JavaScript

INSTITUT VINČA Škola Računara
Ramadani Uzahir

Beograd, 2017

Contents

1 Uvod u JavaScript

Java sa JavaScriptom nema gotovo nikakve veze, ovo su dva potpuno različita programska jezika.

Sličnost ova dva jezika je pre svega u sintaksi (pravila za pisanje naredbi) koja proizilaze iz jezika C. Danas postoji više jezika koji nasledjuju sintaksu C-a, a osim ova dva, poznati su još i C++ i PHP. Druga sličnost je u ideji "programiraj jednom, izvršavaj svuda".

Java je zasnovana na toj ideji - njeni programi se ne kompajliraju za neki odredjeni sistem, već se izvršavaju unutar virtuelne mašine. Pošto je JavaScrvipt zamišljen da se izvršava unutar web browser-a, to važi i za njegove programe. Jednom napisan proogram u JavaSmcript-u izvršavaće se na isti način u svim web browser-ima koji ga podržavaju, a samim tim i na različitim operativnim sistemima i računarima.

Medjutim, medju ovim jezicima postoje značajne razlike. Java je jezik koji podržava klasično objektno-orijentisano programiranje, zasnovano na klasama i nasledjivanju. Ako želite da učite OOP, Java (kao veoma "čist" jezik) je pravi izbnor. Sa druge strane, JavaScrvipt je jezik komji je zasnovan na objektima, ali nenma zaista "pravo" objektno-orijentisano programiranje. U stvari, u JavaScriptu se može programirati i na talkav način, ali jezik je tolliko fleksibilan da možete izabrati stil/paradigmu programiranja koju želite - objektno-orijentisano, funkcionalno, klasično proceduralno...

Druga velika razlika je što JavaScript nije strogo tipiziran jezik. U Javi se tipovi promenljivih moraju odrediti unapred. U JavaScriptu promenljive u jednom trenutku mogu sadržati broj, u drugom tekst, posle toga mogu biti referenca na objekat... Ovo svakako zvuči primamljivo programerima-početnicima, koji ne vole disciplinu koju nameću strogo tipizirani jezici, ali zbog velike slobode, u JavaScriptu je moguće napraviti velike greške koje se kasnije te sko otkrivaju.

1.1 Odakle ime JAVAscript?

Odakle onda sličnost u imenima Java i JavaScript? U vreme kada je Netscape uveo JavaScript u svoj web browser, Java je bila veoma popularna, i proglašena "programskim jezikom Interneta". U Netscape-u su želeli da zgrabe taj talas i "ukradu" malo popularnosti i za svjoj jezik, pa se tako sticao (lažni) utisak da je u pitanju neka pojednostavljena verzija Jave, koja služi za web stranice. Ironično, Java nikada nije zaživela kao jezik za Internet i poptularnost joj je polako splasnula (iako nikada nije prestala da se koristi). Tek mnogo kasnije doživljava svoj veliki bum, kao jezik u kome se programiraju aplikacije za smartfone.

2 Sintaksa u JavaScript-u

Sintaksa predstavlja skup pravila u pisanju komandi nekog programskog jezika. U daljem izlaganju obradićemo osnovna pravila u pisanju JavaScript-a.

2.1 JavaScript i web browser

JavaScript jako jednostavno funkcioniše i interaguje sa web browser-ima. Na browser-e ga uvodim unutar html zaglavlja, učitavanjem celog koda u $< script >$ tagovima ili učitavanjem externog JavaScript fajla.

Primer: Učitavanje JavaScript koda unutar $< script >$ taga (Interni stil).

```
<html>
  <head>
    ...
    <script>
      // deo programa unutar zaglavlja
      function radi()
      {
        console.log("Nesto je uradjeno);
      }
    </script>
    ...
  </head>

  <body>
    ...
    <script>
      // deo programa unutar stranice
      radi();
    </script>
    ...
  </body>
</html>
```

Primer: Učitavanje Preko externog fajla.

```
<html>
  <head>
    ...elementi zaglavlja

    <script src="program.js"></script>

  </head>

  <body>
    ...sadr aj stranice
  </body>
</html>
```

