.
- 19 B. Pfaffenberger, S. M. Schafer, C. White i B. Karow, *HTML, XHTML and CSS Bible*, Wiley Publishing, 2004.
- 20 L. Sikos, Web Standards: Mastering HTML5, CSS3 and XML, Apress, 2014.
- 21 L. McKinnon i Al McKinnon, *XML in 60 minutes a day*, Wiley Publishing Inc., Indiana, 2003.
- 22 H. W. Lie i B. Bos, *Cascading Style Sheets: Designing for the Web*, Third Edition, Addison Wesley Professional, 2005.
- 23 M. E. Holzschlag, *250 HTML and Web design secrets*, Wiley Publishing Inc., Indiana, 2004.
- 24 J. Cohen, *The Unusually Useful Web Book*, New Riders Publishing, 2003.
- 25 M. J. Young, *Korak po korak XML*, CET, Beograd, 2001.

- 26 E. van der Vlist, A. Vernet, E. Bruchez, J. Fawcett i D. Ayers, *Professional Web 2.0 Programming*, Wiley Publishing, 2007.
- 27 R. Connolly i R. Hoar, *Fundamentals of Web Development* Mar 7, Addison-Wesley, 2014.
- 28 R. M. Roberts, *Networking Fundamentals*, Goodheart-Willcox, Laboratory Manual edition, 2011.
- 29 R. Stair i G. Reynolds, *Fundamentals of Information Systems*, Cengage Learning, 2013.
- 30 R.L. Adams, *SEO Fundamentals: An Introductory Course to the World of Search Engine Optimization*, The SEO University Book, Amazon Digital Services, Inc., 2014.
- 31 S. Sarris, *HTML5 Unleashed* Jul 26, Sams Publishing, 2013.
- 32 T. Felke-Morris, *Web Development and Design Foundations with HTML5*, Addison-Wesley, 2014.
- 33 K. Patel, *Incremental Journey for World Wide Web: Introduced with Web 1.0 to Recent Web 5.0—A Survey Paper*, International Journal of Advanced Research in Computer Science and Software Engineering, 3(10), 410-417, 2013.
- 34 B. P. Hogan, *HTML5 and CSS3: Level Up with Today's Web Technologies*, Pragmatic Bookshelf, 2013.
- 35 N. Shadbolt, W. Hall, T. Berners-Lee, *The semantic web revisited*, Intelligent Systems, 2006.
- 36 L. Richardson, S. Ruby, *RESTful web services*, O'Reilly Media, Inc, 2008.
- 37 A. Maedche, *Ontology learning for the semantic web*, Springer Science & Business Media, 2002.
- 38 P. Andersen, *What is Web 2.0?: ideas, technologies and implications for education*, Vol. 1, No. 1. Bristol, UK, 2007.
- 39 C. Fuchs, K. Boersma, A. Albrecht, M. Sandoval, *Internet and surveillance: The challenges of Web 2.0 and social media*, Vol. 16, Routledge, 2013.
- 40 M. Eisenstadt, T. Vincent, *The knowledge web: Learning and collaborating on the net*, Routledge, 2012.
- 41 B. Clifton, *Advanced web metrics with Google Analytics*, John Wiley & Sons, 2012.
- 42 B. Frain, *Responsive web design with HTML5 and CSS3*, Packt Publishing Ltd, 2012.
- 43 S. Aghaei, M. A. Nematbakhsh, H. K. Farsani, *Evolution of the world wide web: From WEB 1.0 TO WEB 4.0*, International Journal of Web & Semantic Technology, 3(1), 1-10, 2012.
- 44 Deni Gudman, "JavaScript Biblija", Mikro knjiga, Beograd, 2001.
- 45 Michael J. Young, "Korak po korak XML", CET, Beograd, 2001.

