Veb programiranje

Beleške sa vežbi

Aleksandar Veljković

Radna verzija koja se proširuje nakon svakog časa, greške su moguće i ukoliko ih primetite pošaljite obaveštenje na email kako bi greška u najskorijem roku bila ispravljena. Moguće su i blagovremene dopune i manje izmene sadržaja uz obaveštenje o izmenama.

Jezik HTML

HTML (*HyperText Markup Language*), je jezik za obeležavanje tekst koji se koristi za opisivanje sadržaja i strukture Veb stranica. Jezik se sastoji od **elemenata** koji se predstavljaju korišćenjem etiketa (tagova), pri čemu postoji tri vrste oznaka:

- **Otvarajuće** – `<ETIKETA>`
- **Zatvarajuće** - `</ETIKETA>`
- **Samozatvarajuće** - `<ETIKETA/>`
  u HTML verziji 5 dopušteno je korišćenje oblika `<ETIKETA>` I za samozatvarajuće etikete

Nazivi etiketa nisu "osetljivi" na veličinu slova (*case sensitive*). Etikete moraju biti **pravilno uparene**, posle otvarajuće etikete mora slediti ili odgovarajuća zatvarajuća ili druga otvarajuća (ili samozatvarajuća) koja zatim mora biti pravilno uparena svojom odgovarajućom zatvarajućom etiketom pre zatvaranja prethodno otvorene.

Ispravan primer:

```
<ETIKETA1>
   <ETIKETA2>
   </ETIKETA2>
</ETIKETA1>
```

Neispravan primer:

```
<ETIKETA1>
      <ETIKETA2>
      </ETIKETA1>
</ETIKETA2>
```

Uvlačenje redova (indentacija) obezbeđuje preglednost koda i nije obavezujuće, ali je korišćenje istog vrlo preporučljivo radi čitljivosti.

Jezik HTML ima ulogu opisivanja **logičke structure sadržaja stranice** i **ne određuje način vizuelnog prikaza elemenata**. Elementi se bliže opisuju korišćenjem **atributa**. Atributi se zadaju u okviru otvarajućih (ili samozatvarajućih) etiketa u obliku:

```
NAZIV=“VREDNOST“ ili NAZIV=’VREDNOST’
```

Struktura HTML dokumenta

Prva linija HTML dokumenta sadrži deklaraciju tipa (*doctype*) koja **ne pripada** jeziku HTML ali omogućava pregledaču da razume šta predstavlja sadržaj u ostatku datoteke. Deklaracija tipa za jezik HTML 5 je:

Primer

```
<!DOCTYPE html>
```

Svaki HTML document sadrži koreni **html** element:

Primer

```
<html> </html>
```

Unutar elementa html navode se dva osnovna strukturna segmenta HTML dokumenta, **head** (zaglavlje) I **body** (telo).

Primer:

```
<html>
      <head>
</head>

     <body>
</body>
</html>
```

Zaglavlje (**head**) sadrži metapodatke o stranici, kao što su naziv autora, jezik kojim je stranica pisana, kodiranje korišćeno za zapis teksta, ključne reči i slično. Unutar zaglavlja zadaju se i datoteke koje će dodatno biti uključene, kao što su datoteke sa stilovima, skriptovima i slično. Metapodaci se zadaju u okviru **meta** elemenata, najčešće atributima **name**/**content**. Meta elementi koriste samozatvarajuće etikete.

Primeri sadržaja nekih meta elemenata:

```
<meta charset="UTF-8">
<meta name="description" content="Opis stranice">
<meta name="keywords" content="Ovo,su,kljucne,reci">
<meta name="author" content="Ime Autora">
```

Najbitniji element zaglavlja je naslov stranice (**title**). Tekst naslova se prikazuje u kartici pregledača, kao i u rezultatima pretrage kao naslov rezultata.

Primer:

```
<head>
      <title> Naslov </title>
</head>
```

Telo dokumenta (**body**) sadrži elemente koji će se prikazivati na stranici, u prozoru pregledača. Body je element omotač za sve sve elemente koje sadrži u sebi I takođe se prikazuje u prozoru pregledača (zajedno sa **html** elementom koji obuhvata sve elemente HTML dokumenta prikazane u prozoru pregledača).

Jedan od vidljivih strukturnih elemenata koji se navode unutar body elementa je naslov (*heading*) – **h1**

Primer:

```
<h1> Naslov teksta </h1>
```

Konačno, primer strukture jednog HTML5 dokumenta bio bi:

```
<!DOCTYPE html>

<html>
      <head>
            <title> Naslov stranice </title>
            <meta charset="UTF-8">
            <meta name="description" content="Opis stranice">
      </head>

      <body>
            <h1> Naslov teksta </h1>
      </body>
</html>
```

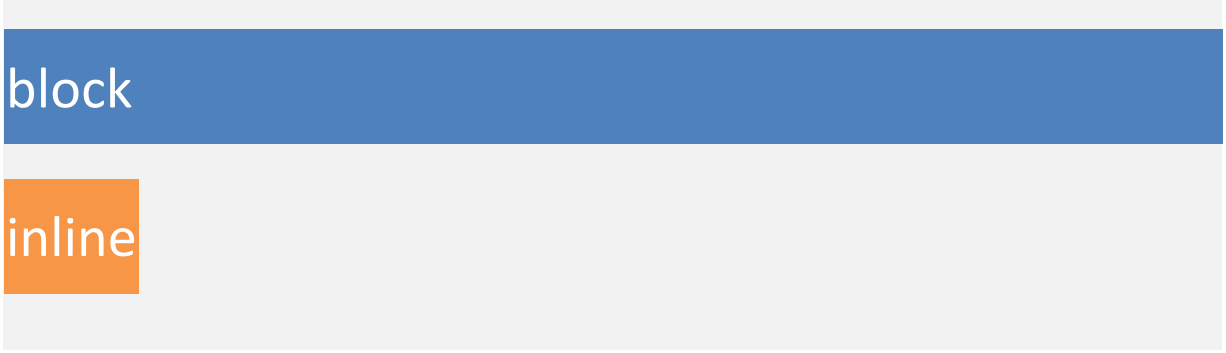
HTML elementi

Osnovna podela vidljivih HTML elemenata je podela na:

- blokovske (**block**)
- i linijske (**inline**) elemente

Block elementi zauzimaju celokupnu širinu elementa koji ih sadrži. Ukoliko je roditelj **block** elementa **body** element, element će zauzimati celu širinu prozora pregledača (*viewport*). **Inline** elementi zauzimaju širinu dovoljnu za prikaz sadržaja elementa (*Slika 1)*. Po pravilu, **block** elementi mogu sadržati druge **block i inline** elemente dok **inline** elementi mogu sadržati **samo druge inline elemente**. Odstupanje od pravila će pregledač pokušati da prikaže ali ispravan ishod prikaza nije zagarantovan.

Primer:



Slika 1. Inline i block elementi

Uvodni pregled nekih HTML elemenata

Naslovi u HTML-u imaju **6 mogućih nivoa** i označavaju se etiketama od **h1** do **h6**. Naslov **h1** je naslov najvišeg nivoa, dok naslovi sa manjih rednim brojevima označavaju podnaslove nižih nivoa. Naslovi su **block** elementi.

Primer:

```
<h1> Naslov </h1>
<h2> Podnaslov </h2>
<h3> Podpodnaslov </h3>
     . . .
```

Pasus teksta prikazuje se elementom p (*paragraph*). Pasusi su takođe block elementi.

Primer:
```
<p> Ovo je jedan pasus teksta </p>
<p> Ovo je drugi pasus teksta </p>
```

Za prikaz naglašenog teksta možemo koristiti element **strong**.

Primer:
```
<p> Ovaj pasus sadrzi <strong> naglašenu </strong> rec </p>
```

Ukoliko je potrebno preći u novi red teksta, to se može učiniti korišćenjem elementa **br** (*break)*.

Primer:
```
<p> Prvi red <br/> drugi red </p>
```

Za zapis teksta uključujući sve beline koristi se element **pre**.

Primer:
```
<pre> Ovaj tekst       sadrzi
      Beline
 </pre>
```

Element **strong** se najčešće prikazuje kao boldovan tekst, ali sam prikaz je volja pregledača jer **HTML ne propisuje način prikaza elementa**. Element **strong** je inline **element**.

Za prikaz istaknutog teksta koristi se element **em**.

Primer:
```
<p> Ovaj pasus sadrzi <em> naglasenu </em> rec </p>
```

Element **em** se najčešće prikazuje kao *italic* tekst, ali, ponovo, **HTML ne propisuje način prikaza elemenata**. Element **em** je **inline** element.

Veze predstavljaju mehanizam povezivanja stranica. Putem linkova vrši se, najčešće, prelazak sa jedne stranice na drugu, moguć je prelaz i na određeni segment trenutne stranice (više o linkovima u narednim poglavljima). Veze su zadate elementom **a** (*emphasize*) dok se putanje navode u okviru atributa **href**, putanja ne počinje nazivom protokola putanje su relativne. Veze su **inline** elementi.

Primer:
```
<a href=”stranica2.html”>Ovo je veza ka stranici2</a>
<a href=http://www.google.com>Google.com</a>
```

Jezik HTML definiše tri vrste lista, uređene (**ol** – *Ordered List*), neuređene (**ul** – *Unordered List*), definicione (**dl** – *Definition List*). Elementi uređenih i neuređelih lista su **block** elementi i navode se etiketama **li** (*List Item*). Elementi definicionih lista su **dt** (naziv pojma) I **dd** (definicija pojma).

Primer:

```
<ul>
  <li> Stavka </li>
  <li> Stavka </li>
  <li> Stavka </li>
</ul>

<ol>
  <li> Stavka </li>
  <li> Stavka </li>
  <li> Stavka </li>
</ol>

<dl>
  <dt> Pojam1 </dt>
  <dd> Definicija pojma 1 </dd>
  <dt> Pojam2 </dt>
  <dd> Definicija pojma 2 </dd>
</dl>
```

Slike se navode korišćenjem **img** elementa. Atributi img elementa su **src** (*source*, putanja do slike) i **alt** (*alternative*, alternativni tekst koji se ispisuje u slučaju da slika nije učitana ili je putanja do slike neispravna. Slike su **inline** elementi.

Primer:

```
<img src=”slika.jpg” alt=”Ovo je slika”>
```

Tabele su **inline** elementi I navode se etiketama **table**. Redovi tabela su posebni elementi koji se označavaju etiketama **tr** (*Table Row*), dok su polja u okviru redova označena etiketama **td** (*Table Data*). Naslovi kolona označavaju se etiketama **th** (*Table Heading*). Po pravilu, red koji sadrži naslove kolona smešta se u element zaglavlja tabele, **thead**, dok se ostali redovi smeštaju u element tela tabele, **tbody**.

Primer:

```
<table>
  <thead>
    <tr>
      <th> Kolona 1 </th>
      <th> Kolona 2 </th>
    </tr>
  <thead>

  <tbody>
    <tr>
      <td> A </td>
      <td> B </td>
    </tr>
  </tbody>
<table>
```

Atributom **title** se elementima dodaje naslov koji je vidljiv u okviru malog pravougaonika (*tool tip*) koji se pojavljuje prilikom prelaska kursora preko elementa.

Primer:

```
<h1 title=”Ovo je naslov> Naslov </h1>
```

CSS – Cascading Stylesheet

Jezik **CSS** se koristi za opisivanje vizuelnog prikaza HTML elemenata. Osobine elemenata navode se naredbama oblika:

Primer:

```
NAZIV_SVOJSTVA : VREDNOST;
```

Komentari se u jeziku CSS navode unutar /* i */.

Primer:

```
/* Ovo je komentar */
```

CSS naredbe se mogu navoditi u okviru **style atributa** elemenata, u okviru zasebnog **style elementa**, koji se navodi u okviru zaglavlja stranice ili **u odvojenoj datoteci** koja se uključuje korišćenjem elementa **link**.

Primer:

```
<h1 style=”color: red”> Crveni naslov </h1>
```

```
<head>
   <style>
     h1 {
      color: red;
    }
   </style>
</head>
```

```
<head>
  <link rel=”stylesheet” type=”text/css” href=”style.css”>
</head>
```

Korišćenjem elementa link uključuju se dodatne datoteke, pa je moguće i uključivanje ikonice koja će se prikazivati na kartici pregledača

```
<link rel="icon" type="image/png" href="ikonica.png">
```

Različite pozicije navođenja CSS naredbi imaju različite prioritete. Najviši prioritet imaju svojstva zadata unutar **style atributa**, zatim svojstva u okviru **style elementa** i najniži prioritet imaju svojstva iz **uključene datoteke**.

Dok je za naredbe zadate u okviru **style atributa** jasno na koji element se odnose, za naredbe navedene u okviru **style elementa** ili **uključene datoteke** potrebno je naglasiti na koji element se svojstva odnose. Ovo je omogućeno korišćenjem **CSS selektora** (*CSS Selectors*). Više o CSS selektorima biće rečeno na narednim časovima, ali će nekoliko osnovnih selektora ovde biti navedeno kao primer.

Svojstva koja važe za sve elemente nekog tipa zadaju se **selektorom elementa**.

Primer:

```
h1 {
color: red;
}
```

Elementima se mogu zadavati identifikatori korišćenjem atributa **id**. Identifikatori su po pravilu **jedinstveni za svaki element i neispravno je navoditi dva elementa koji imaju isti identifikator**.

Svojstva koja važe za element sa zadatim identifikatorom zadaju se **selektorom identifikatora**, koji je oblika **#naziv_identifikatora**.

Primer:

```
<h1 id=”naslov”> Naslov teksta </h1>
```

```
. . .

#naslov {
color: red;
}
```

Ukoliko je potrebno zadati svojstva određenoj grupi elemenata, takva grupa se određuje klasom. Naziv klase navodi se atributom class. Dozvoljeno je da više elemenata pripada istoj klasi.

