

Uvod u JavaScript



Saša Fajković

2015.

SADRŽAJ:

1) Uvod	2
2) Internet preglednik i konzole	3
3) <script>	4
4) Prikaz podataka korištenjem JavaScripta	8
5) Varijable.....	11
6) Funkcije.....	14
7) Operatori	17
8) Polja	23
9) Komentari.....	24
10) Globalne i lokalne varijable	25
11) Kontrola toka – IF, ELSE IF, ELSE	26
12) Petlje – For, While, Do While	29
13) Događaji (events).....	39
14) Metode nad String tipom podataka	43
15) Metode nad numeričkim tipovima podataka.....	48
16) toString().....	50
17) HTML DOM	51
18) Rad s kontrolama.....	54
19) Popis slika	63

1) Uvod

1. a Što je JavaScript

JavaScript nam omogućava izradu dinamičkih web stranica u kombinaciji s HTML i CSS tehnologijama. Korištenjem JavaScript jezika možemo mijenjati sadržaj na stranici ovisno o načinu interakcije korisnika sa stranicom.

Primjerice, na pritisak gumba (*button*) možemo promijeniti tekst u nekom paragrafu. U JavaScriptu možemo kreirati varijable za pohranu podataka te pozivati mnogobrojne metode kako bi stranicu učinili zanimljivijom za korisnika. Svatko tko ima prethodnog iskustva u programiranju će vrlo lako svladati JavaScript, posebice jer je sintaksa JavaScript jezika jako slična drugim popularnim jezicima poput C++, C#, Java i drugih.

Pretpostavka je da ste već upoznati s HTML i CSS tehnologijama i objašnjavanje HTML elemenata i CSS svojstava se ovdje neće objašnjavati.

Primjeri koji će biti prikazani su izrađeni u Brackets ¹programu (editoru). (<http://brackets.io/>). Vi možete naravno koristiti bilo koji drugi program (ili punokrvno razvojno sučelje) koji podržava JavaScript jezik. Naravno, ako želite možete raditi i u klasičnom Notepad programu no to nije preporuka jer Notepad ne podržava obilježavanje sintakse (*syntax highlighting*).

1. b Kada NE koristiti JavaScript

Zahvaljujući novitetima predstavljenima u CSS3, za neke animacije i reagiranje na akcije korisnika nije više potrebno koristiti JavaScript. Za prikaz pomoćnih obavijesti (*tooltip*), menija, zamjene slika i razne tranzicije, danas možemo koristiti samo CSS3 tehnologiju.

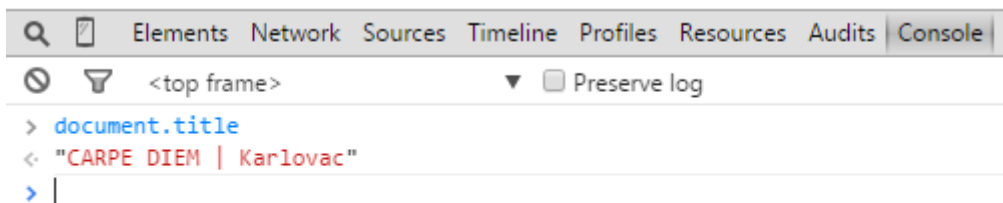
¹ <http://brackets.io/>

2) Internet preglednik i konzole

Prije početka rada s JavaScriptom dobro je upoznati se s Internet preglednikom i konzolama koje su ugrađene u njih. Svaki moderan Internet preglednik ima konzolu ugrađenu u sebe. U pregledniku Google Chrome otvaramo Developer Tools, u Safari pregledniku imamo Web Inspector, a Mozilla Firefox ima koristan dodatak (*add-on*) Firebug.

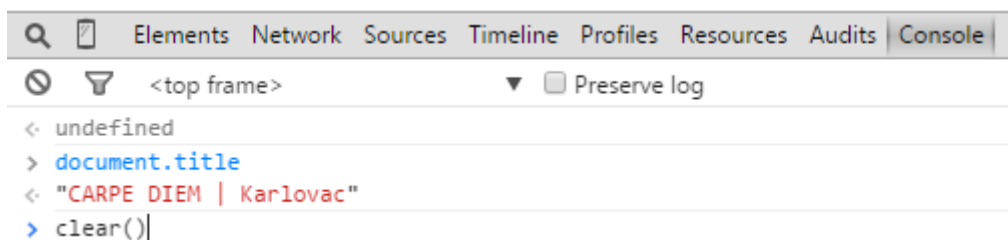
U Google Chrome je dovoljno pritisnuti F12 ili Ctrl+Shift+i kako bi otvorili Developer Tools. Svaki Internet preglednik ima svoje kratice i dobro je upoznati se s njima prije početka tada.

Konzola (*Console*) omogućava direktno izvršavanje JavaScript koda unutar Internet preglednika.



Slika 1 - Konzola u Google Chrome pregledniku

Ako želimo očistiti konzolu pozivamo funkciju **clear()**.



Slika 2 - clear() funkcija za čišćenje konzole

3) <script>

~~Na „Internetu“ postoje razne vrste skripti koje naši Internet preglednici prepoznaju. Kako bi Internet preglednik znao o kojoj vrsti skripte se radi moramo mu specificirati da je tip naše skripte JavaScript.~~

Gornji odlomak je s razlogom prekrížen. Iako je ovaj postupak bio potreban prije, danas to više nije slučaj jer je JavaScript pretpostavljeni (*default*) skriptni jezik HTML-a.

Vidjet ćemo 3 mogućnosti kreiranja skripti. Prvo ćemo upoznati kreiranje skripti unutar samog HTML dokumenta na dva načina, a zatim ćemo vidjeti kako skripte smjestiti u zasebnu datoteku te kako pozvati skriptu u naš HTML dokument.

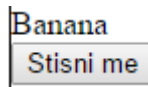
3. a Postavljanje u <head>

JavaScript skriptu možemo postaviti u <head> dio HTML dokumenta. Unutar skripte ćemo kreirati funkciju koju ćemo pozvati na pritisak gumba.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
  <script type="text/javascript">
    function ispis1() {
      document.write("Učim JavaScript");
    }
  </script>
</head>
<body>
  <div class="glavni">
    <p id="demo">Banana i jabuka</p>
    <button type="button" onclick="ispis1()" id="gumb">Stisni me</button>
  </div>
```

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

Ovaj kod će kreirati jedan gumb (button) s tekстом „Stisni me“. Važno je uočiti događaj **onclick** koji je vezan za taj gumb. On označava koja funkcija će se pokrenuti kada se pritisne taj gumb. U našem slučaju će se pozvati funkcija „ispis1()“ koja će u HTML dokument upisati „Učim JavaScript“.



Slika 3 - Gumb koji će pokrenuti funkciju ispis1()

Pogledajmo što se dogodilo nakon pritiska gumba.

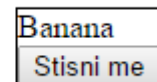
Učim JavaScript

Slika 4 - tekst koji je ispisan nakon poziva funkcije ispis1()

Ako bismo u <head> dodali samo <script> tag (bez function() { }) tada bi se naredba document.write(„Učim JavaScript“); izvršila odmah prilikom učitavanja HTML dokumenta.

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
  <script type="text/javascript">
    document.write("Učim JavaScript");
  </script>
</head>
```

Učim JavaScript



Slika 5 - Izgled HTML dokumenta nakon izvršavanja JavaScript skripte

Isto bi se dogodilo ako bismo u <body> dijelu kreirali samo skriptu umjesto da kod koji se izvršava dodamo u funkciju koja je zatim smještena unutar skripte.

3. b Postavljanje u <body>

Isto kao i kod postavljanja skripte u <head> dio, kada kreiramo skriptu unutar <body> dijela dokumenta pišemo unutar <script> tagova funkciju koja će se izvršiti u nekom trenutku odnosno na neki događaj.

```
<html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <script>
    function ispis2() {
      document.write("Skripta u body dijelu.");
    }
  </script>
  <div class="glavni">
    <p id="demo">Banana</p>
    <button type="button" onclick="ispis2()" id="gumb">Stisni me</button>
  </div>
</body>
</html>
```

Ovaj kod će (kao i u <head> dijelu) generirati gumb na kojem piše „Stisni me“. Kada pritisnemo gumb poziva se funkcija „ispis2()“ koja u HTML dokument ispisuje tekst „Skripta u body dijelu.“

3. c Pozivanje skripte iz drugog dokumenta

Kako bi mogli pozvati skriptu iz drugog dokumenta, potrebno je kreirati JavaScript datoteku koja ima ekstenziju „js“.

Pogledajmo kod koji piše u JavaScript datoteci js1.js

```
function test1() {  
    document.write("Testiram ispis iz JS dokumenta.");  
}
```

Vidimo da će se izvršiti samo upisivanje u dokument kada se pozove funkcija pod imenom test1().

Pogledajmo HTML dokument.

```
<html>  
<head>  
    <meta charset="utf-8">  
    <link rel="stylesheet" href="style.css">  
    <script src="js1.js"></script>  
</head>  
<body>  
    <div class="glavni">  
        <p id="demo">Banana</p>  
        <button type="button" onclick="test1()" id="gumb">Stisni me</button>  
    </div>  
</body>  
</html>
```

Unutar <head> dijela smo naveli lokaciju do datoteke koja je JavaScript skripta koja unutar sebe ima funkciju. Dakle, kreirali smo skriptu kao što bismo to napravili i da radimo JavaScript kod unutar samog HTML koda samo smo dodali putanju tj. izvor do JavaScript datoteke koristeći *source* tj. **src** atribut. Kada smo na pritisak gumba (onclick) pozvali funkciju test1(), ta se funkcija tražila na lokaciji skripte koju smo odredili u <head> dijelu.

4) Prikaz podataka korištenjem JavaScripta

U ovom dijelu ćemo naučiti kako prikazati informacije u HTML dokumentu odnosno kako utjecati na njihov prikaz korištenjem JavaScripta.

4. a **window.alert()** – iskočni prozor

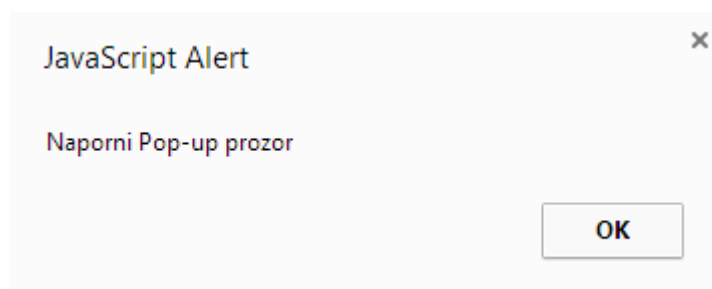
Izbacivanje poruke korisniku putem iskočnog prozora (*pop-up*) se radi korištenjem `windows.alert()` naredbe. U ovom slučaju je `alert()` metoda (funkcija) u koju možemo upisati tekst koji da se prikaže u iskočnom prozoru.

