JavaScript Uvod

Autori:
Milan Segedinac
Goran Savić

1. Programski kod u veb stranici

Pored HTML elemenata i CSS stilova, veb stranica može da sadrži i *programski kod* koji će se izvršavati u pregledaču. Ovaj programski kod piše se u programskom jeziku JavaScript.

Iako imaju slične nazive, JavaScript i Java su veoma različiti programski jezici. Sličnosti između ova dva jezika završavaju se na sličnoj sintaksi – i jedan i drugi jezik imaju sintaksu inspirisanu programskim jezicima C i C++. Obzirom da smo programski jezik Java obradili vrlo detaljno moći ćemo da koncepte programskog jezika JavaScript uporedimo sa analognim konceptima Jave.

Prva važna karakteristika JavaScript programskog koda je da je to *skript* (*script*). Skript je programski kod koji se izvršava u već postojećim softverskim aplikacijama. Skriptovi se razlikuju od same aplikacije u tome što su, tipično, pisani u različitim programskim jezicima (skript jezicima), često se distribuiraju kao programski kod (a ne binarne datoteke) i, u velikom broju sličajeva, mogu da ih menjaju korisnici. Za skript jezike je karakteristično da su interpretirani umesto kompajlirani. U slučaju kompajliranih programskih jezika, kompajler prevodi izvorni kod u izvršni kod, a program se izvršava pokretanjem izvršnog koda. U interpretiranim programskim jezicima, posebnom programu (interpreteru) se daje izvorni kod i intepreter ga direktno izvršava. Skriptovi se gotovo uvek ugrađuju u aplikacije u kojima se izvršavaju. Neki od primera skript jezika su VBA (Visual Basic for Applications, programski jezik u kojem se pišu Microsoft Office makroi), AutoLISP (programski jezik za skriptovanje AutoCAD softverskog alata) i JavaScript – programski jezik za skriptovanje pregledača. Naravno, važno je napomenuti da JavaScript ima i širu primenu. Na primer, JavaScript je veoma popularan jezik za razvoj serverskih aplikacija. Mi ćemo se baviti jedino klijentskim JavaScript programima, odnosno JavaScript programima koji se izvršavaju u pregledaču.

Činjenica da se korisnik često nije svestan izvršavanja JavaScript programa u njegovom pregledaču nameće seriju ograničenja:

- JavaScript program ne može da čita i piše fajlove na klijentskom
- JavaScript program ne može da izvrši druge programe na klijentskom računaru
- JavaScript program ne može da uspostavi vezu ka drugom računaru u mreži, osim da preuzme HTML stranicu ili da pošalje email.

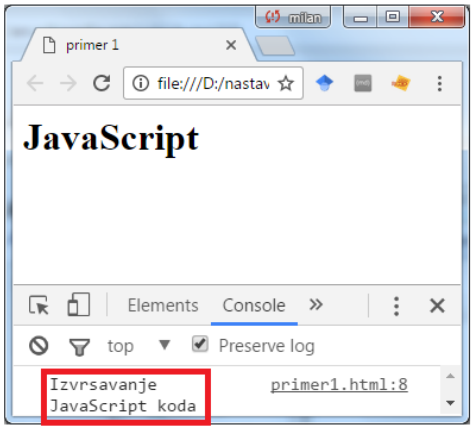
Dakle, klijentski JavaScript program koji se izvršava u okviru HTML stranice ima uticaj samo na elemente te stranice i na pregledač.

Element <script>

JavaScript programski kod se u HTML stranicu ugrađuje u element `<script>` tako što je sadržaj ovog elementa JavaScript programski kod. Prilikom učitavanja HTML stranice, kada se u obradi stranice naiđe na <script> element, zaustavlja se dalja obrada stranice i programski kod iz tog `<script>` elementa se izvršava.

Korišćenje `<script>` elementa prikazano je na listingu ispod. Funkcija `console.log('tekst')` ispisuje tekst u konzolu, kao što je metoda `System.out.println("tekst")` radila u programskom jeziku Java.

```
<html>
<head>
    <title>primer 1</title>
</head>
<body>
    <h1>JavaScript</h1>
    <script>
        console.log('Izvrsavanje JavaScript koda');
    </script>
</body>
</html>
```



Slika - `<script>`

JavaScript programski kod može biti zadat i u zasebnoj datoteci. I u tom slučaju se koristi `<script>` element, a putanja do .js datoteke zadaje se pomoću atributa `src`. Uvoz eksterne datoteke izgleda kao `<script src="skript.js"></script>`. Time se specificira preuzimanje JavaScript fajla u brauzeru.