2.2 Promenljive

Promenljiva predstavlja jedan od osnovnih elemenata programskog jezika. Promenljiva predstavlja memorijsku lokaciju na koja sadrži neku vrednost. Ta vrednost se tokom izvršavanja programa može menjati pa odatle i njeno ime "promenljiva". Pošto je JavaScript slabo tipiziran programski jezik ona može u toku svog izvršavanja da menja svoj tip.
Svaka promenljiva se označava engleskim slovima abecede i znakom "_". Promenljiva ne sme da

počinje brojem.
Iako je slabo tipiziran programski jezik ipak postoji deklaracija promenljivih.

```
var promenljiva;         // deklaracija promenljive
var promenljiva = 10;// deklaracija i definicija promenljive(dodela vrednosti)
```

Reč "var" je definisana za označavanje promenljivih.

Primer:

```

var prom;   // deklarisana globalna promenljiva "promenljiva"
var a, b, c;  // deklarisane promenljive a, b i c (isto globalne)
var x = "string"; // deklarisana globalna promenljiva x i dodeljena vrednost string

x = 15; //menjamo vrednost x, ali i njen tip (bila je string, sada je broj)
prom = x * 2; // dodeljujemo vrednost promenljivoj prom

fun();        // izvr avamo funkciju da bi se "izvrsile" naredbe unutar nje

prom = p1; // GRESKA - p1 ne postoji van funkcije
var y = p2;// promenljiva y je deklarisana od pocetka
           // (uvek se prvo "prodje" kroz sve deklaracije
           // ali je definisana (ima vrednost) tek od ove tacke u programu
           // dobija vrednost "Tekst" jer je promenljiva p2 globalno vazeca

function fun()
{
  var a=1, b=2;
  var p1 = x+1;// promenljiva p1 ce biti definisana samo unutar funkcije fun
               // promenljiva x je globalna i samim tim dostupna u svim funkcijama
  p2 = "Tekst";// promenljiva p2 je uvedena bez iskaza var - bice globalno dostupna
               // izbegavajte ovakvu praksu - program ce biti mnogo razumljiviji
}
```

2.3 Funkcije

U prograiranju se dešava da nam je potrebno izvršavanje jedne te iste funkcionalnosti na više mesta.

Funkcije su potprogrami koji se izvršavaju na poziv i oni vraćaju neku vrednost, medjutim postoje i funkcije koje izvrsavaju svoje zadatke ali njihov zadatak nije vraćanje vrednosti.

Funkciju moramo prvo deklarisati i deklarisati njene ulazne parametre.

```
 function imeFunkcije(ulazni parametri)
{
  ...naredbe unutar funkcije
  return rezultat;
}
```

Primer1. Deklarisaćemo jednu funkciju koja sabira dva broja:

```
function zbir(a, b)
{
    var rezultat = a + b;
}
```

Treba zapamtiti da sve sto je deklarisano unutar funkcije ono važi samo u toj funkciji i samo se tu može koristiti. Takodje naredba return služi za vraćanje vrednosti iz funkcije. Svaka funkcija moze da vrati samo jednu vrednost, osim ako ta vrednost nije niz koji ima vise elemenata. Naredba return se uvek pise na kraju funkcije.

Primer 2: Deklaracija funkcije

```
function fun()  // deklaracija funkcije bez parametara
{
  return x + y; // vraca rezultat x i y moraju biti globalne promenljive
}

// deklaracija funkcije stepen
function stepen(osnova, step)  // funkcija ima dva parametra
{
  var rez = 1;                      // rez dobija pocetnu vrednost 1
  for (var i=0; i<step; i++) {      // ciklus se "vrti" onoliko puta koliki je step
    rez *= osnova;                  // toliko puta mnozimo rez sa osnovom
  }
  // tako smo dobili vrednost osnova na stepen step
  return rez;       // vra amo rez kao rezultat funkcije
}

var x = 12;
var y = 8;

fun();  // poziv funkcije - iako funkcija vraca vrednost,
        // mi nista ne radimo sa njom
var s = stepen(x, 3); // pozivamo funkciju stepen za dva parametra,
          // a rezultat smestamo u s
console.log(s);       // ispisujemo vrednost iz promenljive s
```

3 Operatori

Aritmetičke operacije su nam svima dobro poznate. To su osnovne operacije, bez kojih se ni ne može. Ovde ćemo videti kako se koriste u JavaScriptu (a samim tim i u drugim programskim jezicima). Operandi su brojevi, a u JavaScriptu se ne pravi razlika izmedju celih i racionalnih brojeva.