Prvo ćemo kreirati skriptu koja sadrži funkciju.

```
<script>
function test1() {
    window.alert("Naporni Pop-up prozor");
}
</script>
```

Funkciju ćemo pozvati na pritisak gumba, a funkcija će prikazati iskočni prozor s porukom „Naporni Pop-up prozor“.

```
<button type="button" onclick="test1()" id="gumb">Stisni me</button>
```



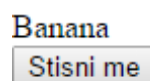
Slika 6 - Iskočni prozor (Pop-up)

4. b document.write() – pisanje u HTML dokument

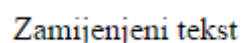
Ako želimo zamijeniti ono što je upisano u HTML dokument koristimo metodu write() koju pozivamo nad objektom document. Unutar obliha zagrada metode write() navodimo što želimo ispisati u HTML dokument. Imajte na umu da će ovaj postupak prebrisati postojeće informacije u tom HTML dokumentu s onima koje predamo metodi write() kao parametar.

U ovom primjeru imamo skriptu koja sadrži funkciju za promjenu teksta u HTML dokumentu, a funkcija će se izvršiti pritiskom na gumb.

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="style.css">
  <script>
  function test2() {
    document.write("Zamijenjeni tekst");
  }
  </script>
</head>
<body>
  <div>
    <p id="demo">Banana</p>
    <button type="button" onclick="test2()" id="gumb">Stisni me</button>
  </div>
</body>
```



Slika 7 - HTML dokument prije izvršavanje skripte



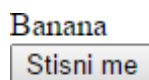
Slika 8 - HTML dokument nakon izvršavanja skripte

4. c `document.getElementById(id).innerHTML`

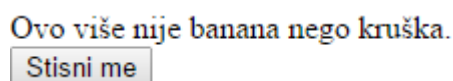
Često ćemo htjeti utjecati na sadržaj ili prikaz samo nekog određenog elementa unutar HTML dokumenta. U tom slučaju koristimo `document.getElementById()` metodu kojoj kao parametar predajemo ID elementa kojeg želimo mijenjati. Nakon zatvorene okvire zagrade stavljamo svojstvo `innerHTML` čime određujemo da se radi o HTML sadržaju.

U ovom primjeru će se pokrenuti funkcija na pritisak gumba koja će promijeniti sadržaj `<p>` elementa (*paragraph*).

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="style.css">
  <script>
  function test3() {
    document.getElementById("demo").innerHTML = "Ovo više nije banana nego kruška.";
  }
  </script>
</head>
<body>
  <div>
    <p id="demo">Banana</p>
    <button type="button" onclick="test3()" id="gumb">Stisni me</button>
  </div>
</body>
```



Slika 9 - HTML dokument prije izvršavanja funkcije



Slika 10 - HTML dokument nakon izvršavanja funkcije

5) Varijable

Varijable nam služe za pohranu podataka. Ljudima je često jako teško pamtiti dugačke podatke (tekst, brojevi s 10 decimalnih mjesta) pa koristimo varijable koje simbolički označavaju neku vrijednost. Primjerice možemo imati varijablu imena „marko“ koja će zapravo označavati ime, prezime, datum rođenja, oib, kontakt broj i kontakt email. Puno nam je lakše upamtiti samo ime varijable „marko“ nego sve te podatke.

Varijable kreiramo korištenjem ključne riječi „var“.

```
var marko = "Ovo je neki dugački tekst koji mi je teže upamtiti nego ime varijable";
```

Ovaj kod će kreirati varijablu imena „marko“ koja će zapravo obilježavati cijeli tekst koji je naveden unutar dvostrukih navodnika.

Varijablama pridjeljujemo vrijednosti korištenjem operatora za dodjeljivanje -> = (simbol jednakosti).

Kao i do sada, kraj naredbene linije moramo završiti simbolom točka-zarez (;).

Varijable će nam omogućiti da primjerice ispišemo neki tekst koji nismo unaprijed odredili. Primjerice, tražimo od korisnika da unese neki tekst, taj tekst spremimo u varijablu, a vrijednost te varijable naknadno ispišemo. Ovakvim primjerom smo ispisali tekst na koji ne možemo utjecati i nismo ga bili svjesni prilikom stvaranja varijable.

Velika i mala slova – JavaScript je *case sensitive* što znači da jezik raspoznaje razliku između velikih i malih slova. Primjerice varijabla **marko** je sasvim druga varijabla od **Marko**.

Ograničenja u imenima varijabli – Korištenje razmaka u imenu varijable nije dopušteno. Varijabla ne smije započeti s brojem odnosno smije započeti slovom ili simbolom podcrte (_). Načelno je pravilo da ne koristite hrvatska slova nego engleska u imenu varijable.

5. a var

Za razliku od velikog broja programskih jezika u kojima je potrebno navesti striktan tip podatka za svaku varijablu, u JavaScriptu to nije potrebno. Koristimo ključnu riječ **var** i dodjeljujemo neku vrijednost koja može biti cjelobrojna vrijednost, broj s decimalnom komponentom, pozitivni i negativni brojevi, jedan znak ili tekst (skup znakova). JavaScript će sam interno prebacivati između sebe tipove podataka po potrebi.

5. b string

Važno je paziti da prilikom pisanja **String** tipa podatka (**tekst**) moramo koristiti dvostruke ili jednostruke navodnike i unutar tih navodnika navodimo vrijednost odnosno tekst.

```
var stringVarijabla = "Ovo je string";  
var cijeliBroj = 3;  
var decimalniBroj = 3.14;
```

Problem kod ispisa teksta se može pojaviti želimo li ispisati citat. Primjerice : Saša je rekao: „Učimo JavaScript“.

Ako pokušavamo ispisati ovako tekst:

```
document.write("Saša je rekao: "Učim JavaScript" ");
```

dobit ćemo pogrešku jer dio koji smo smatrali tekстом „Učim JavaScript“ neće biti prepoznat kao string tip podatka već kao varijable.

U ovakvim slučajevima koristimo tzv. *Escape character* koji nam omogućava da kažemo jeziku da slijedeći simbol ne tretira kako bi ga inače tretirao. U JavaScript jeziku je to simbol *backslash* koji se piše -> „\“ (bez navodnika). To nije simbol „/“ koji dobijemo pritiskom na Shift+7. Primijetite da je okrenut na drugu stranu. Nama treba simbol koji dobijemo pritiskom na AltGr+Q.

```
document.write("Saša je rekao: \"Učim JavaScript\" ");
```

Ovakav kod će ispravno prikazati tekst baš kako smo zamislili; da se „Učim JavaScript“ prikaže unutar dvostrukih navodnika.

5. c Spajanje stringova i spajanje varijabli

U JavaScript jeziku string tipove podataka međusobno povezujemo simbolom „+“. Primjerice

```
var a = "Ovo je ";  
var b = "neki tekst! ";  
document.write(a+b);
```

će rezultirati ispisom teksta „Ovo je neki tekst“.

Primjerice:

```
var ime = "Saša";  
var godine = 25;  
document.write("Moje ime je " + ime + " i imam "+godine+" godina.");
```

će ispisati tekst „Moje ime je Saša i imam 25 godina.“.

5. d Boolean

Bool odnosno Boolean je još jedan tip podatka koji ima samo dvije mogućnosti, **True** ili **False**. Te dvije su sve moguće teoretske mogućnosti. Dakle, ili nešto je ili nešto nije. Ili je nešto istina ili nije istina.

```
Var istina = true;
```

Primijetite kako je ova vrijednost (true) posebno označena naspram primjerice string tekstu.

5. e Null

Null je specifična vrijednost jer obilježava da nešto nema vrijednost. Zamislimo ju kao praznu varijablu. Dakle, ako imamo varijablu čija je vrijednost nuli, to NIJE isto kao i **Null**. Varijabla koja ima vrijednost nula zapravo ima vrijednost, a to je nula. Varijabla čija je vrijednost Null nema definiranu nikakvu vrijednost.

6) Funkcije

Funkcije nam omogućavaju bolju čitljivost, lakše održavanje i brže korištenje koda. Posebice je korisno ako imamo dio koda koji se izvršava više od jednom jer umjesto da na dva mjesta imamo po primjerice 20 linija koda, stvorimo funkciju koja sadrži te linije koda i onda samo pozovemo tu funkciju što je jedna linija koda.

6. a **Pravilo pisanja funkcija**

Funkcije započinjemo pisati ključnom riječi **function** nakon čega slijedi **ime funkcije**. Nakon imena idu otvorena i zatvorena obla zagrada unutar kojih se navode eventualni **parametri** funkcije. Kažemo da **pozivamo funkciju**, da **funkcija vraća** neki podataka i da funkcija **prima parametre**. Unutar otvorenih i zatvorenih vitičastih zagrada navodimo kod koji će se izvršiti kada pozovemo funkciju.

6. b **Funkcija bez parametara**

Primjerice,

```
function funkcija1()
{
    alert("Ovo je iskočni prozor");
}
```

će kreirati iskočni prozor u kojem piše „Ovo je iskočni prozor“: Vidimo da je ime funkcije „funkcija1“. Ova funkcija ne prima niti jedan parametar. U tijelu funkcije se nalazi kod koji će se izvršiti kada se funkcija pozove, a to je da prikaže iskočni prozor s tekстом „Ovo je iskočni prozor“ unutar njega.

6. c Funkcija s parametrom

Jako često ćemo se susresti s funkcijama koje imaju jedan ili više parametara.

Definicija funkcije:

```
function funkcija1(tekst)
{
    alert("Ovo je " + tekst);
}
```

HTML kod:

```
<button type="button" onClick=funkcija1("PeroDjetlić") id="gumb">Stisni me</button>
```

Ovaj kod će generirati button kontrolu s natpisom „Stisni me“. Kada se taj button pritisne pojavit će se iskočni prozor u kojem će pisati „Ovo je PeroDjetlić“.