Svojstva koja važe za elemente koji pripadaju zadatoj klasi zadaju se **selektorom klase**, koji je oblika **.naziv_klase**. Ukoliko je potrebna preciznija selekcija, ograničena samo na određene elemente koji pripadaju klasi, selektor je oblika **element.klasa**

Primer:

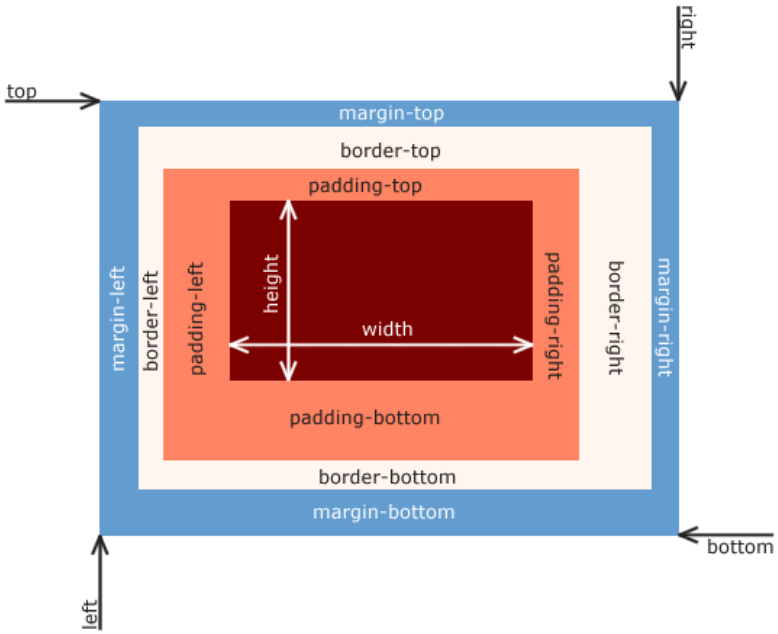
```
<h1 class=”naslovi”> Naslov teksta </h1>
<h2 class=”naslovi”> Još jedan naslov </h1>



. . .

.naslovi {
color: red;
}
```

Ili

```
h1.naslovi {
color: red;
}
```

Box model

Box model (model kutije) predstavlja pravougaoni omotač prostora koji element zauzima prilikom prikaza. Sastoji se od nekoliko omotača, **content** – sadržaj elementa, **padding** – unutrašnje margine, **border** – ivice i **margin** – spoljašnje margine elementa (*Slika 2*).



Slika 2. Box model

Vidljiva površina elementa, korišćenjem podrazumevanih parametara, računa se kao zbir content + padding + border. Svojstva **width** i **height** odnose se na **content box**, ali su vidljiva visina i širina izračunate kao zbir veličina omotača **content box**, **padding box** i **border box**. Navođenjem CSS svojstva **box-sizing** može se navesti na koji bi način pregledač trebalo da tumači svojstva **width** i **height**. Podrazumevana vrednost **box-sizing** svojstva je **content-box**, kojim se navodi da se visina i širina odnose na **content box**. Navođenjem vrednosti **border-box** zadaje se pregledaču da, prilikom prikaza, vrednosti svojstava **width** i **height** primenjuje na ukupnu površinu obuhvaćenu **border box** omotačem. Vrednost **padding-box** ima značenje analogno prethodno navedenim vrednostima sa razlikom da se visina i širina odnose na površinu ograničenu **padding box** omotačem.

Dimenzije omotača **padding** zadaju se u kao odstojanje od **content** omotača. Vrednosti mogu biti navedene na nekoliko načina:

Primer:

```
/* 1. Unutrašnje margine su zadate odstojanjem 10 piksela od donje,
gornje, leve i desne ivice content omotača: */
```

```
padding: 10px;

/* 2. Unutrašnje margine su zadate odstojanjem 10 piksela od gornje
i donje ivice content omotača, dok odstojanje od leve i desne ivice
content omotača iznosi 20 piksela:  */

padding: 10px 20px;

/* Unutrašnje margine su zadate odstojanjem 10 piksela od gornje
ivice, 20 piksela od desne, 30 piksela od donje i 40 piksela od
leve ivice content omotača (pravac kretanja kazaljke na satu):  */

padding: 10px 20px 30px 40px;
```

Zadavanje veličine margina zadaje se **margin** svojstvom, analogno zadavanjem veličina unutrašnjih margina. Veličine ivica zadaju se svojstvom **border** navodeći širinu ivice, tip ivice (puna linija – **solid**, isprekidana – **dashed**, „tačkasta“ – **dotted**) i boju ivice.

Primer:
```
border: 2px solid green;
```

Pozicioniranja i kontejnerski elementi

Pozicioniranje elementa određuju poziciju na kojoj će se element prikazati. Način pozicioniranja elementa navodi se korišćenjem CSS svojstva **position**. CSS propisuje četiri načina pozicioniranja elemenata – **static**, **relative**, **fixed** i **absolute**. Ako vrednost svojstva **position** nije navedena, podrazumevana vrednost je **static**.

Elementi sa **static** pozicioniranjem se prikazuju u normalnom toku, element se pozicionira na prvom slobodnom prostoru dovoljno velikom za njegovo smeštanje.

Relativno pozicioniranje (**relative**) pozicionira element relativno u odnosu na poziciju na kojoj bi se u normalnom toku našao. Odstupanje od početne pozicije navodi se svojstvima **top**, **bottom**, **left** i **right**. Elementi koji slede nakon relativno pozicioniranog elementa pozicioniraju se u normalnom toku dok prostor na kome bi se relativno pozicionirani element našao (bez ikakvog pomeranja) ostaje nepopunjen. Ukoliko se nađu dva ili više relativno pozicioniranih elemenata jedan za drugim, pojavljuje se potreba za novom, trećom dimenzijom, elementi koji su kasnije definisani prikazuju se preko elemenata koji su prethodno definisani. Redosled elemenata po „z osi“ zadaje se korišćenjem svojstva **z-index**. Elementi sa većim vrednostima **z-index** prikazuji se **ispred** elemenata sa nižim vrednostima.

Fiksno pozicioniranje (**fixed**) pozicionira element fiksno u odnosu na prozor pregledača (*viewport*). Elementi sa fiksnim pozicioniranjem se ne skroluju zajedno sa stranicom. Svojsvima **top**, **bottom**, **left** i **right** zadaje se **pomeraj od ivica pregledača**. Svojstvo **z-index** se može primenjivati na fiksno pozicionirane elemente.

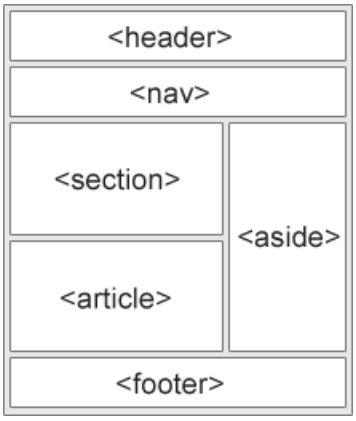
Apsolutno pozicioniranje (**absolute**), pozicionira element u odnosu na prvog **ne-static** pretka (prvog elementa u hijerarhiji koji nije statično pozicioniran), ukoliko takav predak ne postoji, pozicioniranje se vrši u odnosu na **body** element. Svojstvima **top**, **bottom**, **left** i **right** zadaje se pomeraj **od ivica ne-static roditelja ili body elementa**. Element koji sledi nakon apsolutno pozicioniranog elementa zauzima prostor koji je ostao slobodan iza apsolutno pozicioniranog elementa dok se za apsolutno pozicionirani element može smatrati da je u zasebnom sloju, nezavisan od normalnog toka.

Element **div** (*division*) je element kojim se grupišu drugi elementi u zajedničke celine. Element **div** je **block** element pa može sadržati I **block** I **inline** elemente. **Inline** element koji se može koristiti za grupisanje drugih **inline** elemenata je element **span**.

Primer:

```
<div>
  <h1> Naslov teksta </h1>
  <p> Pasus teksta </p>
</div>

<span>
  <a href="#"> Link 1 </a>
  <a href="#"> Link 2 </a>
</span>
```

Jezik HTML5 uvodi nove kontejnerske, strukturne, elemente koji se ponašaju kao **div** elementi, ali su nazvani po nameni koji imaju na stranici (*slika 3*). To su elementi: **header**, **nav**, , **aside**, **article**, **section**, **footer.**



Slika 3. Organizacija novih sturkturnih HTML5 elemenata

CSS selektori i pseudo klase

CSS selektorima se navodi za koji element se zadaju svojstva, u nekom od prethodnih poglavlja prikazani su selektori po tipu elementa, identifikatoru i klasi. Spisak svih CSS selektora može se naći na stranici: https://www.w3schools.com/cssref/css_selectors.asp.

Neki od češće korišćenih selektora su:

div, p – Zarezom su odvojeni selektori svih elemenata na koje je potrebno primeniti svojstva

div p – Razmakom se selektuju svi naslednici elementa, selektor u primeru obuhvata sve pasuse koji se nalaze unutar div elemenata.

div > p – Znakom **>** selektuju se svi elementi kojima je navedeni element **direktni predak** (roditelj). Selektor u primeru obuhvata sve pasuse kojima je roditelj div element.

div + p – Znakom **+** selektuju se svi elementi koji neposledno u HTML dokumentu slede nakon zadatog elementa. Selektor u primeru obuhvata sve paragrafe pre kojih je neposredno definisan div element.

Pseudo klase elementi dobijaju kada dođu u neko stanje, to stanje može biti prelazak kursora preko elementa, klik na element, ili čak kada je element n-ti po redu potomak nekog elementa. Selektori pseudo klasa počinju znakom **:**

Primeri nekih selektora pseudo klasa

:hover – pseudo klasa koja se dodeljuje prelaskom kursora preko elementa.

:focus – pseudo klasa koja se dodeljuje dovođenja elementa u fokus, klikom na element ili korišćenjem tab dugmeta.

:active – pseudo klasa aktivnih linkova (klik miša je pritisnut ali ne i pušten).

:visited – Stranica ka kojoj vodi link je posećena već putem tog linka

:link – Link je neposećen

:nth-child(n) – Element sa ovom pseudo klasom je n-ti direktni naslednik. n je fiksni broj koji se zadaje, ali je moguće i navođenje vrednosti **odd** ili **even** (neparni ili parni brojevi).

Uređivanje pozadine elemenata

Uređivanje izgleda pozadine elemenata vrši se **background** svojstvima. Boja pozadine elementa uređuje se svojstvom **background-color**.

Primer:
```
h1 {
  background-color: red;
}
```

Pozadinska slika elementa zadaje se svojstvom **background-image**. Putanja do slike koja se prikazuje na pozadini elementa zadaje se funkcijom **url**.

Primer:
```
h1 {
  background-image: url('slika.jpg');
}
```

Veličina prostora koji će pozadinska slika zauzimati unutar elementa zadaje se svojstvom **background-size**. Moguće vrednosti ovog svojstva su: **auto** (podrazumevana vrednost, slika se prikazuje u punoj veličini), **contain** (slika zauzima maksimalan prostor koji može zauzeti unutar granica elementa, ali tako da se odnos visine i širine ne naruši), **cover** (slika se „razvlači" tako da je ceo element pokriven slikom, odnos visine i širine slike nije narušen) i **eksplicitnim zadavanjem dimenzija slike**.

Primer:
```
h1 {
```

```
  background-image: url('slika.jpg');
  background-size: cover;
}
```

Pozicija pozadinske slike određena je svojstvom **background-position**. Vrednosti ovog svojstva određuju pozicije elementa horizontalno i vertikalno u odnosu na element. Ukoliko je navedena samo jedna vrednost pozicije (umesto dve za horizontalno i vertikalno) druga vrednost je podrazumevano **center**.

Primer:

```
h2 {
  background-image: url('slika.jpg');
  background-position: top left;
}

h1 {
  background-image: url('slika.jpg');
  background-position: center;
  /* Druga vrednost je podrazumevano center */
}
```

Svojstvom **background-attachment** može se zadati da se pozadinska slika skroluje – **scroll** ili ne skroluje **fixed** zajedno sa stranicom.

Primer:

```
h1 {
  background-image: url('slika.jpg');
  background-attachment: fixed;
}
```

Prikaz elemenata

U odeljku o HTML elementima pomenuta su dva osnovna načina prikaza elemenata, **block** i **inline**. Elementu se eksplicitno može zadati da se ponaša kao **block** ili **inline** element koristeći svojstvo **display**.

Primer:
```
h1 {
  display: inline;
}
```

Vrednost **inline-block** svojstva **display** nalaže elementu da se ponaša delom kao **inline** a **delom** kao blok element. U tom slučaju, element će zauzimati samo onoliko prostora koliko mu je potrebno za prikaz sadržaja (kao što je slučaj sa **inline** elemetima), ali je moguće zadati određenu širinu elementa (kao što je slučaj sa **block** elementima).

Primer:
```
h1.inline {
  display: inline;
  background-color: red;
  width: 500px; /* Ova svojstvo neće biti prihvaćeno */
}

h1.inline_block {
  display: inline-block;
  background-color: blue;
  width: 500px; /* Ovo svojstvo će biti prihvaćeno */
}
```

Ukoliko je potrebno sakriti element, ne prikazati ga, to se može ostvariti zadavanjem vrednosti **none** svojstvu display. **Napomena,** `display:none` i `opacity: 0` **ne daju isti rezultat**. Svojstvom **opacity** se zadaje **providnost elementa**, ali element iako je providan, **ostaje vidljiv** na ekranu u smislu da zauzima prostor predviđen za njegovo prikazivanje.