- 46 Arman Danesh, "JavaScript in 10 simple steps or less", Wiley Publishing Inc., Indiana, 2004.
- 47 Thomas Powell and Fritz Schneider, "JavaScript 2.0: The Complete Reference", Second Edition, McGraw-Hill/Osborne, 2004.
- 48 Tom Negrino, Dori Smith, "Java Script za world wide web", CET, 2005.
- 49 Benoit Marchal, "XML by example", Que, Indiana, 2000.
- 50 David Hunter, Kurt Cagle, Dave Gibbons, Nikola Ozu, Jon Pinnock, Paul Spencer, "XML od početka", CET, Beograd, 2001.
- 51 Michael J. Young, "Korak po korak XML", CET, Beograd, 2001.
- 52 Linda McKinnon, Al McKinnon, "XML in 60 minutes a day", Wiley Publishing Inc., Indiana, 2003.
- 53 Molly E. Holzschlag, "250 HTML and Web design secrets", Wiley Publishing Inc., Indiana, 2004.
- 54 Earle Castledine, Craig Sharkie, "JQUERY-Novice to NINJA", SitePoint Pty. Ltd. 2010.
- 55 B.M. Harwani, "JQUERY recipes", Paul Manning 2010.
- 56 Yehuda Katz, John Resig, "jQuery in Action", Manning Publications 2008.
- 57 <http://www.w3.org>
- 58 <http://www.javascript.com>
- 59 <http://jquery.com/>
- 60 J. Larsen, Get Programming with JavaScript. Manning Publications Co., 2016.
- 61 L. Atencio, Functional Programming in JavaScript: How to improve your JavaScript programs using functional techniques. Manning Publications Co., 2016.
- 62 F. Alvaro, "JAVASCRIPT: Easy JavaScript Programming For Beginners. Your Step-By-Step Guide to Learning JavaScript Programming." (2016).
- 63 J. Duckett, JavaScript & jQuery. Wiley VCH, 2015.
- 64 S. Powers, JavaScript Cookbook: Programming the Web. " O'Reilly Media, Inc.", 2015.
- 65 P. Ballard, Sams teach yourself JavaScript in 24 hours. Sams, 2015.
- 66 M. Chaudhary, K. Ankur, Practical jQuery. Apress, 2015.
- 67 T. Miles, jQuery Essentials. Packt Publishing Ltd, 2016.
- 68 N. Kojić, Web dizajn - detaljan priručnik, II izdanje, Visoka ICT škola, Beograd, 2017.

<html>...

Kako se i kod web strane približava kraju elementa HTML tako smo se i mi približili kraju ove knjige. Primarni zadatak ovog udžbenika je bio da čitaoca uvede u svet HTML-a, CSS-a i JavaScript-a. Da ga od početka upozna sa ciljevima ovih jezika, njihovim pravilima i primenama. Kako se ovi jezici konstantno unapređuju, menjaju i modifikuju, jasno je da ni jedna knjiga ne može prikazati sve detalje i moguće primene. Zato je zadatak da se korisnici, nakon ove materije, upoznaju sa jezicima koji se ovde objašnjavaju i osposobe da samostalno nastave da uče i unapređuju svoja znanja i veštine.

Najbitniji aspekt učenja web-a je „kucanje“ koda. Često se najbolje uči na greškama, na koje se izgubi mnogo vremena da se pronađu i reše. Ovo su realne i očekivane situacije koje nikako ne treba da usporavaju ili obeshrabre početnike. Naprotiv, treba da im daju volju da ih što pre reše i idu dalje na svom putu u izuzetan svet web dizajna i web programiranja.

```
<script type="text/javascript">
$(document).ready(function(){
    $('#kraj).autocomplete(podaci,{
        autor: Nenad Kojic,
        poruka: Kucamo se na web-u
    });
});
```

...</html>

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.42:004.738.12(075.8)

КОЈИЋ, Ненад С., 1977-

WEB dizajn : HTML, CSS i JavaScript / Nenad Kojić. - 4. izd. - Beograd : Univerzitet
Singidunum, 2024 (Beograd : Birograf). - 379 str. : ilustr. ; 24 cm

Tiraž 400. - Bibliografija: str. 375-377.

ISBN 978-86-7912-661-0

a) Web презентације -- Програмирање

COBISS.SR-ID 148313097

© 2024.

Sva prava zadržana. Nijedan deo ove publikacije ne može biti reprodukovan u bilo kom vidu i putem
bilo kog medija, u delovima ili celini bez prethodne pismene saglasnosti izdavača.



Nenad Kojić

Web dizajn

HTML, CSS i JavaScript



Intenzivna primena Interneta doprinela je velikom povećanju broja i kvaliteta web sajtova. Skoro svaka firma, pojedinac, roba ili usluga može se pronaći na web-u tj. na nekom od web sajtova. Danas postoji preko 700 miliona sajtova i njihov broj se stalno povećava kao i potreba za web programerima. Ova knjiga se bavi programskim jezicima i tehnikama kojima se prave komercijalni interaktivni web sajtovi. Korisnici se od početka uvode u materiju HTML-a, CSS-a i JavaScript-a sa ciljem da steknu veštine i znanja za samostalnu izradu web sajtova.