Korištenje **razmaka** prilikom predaje string parametra **nije dozvoljeno**. Pokušamo li kao parametar predati „Pero Djetlić“ funkcija neće raditi.

```
<button type="button" onClick=funkcija1("Pero Djetlić") id="gumb">Stisni me</button>
```

6. d Funkcija s više parametara

Često ćemo imati potrebu predati više parametara jednoj funkciji pa za takve funkcije kažemo da primaju više parametara.

```
function spajanje(a, b) {
    alert("Moje ime je "+a+", a prezime je "+b);
}
```

U HTML dijelu ponovno pozivamo ovu funkciju pritiskom na gumb pa će taj HTML kod izgledati ovako:

```
<button type="button" onClick=spajanje("Pero","Djetlić") id="gumb">Stisni me</button>
```

6. e Return naredba

Return naredbu koristimo kada želimo da funkcija vrati neki podatak. Primjerice, ne želimo da funkcija ispiše nešto u HTML dokument, nego želimo da izradi neku matematičku operaciju i vrati nam rezultat. Na osnovu rezultata ćemo dalje određivati kako će se izvršavati kod. Kažemo dakle da **funkcija vraća rezultat**.

Primjer funkcije koja vraća tekst:

```
function vracanjeTeksta() {  
    return "Vježbanje";  
}
```

HTML dio:

```
<script>  
    document.write(vracanjeTeksta());  
</script>
```

Korištenjem `document.write()` ćemo upisati nešto u HTML dokument. To nešto je u našem slučaju vrijednost koju će vratiti funkcija „`vracanjeTeksta()`“. Vidimo da će ta funkcija vratiti vrijednost „Vježbanje“ i to je vrijednost koja će se upisati u HTML dokument.

Sličan princip bi bio i prilikom zbrajanja dva broja. Imamo funkciju koja prima dva broja i vraćamo njihov zbroj.

```
function zbroji(a, b) {  
    return a+b;  
}
```

U HTML dijelu ispisujemo ono što vrati funkcija korištenjem `document.write()`;

```
<script>  
    document.write(zbroji(3,2));  
</script>
```

7) Operatori

7. a Matematički operatori

JavaScript koristi standardne matematičke operatore koje ste navikli vidati u gradivu matematike u srednjoj školi.

Za **zbrajanje** koristimo simbol **+**.

Za **oduzimanje** koristimo simbol **-**.

Za **množenje** koristimo simbol *****.

Za **dijeljenje** koristimo simbol **/**.

Dodatak na ova četiri osnovna operatora je operator **modulo** koji nam kao rezultat daje ostatak pri cjelobrojnom dijeljenju. U JavaScriptu se on obilježava simbolom **%**. Primjerice, želimo li napisati 15 modulo 6 to bismo napravili ovako: `15%6`. Rezultat ove operacije je 3 jer je ostatak pri cjelobrojnom dijeljenju 3. Broj 6 stane dva puta u broj 15 i još nam ostaje 3. Upravo taj 3 je rezultat.

S obzirom da se često koriste povećanje vrijednosti varijable za jedan ili smanjenje varijable za jedan, lijepi ali mudri programeri su osmislili skraćenicu za te postupke. Zamislimo da imamo varijablu imena „pero“ čija je vrijednost 10. Želimo li povećati za jedan, dovoljno je napisati `pero++`, odnosno želimo li smanjiti njenu vrijednost za jedan napisali bi `pero--`. Postupak povećanja za jedan se zove **inkrementacija**, a postupak smanjivanja za jedan se zove **dekrementacija**.

7. b Operatori pridruživanja

Dodjeljivanje vrijednosti varijabli smo već upoznali korištenjem operatora za pridruživanje „ = „. Ako bismo željeli vrijednost varijable uvećati za drugu vrijednost možemo koristiti najosnovniji pristup:

```
var a = 20;  
var b = 10;  
a = a + b;
```

Sada će varijabla „a“ iznositi 30. Drugi, brži način upisivanja (a time i učestalije korišten) je:

```
var a = 20;  
var b = 10;  
a +=b;
```

Na sličan način ćemo raditi i oduzimanje vrijednosti.

```
function ispis() {  
var a = 20;  
var b = 10;  
b-=a;  
return b;  
}
```

Operacije množenja i dijeljenja s pridruživanjem vrijednosti se pišu na isti način. Množenje i pridruživanje:

```
var a = 20;  
var b = 10;  
b*=a;
```

Dijeljenje i pridruživanje:

```
var a = 20;  
var b = 10;  
b/=a;
```

7. a Logički operatori

Logički operatori imaju samo dva moguća rješenja, a to su **True** i **False**. Dakle, ili nešto je zadovoljeno ili nešto nije. Upoznat ćemo operator „logičko i“, „logičko ili“ i „logičko ne“.

LOGIČKO I

Operator „logičko i“ zahtijeva da svi njegovi uvjeti budu zadovoljeni kako bi rezultat bio **True**. Ovaj operator se piše ovako: **&&**. Dva znaka „and“ (Shift+6).

```
var a = 5;  
var rezultat = (a>1 && a<10);
```

U ovom primjeru će varijabla „rezultat“ poprimiti vrijednost **True** jer su istovremeno zadovoljena oba uvjeta. Prvi uvjet je da je vrijednost varijable „a“ veća od 1. Drugi uvjet je da je vrijednost varijable manja od 10. S obzirom da su oba uvjeta **istovremeno** zadovoljena, rezultat je **True**. Da je samo jedan od uvjeta nezadovoljen, operator vraća vrijednost **False**.

LOGIČKO ILI

Za razliku od operatora „logičko i“ koji zahtijeva da su svi njegovi uvjeti istovremeno zadovoljeni kako bi rezultat bio **True**, operator „logičko ili“ zahtijeva da je barem jedan od uvjeta ispunjen kako bi operator dao vrijednost **True**. Ako niti jedan od uvjeta nije zadovoljen, rezultat operatora će biti **False**.

Logički operator „ili“ pišemo ovako: **||** (dvije vertikalne linije – AltGr+W)

```
var a = 95;  
var rezultat = (a<1 && a>90);
```

U ovom slučaju će varijabla „rezultat“ poprimiti vrijednost **True** jer iako nije ispunjen uvjet da je vrijednost varijable manja od 1, ispunjen je drugi uvjet da je vrijednost varijable veća od 90. Pravilo operatora „logičko ili“ je takvo da mora barem jedan od uvjeta biti zadovoljen da bi rezultat operatora bio **True**, i u ovom primjeru je to zadovoljeno.

LOGIČKO NE

Operator „logičko ne“ će dati rezultat True ako niti jedan od uvjeta nije zadovoljen. Dakle, suprotno od operatora „logičko i“. Operator se obilježava simbolom „!“ (uskličnik, Shift+1).

```
var a = 95;  
var rezultat = !(a==90);
```

U ovom primjeru gledamo je li istina da vrijednost varijable nije jednaka 90. S obzirom da je a=95 uvjet (a==90) nije zadovoljen. S obzirom da uvjet nije zadovoljen, operator „logičko ne“ će vratiti rezultat True.

7. b Operatori usporedbe

Za uspoređivanje vrijednosti koristimo klasične „veće od“, „manje od“ ili „jednako nečem“ operatore. Za uspoređivanje vrijednosti koristimo dva simbola znaka jednakosti.

```
var a = 7;  
a==7;  
a==8;
```

u prvom slučaju (a==7) rezultat je True. U drugom slučaju rezultat usporedbe je False.

U slučaju da radimo s različitim tipovima podataka, a uspoređujemo vrijednosti JavaScript će napraviti pretvorbu između tipova podataka kako bi usporedio vrijednosti.

```
var a = 5;  
a == „5“;
```

U ovom slučaju će rezultat biti True jer iako je varijabli „a“ dodijeljena numerička vrijednost 5, a radimo usporedbu sa String vrijednosti. Napravljena je pretvorba između tipova i izvršila se usporedba vrijednosti koje su u ovom slučaju iste pa je rezultat usporedbe True.

Ako želimo usporediti vrijednost i tip podatka koristimo tri znaka jednakosti.

```
var a = 5;  
a === „5“;
```

U ovom slučaju će rezultat biti False jer iako su vrijednosti iste, njihov tip podatka nije isti.

```
var a = 5;
a != 6;
```

U ovom primjeru vidimo novi operator „ != „ koji označava „nejednakost. Operator će vratiti **True** ako dvije vrijednosti **nisu jednake** odnosno vratiti će **False** ako dvije vrijednosti **jesu jednake**.

Kao što imamo usporedbu jesu li dvije vrijednosti jednake po vrijednosti i po tipu, isto tako imamo usporedbu da dvije vrijednosti nisu jednake po vrijednosti i po tipu.

```
Var a = 5;
a != „5“;
```

Rezultat ove usporedbe će biti **True** jer varijabli „a“ smo dodijelili vrijednost 5 no usporedbu radimo sa String vrijednost 5. Njihove su vrijednosti iste ali je tip podatka drugi što znači da nisu jednaki. Naš uvjet usporedbe je da te dvije vrijednosti nisu jednake po vrijednosti i po tipu podatka i s obzirom da je zadovoljeno da te dvije vrijednosti nisu jednake po vrijednosti i tipu podatka, rezultat usporedbe je True.

```
Var a = 5;
a < 4;
```

U ovom primjeru je rezultat False jer vrijednost varijable „a“ nije manja od 4.

```
Var a = 5;
a > 5;
```

U ovom primjeru je rezultat usporedbe također False jer vrijednost varijable nije veća od 5. Vrijednost varijable „a“ je jednaka 5 no mi radimo usporedbu je li vrijednost veća od 5 što u ovom slučaju nije i rezultat je False.

```
Var a = 5;
a <= 5;
```

Ova usporedba će dati rezultat True jer testiramo je li vrijednost varijable „a“ manja ili jednaka 5 što je u ovom slučaju zadovoljeno pa je rezultat True.

```
Var a = 5;
a >= 5;
```

Kao i u prethodnom primjeru, testiramo je li vrijednost varijable veća ili jednaka 5. U ovom primjeru je to zadovoljeno jer je 5 jednako 5 pa je rezultat usporedbe True.

8) Polja

Polja (*engl. Array*) koristimo za spremanje više vrijednosti koje su često logički povezane. Primjerice, želimo imati popis osoba u učionici pa možemo kreirati jedno polje koje će sadržavati popis učenika. Kažemo da se **polje sastoji od elemenata** i da svaki element ima svoj jedinstveni indeks.