Operator	Primer
Umanjenje	x–
Uvečanje	x++
Prefiksno uvećanje	++x
Prefiksno umanjenje	–x
Množenje	x*y
Deljenje	x/y
Ostatak celobrojnog deljenja	x%y
Sabiranje	x+y
Oduzimanje	x-y

4 Kontrola toka

Grananje je osnovna kontrolna struktura. Ideja je da se u programu postavi pitanje vezano za neko trenutno stanje, a onda zavisno od odgovora, izvršava se jedan ili drugi deo programa. Na taj način se vrši odlučivanje o daljem toku programa.

Zašto dolazi do ovakvih situacija pa nam je grananje potrebno? Jednostavno, zato što naš program treba da bude "spreman" za razne situacije. To se najčešće dešava jer ne možemo da znamo kakve ulazne vrednosti (aktivnosti) će zadati korisnik, već samo da predvidimo razne mogućnosti.

4.1 if else grananje

Unutar naredbe if zadaje se izraz (unutar zagrade) na osnovu koga se odredjuje dalji tok programa. Uobičajeno je da izraz kao rezultat vraća logičku vrednost - tačno ili netančno (true ili false). Ako je rezultat izraza "tačno", izvršava se naredba koja sledi ključnu reč if. U suprotnom slučaju, izvršava se naredba koja sledi posle else. Ako ne postoji ništa što bi se izvršilo u "suprotnom" slučaju, else deo je moguće izostaviti.

```
if ([izraz]) {
  [naredbe]
}
else {
  [naredbe]
}
```

Primer:

```
if ( (i>=0) && (i<=9) ) {
  x += i;
  y -= i;
}
else {
  x = i*2;
  y = i*4;
}
```

Primer:

```
var x = 5;
var y = 10;
var z = 0;
if (x>y) {
  if (x>20)
    z = y;
}
else
  z = x;
console.log ("Rezultat je " + z);
```

Sa običnim if-else grananjem, u programu monžemo razdvojiti samo dve mogućnosti: ako je uslov zadovoljen i ako nije. Sada ćemo se upoznati sa jednom konstrukcijom koja nam omogućava da napravimo delove programa za više mogučnosti.

U suštini ideja je jednostavna. Proverimo jedan uslov, pa ako je zadovoljen, izvršava se zadati deo programa. Ako nije, proveravamo sledeći uslov i tako sve dok ne stignemo do dela programa koji se izvršava ako ni jedan uslov nije bio zadovoljen.

```
if ([izraz]) {
  [naredbe]
}
else if ([izraz]) {
  [naredbe]
}
else if ([izraz]) {
  [naredbe]
}
...
else {
  [naredbe]
}
```

Primer:

```
if (ocena < 6) {
  console.log("Lose");
}
else if (ocena < 8) {
  console.log("Dobro");
}
else if (ocena < 10) {
  console.log("Solidno");
}
else {
  console.log("Odlicno");
```

```
}
```

4.2 switch grananje

Višestruko grananje nam je potrebno kada postoji više mogućnosti. Ako se svaka mogućnost ostvaruje u slučaju da je zadovoljen neki uslov jednakosti, možemo koristiti komandu switch. Najpre se zadaje komanda switch, posle koje se obavezno navodi izraz unutar zagrade. Ovo je izraz čiju ćemo vrednost proveravati. Onda se navodi blok (vitičaste zagrade). Unutar bloka se navode case izrazi za koje se proverava da li im je vrednost jednaka switch izrazu.

Znači, navede se reč case, zatim sam izraz i dvotačka. Posle toga se navode naredbe koje se izvršavaju u slučaju da je vrednost switch izraza jednaka vrednosti tog case izraza. Ove naredbe ne moraju biti u posebnom bloku (nisu potrebne dodatne vitičaste zagrade).

Ako posle tih naredbi dolazi novi case, poslednja naredba mora biti naredba break. Ova naredba će naterati program da iskoči iz switch strukture. Ako je propustimo, jednostavno ćć nastaviti da se izvršavaju naredbe za sledeći case, što najčešće nije ono što smo želeli da se desi.