Kada se eksplicitno zadaju visina i širina elementa moguće je da sadržaj koji se nalazi u okviru elementa zahteva veću visinu ili širinu za ispravan prikaz nego što je eksplicitno navedeno. U tom slučaju prikaz sadržaja koji izlazi van granica elementa možemo kontrolisati svojstvom **overflow**. Ukoliko je potrebno sakriti sadržaj koji izlazi izvan okvira elementa, zadaje se vrednost **hidden** svojstva **overflow**. Podrazumevana vrednost **overflow** svojstva je **visible**, pri čemu se sav sadržaj, koji izlazi iz okvira elementa koji ga sadrži, prikazuje. Moguće je zahtevati da se sadržaj koji prelazi izvan okvira elementa skroluje unutar elementa i to se ostvaruje postavljanjem vrednosti

svojstva **overflow** na **scroll**. Moguće je pojedinačno postaviti pravila za prikaz sadržaja koji izlazi ivan okvira elementa po horizontali i po vertikali korišćenjem svojstava **overflow-x** i **oveflow-y**.

Float elementi

Float ili plutajući elementi postavljaju se uz zadatu ivicu elementa koji ih sadrži. Element postaje plutajući element postavljanjem svojstva **float** na vrednost **left** ili **right**, u zavisnosti od ivice uz koju je potrebno prilepiti element. Susedni elementi sa istim vrednostim **float** svojstva pozicioniraju se jedan do drugog na zadatoj strani. Ukoliko na ekranu nema prostora za prikaz **float** elemenata u istoj linije, element za koju nema dovoljno prostora prelazi u narednu liniju. **Flex** elementi ne narušavaju normalan tok, ali utiču na prikaz teksta, na taj način što se tekst obmotava oko njih. Ako je potrebno da float elementi ne utiču na sadržaj narednog element u toku, narednom elementu se svojstvo **clear** postavlja na **left**, **right** ili **both** u zavisnosti od toka uticaj kojih **float** elemenata je potrebno poništiti. Margine elementa koji ima postavljenu svojstvo **clear** na odgovarajuću vrednost i dalje zalaze iza **float** elemenata. Ovaj problem je moguće rešiti smeštanjem **float** elemenata unutar drugog omotač elementa a zatim pseudoklasi **::after** omotač elementa postaviti svojstva **display**, **clear** i **content** na **block**, **both** i “” (prazna niska) u navedenom redosledu vrednosti.

Primer:

HTML:

```
<div id=”omotac”>
  <div id=”levi”></div>
  <div id=”desni></div>
</div>
<p> Tekstualni sadrzaj pasusa </p>
```

CSS:

```
#omotac::after {  /* Testirati promene bez navedene pseudo klase */
  display: block;
  clear: both;
  content: “”;
}

#levi {
  float: left;
  height: 100px;
  background-color: red;
```

```
}

#desni {
  float: right;
  height: 100px;
  background-color: blue;
}

p {
  margin-top: 20px;
}
```

Flex elementi

Trenutno poslednja verzija jezika CSS, verzija 3, donela je novi način prikaza elemenata, prikaz **flex** (flexible box). Da bi se element ponašao kao **flex** element potrebno je postaviti vrednost svojstva **display** na **flex**. Prikaz **flex** ne utiče na nači prikaza elemenata unutar elementa, već utiče na njihov raspored. Elementima unutar **flex** elementa, korišćenjem **svojstva flex** zadaje se relativni odnos veličina elemenata u odnosu na omotač.

Primer:

HTML:

```
<div id=”omotac>
  <div id=”element1”></div>
  <div id=”element2”></div>
  <div id=”element3”></div>
</div>
```

CSS:

```
#omotac {
  display: flex;
}

#element1 {
  flex: 1;
  height: 100px;
  background-color: red;
}

#element2 {
  flex: 2;
  height: 100px;
```

```
  background-color: green;
}

#element3 {
  flex: 1;
  height: 100px;
  background-color: blue;
}
```

Elementi unutar elementa čiji je **id** „omotac“ će podeliti prostor omotača horizontalno u odnosu 1 : 2 : 1. Ukoliko je potrebno deliti prostor vertikalno, elementu omotaču se navodi svojstvo **flex-direction** sa vrednošću **column** (podrazumevana vrednost svojstva **flex-direction** je **row**). Za eksplicitno zadavanje širine elemenata unutar flex elementa koristi se svojstvo **flex-basis** koje je ekvivalentno svojstvu **width**.

Redosled prikaza elemenata sa **svojstvom flex** se može eksplicitno zadati svojstvom **order** čija vrednost predstavlja redni broj elementa prilikom prikaza.

Primer:

HTML:

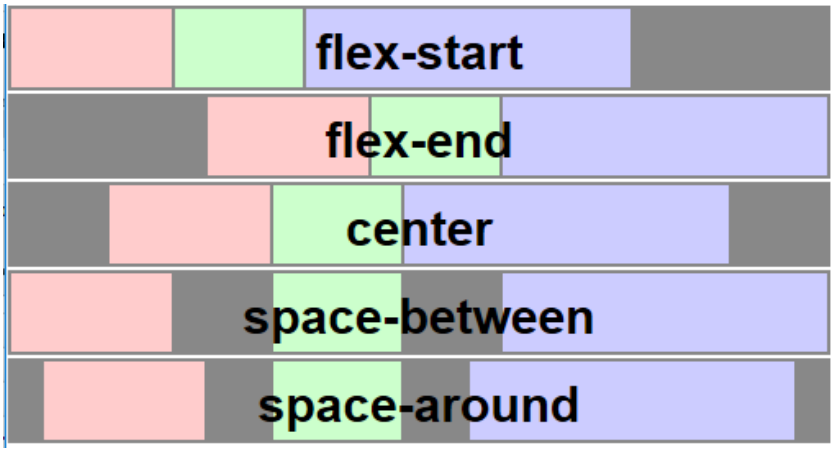
```
<div id=”omotac>
  <div id=”element1”></div>
  <div id=”element2”></div>
  <div id=”element3”></div>
</div>
```

CSS:

```
#omotac {
  display: flex;
}

#element1 {
  flex: 1;
  height: 100px;
  background-color: red;
  order: 3; /* Element se prikazuje treci po redu */
}

#element2 {
  flex: 2;
  height: 100px;
  background-color: green;
  order: 1; /* Element se prikazuje prvi po redu */
}
```

```
#element3 {
  flex: 1;
  height: 100px;
  background-color: blue;
  order: 2 /* Element se prikazuje drugi po redu */
}
```

Način **horizontalnog** prostiranja elemenata unutar elementa sa prikazom **flex** zadaje se svojstvom **justify-content**. Moguće vrednosti ovog svojstva su **flex-start** (elementi se prikazuju jedan do drugog počevši od početka **flex**), **flex-end** (elementi se prikazuju jedan do drugog od kraja **flex** omotača), **center** (elementi se prikazuju na sredini **flex** omotača), **space-between** (elementi se prikazuju ravnomerno raspoređeni unutar **flex** omotača pri čemu se prazan prostor između elemenata deli ravnomerno **oko elemenata**) i **space-around** (elementi se prikazuju ravnomerno raspoređeni unutar **flex** omotača ali se prazan prostor deli ravnomerno **između elemenata**). Prikaz rezultata različitih vrednosti svojstva **justify-content** može se videti na *slici 4*.



Slika 4. Prikaz rezultata vrednosti svojstva justify-content ukoliko se elementi unutar flex omotača prikazuju u redu (flex-direction: row)

Vertikalni respored elemenata unutar **flex** omotača zadaje se svojstvom **align-items** čije su moguće vrednosti **flex-start** (elementi se prikazuju od početka **flex** omotača), **flex-end** (elementi se prikazuju od kraja **flex** omotača), **center** (elementi se prikazuju na sredini **flex** omotača), **strech** (elementi se prikazuju tako da im visina bude jednaka visini **flex** omotača).

Napomena, pojmovi horizontalno i vertikalno su relativni u zavisnosti od vrednosti svojstva **flex-direction**. Prema podrazumevanoj vrednosti svojstva **flex-direction** kao **row**, važi navedena podela na horizontalno i vertikalno prostiranje .

Pri fiksnom zadavanju veličine elementa unutar **flex** omotača moguće je da se elementi na ekranima manjih veličina prikazuju jedan preko drugog, trudeći se da zadrže zadatu veličinu. Za rešavanje ovog problema može se iskoristiti svojstvo **flex-wrap** čija se vrednost postavlja na **nowrap**. Na ovaj način, elementi koji ne mogu da zadrže zadato rastojanje od ostalih elemenata i zadate dimenzije, prelaze u novi red. Podrazumevana vrednost ovog svojstva je **no-wrap**.

Svojstva koja utiču na prikaz teksta

Na prikaz teksta može se uticati korišćenjem većeg broja svojstava, neka od tih svojstava su:

font-size – Zadaje se veličina fonta kojim se prikazuje tekst.

Primer:
```
p {
  font-size: 20px; /* Veličina fonta je 20 piksela */
}
```

font-family – Zadaje se familija fontova ili konkretan naziv fonta kojim će se tekst ispisivati. Moguće je navesti listu familija i naziva fontova pri čemu će pregledač iskoristiti prvi font sa zadatim nazivom ili iz zadate familije koji može da prikaže.

Primer:
```
p {
  font-family: “Arial”, sans-serif, serif;
}
```

font-weight – Zadaje se „debljina“ slova, ukoliko font može da prikaže karaktere sa zadatom debljinom. Vrednosti svojstva su **normal**, **bold**, **bolder**, **lighter** ili brojevi od 100 do 900 koji označavaju deblinu.

Primer:
```
p {
  font-weight: 900;
}
```

font-style – Zadaje se stil fonta, moguće vrednosti su **normal**, **italic**, **oblique**.

Primer:
```
p {
  font-style: italic;
}
```

text-decoration – Zadaju se dodatne dekoracije deksta, moduće vrednosti su **underline**, **overeline**, **line-through** i **none**.

Primer:
```
a {
  text-decoration: none;
}
```

text-align – Zadaje se poravnanje teksta, moguće vrednosti su **left**, **center**, **right** i **justify**.

Primer:
```
p {
  text-align: center;
}
```

text-transform – Zadaje se transformacija teksta, moguće vrednosti su **uppercase**, **lowercase**, **capitalize**.

Primer:
```
p {
  text-transform: uppercase;
}
```

text-indent – Zadaje se veličina uvučenog dela pasusa, vrednosti su izražene u nekoj mernoj jedinici.

Primer:
```
p {
  text-indent: 20px;
}
```

line-height – Zadaje se visina linije teksta, moguće vrednosti su fiksna visina izražena u nekoj mernoj jedinici ili procentualno u odnosu na veličinu fonta teksta koji se prikazuje.

Primer:

```
p {
  line-height: 140%;
}
```

word-spacing – Zadaje se razmak između reči u tekstu, vrednosti su izražene u nekoj mernoj jedinici.

Primer:

```
p {
  word-spacing: 30px;
}
```

Učitavanje fonta iz datoteke

Primer:

```
@font-face {
  src: url(fontovi/datoteka.otf);
  font-family: 'moj font';
}
```

Font se može uključiti u veb stranicu iz datoteke korišćenjem pravila **@font-face** jezika css. Pravilo **@font-face** se sastoji od putanje do datoteke (**src**) sa fontom i naziva familije fonta pod kojom će biti dostupan za korišćenje (**font-family)**.

Responsive design

Responsive design ili „prilagodljiv" dizajn omogućava prikaz sadržaja koji stranice na uređajima sa različitim veličinama ekrana, npr. Telefon, tablet ili laptop. Svi uređaji otvaraju istu stranicu, ali se sadržaj stranice prilagođava veličini ekrana. U jeziku CSS moguće je navođenje drugačijih vrednosti svojstava za uređaje sa različitim osobinama korišćenjem pravila **@media**. Moguće je navođenje za koje uređaje važi pravilo, kao i koje osobine treba da ispunjava uređaj. U slučaju prilagodljivog dizajna, osobina uređaja koja se najčešće navodi je širina ekrana, tačnije da svojstvo važi za sve uređaje sa ekranima manjim, odnosno većim, od zadate vrednosti.

Primer:

```
h1 {
  font-size: 50px;
}

@media screen and (max-width: 768px) {
  h1 {
    font-size: 20px;
  }
}

/* Velicina fonta naslova ce na ekranima vecim od 768px sirine biti
50px dok ce na ekranima sa sirinom manjom od 768px velicina fonta
naslova biti 20px*/
```

Mobilni uređaji imaju osobinu da za širinu sadržaja stranice uzimaju najmanju širinu potrebnu za prikazivanje celokupnog sadržaja stranice. Ideja responsive dizajna je da za širinu sadržaja mobilni uređaj podrazumeva stvarnu širinu ekrana a da se dizajn **prilagođava** veličini ekrana. To se ostvaruje navođenjem odgovarajućih meta podataka.

Primer:

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

Postavlja se pitanje u na kojim tačkama je potrebno prelomiti stranicu da se prilagodi manjem ekranu. Odgovor na to pitanje je da prikaz sadržaja treba preurediti na veličini na kojoj loše izgleda. Ne prilagođavati se svakom uređaju pojedinačno već truditi se da stranica izgleda dobro na svim veličinama popravljajući prikaz na veličinama na kojoj se ne prikazuje dobro. Za veličine elemenata dobra je praksa korišćenjem procenata kao merne jedinice i svojstava koja se odnose na maksimalne, odnosno minimalne, veličine.

Primer:

```
p {
  max-width: 1000px;
  margin: 0 auto;
}

img {
  min-width: 50%;
}
```

Senke i zaobljeni uglovi

Izgled senke koja ostaje iza određenog elementa zadaje se CSS svojstvom **box-shadow**. Vrednost svojstva **box-shadow** sadrži **horizontalnu** i **vertikalnu** udaljenost senke od elementa, **prečnik zamućenja** senke, zadat u pikselima (na koliko će se okolnih piksela „razliti" piksel senke), **prostiranje** senke (na koliko će se piksela prostirati senka oko elementa) i **boju** senke. Boja senke se često navodi RGB modelom proširenim sa alfa kanalom (providnošću) – **rgba**.