Prvi element polja ima indeks 0. Drugi element polja ima indeks 1, i tako dalje.

```
function polje(element) {  
    var ucenici = ["Saša","Marko","Pero"];  
    alert(ucenici[element]);  
}
```

Ovim kodom smo kreirali polje imena **ucenici** koje se sastoji od tri elementa. Prvi element je „Saša“, drugi je „Marko“, a treći „Pero“. To znači da „Saša“ ima indeks nula, „Marko“ ima indeks jedan, a „Pero“ ima indeks dva.

Indeksi elemenata nam trebaju kako bi mogli odrediti koji točno element želimo pročitati ili kojem točno elementu želimo dodijeliti neku vrijednost.

```
<button type="button" id="gumb" onclick="polje(1)">Stisni me</button>
```

Vidimo da smo pritiskom na gumb pozvali funkciju `polje()` koja prima jedan argument odnosno parametar. Kao parametar smo predali broj jedan. Pogledamo li u definiciju funkcije `polje()` vidimo da će se u iskočnom prozoru pojaviti vrijednost koja je upisana u polju *ucenici* na tom indeksu koji smo predali kao parametar funkciji. U ovom konkretnom slučaju to je vrijednost „Marko“.

9) Komentari

Komentari su velika pomoć prilikom pisanja koda. Njihov kod se ne izvršava i služi kao pomoć nama, autorima našeg koda ili osobama kojima ćemo poslati naš kod kako bi mogli objasniti što koja linija koda radi ili barem pojasniti što koja funkcija radi.

Vrlo vjerojatno na početku nećete vidjeti veliku potrebu, a time i korist od komentara, no prilikom rada na većim projektima, komentari su od iznimne važnosti. U JavaScript jeziku komentare započinjemo s dvije kose linije u desno (*slash*) -> „ // „.

Sve napisano iza ova dva simbola će se smatrati komentarom. Primjerice:

```
function spajanje(a, b) {  
    alert("Moje ime je "+a+", a prezime je "+b);  
    //alert("ovo se neće ispisati!");  
}
```

U ovom primjeru vidimo da je treća linija koda označena kao komentar (*//alert(„ovo se neće ispisati!“);*)

Želimo li postaviti više linija pod komentar možemo koristiti za svaku liniju simbol (//) ili jednom koristiti /* i */ simbole. Početak komentara započinjemo simbolom /* , a kraj određujemo simbolom */.

10) Globalne i lokalne varijable

Do sada smo vidjeli kako kreirati varijable unutar funkcije. Takve varijable zovemo **lokalnim** varijablama. Varijabla koja je kreirana unutar funkcije je vidljiva samo unutar te funkcije. To znači da ne možemo dohvatiti njenu vrijednost ili joj pridijeliti vrijednost iz druge funkcije.

Ako ipak želimo kreirati varijable koje su vidljive u više funkcija tada pričamo o **globalnim varijablama**. Vidljivost varijabli nazivamo **scope** koji možemo zamisliti kao vidno polje koje obilježava od kuda sve vidimo varijablu. Ako je varijabla kreirana unutar funkcije, onda je ona vidljiva samo unutar te funkcije dok ako je varijabla globalna (kreirana unutar `<script></script>`) ona postaje vidljiva u svim funkcijama koje se nalaze unutar te skripte.

U `<head>` dijelu imamo ovaj kod:

```
<script>
  var ime = "Saša";
  function ispis() {
    return ime;
  }
</script>
```

A u `<body>` dijelu imamo :

```
<script>
  document.write(ispis());
</script>
```

Vidimo da smo u definiciji funkcije `ispis()` odredili da funkcija vraća vrijednost varijable „ime“ koja je navedena van te funkcije, tj. kao globalna varijabla. Zatim u `<body>` dijelu ispisujemo u HTML dokument vrijednost koju vrati ta funkcija.

11) Kontrola toka – IF, ELSE IF, ELSE

11. a Osnovne kontrole toka

Kontrole toka nam omogućavaju da se kod izvrši na različite načine, ovisno o tome što primjerice korisnik unese u TextBox kontrolu.

Sintaksa (pravilo pisanja jezika) je ovakva:

```
if (uvjet) {  
    kod koji se izvršava ako je uvjet ispunjen;  
}
```

Pogledamo jedan lagani primjer kontrole toka.

```
function ispis() {  
    var a = 5;  
    var b = 10;  
  
    if (a>b) {  
        return "A je veći od B.";  
    }  
    else {  
        return "A nije manji od B";  
    }  
  
    return b;  
}
```

U ovom primjeru vidim osnovnu kontrolu toka. Ako je vrijednost varijable „a“ veća od „b“ funkcija vraća kao rezultat „A je veći od B“. Ako taj uvjet nije zadovoljen (dakle ili je vrijednost varijable „a“ veća od vrijednost varijable „b“ ili je vrijednost varijable „a“ jednaka vrijednost varijable „b“).

S obzirom da je vrijednost varijable „b“ veća od vrijednosti varijable „a“ prvi uvjet (IF dio) nije zadovoljen i zbog toga se izvršava ELSE dio.

Ako bi željeli točnije odrediti odnos između ove dvije varijable (veće, manje ili jednako) trebat će nam više provjera uvjeta za što koristimo **ELSE IF**.

```
var a = 5;
var b = 10;

if (a>b) {
    return "A je veći od B.";
}
else if (a==b) {
    return "A je jednak B.";
}
else
{
    return "B je veći od A.";
}
```

U ovom primjeru prvo uspoređujemo je li vrijednost varijable „a“ veća od vrijednosti varijable „b“. Ako je taj uvjet ispunjen, funkcija vraća vrijednost „A je veći od B.“. Ako taj uvjet nije ispunjen onda se izvršava drugi uvjet gdje testiramo jesu li te dvije vrijednosti jednake. Ako jesu, funkcija vraća vrijednost „A je jednak B.“. Ako ni to nije slučaj, funkcija vraća rezultat „B je veći od A.“.

11. b Ugniježdene kontrole toka

Često ćemo se susresti s potrebom postavljanja jedne kontrole toka unutar druge. Ovakav postupak se zove ugnježdivanje.

```
var a = 5;
var b = 10;
if(a>0) {
    if(a<10) {
        return "A iznosi "+a;
    }
}
```

U ovom slučaju vidimo jednostavan primjer ugniježdene kontrole toka jer imamo jedan „IF“ koji se nalazi unutar drugog „IFa“. Naravno da će u praksi ovakvi primjeri biti kompleksniji i smisleniji od usporedbe dva broja. Uz unutarnji IF se može dalje navoditi i ELSE-IF i ELSE. Isto tako u unutarnji IF možemo dodati još jedan IF. Unutarnji IF možemo dodati i u ELSE-IF vanjskog IF-a i u ELSE. Dakle, sve kombinacije su dopuštene.

12) Petlje – For, While, Do While

Ovo poglavlje će predstaviti tri vrste petlji. Petlje nam omogućavaju da na jednostavan način ponavljamo izvršavanje nekog dijela koda. Primjerice, želimo ispisati brojeve od 1 do 100. Ako bismo to radili „korak po korak“ morali bismo imati 100 linija koda gdje svaka linija ispisuje jedan broj. Korištenjem petlji, ovaj problem ćemo riješiti u samo par linija koda.

Logika koja stoji iza petlji je da omogućimo izvršavanje nekog kôda dok god je neki uvjet zadovoljen. Ovisno od petlje do petlje, mijenja se njihova svrha, *sintaksa* i svojstva.

12. a for petlja

For petlja svoju vrlo čestu primjenu nalazi kako i prilikom izvršavanja istog koda više puta tako i prilikom prolaska kroz polja. Što su polja i koja su njihova svojstva će te saznati u narednim poglavljima no spomenuta su čisto kako bi bili upućeni u njihovo postojanje i često korištenje „for“ petlji u kombinaciji s poljima. *Sintaksa* je nešto kompliciranija za shvatiti naspram ostalih petlji no nakon nekoliko korištenja nećete imati problema.

Sintaksa :

```
for ( inicijalizacija; uvjet; uvećanje ili smanjenje )
{
kôd koji se izvršava dok god je uvjet zadovoljen
}
```

inicijalizacija – postavljamo vrijednost neke varijable na neku određenu vrijednost. Često se koriste slovo „i“ i slovo „j“. Korištenje ovih slova nije striktno definirano ali se radi pridržavanja konvencije najčešće koriste ta dva slova. Važno – varijablu koju inicijaliziramo nije potrebno definirati prije korištenja „for“ petlje već se ona najčešće deklarira prilikom stvaranja „for“ petlje i to unutar „for“ petlje na mjestu „inicijalizacija“. Primjer inicijalizacije : `i = 1;`

uvjet – ovaj parametar nam govori do kada će se for petlja izvršavati. Ako smo u prvom parametru (inicijalizacija) postavili vrijednost varijable „i“ na 1 (jedan) i sada kao parametar „izvođenje“ upišemo „i < 5“ čime smo rekli da se petlja izvrši 4 (četiri) puta.

Parametar „uvjet“ možemo gledati kao uvjet kod „if“-a dok ćemo parametar „inicijalizacija“ gledati kao vrijednost koju uspoređujemo s onom postavljenom u uvjetu. Ne brinite, sve će imati više smisla u prikazu primjera kojih će za potrebe „for“ petlje biti nekoliko.

Uvećanje ili smanjenje – srećom posljednji parametar u ovoj kompliciranoj petlji i vjerojatno najjednostavniji za shvatiti. Ovdje upisujemo za koliko da se uveća varijabla koju smo inicijalizirali u parametru 1 (inicijalizacija) nakon što se izvrši kod u tijelu petlje. Često ćete koristiti inkrementaciju odnosno `i++`;

12. b Sintaksa i primjeri „for“ petlje :

Primjer 1 :

```
function ispis() {  
    for (i=0; i<5; i++) {  
        document.write("i = "+i + "</br>");  
    }  
}
```

Tijelo for petlje sačinjava jedna naredba i to ispisa : `document.write(„i = "+ i + "</br>“`. Kod ove naredbe nemamo ništa novo. Problem u shvaćanju se javlja kod čudnog teksta unutar obliha zagrada.

Prvi parametar (do prvog točka-zarez simbola) je inicijalizacija varijable čije je ime „i“. Vrijednost ove varijable smo postavili na 0.

Dalje smo rekli programu : „sve dok je vrijednost varijable „i“ manja od broja 5 ti izvršavaj kôd koji ti piše u tijelu petlje“. To znači da će program izvršiti naredbu ispisa i ako je vrijednost varijable „i“ još uvijek manja od 5 ponovno će se izvršiti isti kôd i tako sve dok je vrijednost varijable „i“ manja od 5 odnosno, u ovom konkretnom slučaju dok varijabla „i“ ne poprimi vrijednost 5.