Za iste naredbe moguće je navesti i vi se case izraza.Ako je potrebno da se odredjene naredbe izvrše u slučaju da vrednost switch izraza nije jednaka ni jednom od navedenih case izraza, na kraju se navodi reč default (sa obaveznom dvotačkom) i naredbe koje se izvršavraju. Za poslednju grupu naredbi nije potrebno navoditi komandu break.

```
switch ([izraz]) {
  case [izraz]:
    [naredbe]
  break;
  case [izraz]:
  case [izraz]:
  ...
    [naredbe]
  break;
  default:
  ...
    [naredbe]
}
```

Primer:

```
switch (ocena) {
  case 5:
    console.log("Lo e");
  break;
  case 6:
  case 7:
    console.log("Dobro");
  break;
```

```
  case 8:
  case 9:
    console.log("Solidno");
  break;
  case 10:
    console.log("Odli no");
  break;
}
```

Primer:

```
switch (mesec) {
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    brojDana = 31;
    daniTxt = "tridesetjedan";
  break;
  case 4:
  case 6:
  case 9:
  case 11:
    brojDana = 30;
    daniTxt = "trideset";
  break;
  default:
    brojDana = 28;
    daniTxt = "dvadesetosam";
}
```

4.3 Ciklusi "Petlje"

U svakom malo složenijem programu, potrebno je da imamo neke naredbe koje se izvršavaju više puta. Ove naredbe se izvršavaju kroz takozvani "ciklus", odnosno "petlju".

Svaki "prolazak" kroz petlju naziva se "iteracija".Najjednostavniji ciklusi su oni koji obezbedjuju da se naredbe izvršavaju dok je neki uslov zadovoljen. Ovo su tzv "uslovni ciklusi". Zamislite da imate naredbu if koja proverava neki uslov, a da se naredbe unutar if-a izvršavaju više puta - sve dok je taj uslov zadovoljen. Ovi ciklusi funkcionišu upravo na taj način, s tim što se umesto if, koristi naredba while.

```
while ([izraz]) {
  [naredba];
  [naredba];
  ...
}
```

Primer While ciklusa:

```
var a = 0;
var b = 1;
console.log(a);
console.log(b);

```

```
var novi = 0;

while (novi < 1000) {
  novi = a+b;
  console.log(novi);
  a = b;
  b = novi;
}
```

Brojački ciklus je posebna varijanta uslovnog ciklusa. Ovakav ciklus postoji u velikom broju programskih jezika i obično podrazumeva "brojanje" kroz neku promenljivu od početne do krajnje vrednosti. Koliko se različitih vrednosti dodeli toj promenljivoj, toliko puta se izvrši i sam ciklus. U jezicima kao što su Basic ili Pascal, ovi ciklusi imaju veoma strogu formu. U jezicima koji proizilaze iz jezika C (kao što je i JavaScript), ovaj ciklus daje mnogo veću slobodu programeru, ali zato početnici mogu lako napraviti grešku i "zaglaviti" program u beskonačnom ciklusu.

```
for ([inicijalizacija]; [uslov za iteraciju]; [kraj iteracije]) {
  [naredba];
  [naredba];
  ...
}
```

Primer For ciklusa:

```
// na ulasku u ciklus i se postavlja na 0
for (var i=0; i<10; i++) {
  // na po etku svakog ponavljanja se uvek proverava da li je i<10,
  // ako jeste, izvr avaju se naredbe

  console.log("Vrednost broja a je " + i);

  // na kraju svake iteracije i se pove ava za 1 (i++)
}
```

5 Nizovi

JavaScript, kao i svaki ozbiljan programski jezik, može raditi sa različitim strukturama podataka. Verovatno najpoznatija (i najčešće korišćena) struktura je niz. Kao što verovatno do sada znamo, niz je struktura u kojoj se pod istim nazivom može zabeležiti više vrednosti, kojima se pristupa preko rednog broja, odnosno "indeksa".