Primer:

```
img {
  box-shadow: 5px 5px 10px 3px rgba(0, 0, 0, 0.2);
}
```

Za prikaz senke oko teksta koristi se svojstvo **text-shadow**, čija vrednost sadrži **horizontalnu** i **vertikalnu** udaljenost senke od karaktera teksta, **prečnik zamućenja** i **boju**.

Primer:

```
h1 {
 text-shadow: -5px -10px 15px rgba(200, 100, 0, 0.5);
}
```

Zaobljenost uglova elementa navodi se svojstvom **border-radius** u pikselima. Navođenjem samo jedne vrednosti zadaje se zaobljenost svih uglova. Ako su zadate dve vrednosti, prva se odnosi na gornji levi i donji desni ugao, dok se druga odnosi na gornji desni i donji levi. Kod tri zadate vrednosti prva vrednost se odnosi na gornji levi ugao, druga na gornji desni i donji levi i treća na donji desni ugao, dok se kod četiri zadate vrednosti svaka vrednost odnosi na po jedan ugao redom – gornji levi, gornji desni, donji levi, donji desni.

Primer:

```
.div1 {
  width: 200px;
  height: 100px;
  background-color: red;
  border-radius: 15px
}

.div2 {
  width: 200px;
  height: 100px;
  background-color: green;
```

```
  border-radius: 10px 20px 30px 40px;
}
```

Da bi se napravio kružni element potrebno je da su vrednosti visine i širine elementa iste a da je zaobljenje svih uglova 50%.

Primer:

```
.krug {
  width: 100px;
  height: 100px;
  background-color: blue;
  border-radius: 50%;
}
```

Formulari

Fomulari omogućavaju unos korisničkih podataka. Element kojim se predstavlja formular je element sa etiketom **form**. Osnovni atributi formulara su **method** i **action**. Atributom **method** se zadaje kojim metodom će se podaci proslediti na obradu, podržani metodi su **GET** i **POST**. Ukoliko nije navedena vrednost atributa **method**, podrazumevana je vrednost **GET**. Atributom **action** se navodi kojoj se strani prosleđuju podaci za obradu. Ukoliko nije navedena vrednost atributa **method**, podrazumevana vrednost je strana na kojoj se nalazi formular. Detalji vezani za prosleđivanje i obradu podataka biće objašnjeni u poglavlju o PHP-u.

Primer:

```
<form method=”POST” action=”obrada.php”>
</form>
```

Polja za unos podataka predstavljena su elementom sa etiketom **input**, čiji atribut **type** određuje kakav tip unosa se očekuje. Kako bi se vrednosti polja mogle kasnije obraditi, neophodno je navodjenje atributa **name** čija će se vrednost koristiti za referisanje na prosleđene vrednosti forme. Vrednosti **input** polja su vrednosti **value** atributa. Prikaz dodatnog objašnjenja **input** polja, u obliku teksta koji se prikazuje unutar **input** elementa dok vrednost polja nije postavljena, zadaje se atributom **placeholder** čija je vrednost tekst objašnjenja koji će se prikazivati. Dodatno, za označavanje naziva polja za unos koristi se element sa etiketom **label**. Važan atribut elementa **label** je atribut **for** čija je vrednost naziv identifikatora **input** elementa na koji se labela odnosi. Ako je potrebno da je **input** polje odmah u fokusu čim je stranica učitana navodi se atribut **autofocus** (vrednost nije potrebno navoditi, već samo naziv). Atributom **autocomplete** se navodi da li je dozvoljeno predlaganje ranije unetih vrednosti u toku popunjavanja

polja. Za uključivanje ove opcije (koja je često podrazumevano uključena) navodi se naziv atributa dok se za isključivanje vrednost atributa postavlja na **off**. Da bi se naznačilo da je polje neophodno popuniti, navodi se atribut **required**. Ukoliko polje sa atributom **required** nije popunjeno, podaci iz formulara se neće proslediti na obradu. Za onesposobljavanje unosa za neki **input** element koristi se atribut **disabled**.

Grupa polja može se obuhvatiti u jednu celinu smeštanjem unutar elementa sa etiketom **fieldset**. Elementom sa etiketom **legend** unutar elementa **fieldset** zadaje se naziv grupe elemenata.

Tip **input** elementa kojim se omogućava unos proizvoljnog teksta je **text**. Atributima **minlength** i **maxlength** se mogu zadati minimalni i maksimalni dozvoljeni broj karaktera tekstualnog polja. Ako vrednost nije u zadatim granicama na nekom od dozvoljenih koraka, podaci iz formulara neće biti prosleđeni na obradu. Korišćenjem elementa **datalist** navodi se spisak vrednosti se mogu koristiti kao pomoć pri popunjavanju tekstualnog polja, u smislu predlaganja mogućih vrednosti. Element **datalist** sadrži vrednosti u okviru **option** elemenata. Elementu **option** se navodi vrednost na koju se odnosi atributom **value** dok se između **option** etiketa navodi natpis koji će se koristiti kao predložena vrednost prilikom popunjavanja tekstualnog polja. Da bi se **datalist** povezao sa tekstualnim poljem, tekstualnom polju se dodaje atribut **list** čija je vrednost **id** elementa **datalist** sa kojim se vezuje.

Primer:

```
<form method=”POST” action=”obrada.php”>
    <fieldset>
        <legend>Licni podaci</legend>
        <label for=”ime_id”>Ime</label>
        <input type=”text” name=”ime” id=”ime_id” minlength=”5”>
        <br>
        <br>
        <label for=”prezime_id”>Prezime</label>
        <input type=”text” name=”prezime” id=”prezime_id”
                                    maxlength=”10” required>
        <br>
        <br>
        <label for=”grad_id”>Mesto rodjenja</label>
        <input type=”text” name=”grad” id=”grad_id” list=”gradovi”>
        <datalist id=”gradovi”>
            <option value=”BG”>Beograd</option>
            <option value=”NS”>Novi Sad</option>
            <option value=”PO”>Pozarevac</option>
        </datalist>

    </fieldset>
```

```
</form>
```

Element sa etiketom **textarea** je takođe element za unos teksta, ali se od **input** elementa sa tipom **text** razlikuje po tome što je dozvoljen unos teksta u više linija. Neki od atributa ovog elementa su **rows** – broj vidljivih linija bez skrolovanja, **cols** – broj karaktera po liniji**, wrap** – na koji način će se tekst prelamati na kraju linija (moguće vrednosti su **hard** i **soft**) kao i oni atributi koji se odnose na tekstualni **input** element.

Za unos numeričkih vrednosti koristi se tip **number**. Numeričkim poljima se mogu zadati minimalna i maksimalna dozvoljena vrednost, atributima **min** i **max**, a atributom **step** se zadaje korak kojim se dozvoljene vrednosti raspoređene od minimalne do maksimalne.

Primer:
```
<input type=”number” name=”broj” min=”10” max=”15” step=”5”>
```

Polje koje se koristi za unos e-mail adrese ima tim **email**. Polje se izgledom ne razlikuje od tekstualnog polja, već je samo dodata provera ispravnosti unete adrese u pogledu sintaksnih pravila. Ukoliko adresa nije u ispravnom formatu, podaci iz formulara neće biti prosleđeni na obradu.

Primer:
```
<input type=”email” name=”email-adresa” autocomplete>
```

Tip **checkbox** se koristi za **input** polja koja mogu biti uključena ili isključena (čekirana). Vrednost atributa **value** određuje vrednost koja će biti prosleđena, ukoliko je polje uključeno. U suprotno, vrednost neće biti prosleđena. Podrazumevana vrednost **checkbox** elementa je **on**. Ako je potrebno da polje bude podrazumevano uključeno, navodi se atribut **checked**.

Primer:
```
<input type=”checkbox” name=”remember-me” value=”yes” checked>
```

Za obeležavanje samo jedne od ponuđenih vrednosti koristi se tip **radio**. Da bi se omogućilo da se od nekoliko **input** elemenata sa tipom **radio** odabralo samo jedno, potrebno je da svi **radio** elementi imaju istu vrednost atributa **name**. Samo se vrednost odabranog elementa prosleđuje na obradu pod zajedničkim nazivom zadatim atributom **name**.

Primer:
```
<input type=”radio” name=”godina-studija” value=”1” checked>
<input type=”radio” name=”godina-studija” value=”2” checked>
```

```
<input type="radio" name="godina-studija" value="3" checked>
<input type="radio" name="godina-studija" value="4" checked>
```

Drugi način za odabir jednog od ponuđenih elemenata se ostvaruje korišćenjem elementa sa etiketom **select**. Unutar **select** elementa se nalaze dozvoljene ponuđene vrednosti u obliku elemenata **option**. Struktura ovih **option** elemenata je ista kao struktura **option** elemenata u okviru elementa **datalist**. Za grupisanje **option** elemenata u grupe sa zadatim nazivom koristi se element **optgroup** unutar koga se smeštaju option elementi. Naziv grupe se zadaje atributom **label**.

Primer:

```
<select name="smer">
  <optgroup label="Matematika">
    <option value="M">Teorijska matematika</option>
    <option value="L">Profesor matematike i racunarstva</option>
    <option value="V">Verovatnoca i statistika</option>
    <option value="R">Racunarstvo i informatika</option>
    <option value="N">Numericka matematika</option>
  </optgroup>

  <optgroup label="Informatika">
    <option value="I">Informatika</option>
  </optgroup>
</select>
```

Za potvrdu unosa i prosleđivanje vrednosti formulara na dalju obradu koristi se tip **submit**. Ukoliko ovaj element nije prisutan unutar forme, vrednosti se mogu proslediti pritiskom na dugme **enter** unutar nekog od tekstualnih polja. Natpis koji se prikazuje unutar ovog elementa se zadaje atributom **value**.

Primer:

```
<input type="submit" value="Posalji" name="slanje">
```

Za poništavanje svih unetih vrednosti polja i vraćanje na podrazumevane vrednosti koristi se tip **reset**. Natpis koji se prikazuje unutar ovog elementa se zadaje atributom **value**.

```
<input type="reset" value="Ponisti">
```

Još neki od tipova **input** polja koji se mogu koristiti su **file**, **image**, **date**, **time**, **range**, **url** i drugi, detaljan spisak tipova i dozvoljenih atributa **inpu** elemenata može se naći na adresi: https://www.w3schools.com/tags/tag_input.asp

Indeks obrađenih CSS svojstava

display
color
background-color
background-image
background-position
background-attachment
background-repeat
font-size
font-family
font-weight
font-style
text-align
text-decoration
text-indent
text-transform
line-height
word-spacing
position
height
width
top
bottom
left
right
z-index
box-sizing
margin
padding
border
overflow
opacity
overflow-x
overflow-y
float
clear
content
flex
flex-direction
flex-basis
order
justify-content
align-items
flex-wrap
min-width
max-width
min-height
max-height
box-shadow
text-shadow
border-radius

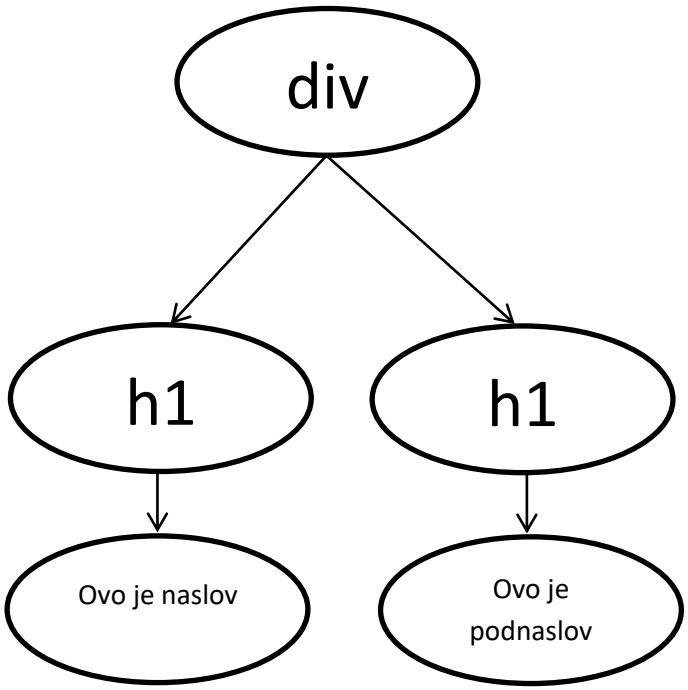
Javascript

Za svaku Veb stranu pregledač formira **DOM** (*Document Object Model*), drvoliku strukturu čiji su čvorovi HTML elementi (i njihovi sadržaji). **DOM** čuva hijerarhiju elemenata pa je onda za element, koji je u HTML jeziku predstavljao roditeljski element u odnosu na neki drugi element, njemu dodeljen čvor koji predstavlja roditelja u odnosu na čvor pridružen drugom elementu. Koren **DOM** stabla je objekat **window**, koji odgovara prozoru pregledača. Elementi koji su napisani jezikom HTML odgovaraju čvorovima podstabla čiji je koren objekat **document**, koji je jedan od sinova čvora **window**.