Promjenu vrijednosti varijable „i“ izvršavamo pomoću trećeg parametra. U ovom slučaju mi smo programu rekli „svaki put kada izvršiš kod koji ti piše u tijelu petlje, ti povećaj vrijednost varijable za jedan.

Detaljno objašnjenje što se i kako događa :

- 1) vrijednost varijable „i“ je 0 (nula)
- 2) program provjerava ako je „i“ manji od 5. (**0 < 5** 😊)
- 3) program uvidi da je „i“ manji od 5
- 4) ispisuje se naredba (**prvo ispisivanje**)
- 5) uvećava se vrijednost varijable za jedan i sada „i“ ima vrijednost 1
- 6) provjera da li je „i“ manji od 5 (**1 < 5** 😊)
- 7) program utvrdi da je „i“ manji od 5
- 8) ispisuje se naredba (**drugo ispisivanje**)

- 9) uvećava se vrijednost varijable za jedan i sada „i“ ima vrijednost 2
- 10) provjera da li je „i“ manji od 5 ($2 < 5$) 😊
- 11) program utvrdi da je „i“ manji od 5
- 12) ispisuje se naredba (**treće ispisivanje**)
- 13) uvećava se vrijednost varijable za jedan i sada „i“ ima vrijednost 3
- 14) provjera da li je „i“ manji od 5 ($3 < 5$) 😊
- 15) program utvrdi da je „i“ manji od 5
- 16) ispisuje se naredba (**četvrto ispisivanje**)
- 17) uvećava se vrijednost varijable za jedan i sada „i“ ima vrijednost 4
- 18) provjera da li je „i“ manji od 5 ($4 < 5$) 😊
- 19) program utvrdi da je „i“ manji od 5
- 20) ispisuje se naredba (**peto ispisivanje**)
- 21) uvećava se vrijednost varijable za jedan i sada „i“ ima vrijednost 5
- 22) provjera da li je „i“ manji od 5 ($5 < 5$) ❌
- 23) program utvrdi da sada „i“ više nije manji od 5
- 24) Petlja se prekida
- 25) Nastavlja se s izvođenjem koda koji se nalazi nakon „for“ petlje

Savjet

Uzmite prazan papir i zapišite 2-3 varijacije for petlji (mijenjajte samo drugi parametar) i pišite si svaki korak i izgovarajte ga na glas. Nakon 10-ak minuta će sve postati jednostavnije.

Primjer 2:

```
for ( int i = 0; i<5;i+=2) {  
    document.write(„Testiram for petlju“);  
}
```

Detaljno objašnjenje :

- 1) vrijednost varijable „i“ je 0 (nula)
- 2) program provjerava ako je „i“ manji od 5. ($0 < 5$) 😊
- 3) program uvidi da je „i“ manji od 5
- 4) ispisuje se naredba (**prvo ispisivanje**)
- 5) uvećava se vrijednost varijable za dva i sada „i“ ima vrijednost 2
- 6) provjera da li je „i“ manji od 5 ($2 < 5$) 😊
- 7) program utvrdi da je „i“ manji od 5
- 8) ispisuje se naredba (**drugo ispisivanje**)
- 9) uvećava se vrijednost varijable za dva i sada „i“ ima vrijednost 4
- 10) provjera da li je „i“ manji od 5 ($4 < 5$) 😊
- 11) program utvrdi da je „i“ manji od 5
- 12) ispisuje se naredba (**treće ispisivanje**)
- 13) uvećava se vrijednost varijable za dva i sada „i“ ima vrijednost 6
- 14) provjera da li je „i“ manji od 5 ($6 < 5$) ❌
- 15) program utvrdi da sada „i“ više nije manji od 5
- 16) Petlja se prekida
- 17) Program nastavlja s izvođenjem kôda koji se nalazi nakon „for“ petlje

Primjer 3:

```
for ( int i = 2; i<0;i--) {  
    document.write(„Pokusavam razumjeti for petlju“);  
}
```

Pogledajte rješenje! Zašto se nije ništa ispisalo?

Pogledajte opet što je upisano unutar uglatih zagrada „for“ petlje. Postavljena je varijabla „i“ na vrijednost 2 a provjera glasi „ako je $i < 0$ izvrši kôd u tijelu petlje“. Kako je od samog početka vrijednost varijable „i“ postavljena na 2 (dva), odmah u startu nije uvjet zadovoljen i petlja se prekida.

12. c while petlja

Ako ste uspjeli svladati „for“ petlju onda nećete imati nikakvih problema sa slijedeće dvije petlje. Prvo ćemo predstaviti „while“ petlju na koju ćemo nadograditi „do-while“ petlju. Ako bismo preveli na hrvatski jezik, ovu petlju bismo nazvali „dok petljom“. Kako već i samo ime sugerira, kôd koji se nalazi unutar tijela ove petlje će se izvršavati dok god je neki uvjet ispunjen. Kao što ste možda zamijetili, ideologija je slična kao „for“ petlji. Obje će se izvršavati dok je neki uvjet zadovoljen ali razlika je što će se „for“ petlja izvršavati točno definiran broj puta (imamo početnu vrijednost i krajnju vrijednost koju može varijabla poprimiti) dok u slučaju „while“ petlje ne znamo koliko će se puta neki dio kôda izvršiti.

12. d Sintaksa i primjeri while petlje

Sintaksa :

```
while ( uvjet )
{
kôd koji će se izvršavati dok god je uvjet ispunjen
}
```





uvjet – opisujemo što mora biti zadovoljeno da bi se naš kôd izvršio

Primjer 1 :

```
var i = 0;
while (i<3) {
    document.write(„Moja prva while petlja“);
    i++;
}
}
```

Uočite – varijabla koja se nalazi unutar uglatih zagrada odnosno dio je našeg uvjeta morala se deklarirati i inicijalizirati prije provjere uvjeta.

Detaljno objašnjenje :

- 1) provjera uvjeta da li je „i“ manje od 3 (0 < 3) 
- 2) uvjet je zadovoljen i ulazimo u petlju
- 3) izvršava se ispis
- 4) uvećavamo vrijednost varijable za jedan i sada „i“ ima vrijednost 1
- 5) provjera uvjeta da li je „i“ manje od 3 (1 < 3) 
- 6) uvjet je zadovoljen i ulazimo u petlju
- 7) izvršava se ispis
- 8) uvećavamo vrijednost varijable za jedan i sada „i“ ima vrijednost 2
- 9) provjera uvjeta da li je „i“ manje od 3 (2 < 3) 
- 10) uvjet je zadovoljen i ulazimo u petlju
- 11) izvršava se ispis
- 12) uvećavamo vrijednost varijable za jedan i sada „i“ ima vrijednost 3
- 13) provjera uvjeta da li je „i“ manje od 3 (3 < 3) 
- 14) uvjet nije zadovoljen i ne ulazimo u petlju
- 15) izvršava se kod iza while petlje

Uočite – na kraju koda unutar tijela petlje se nalazi **i++**. Razmislite što bi se dogodilo da nismo postavili uvećanje varijable za 1 odnosno da nam je kôd ovako izgleda :

```
int i = 0;
while (i<3)
{
    document.write(„Moja prva while petlja");
}
```

Ovakav problem se zove **BESKONAČNA PETLJA** i vrlo vjerojatno vam Internet preglednik sad uporno pokušava izvršiti tu skriptu ali njoj nikada nema kraja.

12. e Beskonačna petlja

Beskonačnu petlju je relativno lagano za uočiti i pri nešto složenijim aplikacijama. Kako joj i samo ime kaže, naredbe unutar tijela petlje će se izvršavati skoro beskonačno puta mnogo. Razlog ovome „skoro“ je zato što se teoretski neće beskonačno puta odviti jer će u jednom trenutku se morati zaustaviti pošto svako računalo ima ograničene resurse koji bi se kad-tad popunili. Naravno, vrijeme potrebno da ovako malo kôda popuni svu vašu memoriju je ovisno o Vašem računalu ali još važnije, skoro je beskonačno.

Koji je razlog ovome? Odgovor je zapravo jednostavan. Pokrenuli smo petlju i rekli joj „dok god je vrijednost varijable „i“ manja od 3, ti izvršavaj kôd koji ti piše u tijelu petlje“. Svaki put kada se sve linije u tijelu izvrše ponovno se radi provjera da li je uvjet još uvijek zadovoljen. Kako mi ni na jedan način ne utječemo na vrijednost varijable „i“ ona ostaje nepromijenjena cijelo vrijeme odnosno njena vrijednost je zauvijek 0 (nula).

12. f do-while petlja

Već iz samog imena možete zaključiti kako je ova vrsta petlje slična prethodno objašnjenjnoj „while“ petlji. *Sintaksa* im je poprilično slična uz jedan dodatak bloka kôda. U „do-while“ petlji prvo navodimo kôd koji želimo da se izvrši a u drugom dijelu navodimo uvjet koji treba biti zadovoljen. Kako znamo da se naš program izvodi od najgornje linije prema dolje možemo uočiti kako u „do-while“ petlji to znači da će prvo doći do dijela u kojem se opisuje koji kôd se treba izvesti tek nakon toga do uvjeta. Upravo to je svojstvo „do-while“ petlje i to je jedina petlja za koju kažemo da će se **sigurno izvršiti barem jednom**.

Sintaksa :

```
do {  
    kôd koji će se izvršiti  
}  
while ( uvjet );
```

Primijetite simbol točka-zarez na kraju retka u kojem se nalazi uvjet i pripazite da ne stvorite beskonačnu petlju.

Primjer 1 :

```
int i = 0;  
do {  
    document.write( "Moja prva do-while petlja");  
    i++;  
}  
while ( i < 3 );
```

U ovom primjeru izradili smo aplikaciju koja će ispisivati rečenicu „Moja prva do-while petlja“ dok god je vrijednost varijable „i“ (koju smo prethodno deklarirali i postavili na vrijednost 0 (nula)) manja od 3 (tri). Svaki put kada je uvjet zadovoljen ispisat će se naša rečenica i vrijednost varijable „i“ će se uvećati za 1 (jedan).