Niz se može kreirati suštinski na dva načina - preko konstruktora objekta Array (ili "stručno": kreiranjem instance klase Array) i korišćenjem literala ("nizovne konstante", odnosno "literala").

```
 a = new Array()
```

```
a = []
```

Primer:

```
var a = [];  // prazan niz
var sestrici = ["Marko", "Milena", "Nikola"];  // niz stringova

var m1 = [[25,34], [18,51], [92,44]];  // matrica dimenzije 3x2
var m1 = [[17,49,38], [88,23,39]];  // matrica dimenzije 2x3

var obj = [{}, {}, {}, {}];  // niz od   etiri   objekta

var a1 = new Array(100);  // niz od 100 nedefinisanih elemenata, numerisanih od
   0..99
var a2 = new Array(100, 200, 50);  // niz od 3 elemenata (brojevi 100, 200 i 50)
```

Primer:

```
var m = [[25,34], [18,51], [92,44]];  // matrica dimenzije 3x2

  // u konzoli ispisuje jedan po jedan element matrice
  for (var i=0; i<3; i++) {
    for (var j=0; j<2; j++) {
      console.log( a[i][j] );
    }
  }
```

Uprethodnim primerima možemo primetiti da se niz i objekat poistovećuju. U principu u JavaScriptu niz predstavlja objekat koji ima svoja svojstva i metode.

Operator	Primer
Array.isArray(obj)	Vraća TRUE ako je zadati objekat JavaScript niz.
a.length	Broj elemenata niza
a.push(E)	Dodaje jedan ili više novih elemenata na kraj niza.
a.pop()	Uklanja poslednji elemenat niza i vraća ga kao rezultat.
a.unshift(E)	Dodaje jedan ili više novih elemenata na početak niza.
a.shift()	Uklanja prvi elemenat niza i vraća ga kao rezultat.
a.splice(P,N,E1,E2,...)	Uklanja N elemenata počev od indeksa P i umesto njih ubacuje eleme
a.indexOf(V,P)	U nizu pronalazi prvi elemenat sa vrednošću V, počev od indeksa P i vra
a.forEach(F)	Za svaki elemenat niza samo poziva callback funkciju F.
a.map(F)	Za svaki elemenat niza poziva funkciju F i vraća niz koji se sastoji iz vraćenih

Primer: U prikazanom primeru koristimo svojstvo length da bismo prošli kroz sve članove niza. Ciklus se "vrti" počev od nultog elementa pa do poslednjeg koji je manji od broja članova niza (a.length).

```
var a = [5, 10, 3, 6, 22];

  for (var i=0; i<a.length; i++) {
    console.log(i + ": " + a[i]);
  }
```

Primer: Provera da li je neki objekat tipa Array

```
var a = [5, 10, 3, 6, 22];

  if (a.length) {   // u ovom slu aju  e  raditi, ali izbegavajte taj na in
    // ...
  }
```

```
6
7   // ovo je ispravno
8   console.log(Array.isArray(a));  // true
9
10  console.log(Array.isArray([]));        // true
11  console.log(Array.isArray("Pera"));    // false
12  console.log(Array.isArray(["Pera"]));  // true
13
14  // alternativno, za jako stare web    itae
15  if( Object.prototype.toString.call( niz ) === '[object Array]' ) {
16    // ...
17  }
```

Primer: Dodavanje elemenata u niz.

```
1   var a = [];  // novi niz sa 0 elemenata
2
3   a[3] = "Pera";
4   a[8] = "Mika";
5   // niz ima 9 elemenata: [-, -, -, "Pera", -, -, -, -, "Mika"]
6
7   a.unshift(" ika ");
8   // niz ima 10 elemenata: [" ika ", -, -, -, "Pera", -, -, -, -, "Mika"]
9
10  a.push("Joca", "Pe a");
11  // niz ima 12 elemenata: [" ika ", -, -, -, "Pera", -, -, -, -, "Mika", "Joca", "
      Pe a"]
```

Primer: Uklanjanje elemenata iz niza.

```
1  var a = ["Pera", "Mika", "Joca", " ika "];  // novi niz sa 4 elemenata
2
3   var x = a.pop();  // x = " ika "
4   // a = ["Pera", "Mika", "Joca"]
5
6   var y = a.shift();  // x = "Pera"
7   // a = ["Mika", "Joca"]
```

Primer: Pretrage po nizu

```
1  var a = ["Pera", "Mika", "Pera", " ika ", "Ana", "Pera"];
2 var ind = [];
3
4 var val = "Pera";
5 var p = a.indexOf(val);
6 while (p > -1) {
7   ind.push(p);
8   p = a.indexOf(val, p+1);
9 }
10 console.log(ind);   // [0, 2, 5]
```