HTML elementi:

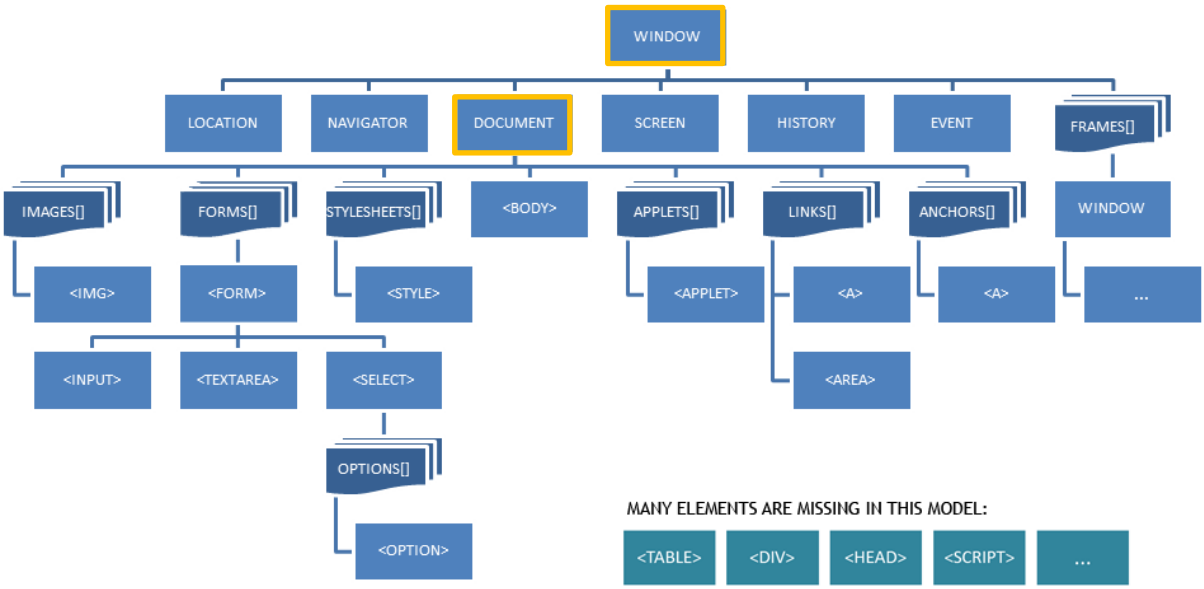
```
<div>
    <h1>Ovo je naslov</h1>
    <h2>Ovo je podnaslov</h2>
</div>
```

Se drvoliko predstavljaju kao na *slici 5*.



Slika 5. Primer DOM stabla

Struktura celokupnog **DOM** stabla predstavljena je primerom na *slici 6.*



Slika 6. Primer DOM stabla počevši od **window** objekta

Javascript je skrip jezik čija je osnovna uloga na Veb-u programiranje korisničkog interfejsa. Korišćenjem jezika **Javascript** moguća je interakcija sa **DOM** stablom, njegovo menjanje i obilaženje. Izmene napravljene nad čvorovima **DOM** stabla biće vidljive na stranici u okviru elemenata čiji su odgovarajući čvorovi **DOM**-a menjani. Programi napisani jezikom **Javascript** izvršavaju se na klijentskoj mašini (iako postoje i upotrebe na serveru).

Kod napisan na jeziku javascript može se naći između etiketa elementa **script** ili u posebnoj datoteci koja se uključuje takođe korišćenjem elementa **script** navođenjem putanje do datoteke u okviru atributa **src**. Opciono je navodjenje atributa **type** sa vrednošću “text/javascript”

Primer:

```
<script type=”text/javascript”>
      a = 5;
</script>

<script src=”datoteka.js”></script>
datoteka.js:

// Ovo je komentar u JS
/* I ovo je komentar u JS */
```

```
b = 3;
```

Dobra je praksa navođenje **script** elemenata nakon svih elemenata unutar elementa **body**, kako bi se svi HTML elementi smestili u DOM pre pozivanja **Javascript** programa.

Osnove jezika Javascript

Javascript je slabo tipiziran jezik, pa se nazivi tipova promenljivih ne navode eksplicitno. Nova promenljiva deklariše se navođenjem njenog imena pre koga se mogu naći ključne reči **var** ili **let**. Tip promenljive određen je tipom vrednosti koju čuva, tako da jedna promenljiva može promeniti svoj tip u toku izvršavanja programa.

Primer:

```
var a = 5; // Promenljiva a je celobrojnog tipa,
a = "abc"; // sada je promenljiva a niska,
a = 'a';   // a sada karakter
```

Niske se mogu navoditi unutar jednostrukih ili dvostrukih navodnika.[1]

Primer:

```
var tekst1 = 'Ovo je niska';
var tekst2 = "I ovo je niska";
```

Ispisivanje vrednosti na izlaz konzole vrši se funkcijama **console.log()**, **console.info()**, **console.warn()** i **console.error()**. Konzola se otvara u pregledaču najčešće pritiskom na dugme **F12**.

Primer:

```
console.log("Ovo je neki ispis");
console.log("Ovo", "je", "ispis", "vise", "argumenata", 1, 2, 3);
console.info("Ovo je obavestenje");
console.warn("Ovo je upozorenje");
console.error("Ovo je greska");
```

Nadovezivanje niski vrši se operatorom **+**.

Primer:

```
var tekst = "Prvi deo teksta" + " " + "Drugi deo teksta";
console.log(tekst);
```

[1] Pri kopiranju primera iz skripte moguće je da navodnici nisu odgovarajući, pa ih po potrebi treba zameniti ručno napisanim

Pri nadovezivanju niski sa numeričkim izrazima numerički izrazi se prevode u nisku, ukoliko je potrebno prvo izračunati vrednost izraza pa tek onda ga ispisati onda je dovoljno izraz staviti unutar zagrada

Primer:
```
// Ispis: Vrednost izraza je 12
console.log(“Vrednost izraza je: ” + 1 + 2);

// Ispis: Vrednost izraza je 3
console.log(“Vrednost izraza je: ” + (1 + 2));
```

Za prikaz iskačućeg prozora sa obaveštenjem koristi se funkcija **alert()** čiji je argument niska koja se prikazuje pri pojavi prozora. Iskačući prozor koji od korisnika zahteva da na neko pitanje odgovori pozitivno ili negativno koristi se funkcija **confirm()**, čiji je argument ponovo niska koja se prikazuje. Rezultat korisnikovog odgovora vraća se kao povratna vrednost funkcije. Ukoliko se od korisnika očekuje unos nekog tekstualnog podatka unutar iskačućeg prozora, koristi se funkcija **prompt()**. Prvi argument funkcije je niska koji se ispisuje u prozoru a drugi podrazumevana vrednost polja za unos teksta. Uneta vrednost se vraća kao povratna vrednost funkcije

Primer:
```
alert(“Ovo je obavestenje!”);

if(confirm(“Da li ste sigurni?”))
    console.log(“Pozitivan odgovor”);
else
    console.log(“Negativan odgovor”);

console.log(“Vase ime je: “, prompt(“Unesite vase ime”));
```

Funkcije se u jezikom **Javascript** navode korišćenjem ključne reči **function** nakon koje sledi naziv funkcije i spisak argumenata.

Primer:
```
function nadovezivanje(niska1, niska2)
{
  return niska1 + “ “ + niska2;
}
```

Za eksplicitno kastovanje niske u broj koriste se funkcije **parseInt** i **parseFloat** koji nisku prosleđenu kao argument pokušavaju da prevetu u numeričku (redom celobrojnu ili razlomljenu) vrednost. Funkcije uzimaju karakter sa početka niske koji mogu biti deo broj sve do karaktera koji ne može biti deo broja. Ukoliko niska ne sadrži cifre na početku, funkcije vraćaju vrednost **NaN** (*Not a Number*).

Primer:
```
// Vrednost promenljive broj1 je 123
```

```
var broj1 = parseInt('123');

// Vrednost promenljive broj2 je 1
var broj2 = parseFloat('12.3abc');

// Vrednost promenljive broj3 je NaN
Var broj3 = parseInt('abc');

// Ispis je NaN jer se sabiraju dve numericke vrednosti I jedna
// nenumericka, pa je rezultat nenumericka vrednost
console.log(broj1 + broj2 + broj3);
```

Nizovi u jeziku Javascript

Nizovi u jeziku **Javascript** su asocijativni nizovi, moguće je indeksiranje elemenata korišćenjem vrednosti različitih tipova kao i čuvanje promenljivih različitih tipova. Deklaracija niza može se vršiti inicijalizacijom promenljive na novi objekat tipa **Array** ili navođenjem inicijalne vrednosti. Ukoliko indeksi elemenata nisu eksplicitno navedeni, niz je podrazumevano indeksiran celim brojevima počevši od indeksa 0.

Primer:

```
var niz1 = new Array();
var niz2 = [];
var niz3 = [23, 42, 15];
var niz4 = [1, "tekst", '?'];
niz4['kljuc'] = 5;

// Prazan niz.
console.log(niz1);

// Takodje prazan niz.
console.log(niz2);

// Niz [23, 42, 15] indeksiran sa 0, 1, 2 (niz[0] == 23).
console.log(niz3);

// Elementi 1, "tekst" i '?' su indeksirani sa 0, 1, 2 dok je
// vrednos 5 indeksirana niskom "kljuc".
console.log(niz4);
```

Element niza može takođe biti niz, za koji važe navedena pravila indeksiranja.

Primer:

```
var matrica = [];
matrica[0] = [1, 2];
matrica[1] = [3, 4];
```

```
/* Matrica: 1 2
            3 4
*/

// Ispis elementa u nultom redu i prvoj koloni:
console.log(matrica[0][1]);

var niz = []
niz['x'] = [1, 2, 3, [4, 5, 6]];




/* Niz ima jedan element sa indeksom 'x', njegova vrednost je niz
sa elementima 1, 2, 3 koji su indeksirani sa 0, 1, 2 i podniz sa
elementima 4, 5, 6 koji se nalazi na indeksu 3. Elementi 4, 5, 6 su
indeksirani sa 0, 1, 2 u svoj nizu.


'x':[
     0: 1
     1: 2
     2: 3
     3: [
        0: 4
        1: 5
        2: 6
        ]
     ]
*/

// Ispis vrednosti 6
console.log(niz['x'][3][2]);
```

Petlje u jeziku Javascript

Petlje koje se mogu koristiti u jeziku **Javascript** su **for** i **while**, kao i njihovi derivati **for/in** i **do/while**. **For/in** petlja iterira kroz kolekciju objekata (niz) i koristi se kada nije poznato na koji način su elementi indeksirani.

Primer:

```
// Ispis brojeva od 0 do 4
for(i = 0; i < 5; i++)
{
   console.log(i);
}

var j = 5;

// Ispis brojeva od 5 do 1
while(j > 0)
{
   console.log(j);
   i--;
}

// Ispis brojeva od 5 do 0
do
{
  console.log(j);
  j--;
} while(j > 0)

var niz = [1, 'abc'];
niz['x'] = 1.5;
// Promenljiva i dobija vrednost indeksa svakog elementa u nizu,
// petlja ispisuje vrednosti svakog indeksa i kao i elemente na
// pozicijama indeksa i.

for(i in niz)
{
    console.log(i + ": ", niz[i]);
}

// Ispis:
//
// 0: 1
// 1: abc
// x: 1.5
```

Pristupanje pojedinačnim karakterima niske vrši se funkcijom **charAt()**, čiji je argument indeks karaktera u niski. Uklanjanje belina sa početka i kraja niske vrši se funkcijom **trim()** a dužina niske je vrednost **svojstva length**. ASCII kod karaktera u niski se može dobiti funkcijom **charCodeAt()** čiji je argument indeks karaktera u niski, dok se od celobrojne vrednosti koja odgovara ASCII kodu karaktera može napraviti karakter korišćenjem funkcije **fromCharCode()**, čiji je argument celobrojna vrednost.

Primer:

```
var niska = “   abc   “;

// Duzina niske je 9
console.log(niska.length);

niska = niska.trim();

// Duzina niske bez belina na pocetku i kraju je 3 (“abc”)
console.log(niska.length);

// Ispisivanje pojedinacnih karaktera niske i njihovih ASCII kodova
for(i = 0; i < niska.length; i++)
{
   console.log(“Karakter: “, niska.charAt(i), “ kod:
“,niska.charCodeAt(i));
}

// ASCII kod velikog slova A je 65
var slovo_a = 65;

// Ispis: A
console.log(fromCharCode(slovo_a));
```

Interakcija sa DOM-om iz jezika Javascript

Dohvatanje (selektovanje) **DOM** elemenata vrši se funkcijama definisanim u interfejsu **document**. Neke od tih funkcija su:

document.getElementsByTagName() – Dohvatanje elementa čiji je naziv etikete prosleđen kao argument funkcije. Funkcija vraća niz objekata (čvorova stabla) koji odgovaraju pronađenim elementima. Niz je indeksiran celim brojevima počevši od 0.

document.getElementById() – Dohvatanje elementa čiji je ID prosleđen kao argument funkcije.

document.getElementsByClassName() – Dohvatanje svih elementa sa klasom čiji je naziv prosleđen kao argument funkcije. Funkcija vraća niz objekata (čvorova stabla) koji odgovaraju pronađenim elementima. Niz je indeksiran celim brojevima počevši od 0.

document.getElemensByName() – Dohvatanje elementa čijia je vrednost atributa name prosleđena kao argument funkcije. Funkcija vraća niz objekata (čvorova stabla) koji odgovaraju pronađenim elementima. Niz je indeksiran celim brojevima počevši od 0.

document.getElemensByName() – Dohvatanje elementa čijia je vrednost atributa **name** prosleđena kao argument funkcije. Funkcija vraća niz objekata (čvorova stabla) koji odgovaraju pronađenim elementima. Niz je indeksiran celim brojevima počevši od 0.

document.querySelector() – Dohvatanje elemenata CSS selektorom navedenim kao argument funkcije. Ukoliko postoji vise elemenata koji odgovaraju zadatom CSS selektoru, funkcija vraća **samo prvi** takav pronađeni element.

document.querySelectorAll() – Dohvatanje elemenata CSS selektorom navedenim kao argument funkcije. Funkcija vraća niz **svih** objekata (čvorova stabla) koji odgovaraju pronađenim elementima. Niz je indeksiran celim brojevima počevši od 0.