Detaljno objašnjenje :

- 1) Stvori varijablu imena „i“ te postavi joj vrijednost na 0
- 2) izvrši dio koda koji se sastoji od :
 - a. ispis rečenice u konzolu
 - b. uvećaj varijablu „i“ za jedan (**i = 1**)
- 3) provjeri da li je vrijednost varijable „i“ manja od 3 (**1 < 3**)
- 4) izvrši dio koda koji se sastoji od :
 - a. ispis rečenice u konzolu
 - b. uvećaj varijablu „i“ za jedan (**i = 2**)
- 5) provjeri da li je vrijednost varijable „i“ manja od 3 (**2 < 3**)
- 6) izvrši dio koda koji se sastoji od :
 - a. ispis rečenice u konzolu
 - b. uvećaj varijablu „i“ za jedan (**i = 3**)
- 7) provjeri da li je vrijednost varijable „i“ manja od 3 (**3 < 3**)
- 8) prekini izvođenje kôda u ovoj petlji i nastavi dalje s izvođenjem programa

Primjer 2 :

```
int i = 5;
do {
    document.write( "Moja prva do-while petlja ");
    i++;
}
while ( i < 3 );
```

Uočite kako je vrijednost varijable „i“ u ovom primjeru iznosi 5 (pet) a da nam je uvjet za izvršavanje petlje da je vrijednost varijable „i“ manja od 3.

Detaljno objašnjenje :

- 1) stvori varijablu imena „i“ i dodijeli joj vrijednost 5
- 2) izvrši dio kôda koji se sastoji od :
 - a. ispiši rečenicu
 - b. uvećaj vrijednost varijable „i“ (**i = 6**)
- 3) provjeri da li je vrijednost varijable „i“ manja od 3 (**6 < 3**)
- 4) prekini izvršavanje petlje i nastavi s daljnjim izvođenjem koda

Uočite da iako odmah u početku uvjet petlje nije bio zadovoljen, svejedno se kod u tijelu petlje izvršio. Ovdje se lijepo može uočiti slijed kojim se „do-while“ petlja izvršava odnosno da se prvo izvršava kôd u tijelu petlje a zatim se radi provjera da li je uvjet zadovoljen.

13) Događaji (events)

Od ovog poglavlja ćemo početi uvidati na koje načine možemo koristiti JavaScript mijenjanjem sadržaja u HTML dokumentu ovisno o događaju koji korisnik napravi na stranici odnosno napokon ćemo vidjeti kako kreirati dinamičke web stranice koje reagiraju na akcije korisnika.

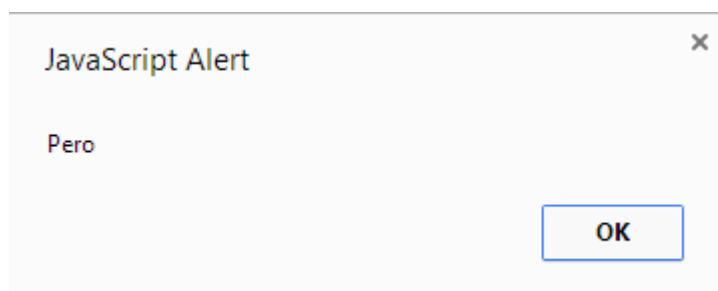
Trebamo razlikovati događaje (**events**) i **event-handlers** koji obrađuju neki događaj. Primjerice, događaj je pritisak na gumb, a event-handler je onclick() metoda.

13. a onclick

Za početak ćemo izraditi jedna gumb koji će imati event-handler onclick(). Kao što ime kaže, neki JavaScript kod ćemo pokrenuti kada mišem kliknemo na neku kontrolu (u našem slučaju na button).

```
<input type="button" value="Stisni me" onclick="alert('Pero');"/>
```

Vidimo da je event-handler **onclick()** koji poziva funkciju alert da ispiše riječ „Pero“. Važno je uočiti kako je pero postavljen unutar **jednostrukih navodnika**. HTML elementi imaju svoje atribute. U našem primjeru button ima atribut „type“ i „value“. Svojstva atributa se stavljaju u dvostruke navodnike i s obzirom da je takav način pisanja definiran za atribute HTML elemenata, moramo argument funkcije alert() postaviti unutar jednostrukih navodnika kako bi se mogli razlikovati odnosno da se „Pero“ ne bi smatrao argumentom.



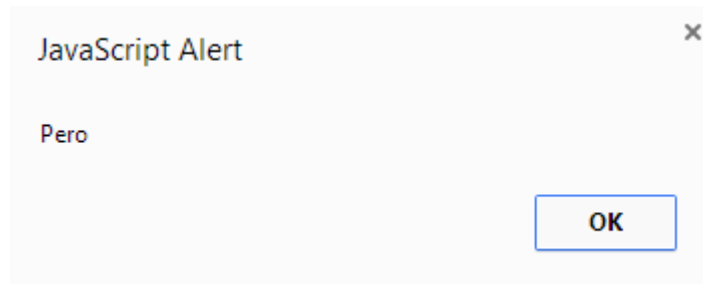
Slika 11 - Iskočni prozor

Pokušamo li s dvostrukim navodnicima, vidjet ćemo da se funkcija neće izvršiti.

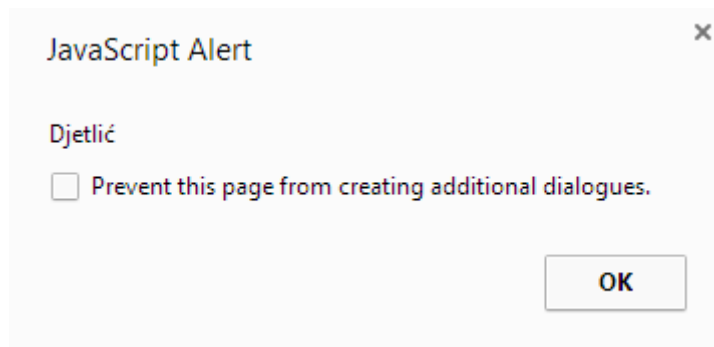
Za onclick() možemo dodati više funkcija. Primjerice, možemo postaviti dvije alert() funkcije.

```
<input type="button" value="Stisni me" onclick="alert('Pero'); alert('Djetlić');"/>
```

Ovaj kod će izbaciti dva iskočna prozora kada se pritisne gumb. Prvo će se prikazati „Pero“. Nakon toga će se prikazati „Djetlić“.



Slika 12 - Iskočni prozor 1



Slika 13 - Iskočni prozor 2

Vidimo da se prilikom drugog izbacivanja pojavila obavijest. Ovo nas sam Internet preglednik pokušava zaštititi jer smatra da stranica pokušava prikazati veliki broj iskočnih prozora (primjerice reklame).

13. b onmouseover

Ovaj događaj će se dogoditi svaki put kada kursorom prijeđemo preko zadanog elementa. Nije potrebno kliknuti na taj miš već samo prijeći preko tog elementa kako bi se aktivirao JavaScript kod.

```
<p onmouseover="alert('Test');">Banana</p>
```

13. c onmouseout

Ovaj događaj će se dogoditi kada se maknemo s nekog elementa. Primjerice, možemo kliknuti na neki element i neće se ništa dogoditi no čim uklonimo kursor s tog elementa, pokrenut će se kod.

```
<p onmouseout="imeStranice();">Banana</p>
```

Napravili smo *paragraph* koji će na događaj „onmouseout“ pokrenuti funkciju „imeStranice()“.

Pogledajmo definiciju funkcije:

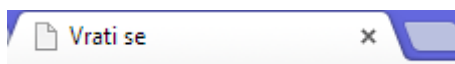
```
<script>
  function imeStranice()
  {
    document.title = "Vrati se !!!";
  }
</script>
```

Funkcija će izvršiti promjenu naziva HTML dokumenta iz koji god da je trenutno naveden u „Vrati se !!!“.



Slika 14 - onmouseout 1

Nakon što postavimo kursor nad *paragraph* elementom i zatim ga uklonimo, naslov HTML dokumenta se promijeni.



Slika 15 - onmouseout 2

13. d onload

Ovaj događaj se pokreće kada stranica završi s učitavanjem.

```
<body onload="alert('Stranica je učitana');">
```

U `<body>` element smo postavili `onload` event-handler. U trenutku kada stranica završi s učitavanjem, izbacit će se iskočni prozor s tekstom „Stranica je učitana“.

Naravno, ako trenutno na stranici nemate sadržaja, učitavanje će biti dovršeno praktički čim stisnete *Refresh*.

14) Metode nad String tipom podataka

String tipovi podataka su najčešće nakupine znakova (slova, brojevi i simboli). S takvim podacima možemo raditi raznovrsne manipulacije. Možemo mijenjati vrijednosti, pročitati dio teksta, obrisati dio i pronaći dio teksta unutar cjeline teksta.

Funkcije koje izvodimo nad String tipom podataka nazivamo **metodama**. Iako je princip korištenja identičan (imaju svoj tip, naziv, parametre i tijelo) razlika je u prirodi korištenja odnosno **metode** vežemo uz objekte odnosno objektno orijentirani pristup.

14. a indexOf()

Ova metoda će pronaći indeks pojavljivanja nekog znaka ili skupa znakova. Podsjetimo se, svaki znak u String podatku ima svoj jedinstveni indeks gdje prvi znak ima indeks nula (kao i polja).

```
function indeks()
{
    var znak = 'r';
    var rijec = "udruga"
    var pozicija = rijec.indexOf(znak);
    alert(pozicija);
}
</script>
```

Funkcija sadrži tri varijable. Prva varijabla (znak) ima vrijednost 'r' što nama predstavlja slovo koje ćemo tražiti. Varijabla „rijec“ sadrži vrijednost „udruga“ što je riječ u kojoj ćemo tražiti znak 'r'. U treću varijablu (pozicija) ćemo spremiti indeks na kojem se nalazi slovo 'r' u riječi „udruga“. Na kraju ćemo to prikazati u iskočnom prozoru. U ovom primjeru će se prikazati brojka 2 jer je slovo 'r' treće po redu slovo u riječi „udruga“ odnosno ima indeks 2.

Ako funkcija ne može pronaći neki znak, vratit će vrijednost -1. Ako je potrebno, možemo predati drugi parametar funkciji koji obilježava startnu poziciju traženja.

14. b lastIndexOf()

Suprotno od metode „indexOf()“, ova će metoda tražiti **zadnje pojavljivanje** nekog znaka ili skupine znakova.

```
function indeks()
{
    var znak = 'o';
    var rijec = "otorinolaringologija"
    var pozicija = rijec.lastIndexOf(znak);
    alert(pozicija);
}
```

U ovom primjeru tražimo zadnje pojavljivanje znaka 'o'. U ovom slučaju je to na poziciji 16 odnosno indeksu broj 15.