Primer: Sortiranje niza.

```
1 var imena = ["Borko", "Anastazija", "Ana", "Joca"];
2 imena.sort();
3 console.log(imena);
4 // Rezultat: ["Ana", "Anastazija", "Borko", "Joca"]
```

```
5
6  var brojevi = [1, 5, 100, 234, 10, 22];
7  brojevi.sort();
8  console.log(brojevi);
9  // Rezultat: [1, 10, 100, 22, 234, 5]
10
11 console.log(niz[0]+niz[1]);
```

6 Funkcije

U programiranju često dolazimo u situaciju da nam se deo programa ponavlja na više različitih mesta. Idealno bi bidlo da taj deo programa nekako izdvojimo i samo ga pozivamo kada nam zatreba. Ovaj problem je veoma dobro poznat u programiranju i rešava se korišćenjem potprograma.

Generalno, postoje dve vrste potprograma - procedure i funkcije. Procedure su klasični potprogrami, dok funkcije predstavljaju potprograme koje vraćaju rezultat. Pošto vraćanje rezultata u JavaScript funkcijama niije obavezno, procedure postaju potpuno nepotrebne, tako da se u JavaScriptu koriste samo funkcije.

Da bismo koristili funkciju, moramo je deklarisati. Deklaracija obavezno sadrži ključnu reč function. Pasle nje se može navesti naziv funkcije, a onda obavezno zagrade unutar kojih se može navesti lista formalnih parametara. Konačno, obavezno se navode vitičaste zagrade unutar kojih se može navesti telo funkcije, odnosno njene naredbe.

```
1 function nazivFunkcije(formalni parametri)
2 {
3   ...naredbe unutar funkcije
4   return rezultat;
5 }
```

Primer funkcije koja sabira dva broja

```
1  function zbir(a, b)
2 {
3     var rezultat = a + b;
4 }
5
6 zbir(3,7);
```

```
1  function zbir(a, b)
2 {
3     var rezultat = a + b;
4     return rezultat;
5 }
```

Kompletan primer:

```
1  function fun()    // deklaracija funkcije bez parametara
2 {
```

```
  return x + y;  // vra a rezultat x i y moraju biti globalne promenljive jer ih nema u funkciji
}

// deklaracija funkcije stepen
function stepen(osnova, step)  // funkcija ima dva parametra
{
  var rez = 1;                      // rez dobija po etnu vrednost 1
  for (var i=0; i<step; i++) {      // ciklus se "vrti" onoliko puta koliki je step
    rez *= osnova;                  // toliko puta mno imo rez sa osnovom
  }
  // tako smo dobili vrednost osnova na stepen step
  return rez;      // vra amo rez kao rezultat funkcije
}

var x = 12;
var y = 8;

fun();                // poziv funkcije – iako funkcija vra a vrednost, mi ni ta ne radimo sa njom
var s = stepen(x, 3); // pozivamo funkciju stepen za dva parametra, a rezultat sme tamo u promenljivu s
console.log(s);       // ispisujemo vrednost iz promenljive s
```

7 Objektno orijentisano programiranje

Objekat je kombinacija promenljivih i funkcija. Zamislite da postoji promenljiva koja u sebi sadrži podpromenljive, a kao dodatak i funkcije. Jedna od najvažnijih mogućnosti programiranja sa objektima je što i te "podpromenljive" takodje mogu da budu objekti. Suštinski, objekat je jedna "skup" podataka i programa koji rade sa tim poodacima.

Promenljive koje pripadaju objektu nazivaju se svojstva (properties), a funkcije objekta nazivaju se metodi (methods).

Bilo kom elementu objekta se pristupa tako što navedmo objekat i onda svojstvo ili metod, odvojen tačkom. Svojstvo objekta se koristi kao i bilo koja druga promenljiva.

```
objekat.svojstvo = vrednost;
objekat.metod();
```

Primer:

```
var tekst = "JavaScript ima objekte";

var x = tekst.length;                // duzina stringa
var promena = tekst.toUpperCase();   // string pretvoren u velika slova

window.alert("Duzina stringa je " + x);
console.log(promena);
```

7.1 Kreiranje objekta

U JavaScriptu se stalno susrečemo sa objektima. Ako ste pažljivo čitali sve tekstove do sada, videli ste da JapvaScript ima neke ugradjene objekte koje možemo koristiti u programiranju. Tako smo naučili da radimo sa objektima Math, String, Array, Date i sl. koji nam prilično "olakšavaju život". Navedeni objekti su nam pre svrega važni zbog svojih metoda (funkcija koaje pripadaju objektu, eng. methods), ali objekat takodje može sadrzati i podatke, odnosno svojstva (ili "atribute", eng. properties).