Primer:

```
HTML:

<p class=”pasus”>Neki pasus</p>
<div id=”omotac”>
    <p class=”pasus” id=”pasus1”>Prvi pasus</p>
    <p class=”pasus” id=”pasus2”>Drugi pasus</p>
    <p class=”pasus” id=”pasus3”>Treci pasus</p>
</div>

JS:

var omotac_element = document.getElementById(‘omotac’);
var svi_pasusi = document.getElementsByTagName(‘p’);
var pasusi_u_omotacu = document.querySelectorAll(‘#omotac .pasus’);

var prvi_pasus = document.querySelector(“#omotac .pasus”);
var drugi_pasus = document.querySelector(‘p:nth-child(2)’);
var treci_pasus = pasusi_u_omotacu[2];
```

Svojstvima dohvaćenog objekta pristupa se korišćenjem “tačka notacije” (*dot notation*). Nakon naziva promenljive navodi se tačka i ime svojstva. Sadržaj unutar etiketa HTML elementa nalazi se u svojstvu **innerHTML** objekta koji odgovara dohvaćenom element. Sadržaj ima tip niske.

Primer:

```
HTML:

<p id=”pasus”>Neki pasus</p>

JS:

var sadrzaj_pasusa = document.getElementById(‘pasus’).innerHTML;

// Ispis: Neki pasus
```

```
console.log(sadrzaj_pasusa);
```

Za pristupanje vrednostima **input** elemenata koristi se svojstvo **value**.

Primer:

HTML:

```
<input type=”text” id=”ime”>
```

JS:

```
var input_polje = document.getElementById(‘id’);
var vrednost_polja = input_polje.value;
```

CSS svojstvima elementa pristupa se pomoću svojstva **style**. Objekat koji odgovara svojstvu **style** sadrži **samo eksplicitno zadate CSS vrednosti**, ne i one koje nisu navedene CSS-om a koje je pregledač samostalno izračunao. Vrednosti CSS svojstva se pristupa preko njenog naziva, ukoliko naziv sadrži crticu za pristup svojstvu se koristi naziv u kome crtica ne postoji ali je prvo slovo posle crtice u originalnom zapisu veliko (npr. background-color → backgroundColor). Izračunate CSS vrednosti dobijaju se funkcijom **getComputedStyle** iz interfejsa **windows**, element čija se izračunata svojstva traže se prosleđuje kao argument funkcije.

Primer:

HTML:

```
<p>Pasus<p>
```

CSS:

```
p {
    background-color: red;
}
```

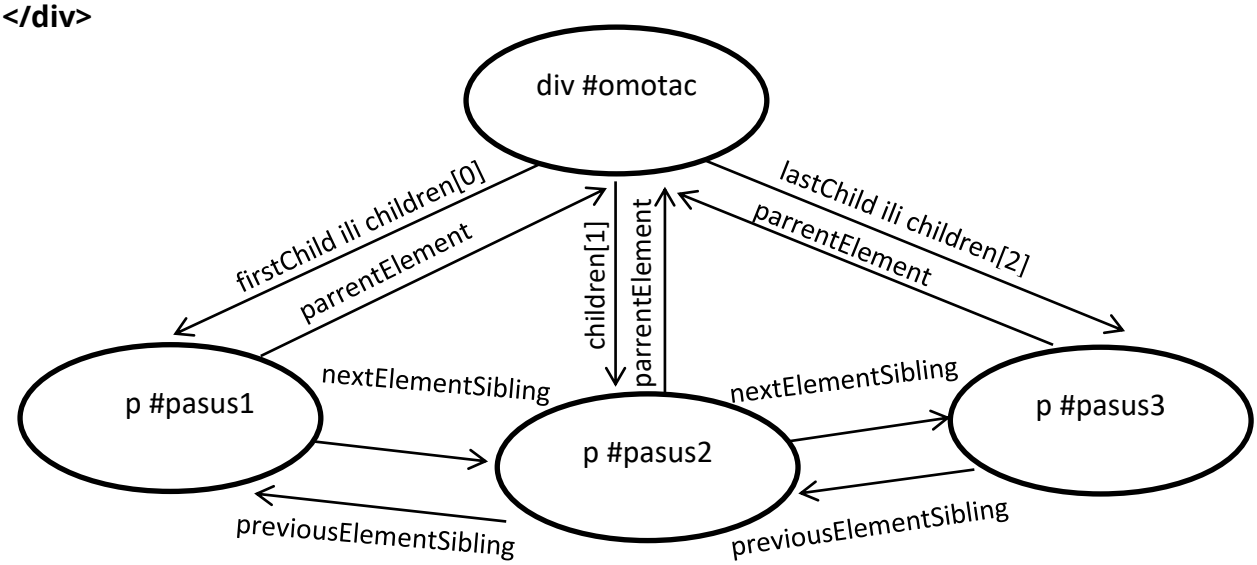
JS:

```
var pasus = document.getElementsByTagName(‘p’)[0];
var stil_pasusa = pasus.style;

// Boja teksta pasusa nije eksplicitno zadata CSS-om pa je njena
// vrednost u okviru style nije postavljena.

console.log(stil_pasusa.color);

// Boja pozadine pasusa jeste eksplicitno zadata CSS-om pa ce se
```

```
// njena vrednost ispisati.

console.log(stil_pasusa.backgroundColor);

var izracunati_stil = window.getComputedStyle(prvi_pasus);

// Pregledac je najverovatnije kao podrazumevanu boju pasusa
// postavio na crnu, rgb(0,0,0).

console.log(izracunati_stil.color);
```

Kada je dohvaćen HTML element, moguće je kretanjem kroz **DOM** stablo doći do ostalih elemenata prateći njegove „rodbinske veze“ sa ostalim elementima. Roditeljskom elementu pristupa preko svojstva **parrentElement** koje sadrži pokazivač na roditeljski element. Elementu na istom nivou **DOM** stabla („bratu/sestri“) koji u HTML strukturi sledi nakon dohvaćenog elementa pristupa se svojstvom **nextElementSibling**, dok se prethodnom elementu sa istog nivoa pristupa svojsvtom **previousElementSibling**. Prvom direktnom potomku elementa („detetu“) pristupa se svojstvom **firstElementChild**, dok se poslednjem direktnom potomku pristupa svojstvom **lastElementChild**. Svim direktnim potomcima elementa pristupa se svojstvom **children**, koje sadrži niz objekata (indeksiran od 0) koji odgovaraju direktnim potomcima elementa. Veze između elemenata prikazane su primerom na *slici 7*. Spisak svih svojstava i funkcija nad **DOM** elementima može se naći na adresi: https://www.w3schools.com/jsref/dom_obj_all.asp

```
<div id="omotac">
   <p id="pasus1">Prvi pasus</p>
   <p id="pasus2">Drugi pasus</p>
   <p id="pasus3">Treci pasus</p>
</div>
```



Slika7. Primer odnosa između **DOM** stabla

Primer:

HTML:

```
<h1>Naslov</h1>
<div id=”omotac”>
    <p>Pasus1</p>
    <p>Pasus2</p>
    <p>Pasus3</p>
</div>
```

JS:

```
var omotac = document.getElementById(‘omotac’);

// Prvi pasus je prvo dete omotaca
var prvi_pasus = omotac.firstElementChild;

// Drugi pasus je sledeci brat prvog pasusa
var drugi_pasus = prvi_pasus.nextElementSibling;

// Treci pasus je poslednje dete roditelja drugog pasusa
var treci_pasus = drugi_pasus.parentElement.lastElementChild;

// Naslov je prethodni brat omotaca
var naslov = omotac.previousElementSibling;
```

Atributima dohvaćenih elemenata pristupa se funkcijom **getAttribute()** kojoj se kao argument prosleđuje naziv atributa čija se vrednost traži. Postavljanje atributa elementima vrši se funkcijom **setAttribute()** čiji je prvi argument naziv novog atributa a drugi argument željena vrednost novog atributa.

Primer:

HTML:

```
<p id=”pasus” title=”neki pasus”>Neki pasus</p>
```

JS:

```
var pasus = document.getElementById(‘pasus’);

// Dohvatanje vrednosti atributa title
var title = pasus.getAttribute(‘title’);
console.log(title); // ispis je: neki pasus
```

```
// Pasusu se postavlja vrednost atributa class na klasa1
pasus.setAttribute('class', 'klasa1');
```

Pravljenje novih elemenata vrši se funkcijom **createElement** iz interfejsa **document** (**document.createElement()**) kojoj se kao argument prosleđuje naziv etikete novog elementa. Element se vezuje u **DOM** stablo kao poslednji sin nekog elementa funkcijom **appendChild** čiji je argument element koji se dodaje u stablo.

Primer:

HTML:

```
<div id="omotac">
    <p id="pasus1">Prvi pasus</p>
    <p id="pasus2">Drugi pasus</p>
</div>
```

JS:

```
var omotac = document.getElementById('omotac');

// Pravljenje novog pasusa
var novi_pasus = document.createElement('p');

// Postavljanje teksta unutar novog pasusa
novi_pasus.innerHTML = "Treci pasus";

// Postavljanje id atributa novog pasusa
novi_passus.setAttribute('id', "pasus3");

// Vezivanje novog pasusa kao poslednje dete omotaca, do ovog
// trenutka novi pasus nije vezan za stablo postojece HTML
// strukture

omotac.appendChild(novi_pasus);

/* Novodobijena HTML struktura je:
<div id="omotac">
    <p id="pasus1">Prvi pasus</p>
    <p id="pasus2">Drugi pasus</p>
    <p id="pasus3">Treci pasus</p>
</div>
*/
```

Reagovanje na događaje

Svaka interakcija sa elementima DOM-a predstavlja događaj. Prilikom pojave događaja moguće je reagovati i izvršiti određene naredbe. Da bi se pojava nekog događaja vezalo sa akcijom koju je potrebno preduzeti potrebno je prijaviti se, „osluškivati“ pojavu događaja. Prvi način za to je navođenje odgovarajućeg atributa elementa na kome se događaj očekuje. Takvi atributi imaju prefix **on** nakon koga se nastavlja naziv događaja. Na primer, ako se naziv događaja **click**, atribut će imati naziv **onclick**. Vrednost atributa je naziv funkcije koju je potrebno izvršiti ukoliko se događaj desio. Element na kome se događaj dogodio može se dohvatiti korišćenjem ključne reči **this**.

Primer:

HTML:

```
<button onclick=”prikaziObavestenje()”>Prikazi</button>
```

JS:

```
function prikaziObavestenje()
{
    alert(“Obavestenje je prikazano klikom na dugme
           sa natpisom: “ + this.innerHTML);
}
```

Drugi način osluškivanja događaja je dodavanje „osluškivača“ (*listener*) u okviru **Javascript** koda na element na kome se događaj očekuje, korišćenjem funkcije **addEventListener()**. Prvi argument funkcije **addEventListener** je naziv događaja čija se pojava osluškuje dok je drugi argument naziv funkcije koja će se pozvati kada se događaj dogodi. Moguće je kao drugi argument navesti i **anonimnu** funkciju koja će biti definisana odmah na mestu argumenta

Primer:

HTML:

```
<button id=”dugme1”>Prikazi 1</button>
<button id=”dugme2”>Prikazi 2</button>
```

JS:

```
var dugme1 = document.getElementById(‘dugme1’);
var dugme2 = document.getElementById(‘dugme2’);

// Pozivanje imenovane funkcije pri pojavi dogadjaja
dugme1.addEventListener(‘click’, prikaziObavestenje);
```

```
function prikaziObavestenje()
{
    alert(“Obavestenje je prikazano klikom na dugme
           sa natpisom: “ + this.innerHTML);
}

// Pozivanje anonimne funkcije pri pojavi dogadjaja
dugme1.addEventListener(‘click’, function(){
    alert(“Obavestenje prikazano iz anonimne funkcije
           klikom na dugme sa natpisom: ” + this.innerHTML);
});
```

Neki od događaja na koje se može reagovati su:

click – klik na element

dblclick – dupli klik na element

focus – dovođenje elementa u fokus (klikom, dugmetom **tab**, programski...)

blur – promena fokusa sa fokusiranog elementa na drugi element

blur – promena fokusa sa fokusiranog elementa na drugi element

mouseenter – ulaz kursora u okvir elementa

mouseover – pomeranje kursora preko elementa

mouseleave – izlaz kursora iz okvira elementa
onload – pri završenom učitavanju elementa (iscrtavanju i smeštanju u DOM)

keypress – Pritisnuto je dugme na tastaturi dok je element u fokusu, objekat prosleđen kao argument funkciji za obradu događaja sadrži informacije o tome koje dugme je pritisnuto

Primer:

```
tekstualno_polje.addEventListener(‘keypress’, function(dogadjaj){
    alert(“Pritisnuto je dugme sa kodom: “ + (dogadjaj.which ||
dogadjaj.keycode));
});
// U zavisnosti od pregledaca, kodni broj karaktera se dobija
// koriscenjem funkcija which ili keycode
```

input – pri korisničkom unosu podatka u input polje

submit – slanje podataka iz formulara. Ukoliko je povratna vrednost funkcije koja se izvršava pri pojavi ovog događaja podaci **neće biti poslati**. Koristi se pri validaciji formulara.

Primer:

HTML:

```
<form onsubmit=”return validacija()”>
…
</form>
```

JS:

```
function validacija()
{
 …
    If(formaValidna)
        return true;
    else
        return false;
}
```

change – promena vrednosti **input** polja, odabrane vrednosti **select** elementa ili elementa **textarea**. Indeksa odabrane opcije **select** elementa je vrednost svojstva **selectedIndex**.