Ako funkcija ne može naći traženi znak, vratit će vrijednost -1. Ako je potrebno, možemo predati i drugi parametar koji obilježava startnu poziciju za početak pretraživanja.

14. c slice()

Metoda slice() će uzeti dio nekog stringa i vratiti nam ga. Kao parametre predajemo startnu poziciju od kuda da se započne dohvaćanje podataka te do kuda.

```
function vjezba()
{
    var rijec = "otorinolaringologija"
    var novaRijec = rijec.slice(2,5);
    alert(novaRijec);
}
```

Ovim kodom ćemo dohvatiti znakove od drugog indeksa do petog indeksa. U našem slučaju će se sada ispisati „ori“.

14. d substring()

Substring() metoda je identična slice() metodi no za razliku od slice() metode, substring() nema mogućnost unosa negativnog broja kao parametra.

14. e replace()

Metoda `replace()` će odabrani skup znakova zamijeniti novim skupom znakova. Primjerice, u rečenici „Ja sam početnik u JavaScript jeziku“ možemo zamijeniti riječ „početnik“ s riječi „profesionalac“.

```
function zamjena() {  
    var recenica = "Ja sam početnik u JavaScript jeziku."  
    var novaRecenica = recenica.replace("početnik","profesionalac");  
    document.write(novaRecenica);  
}
```

14. f concat()

Korištenjem funkcije `concat()` možemo dva ili više stringa spojiti u jedan.

```
function spajanje() {  
    var recenica = "Ja znam";  
    var recenica2 = recenica.concat(" JavaScript!");  
  
    document.write(recenica2);  
}
```

Ovaj kod će ispisati „Ja znam JavaScript!“.

14. g charAt()

Funkcija `charAt()` će pronaći znak koji se nalazi na nekom indeksu unutar nekog stringa. Treba imati na umu da je **indeks prvog elementa nula**. Primjerice:

```
function pozicija() {
    var recenica = "Ja znam";
    var recenica2 = recenica.concat(" JavaScript!");

    var pozicija = recenica2.charAt('5');

    document.write(pozicija);
}
```

Rezultat u ispisu će biti „a“. To slovo „a“ se nalazi u riječi „znam“. Prvo imamo dva znaka u riječi „Ja“ nakon čega slijedi razmak. Razmak se nalazi na indeksu broj 2. Zatim počinje riječ „znam“ iz koje smo pročitali slovo „a“.

14. h toUpperCase()

Funkcija `toUpperCase()` će pretvoriti slova unutar nekog stringa u velika slova. Ako je slovo već bilo veliko, ostat će veliko. Ako je bilo malo slovo, postat će veliko slovo.

```
function zamjena() {
    var recenica = "Ja znam";
    var recenica2 = recenica.concat(" JavaScript!");

    recenica2 = recenica2.toUpperCase();

    document.write(recenica2);
}
</script>
```

Rezultat će biti „JA ZNAM JAVASCRIPT!“.

14. i toLowerCase()

Suprotno funkciji toUpperCase(), ova će funkcija pretvoriti sva slova u mala slova. Ako je slovo već bilo malo, ostatak će malo, a velika slova će postati mala slova.

```
function zamjena() {  
    var recenica = "Ja znam";  
    var recenica2 = recenica.concat(" JavaScript!");  
  
    recenica2 = recenica2.toLowerCase();  
  
    document.write(recenica2);  
}
```

Ovaj kod će ispisati u dokument „ja znam javascript!“.

15) Metode nad numeričkim tipovima podataka

U ovom poglavlju ćemo upoznati funkcije koje možemo koristiti nad numeričkim tipovima podataka. Ove pretvorbe nam trebaju kada primjerice imamo neki broj koji je spremljen kao string. U tom slučaju ne možemo izvršiti matematičke operacije. Primjerice:

```
function brojevi() {  
    var a = "5";  
    document.write(a+5);  
}
```

Rezultat ovog ispisa neće biti 10 nego 55 jer će dogoditi spajanje dva stringa.

15. a **parseFloat()**

Funkcija `parseFloat()` nam omogućava pretvorbu iz string tipa podatka u numerički tip podatka. Float su brojevi s decimalnim dijelom.

```
function floatBroj() {  
    var a = "5.123";  
    document.write(parseFloat(a)+5.456);  
}
```

Korištenjem funkcije `parseFloat()` smo prvo izvršili pretvorbu iz string tipa podatka u float tip podatka nakon čega se izvršila matematička operacija zbrajanje. Rezultat će u ovom primjeru biti 10.579.

15. b parseInt()

Slično kao `parseFloat()`, ovu funkciju koristimo kada želimo pretvoriti string tip podatka u Integer tip podatka. Integeri su cijelobrojne vrijednost (primjerice, 3,7,15,25, itd).

```
function intBroj() {  
    var a = "5.923";  
    document.write(parseInt(a)+5);  
}
```

Rezultat koji će se ispisati je 10. Iako je rezultat zbrajanja 10.923, cjelobrojna komponenta je 10.

16) toString()

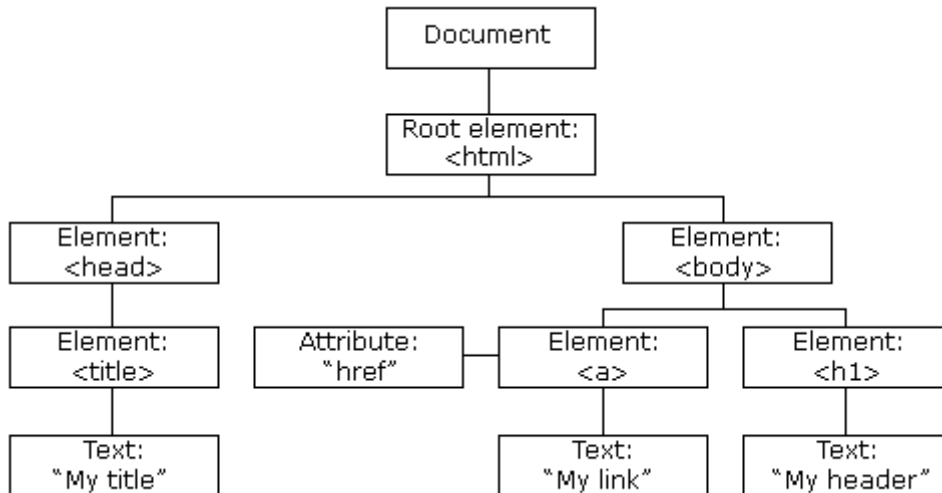
Funkcija `toString()` će druge tipove podataka prebaciti u `string` tip podataka. Svaki tip podatka možemo prebaciti u `string` tip podatka.

```
function stringBroj() {  
    var a = 5.923;  
    a = a.toString();  
    document.write(a+10);  
}
```

Ovaj kod će ispisati „5.92310“. Vidimo da je vrijednost dodijeljena varijabli `a` „5.923“. Ova je vrijednost pohranjena u `string` formatu. Zatim radimo ispis u HTML dokument korištenjem metode `write()` gdje kao parametre predajemo „a+10“. Zato što je „a“ tipa `string` dogodit će se spajanje stringova, a ne matematičko zbrajanje.

17) HTML DOM

HTML DOM je skraćeni za **Document Object Model**. U trenutku kada Internet preglednik učita stranica, on stvara ovakav model naše stranice. Taj model ima ovakvu strukturu:



Slika 16 - HTML DOM struktura²

Korištenjem objektnog modela, JavaScript nam omogućava izradu dinamičnih stranica. Korištenjem JavaScript jezika i mogućnosti koje nam pruža možemo:

- Promijeniti sve elemente na HTML stranici
- Promijeniti sve atribute HTML elemenata na stranici
- Promijeniti CSS stilove na stranici
- Ukloniti postojeće HTML elemente i CSS atribute
- Dodati nove HTML elemente i atribute
- Reagirati na sve postojeće HTML događaje
- Kreirati nove HTML događaje

² http://www.w3schools.com/js/js_htmlDOM.asp

17. a Metode i svojstva

DOM metode smatramo akcijama koje možemo izvesti nad HTML elementima. Svojstva su vrijednosti HTML elementa. **Korištenjem metoda mijenjamo, postavljamo ili uklanjamo svojstva.**

Već smo spomenuli metodu **getElementById()** koja je najčešće korištena metoda za pristup HTML svojstvima. Uz tu metodu, česti ćemo koristiti svojstvo HTML dokumenta **innerHTML** za dohvat ili zamjenu HTML elemenata.

17. b Dohvaćanje HTML elemenata

document.getElementById() metoda pristupa HTML elementu preko njegovog ID svojstva.

document.getElementsByTagName() metoda pristupa HTML elementima po njihovom Tagu

document.getElementsByClassName() metoda pristupa HTML elementima po njihovoj klasi.

17. c Mijenjanje HTML elemenata

element.innerHTML= promjena HTMLa određenog elementa

element.attribute= promjena atributa HTML elementa

element.setAttribute(*attribute, value*) promjena atributa HTML elementa

element.style.property= promjena stila HTML elemnta

17. d Kreiranje i uklanjanje HTML elemenata

document.createElement() stvara HTML element

document.removeChild() uklanja HTML element

document.appendChild() dodaje HTML element

document.replaceChild() zamjenjuje HTML element

document.write() piše direktno u HTML dokument

17. e Dodavanje Event handlera

Document.getElementById(id).onclick=function(){kod koji se izvršava} dodaje Event Handler

18) Rad s kontrolama

18. a Primjer stranice na kojoj se baziraju primjeri

Svi daljnji primjeri će biti

```
<div class="glavniDiv">
  <form id="osobniPodaci">
    Unesi ime: <input type="text" placeholder="Ime" id="ime"> <br>
    Unesi prezime: <input type="text" placeholder="Prezime" id="prezime"> <br>
    Datum rođenja: <input type="date" id="datumRodenja"><br>

    <p>Zaintereseiran/a sam za:</p>
    <input type="checkbox" id="prijateljstvo"> Prijateljstvo <br>
    <input type="checkbox" id="dopisivanje"> Dopisivanje <br>
    <input type="checkbox" id="veza"> Veza <br>

    <p>Ja sam:</p>
    <input type="radio" id="muško"> Muško <br>
    <input type="radio" id="žensko"> Žensko <br>

    <input type="button" value="Izračunaj" id="izracunaj"> <br>
    Pronašli smo kompatibilnih osoba: <input type="text" id="izracun">

  </form>

</div>
```


Unesi ime:

Unesi prezime:

Datum rođenja:

Zaintereseiran/a sam za:

Prijateljstvo

Dopisivanje

Veza

Ja sam:

Muško

Žensko

Pronašli smo kompatibilnih osoba:

Slika 17 - Prikaz stranice za vježbu

CSS je ovdje nebitan no ako želiš da ti stranica izgleda identično, CSS kod slijedi u nastavku.

```
.glavniDiv {
  border: 1px solid black;
  width: 500px;
  height: 700px;
  margin: 0 auto;
}

.footer {
  margin-top: 10px;
  border: 1px solid red;
  width: 500px;
  height: 100px;
  margin: 0 auto;
  margin-top: 20px;
}

.header {
  margin-bottom: 50px;
  margin-top: 50px;
  border: 1px solid blue;
  width: 500px;
  height: 100px;
  margin: 0 auto;
  margin-bottom: 20px;
}

input[type="button"] {
  margin-top: 50px;
  margin-bottom: 10px;
}
```

```
p {
  margin: 10px;
  margin-left: 20px;
  text-decoration: underline;
  font-weight: bold;
}
```

18. b Rad s formama – Dohvaćanje vrijednosti elementa ulaskom u dubinsku strukturu HTML dokumenta

Upišimo u konzolu `document.forms` kako bi dobili popis svih formi unutar HTML dokumenta. Prikazat će se popis formi predstavljen kao polje (*array*). Svaka forma je jedan element tog polja.

```
document.forms
```

```
< undefined
> document.forms
< [ ▼<form id="osobniPodaci"> ]
  "
    Unesi ime: "
    <input type="text" placeholder="Ime" id="ime">
    <br>
    "
    Unesi prezime: "
    <input type="text" placeholder="Prezime" id="prezime">
    <br>
    "
    Datum rođenja: "
    <input type="date" id="datumRodenja">
    <br>
    <p>Zainteresiran/a sam za:</p>
    <input type="checkbox" id="prijateljstvo">
    " Prijateljstvo "
    <br>
    <input type="checkbox" id="dopisivanje">
    " Dopisivanje "
    <br>
    <input type="checkbox" id="veza">
    " Veza "
    <br>
```

Slika 18 - Prikaz forme kao polja

S obzirom da imamo samo jednu formu, dobili smo polje s jednim elementom. Ako želimo dohvatiti točno određenu formu, koristimo indekse za dohvaćanje elemente polja.

```
document.forms[0]
```

```
< undefined
> document.forms[0]
< ▶ <form id="osobniPodaci">...</form>
>
```

Slika 19 - Dohvaćanje točno određene forme

Sada nismo dobili polje u kojem je svaka forma prezentirana kao jedan element nego smo točno odredili kojoj formi pristupamo.

Ako želimo dohvatiti polja/elemente unutar te forme koristimo daljnju dubinsku selekciju. Sada ćemo dobiti popis elemenata unutar te forme. Prikaz elemenata je opet postavljen kao polje gdje je svaki element forme predstavljen kao zaseban element u polju.

```
document.forms[0].elements
```

```
> document.forms[0].elements
< [ <input type="text" placeholder="Ime" id="ime">, <input type="text" placeholder="Prezime"
  <input type="radio" id="muško">, <input type="radio" id="žensko">, <input type="button"
>
```

Slika 20 - Dohvat elemenata unutar forme

Za dohvatiti točno jedan element unutar polja svih elemenata opet koristimo indekse.

```
document.forms[0].elements[0]
```

```
> document.forms[0].elements[0]
< <input type="text" placeholder="Ime" id="ime">
>
```

Slika 21 - Dohvat točno određenog elementa

Konačno, da bi došli do vrijednosti elementa još trebamo dodati svojstvo „value“.

```
document.forms[0].elements[0].value
```

```
> document.forms[0].elements[0].value
< ""
> |
```

Slika 22 - Dohvaćanje vrijednosti elementa unutar forme

Ovakav pristup nije preporučen jer prilikom izmjena dokumenta postoji velika mogućnost da dohvaćanje elemenata neće raditi kako smo zamislili zbog promjene indeksa elemenata.

18. c Dohvaćanje vrijednosti elementa korištenjem getElementById() metode

Metoda `getElementById()` dohvaća element s točno određenom ID vrijednosti neovisno u kojem dijelu HTML dokumenta se nalazi taj dokument. Za primjer ćemo uzeti TextBox kontrolu koja ima ID = „ime“.

```
> document.getElementById("ime")
< <input type="text" placeholder="Ime" id="ime">
> |
```

Slika 23 - Dohvaćanje elementa korištenjem `getElementById()` metode

Ako želimo dohvatiti točnu vrijednost elementa korištenjem ove metode samo još dodajemo da želimo pročitati svojstvo „value“.

```
> document.getElementById("ime").value
< ""
>
```

Slika 24 - Dohvaćanje vrijednosti elementa 1

Drugi način kako možemo doći do vrijednosti je:

```
Document.forms[0].ime.value
```

```
> document.forms[0].ime.value
< ""
> |
```

Slika 25 - Dohvaćanje vrijednosti elementa 2

Ovim pristupom prvo dohvaćamo formu koja se nalazi u polju formi na indeksu nula. Zatim još idemo u dubinu tkao da dohvatimo kontrolu s ID vrijednosti „ime“ i u konačnici određujemo da nas zanima svojstvo „value“ te kontrole.

```
document.getElementById("osobniPodaci").ime.value
```

Ovim pristupom prvo dohvaćamo element čiji je ID „osobniPodaci“ što je u našem slučaju ID forme. Zatim se spuštamo dublje i dohvaćamo kontrolu čiji je ID „ime“ i u konačnici određujemo da dohvaćamo svojstvo „value“ tog elementa.

18. d Rad s TextBox kontrolama

Za ove primjere možemo unijeti testne vrijednosti.

Unesi ime:	Saša
Unesi prezime:	Fajković
Datum rođenja:	dd/mm/yyyy

Slika 26 - Testne vrijednosti u tekstualnim poljima

Za dohvat vrijednosti koristimo svojstvo „value“.

```
document.getElementById(„ime“).value
```

```
> document.getElementById("ime").value  
< "Saša"  
> |
```

Slika 27 - Dohvaćanje vrijednosti iz TextBox kontrole 1

Želimo li u TextBox postaviti vrijednost trebamo navesti svojstvo „value“ tog elementa i korištenjem operatora pridruživanja (simbol =) te vrijednost koju pridružujemo.

```
document.getElementById(„ime“).value = „Marko“;
```

```
> document.getElementById("ime").value = "Marko";  
< "Marko"  
> |
```

Slika 28 - Promjena vrijednosti TextBox kontrole 1

Unesi ime:	Marko
Unesi prezime:	Fajković
Datum rođenja:	dd/mm/yyyy

Slika 29 - Promjena vrijednosti TextBox kontrole 2

18. e Rad s checkbox kontrolama

U našem primjeru imamo tri checkbox kontrole.

Zainterešeiran/a sam za:

- Prijateljstvo
- Dopisivanje
- Veza

Slika 30 - CheckBox kontrole

Za razliku od TextBox kontrola, ovdje nećemo koristiti svojstvo „value“ već svojstvo „checked“ koje označava je li kontrola obilježena ili nije. Rezultat koji ćemo dobiti će biti Boolean tip podatka gdje True predstavlja da je kontrola obilježena, a False da nije obilježena.

```
document.getElementById("prijateljstvo").checked
```

```
> document.getElementById("prijateljstvo").checked  
< false  
> document.getElementById("prijateljstvo").checked  
< true  
> |
```

Slika 31 - Rad s checkbox kontrolama

18. f Rad s radio button kontrolama

Kao i kod checkbox kontrola, ovdje koristimo svojstvo „checked“. Razlika između radio buttona i checkbox kontrola je što u skupini radio buttona možemo imati samo jednu kontrolu selektiranu, dok nam checkbox omogućava selektiranje odabir više kontrola.

U našem primjeru imamo dvije radio button kontrole.

Ja sam:

Muško

Žensko

Slika 32 - Radio button kontrole

Kod koji koristimo za pristup svojstvu „checked“ ovakvih kontrola je:

```
document.getElementById("muško").checked
```

```
> document.getElementById("muško").checked  
< false  
-----  
> document.getElementById("žensko").checked  
< true  
-----  
> |
```

Slika 33 - Rad s radio button kontrolama

19) Popis slika

Slika 1 - Konzola u Google Chrome pregledniku	3
Slika 2 - clear() funkcija za čišćenje konzole	3
Slika 3 - Gumb koji će pokrenuti funkciju ispis1().....	5
Slika 4 - tekst koji je ispisan nakon poziva funkcije ispis1().....	5
Slika 5 - Izgled HTML dokumenta nakon izvršavanja JavaScript skripte.....	5
Slika 6 - Iskočni prozor (Pop-up).....	8
Slika 7 - HTML dokument prije izvršavanja skripte	9
Slika 8 - HTML dokument nakon izvršavanja skripte.....	9
Slika 9 - HTML dokument prije izvršavanja funkcije.....	10
Slika 10 - HTML dokument nakon izvršavanja funkcije	10
Slika 11 - Iskočni prozor.....	39
Slika 12 - Iskočni prozor 1.....	40
Slika 13 - Iskočni prozor 2.....	40
Slika 14 - onmouseout 1	41
Slika 15 - onmouseout 2.....	41
Slika 16 - HTML DOM struktura.....	51
Slika 17 - Prikaz stranice za vježbu	55
Slika 18 - Prikaz forme kao polja.....	57
Slika 19 - Dohvaćanje točno određene forme.....	58
Slika 20 - Dohvat elemenata unutar forme.....	58
Slika 21 - Dohvat točno određenog elementa.....	58
Slika 22 - Dohvaćanje vrijednosti elementa unutar forme.....	58
Slika 23 - Dohvaćanje elementa korištenjem getElementById() metode	59
Slika 24 - Dohvaćanje vrijednosti elementa 1	59
Slika 25 - Dohvaćanje vrijednosti elementa 2	59
Slika 26 - Testne vrijednosti u tekstualnim poljima.....	60
Slika 27 - Dohvaćanje vrijednosti iz TextBox kontrole 1.....	60
Slika 28 - Promjena vrijednosti TextBox kontrole 1	60
Slika 29 - Promjena vrijednosti TextBox kontrole 2	60
Slika 30 - CheckBox kontrole	61
Slika 31 - Rad s checkbox kontrolama	61
Slika 32 - Radio button kontrole.....	62
Slika 33 - Rad s radio button kontrolama	62