Znači objekat je jedan "skup" podataka i funkcija, tj. svojstava i metoda. Ovde ̌ cemo naučiti kako da napravimo sopstvene objekte u JavaScriptu. Za razliku od "pravih" objektno-orijentisanih jezika, u JavaScriptu se ne kreiraju klase za objekte, već se objekat izgradjuje prostim navodjvenjem njegovih svojstava i metoda.

Objektni literal se kreira pomoću vitičastih zagrada. Nemojte mešati ovako definisan objekat i telo funkcije - to su dve sasvim različite stvari. Kreiranje "praznog" objekta je moguće na sledeći način:

```
var obj = {};
obj.svojstvo = vrednost;
obj.metod = funkcija;
obj.podobjekat = {};
obj.podobjekat.svojstvo = vrednost;
```

Isto tako, možemo definisati ceo objekat, sve sa svojstvima i metodima, u jednom koraku:

```
 var obj = {
  svojstvo: vrednost,
  metod: funkcija,
  ...
};
```

Hajde da kreiramo jedan objekat:

```
var obj = {};

obj.ime = "Mika";
obj.prezime = "Mikic";

obj.punoIme = function() {
  console.log(this.ime + " " + this.prezime);
};

//...

obj.punoIme();    // pozivamo metod punoIme() - "Mika Mikic"

obj.prezime = "Zikic";  // menjamo vrednost svojstva prezime

obj.punoIme();    // opet pozivamo punoIme() - sada je rezultat "Mika Zikic"
```

Dakle, kreirali smo objekat obj bez ijednog svojstva. Onda smo mu naknadno dodali svojstva ime i prezime koja sadrže string vrednosti. Dodali smo i svojstvo punoIme koje je referenca na anonimnu

funkciju. Praktično, ovo svojstvo je metod.

Da bi matod pristupio svojstvima "sopstvenog" objekta, koristimo objekat this. Sećate se, da je ovaj objekat "kontekst" za koji se poziva funkcija, odnosno objekat iz koga je funkcija pozvana. Kada pozovemo funkciju kao metod objekta, samim tim taj objekat postaje kontekst funkcije, odnosno unutar metoda, this pokazuje na objekat kome metod pripada.

Na kraju vidimo šta se dešava kada pozovemo funkciju (povezuje ime i prezime i ispisuje rezultat u konzoli web browser-a), i šta se dešava kada menjamo vrednosti svojstava.

Pogledajte sada kako kreiramo isti objekat od malopre, ali navodjenjem liste svojstava unutar literala.

```
var obj = {
 ime: 'Mika',
 prezime: "Mikic",
 punoIme: function() {
   console.log (this.ime + " " + this.prezime);
 }
};
```

8 DOM

Web DOM je druga strana programiranja klijenta. JavaScript je jezik koji ima velike mogućnosti, ali glavnu snagu pokazuje kada se primeni nad elementima HTML dokumenta. DOM (Document Object Model) su objekti koje web browser pruža JavaScriptu. Zahvaljujući tome, naš program može pristjupati web browseru i samom HTML dokumentu.

Drugim rečima, DOM je programski interfejs ka HTML dokumentu. DOM predstavlja HTML dokumenat (ali i web browser) u obliku JavaScript objekta. To znači da pomoću JavaScripta možemo menjati svaki deo dokumenta, ili ga čak kreirati. Web DOM je tokom godina evoluirao u pravcu odvajanja od JavaScripta. Danas je DOM nezavisni objektni model, koji se može koristiti iz svackog programskog jezika zasnovanog na objektima. Ipak, JavaScript je jezik koji je tradicionalno podržan u svim web browser-ima i naši primeri i predstavljanje mogućxnosti DOM-a će biti pomoću ovog jezika.