Primer:

HTML:

```
<select id=”selekcija”>
    <option value=”vrednost1”>Vrednost 1</option>
    <option value=”vrednost2”>Vrednost 2</option>
</select>
```

JS:

```
var selekcija = document.getElementById('selekcija');

    selekcija.addEventListener('change', function(){
      slert("Odabrana je vrednost sa indeksom: ",
                  this.selectedIndex, " i vrednost je: ",
                                              this.value);
    });
```

reset – prilikom poništavanja unetih polja

select – pri selekciji teksta u tekstualnim poljima, **ne mešati sa change**.

Spisak svih događaja na koje je moguće reagovati može se naći na adresi: https://www.w3schools.com/jsref/dom_obj_event.asp

Periodično izvršavanje funkcija

U nekim situacijama, potrebno je izvršavati određeni kod periodično nakon određenog vremenskog intervala. Za te potrebe koriste se funkcije **setTimeout()** i **setInterval()**.

Funkcija **setTimeout()** izvršava zadate naredbe nakon vremenskog intervala zadatog u milisekundama. Prvi argument ove funkcije je funkcija koja se izvršava nakon broja milisekundi koji se zadaje kao drugi argument funkcije. Funkcija koja se prosleđuje kao argument funkcije **setTimeout()** može biti i anonimna funkcija (funkcija definisana odmah unutar argumenta.

Primer:

```
function objava()
{
    console.log(“Isteklo vreme!”);
}

// 1000ms = 1s
setTimeout(objava, 1000);

// 3000ms = 3s
setTimout(function(){
    console.log(“Pozdrav iz anonimne funkcije!”);
}, 3000);
```

Funkcija **setTimeout()** se može iskoristiti i za periodično izvršavanje funkcija tako što će kao poslednja naredba funkcije navesti poziv funkcije **setTimeout()** čiji je prvi argument upravo funkcija iz koje je pozvan **setTimeout()**.

Primer:

```
// Funkcija objava() ce u konzoli ispisati poruku “Pozdrav!” a
// zatim pomocu funkcije setTimeout ponovo pozvati sebe i ponovo
// ispisati poruku nakon 1000ms i tako u krug.

objava(); // Poziv funkcije

function objava()
{
    console.log(“Pozdrav!”);
    setTimeout(objava, 1000);
}
```

Povratna vrednost funkcije **setTimeout()** je identifikator tajmera koji je pokrenut, za prekidanje tajmera koristi se funkcija **clearTimeout()**.

Primer:

```
var tajmer = setTimeout(function(){
    console.log(“Ovaj tekst se nece ispisati!”);
}, 2000);

// clearTimeout ce prekinuti tajmer koji odbrojava do ispisa poruke
// pre nego sto odbroji do 2s.
clearTimeout(tajmer);
```

Funkcija **setInterval()** vrši periodično izvršavanje koda. Redosled i ti argumenata ove funkcije je isti kao i kod funkcije **setTimeout()**.

Primer:

```
setInterval(function(){
    console.log(“Ova poruka ce se ispisivati svake sekunde.”);
}, 1000);
```

Kao što je slučaj i kod funkcije **setTimeout()**, povratna vrednost funkcije **setInterval()** je identifikator pokrenutog tajmera. Za prekid tajmera ove funkcije koristi se funkcija **clearInterval()**.

Primer:

```
var tajmer = setInterval(function(){
    console.log(“Ovaj tekst ce se ispisati samo jednom”);
}, 1000);

// Nakon 1.5s ce se pozvati prekidanje setInterval funkcije
setTimeout(function(){
    clearInterval(tajmer);
}, 1500);
```

Objekat Math

U okviru objekta **Math** definisane su matematičke funkcije i konstante. Pregled celokupnog objekta **Math** nalazi se na adresi: https://www.w3schools.com/js/js_math.asp . Neke od tih funkcija i konstanti su:

Math.random() – Generisanje pseudoslučajnih brojeva u intervalu [0,1]

Math.round(x) – Zaokruživanje broja navedenog kao argument na najbliži ceo broj

Math.sqrt(x) – Računanje korena broja navedenog kao argument

Math.pow(x, y) – Računanje *y*-tog stepena broja *x*.

Math.min(x, y, z, …) – Funkcija vraća minimum brojeva navedenih kao argumenti funkcije, funkcija ima promenljiv broj argumenata.

Math.max() – Funkcija vraća maksimum brojeva navedenih kao argumenti funkcije, funkcija ima promenljiv broj argumenata.

Math.PI – Konstanta π.

PHP

Jezik **PHP** (*PHP: Hypertext Preprocessor*) je serverski skrip jezik čija je uloga dinamičko pravljenje Veb strana. Kod se izvršava na serveru (interpretira, ne kompajlira), u okviru **PHP** modula aplikacije Veb servera. **PHP** kod se piše u datotekama sa ekstenzijom **.php** između oznaka **<?php** i **?>**. Na klijentskoj strani **PHP** kod **nije vidljiv** već je vidljiv samo rezultat njegovog izvršavanja, tačnije, dobijeni ispis.

Ispis iz **PHP-a** vrši se naredbama **echo** ili **print**, dok se za rekurzivno ispisivanje nizova i objekata koristi funkcija **print_r**. Naredba **echo** ispisuje vrednosti koje su navedene odmah nakon ključne reči **echo**. **Svaka PHP naredba se završava karakterom ;**

Primer:
```
<?php
    echo “Ovo je ispis iz PHP-a!”;
?>
```

Za nadovezivanje niski se koristi operator **.**

Primer:
```
<?php
    echo “Prvi deo niske ”.”Drugi deo niske”;
?>
```

Jezik **PHP** je slabo tipiziran jezik i nije potrebno eksplicitno navođenje tipova, već se tipovi promenljivih određuju u toku izvršavanja programa. Promenljive se imenuju dodavanjem znaka **$** ispred naziva.[2]

Primer:
```
<?php
    $a = 5;
```

[2] Pri kopiranju primera iz skripte moguće je da navodnici nisu odgovarajući, pa ih po potrebi treba zameniti ručno napisanim

```
    $b = “Niska”;
    $promenljiva3 = 1.23;
?>
```

Za razliku od jezika **Javascript**, u jeziku **PHP** postoji mala razlika izmedju jednostrukih i dvostrukih navodnika prilikom definisanja niski. U slučaju niske koja je obuhvaćena dvostrukim navodnicima vrši se izračunavanje vrednosti promenljivih koje su se našle unutar niske i pri formiranju niske koriste njihove vrednosti dok to nije slučaj kod jednostrukih navodnika.

Primer:

```
<?php
    $a = 5;

    echo “Vrednost promenljive a je $a”;
    // Ispis: Vrednost promenljive a je 5


    echo ‘Vrednost promenljive a je $a’;
    // Ispis: Vrednost promenljive a je $a
?>
```

Nizovi u jeziku PHP

Nizovi u jeziku **PHP** su asocijativni nizovi, moguće je čuvanje elemenata različitih tipova u istom nizu. Definisanje nizova vrši se pomoću konstruktura **array()** ili eksplicitnim navodjenjem vrednosti niza. Ukoliko indeksi nisu eksplicitno navedeni, podrazumevano je numeričko indeksiranje od 0.

Primer:

```
<?php
    $niz1 = array(1, “a”, 1.23);
    $niz2 = [1, “a”, 1.23];
?>
```

Za eksplicitno navodjenje indeks elemenata niza koristi se operator **=>**, u obliku **indeks => vrednost**.

Primer:

```
<?php
    $niz = array(0 => 1, 1 => “a”, 2 => 1.23);
?>
```

Moguće je i pravljenje višedimenzionalnih nizova.

Primer:

```
<?php
  // Matrica 3x3:

 $matrica1 = array([1, 2, 3], [4, 5, 6], [7, 8, 9]);

 $matrica2 = array(0 => [1, 2, 3], 1 => [4, 5, 6], 2 => [7, 8, 9]);

 $matrica3 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
?>
```

Dimenzija niza dobija se kao povratna vrednost funkcije **count()**;

Primer:

```
<?php
    $niz = [0, 1, 2, 3, 4];

    echo count($niz);
    // Ispis: 5
?>
```

Niske u jeziku **PHP** predstavljaju nizove karaktera pa je moguće pristupanje pojedinačnim karakterima niske pomoću indeksa.

Primer:

```
<?php
    $niska = “Pozdrav!”;

    echo $niska[0].”, “.$niska[1].”, “.$niska[2] .”, “.$niska[2];
    // Ispis: P,o,z,z
?>
```

Petlje u jeziku PHP

Jezik **PHP** obuhvata tri vrste petlji, **for**, **while** (i **do/while**) i **foreach**. Petlja **foreach** je slična petlji **for/in** u jeziku **javascript** i koristi se za prolazak kroz nizove kod kojih nije poznat način indeksiranja.

Primer:

```
<?php
    for($i = 0; $i < 5; $i++)
        echo $i.” “;
    // Ispis: 0 1 2 3 4
```

```
    $i = 5;
    while($i > 0)
    {
        echo $i.” “;
        $i--;
    }
    // Ispis: 5 4 3 2 1


    $niz = array(1, 2, 3, 4, 5);
    $n = count($niz);

    for($i = 0; $i < $n; $i++)
        echo $i;
    // Ispis: 01234

    $niz2 = array(“a” = > 1, “b” => 2, “c” => 3);

    foreach($niz2 as $kljuc => $vrednost)
    {
        echo $kljuc.”: “.$vrednost.”<br>“;
    }
    // Ispis: a: 1
    //        b: 2
    //        c: 3
?>
```

PHP kod se u okviru **.php** stranica može prepletati sa **HTML** kodom, što omogućava dinamičko pravljenje i postavljenje elemenata na stranici u zavisnosti od podataka i **PHP** naredbi. Stranica koju klijent dobije od servera pri zahtevu za neku **PHP** stranu je tekstualni fajl u kome su vidljivi samo ispisi iz **PHP**-a.

Primer:

Index.php:

```
…
<div class=”pasusi”>

<?php echo “<p id=”pasus1”>Pasus 1</p> ?>

<?php

// Sav HTML kod posle { a pre } je obuhvacen petljom i predstavlja
// ispis iz PHP-a

for($i = 2; $i < 5; $i++) {?>
    <p id=”pasus<?php echo $i; ?>>Pasus <?php echo $i; ?></p>
<?php } ?>

</div>
```

```
…

Rezultat koji klijent vidi kao izvorni kod dobijene strane:
…
<div class=”pasusi”>
    <p id=”pasus1”>Pasus 1</p>
    <p id=”pasus2”>Pasus 2</p>
    <p id=”pasus3”>Pasus 3</p>
    <p id=”pasus4”>Pasus 4</p>
</div>
…
```

Funkcije za rad sa nizovima i niskama

Pregled nekih funkcija:

strlen(s) – Funkcija vraća broj karaktera u niski koja je navedena kao argument

substr(s, pocetak, duzina) – Funkcija vraća podnisku niske navedene kao prvi argument, počevši od pozicije navedene kao drugi argument, čiji je broj karaktera jednak vrednosti koja je navedena kao treći argument. Ako dužina nije navedena, podrazumeva se broj karaktera od navedenog početka do kraja niske.

strrev(s) – Funkcija vraća nisku dobijenu okretanjem niske koja je navedena kao argument funkcije.

strpos(s, t, pocetak) – Funkcija pronalazi pojavljivanje niske *t* u niski *s* i vraća indeks početka prvog pojavljivanja. Opciono je navođenje trećeg argumenta kojim se navodi od kog indeksa treba početi pretragu. Funkcija vraća **FALSE** ako niska nije pronađena.

trim(s) – Funkcija vraća nisku koja se dobija uklanjanjem belina sa početka i kraja niske navedene kao argument funkcije.

in_array(element, niz) – Funkcija vraća **TRUE** ili **FALSE** i zavisnosti od toga da li se element naveden kao prvi argument funkcije nalazi u nizu navedenom kao drugi argument funkcije.

array_push(niz, el1, el2, ...) – Funkcija dodaje elemente navedene počevši od drugog argumenta na kraj niza koji je naveden kao prvi argument funkcije. Funkcija ima promenljiv broj argumenata.

array_pop(niz) – Funkcija izbacuje element sa kraja niza i kao povratnu vrednost vraća izbačeni element.

isset(p1, p2, ...) – Funkcija vraća **TRUE** ili **FALSE** u zavisnosti od toga da li je dodeljena vrednost promenljivama koje su navedene kao argumenti funkcije. Funkcija ima promenljiv broj argumenata.

die(poruka) – Funkcija prekida dalje izvršavanje **PHP** naredbi i ispisuje se poruka navedena kao argument funkcije.

Spisak svih PHP funkcija po grupama može se naći na strani:
http://php.net/manual/en/extensions.alphabetical.php

Primer:

```
<?php
    $niska = “Pozdrav!”;
    echo strlen($niska);
    // Ispis: 8

    $podniska = substr($niska, 2, 5);
    echo $podniska;
    // Ispis: zdrav

    echo strrev($niska);
    // Ispis: !vardzoP

    echo strpos(“rav”, “Zdravo”);
    // Ispis: 2

    echo strpos(“ba”, “baba”, 1);
    // Ispis: 2

    $sa_belinama = “   abc   “;
    echo strlen($sa_belinama);
    // Ispis: 9

    $bez_belina = trim($sa_belinama);
    echo strlen($bez_belina);
    // Ispis: 3


    $niz = [1, 2, 3];
    if(in_array(4, $niz))
        echo “Pronadjen.”;
    else
        echo “Nije promadjen.”
```

```
    // Ispis: Nije pronadjen

    array_push($niz, 4, 5);
    // Niz: [1, 2, 3, 4, 5]

    echo array_pop($niz);
    // Niz: [1, 2, 3, 4]
    // Ispis: 5

    if(isset($niz['x']))
        echo "Postavljena vrednost";
    else
        echo "Nije postavljena vrednost";
    // Ispis: Nije posatavljena vrednost

    die('Prekid programa');

    echo "Ovaj tekst se nece ispisati";

?>
```

Obrada vrednosti formulara

Za podsećanje, formulari imaju dva bitna atributa, **action** i **method**. Atributom **action** se navodi putanja se nalazi program ili stranica koji će obrađivati prosleđene podatke formulara. Atributom **method** navodi se na koji način će podaci biti prosleđeni. HTML formulari podržavaju dve vrste metoda, **GET** i **POST**. Metodom **GET** podaci se prosleđuju kroz deo putanje (URL ili URI) u delu *query string* koji se nalazi na kraju putanje počevši od karaktera **?**. Svi prosleđeni podaci se u okviru *query string*-a nalaze u obliku **naziv=vrednost** eventualno razdvojeni karakterom **&**.

Primer:

```
http://localhost/obrada.php?podatak1=vrednost1&podatak2=vrednost2
```

Podaci prosleđeni **GET** metodom na **PHP** stranu dohvataju se iz **PHP** koda pomoću posebnog niza **$_GET[ ]** koji je indeksiran po nazivima prosleđenih podataka.

Primer:

formular.html:

```
…
<form action="obrada.php" method="GET">
    <label for="ime_id">Ime:</label>
    <input type="text" id="ime_id" name="ime">
    <input type="submit" value="Posalji">
</form>
…
```

obrada.php:

```
<?php

    // Provera da li su prosledjene ocekivane vrednosti,
    // ako vrednosti nisu prosledjene, prekida se izvrsavanje
    if(!isset($_GET['ime']))
        die('Nije prosledjen podatak sa imenom!');

    $ime = $_GET['ime'];

    // Ispis imena
    echo "Zdravo, ".$ime."!";

?>
```

Metodom **POST** se podaci ne prosleđuju kroz **URL** već kroz HTTP zahtev. Iz **PHP** koda se prosleđenim podacima pristupa preko specijalnog niza **$_POST[ ]**.

Primer:

formular.html:

```
…
<form action="obrada.php" method="POST">
    <label for="ime_id">Ime:</label>
    <input type="text" id="ime_id" name="ime">
    <input type="submit" value="Posalji">
</form>
…
```

obrada.php:

```
<?php

    // Provera da li su prosledjene ocekivane vrednosti,
    // ako vrednosti nisu prosledjene, prekida se izvrsavanje
    if(!isset($_POST['ime']))
        die('Nije prosledjen podatak sa imenom!');

    $ime = $_POST['ime'];

    // Ispis imena
    echo "Zdravo, ".$ime."!";

?>
```

Moguće je da ista **PHP** strana i šalje i obrađuje podatke, da bi se proverilo da li su podaci poslati na stranu može se dodati atribut **name** na **submit** dugme kojim se

prosleđuju podaci. U tom slučaju, proverom da li je postavljena vrednost navedenog imena u nizovima **$_GET** ili **$_POST** može se utvrditi da li su podaci prosleđeni za obradu ili tek treba da se proslede.

Primer:

formular_i_obrada.php:

```
…
<?php if(isset($_GET[‘slanje’]))
{
    // Podaci su prosledjeni, ali treba proveriti da li su svi
    // prosledjeni
    if(!isset($_GET[‘ime’]) || !isset($_GET[‘prezime’])
        echo “Nisu svi podaci prosledjeni!”;
    else
    {
        // Treba proveriti da li su prosledjeni podaci korektni
        $ime = $_GET[‘ime’];
        $prezime = $_GET[‘prezime’];

        if(strlen(trim($ime)) == 0 || strlen(trim($prezime)) == 0)
            echo “Nekorektni podaci!”;
        else
        {
            echo “Vase ime je: “.$ime.” a prezime “.$prezime;
        }
    }
} ?>

<form method=”GET” action=”formular_i_obrada.php”>
    <label for=”ime_id”>Ime: </label>
    <input type=”text” id=”ime_id” name=”ime”>
    <br>
    <br>
    <label for=”prezime_id”>Prezime: </label>
    <input type=”text” id=”prezime_id” name=”prezime”>
    <br>
    <br>
    <input type=”submit” name=”slanje” value=”Posalji”>
</form>
…
```

AJAX

AJAX (*Asynchronous Javascript And XML)* je skup tehnika koje omogućavaju slanje i dohvatanje podataka sa servera kroz jezik **Javascript** bez potrebe za ponovnim učitavanjem celokupne strane. U biblioteci **jQuery**, **AJAX** zahtevi se izvršavaju pomoću metoda **$.ajax.** Metodu **$.ajax** se kao argument navodi **objekat** koji sadrži podatke o putanji na koju se podaci šalju ili sa koje se dohvataju (**url**), metod kojim se podaci šalju (**method**), podaci koji se šalju (objekat **data** sa parovima **kljuc : vrednost**), *callback* funkciju koja se izvršava nakon što je zahtev uspešno obavljen (**success**), *callback* funkciju koja se izvršava u slučaju da zahtev nije uspešno obavljen (**error**) i druge podatke. Funkcija response kao argument dobija sadrzaj dobijen kao odgovor od servera .

Primer:

obrada.php:

```
<?php
    if(!isset($_GET[‘ime’]))
    {
        die(‘Neispravni podaci!’);
    }

    $ime = $_GET[‘ime’];
    echo “Podatak prosledjen kroz AJAX, vase ime je: ”.$ime ?>
?>
```

index.html:

```
…
<label for=”ime_id”>Ime: </label>
<input type=”text” id=”ime_id”>
<button onclick=”posalji_ajax()”>Posalji</button>
<p id=”odgovor_sa_servera”></p>

<!-- Povezivanje jQuery biblioteke -->
<script src=”jquery.min.3.2.1.js”></script>

<!-- Povezivanje korisnicke javascript datoteke -->
<script src=”skript.js”></script>
```

skript.js:

```
document.ready(function(){

    function posalji_ajax()
    {
        var tekst_imena = $(‘#ime_id’).val();
```

```
        $.ajax({
            url: 'obrada.php',
            method: 'GET',
            data: {ime: text_imena},
            success: function(odgovor){
                    $('#odgovor_sa_servera').html(odgovor);
                },
            error: function(){
                    alert('Neuspesan zahtev!');
                }
        });
    }

});
```

MySQL

MySQL je sistem za upravljanje relacionim bazama podataka. Podaci su smešteni u tabelama, svaki red tabele predstavlja primerak jednog entiteta (npr. proizvod) koji je opisan **atributima** (naziv, cena, barkod...). Za obradu i dohvatanje podataka koriste se upiti napisani na jeziku **SQL** (*Structured Query Language*). Postoje četiri vrste upita kojima se dohvataju i obrađuju podaci – **SELECT**, **INSERT**, **UPDATE** i **DELETE**. Nekad se ovaj skup operacija naziva i **CRUD – Create Read Update Delete**.

SELECT upiti

SELECT upitom se dohvataju podaci koji zadovoljavaju neke navedene uslove. Struktura jednostavnijeg **SELECT** upita je:

SELECT <spisak atributa ili * za sve atribute>
FROM <nazivi tabela u kojima se podaci traže>
WHERE <uslovi koji važe za tražene podatke>

Dohvatanje vrednosti svih atributa svih proizvoda iz tabele proizvod:

Primer:

```
SELECT * FROM proizvod
```

Dohvatanje svih naziva i cena proizvoda iz tabele proizvod:

Primer:
```
SELECT naziv, cena FROM proizvod
```

Dohvatanje svih naziva proizvoda čija je cena veća od 100:

Primer:
```
SELECT naziv FROM proizvod WHERE cena > 100
```

Moć relacionih baza je i u efikasnom spajanju tabela po odgovarajućim parametrima. Za spajanje tabela i dohvatanje zajedničkih podataka iz tabela koristi se naredba **JOIN**. Struktura spajanja je oblika:

JOIN <tabela1> **AND** <tabela2>
ON <zajednicki atribut iz tabele1> **=** <zajednicki atribut iz tabele_2>

Moguće je preimenovanje atributa koji se dohvataju navođenjem novog naziva pored naziva atributa (opciono je dodavanje ključne reči **AS** ispred).

Primer:
```
SELECT naziv AS IME, cena VREDNOST FROM proizvod
```

Dohvatanje naziva proizvoda iz tabele proizvod i naziva odgovarajućih proizvođača proizvoda iz tabele proizvođač, prvi način korišćenjem **JOIN**:

Primer:
```
SELECT proizvod.naziv AS naziv_proizvoda,
       proizvodjac.naziv AS naziv_proizvodjaca

FROM proizvod JOIN proizvodjac
               ON proizvod.id_proizvodjaca = proizvodjac.id
```

Drugi način, bez korišćenja **JOIN**:

```
SELECT proizvod.naziv AS naziv_proizvoda,
       proizvodjac.naziv AS naziv_proizvodjaca

FROM proizvod, proizvodjac
WHERE proizvod.id_proizvodjaca = proizvodjac.id
```

UPDATE upiti

UPDATE upiti se koriste za ažuriranje postojećih podataka u tabelama. Struktura **UPDATE** upita je:

UPDATE <naziv tabele> **SET**
<naziv atributa1> **=** <nova vrednost atributa 1>**,**
<naziv atributa2> **=** <nova vrednost atributa 2>**,**
…
WHERE <uslov koji vazi za podatke koji se menjaju>

Postavljanje cene proizvoda na duplo veću gde je vrednost trenutne cene ispod 100:

Primer:
```
UPDATE proizvod SET cena = cena * 2 WHERE cena < 100
```

INSERT upiti

INSERT upiti se koriste za dodavanje novih podataka u tabele. Struktura jednostavnog **INSERT** upita je:

INSERT INTO <naziv tabele> **(**atribut 1, atribut 2, atribut 3,…**)**
VALUES (vrednost atributa 1, vrednost atributa 2, vrednost atributa 3**)**

Potrebno je obratiti pažnju da se tekstualne vrednosti navode između jednostrukih navodnika. Sve vrednosti koje nisu navedene pri unosu dobijaju podrazumevane vrednosti. Ukoliko podrazumevane vrednosti nisu postavljene upit se neće izvršiti.

Dodavanje novog proizvoda čiji je naziv **novi proizvod**, cena **150**, barkod **1234567891234** i ID proizvođača **1**.

Primer:
```
INSERT INTO proizvod (naziv, cena, barkod, id_proizvodjaca)
VALUES (‘novi naziv’, 150, ‘1234567891234’, 1);
```

DELETE upiti

DELETE upiti se koriste za dodavanje novih podataka u tabele. Struktura **DELETE** upita je:

DELETE FROM <naziv tabele>
WHERE <uslov koji važi za podatke koje je potrebno obrisati>

Brisanje proizvoda čija je cena veća od 500:

Primer:
```
DELETE FROM proizvod WHERE cena > 500
```

MySQL i PHP

Iz jezika **PHP** moguće je korišćenje **MySQL** sistema za upravljanje bazama podataka korišćenjem proširenja **mysqli**. Funkcija kojom se osvaruje konekcija sa bazom je **mysqli_connect()** funkcija. Prvi argument funkcije **mysqli_connect()** je putanja do servera na kome se baza nalazi (za lokalno testiranje putanja je **localhost**), drugi argument je korisničko ime za pristup bazi, treći lozinka i četvrti naziv baze podataka kojoj se pristupa (jedan MySQL sistem može opsluživati više baza podataka). Povratna vrednost ove funkcije je konekcija sa bazom. Ako je došlo do greške prilikom povezivanja, funkcija **mysqli_errno()** sa konekcijom navedenom kao argument će vratiti vrednost različitu od nule, što može signalizirati da se greška dogodila.

Primer:
```
<?php

    function connect()
    {
        $konekcija = mysqli_connect('localhost', 'root', '',
                                                    'prodavnica');

        if(mysqli_errno($konekcija))
            die('Greska prilikom konekcije!')

        return $konekcija;
    }

?>
```

Funkcija za zatvaranje konekcije sa bazom je **mysqli_close()**, kojoj se kao argument navodi konekcija koja se zatvara.

Primer:

```
<?php
    function disconnect($konekcija)
    {
        mysqli_close($konekcija);
    }
?>
```

SQL upiti se izvršavaju korišćenjem funkcije **mysqli_query()** čiji je prvi argument konekcija sa bazom a drugi niska sa upitom koji se izvršava. Povratna vrednost funkcije je objekat sa rezultatima upita ili **FALSE** ukoliko upit nije uspešno izvršen. Funkcijom **mysqli_num_rows()**, kojoj se kao argument prosleđuje promenljiva sa rezultatom upita vraća broj podataka dohvaćenih upitom. Pojedinačni redovi rezultata, koji su predstavljeni asocijativnim nizom indeksiranim nazivima dohvaćenih atributa, dohvataju se funkcijom **mysqli_fetch_assoc()** kojoj se kao argument navodi promenljiva sa rezultatima upita. Kada funkcija **mysqli_fetch_assoc()** više nema redova za dohvatanje, vraća **FALSE**.

Primer:

```
<?php

    // Ukljucivanje korisnicke biblioteke za rad sa bazom, sadrzi
    // funkcije connect i disconnect
    include(db.php);

    $konekcija = connect();

    $upit = “SELECT naziv, cena FROM proizvod”;
    $rezultat = mysqli_query($konekcija, $upit);

    if(!$rezultat)
        die(‘Greska prilikom izvrsavanja upita!’);

    if(mysqli_num_rows($rezultat))
        die(‘Upit nije dohvatio ni jedan podatak!’);

    while($red = mysqli_fetch_assoc($rezultat))
    {
        echo “Naziv proizvoda: $red[‘naziv’]”;
        echo “<br>”
        echo “Cena proizvoda: $red[‘cena’]”;
        echo “<hr>”
    }

    disconnect($konekcija);

?>
```