Da bismo koristili DOM u JavaScriptu, ne moramo da zadajemo nikakve specijalne pozive ili naredbe. Jednostavno koristimo objekte DOM-a kao i sve druge objekte JavaScripta. Kako budemo napredovali kroz DOM, primetićemo da različiti HTML objekti implementiraju iste DOM interfejse - imaju iste metode i svojstva. Primer 1: Ispisuje naslov taba trenutnog dokumenta:

```
<!DOCTYPE html>
<html>
  <head>
  <title>Primer 1</title>
  </head>
```

```
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Naslov je: " + document.title;
</script>

</body>
</html>
```

Primer 2: Prikazuje putanju trenutnog dokuments

```
 <!DOCTYPE html>
<html>\tableofcontents
<body>
<p>URL putanja je: <br><span id="demo"></span>.</p>

<script>
document.getElementById("demo").innerHTML = document.URL
</script>

</body>
</html>
```

Primer 3: U zavisnosti od akcije sakriva i prikazuje tekst.

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">Ja sam teks toji treba da se sakrije i prikaze</p>

<input type="button" value="Sakrij tekst"
onclick="document.getElementById('p1').style.visibility='hidden'">

<input type="button" value="Prikazi tekst"
onclick="document.getElementById('p1').style.visibility='visible'">

</body>
</html>
```

Primer 4: Pronalazi vrednost svih input elemenata iz forme.

```
 <!DOCTYPE html>
<html>
<body>

<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname" value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br><br>
  <input type="submit" value="Submit">
</form>

<p>Klikni "Prebroji" za prikaz vrednosti svakog elementa.</p>

```

```
<button onclick="myFunction()">Prebroji</button>

<p id="demo"></p>

<script>
function myFunction() {
    var x = document.getElementById("frm1");
    var text = "";
    var i;
    for (i = 0; i < x.length ;i++) {
        text += x.elements[i].value + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Primer 5: Skidanje elementa iz select liste.

```
 <!DOCTYPE html>
<html>
<head>
<script>
function removeOption() {
    var x = document.getElementById("mySelect");
    x.remove(x.selectedIndex);
}
</script>
</head>
<body>

<form>
<select id="mySelect">
  <option>Apple</option>
  <option>Pear</option>
  <option>Banana</option>
  <option>Orange</option>
</select>
<input type="button" onclick="removeOption()" value="Remove the selected option">
</form>

</body>
</html>
```

9 Projekat 1

Napravićemo jedan mali programč ić koji će nam služiti za pretvaranje Faranhajte u Celzijuse.

U HTML dokumentu koga ćemo nazvati *index.html* potrebno je da kreiramo jedno input polje i dugme.

```
<h1>Konverzija F to C </h1>

<form>
  <input type="text" class="js-one" value='F' />
  <input type="button" class="js-button" value="Konverzija"/>

</form>
```

u JavaScript dokumentu koga ćemo nazvati *racun.js* ćemo ispisati sledeći kod:

```
var one = document.querySelector('.js-one');
var button = document.querySelector('.js-button');

button.addEventListener('click', function() {
  var first = parseInt(one.value);
  var result = (first - 32) / 1.8;
  if (first !== NaN) {
    alert(result);
  } else {
    alert(" Morate uneti vrednost.");
  }
});
```

10 Projekat 2

Program koji manipuliše CSS-om nekog dokumenta.

```
<html>
    <head>
        <title>JavaScript osobine CSS-a</title>
        <meta charset="UTF-8"/>
    </head>
    <body>

        <span>Pozadinska boja:</span>
        <select id="background">
          <option value="Red">Red</option>
          <option value="Green">Green</option>
          <option value="Blue">Blue</option>
        </select>

        <span>Širina:</span>
        <select id="width">
          <option value="100px">100px</option>
          <option value="200px">200px</option>
          <option value="300px">300px</option>
        </select>
```

```
        <span>Visina:</span>
        <select id="height">
          <option value="100px">100px</option>
          <option value="200px">200px</option>
          <option value="300px">300px</option>
        </select>

        <br/><br/>

        <div id="content" style="background:red; width:100px; height:100px;"></div>

        <script src="script.js"></script>

    </body>
</html>

var arraySelect = document.getElementsByTagName('select');

var element = document.getElementById('content');

function dropdownStyles(){

    var style = this.id;
    var value = this.value;

    element.style[style] = value;

}

for( var i = 0; i < arraySelect.length; i++ ){

    arraySelect[i].addEventListener( 'change', dropdownStyles );

}
```