Činjenica da se JavaScript izvorni programski kod ne kompajlira, već se direktno izvršava omogućuje nam da program izvršavamo zadajući komandu po komandu, što je režim koji se najčešće koristi u toku razvoja programa za probanje komandi „na živo“. Setimo se da Java programe nismo mogli da izvršavamo na taj način: morali smo provo da napišemo ceo program, a zatim da ga prevedemo u bajtkod pomoću `javac` programa i tek onda izvršimo ceo program. Da bismo direktno zadali komandu JavaScript interpreteru potrebno je da pristupimo JavaScript konzoli (u većini pregledača, na primer Chrome, prečica za ovo je F12) i unesemo komandu. Takođe, u programskom jeziku Java programski kod je morao da bude „upakovan“ u klasu da bi se izvršio. U JavaScriptu komande programskog koda ne moraju da budu „upakovane“ ni u šta, već mogu da se zadaju direktno. Ukoliko bismo u konzolu napisali `console.log("hello world");` i izvršili ovu liniju koda, rezultat bi bio ispis `"hello world"`.

2. Tipovi podataka

U programskom jeziku JavaScript postoje dve vrste tipova podataka: *primitivni tipovi podataka* i *objekti*. O objektima će biti više reči u narednoj lekciji. U ovoj lekciji biće obrađeni primitivni tipovi podataka. Podsetimo se da primitivni tipovi podataka predstavljaju osnovne gradivne blokove od kojih se grade objekti.

Brojevi u programskom jeziku JavaScript se predstavljaju primitivnim tipom podataka *Number*. U JavaScriptu ne postoji poseban tip celih brojeva: svi brojevi su razlomljeni u duploj preciznosti (*double*) i u opsegu od $-(2^{53}-1)$ do $2^{53}-1$. Obzirom da celi brojevi nisu nisu zaseban tip, sami moramo da pazimo kada su nam eksplicitno potrebni celi brojevi, na primer prilikom indeksiranja.

Obzirom da su i JavaScript i Java programski jezici koji su sintaksu razvili po uzoru na programski jezik C, numerički operatori (+,-,*,/,%,++,--) i operatori dodele (=,+=,-=,*=,/=) u JavaScriptu su identični kao u programskom jeziku Java.

Logičke vrednosti predstavljaju se primitivnim tipom *Boolean*. Ovaj tip ima samo dve moguće vrednosti: *true* i *false*. Logički operatori (&&, ||, !) su identični kao u programskom jeziku Java. Takođe, relacioni operatori (==, >, >=, <, >=, !=) su slični kao u programskom jeziku Java.

Tekstualne vrednosti reprezentovane su primitivnim tipom *String*. Za razliku od programskog jezika Java, programski jezik JavaScript nema primitivni tip za predstavljanje jednog karaktera. Karakteri se u JavaScriptu predstavljaju stringovima koji imaju samo jedan element. U JavaScriptu su podržane osnovne operacije nad stringovima:

- *Indeksiranje* – počeviši od elementa na poziciji 0, na primer `"abcd"[2]` će vratiti vrednost `"c"`
- *Isecanje* – zadavanjem indeksa početnog i krajnjeg indeksa, na primer `"abcd".substring(1,3)` će vratiti vrednost `"bc"`. Obratite pažnju da su preuzeti elementi na pozicijama 1 i 2, odnosno da se preuzima podstring od početnog do krajnjeg indeksa bez elementa na krajnjem indeksu.
- *Pronalazak podstringa* – metodom `includes`. Na primer `"abcd".includes("bc")` će vratiti true jer je string `"bc"` sadržan u stringu `"abcd"`.
- *Deljenje stringa* – metoda `split` će vratiti listu podstringova zadatog stringa podeljenog po zadatom delimiteru. Na primer `"pera peric; marko markovic; steva stevic".split(";")` će vratiti listu `["pera peric", " marko markovic", " steva stevic"]`
- *Spajanje stringova* – konkatenacija se realizuje operatorom `+`. Na primer `"abc"+"def"` će vratiti `"abcdef"`.
- *Uklanjanje blanko karaktera sa početka i kraja stringa* – metoda `trim`. Na primer `"  abcd  ".trim()` će vratiti `"abcd"`.
- *Pretvaranje u velika/mala slova* – metode `toUpperCase` i `toLowerCase`. Na primer `"aBcD".toUpperCase()` će vratiti `"ABCD"`.

Pre nego što dobije vrednost, varijabla u JavaScriptu **nije definisana**. Za predstavljanje vrednosti nedefinisanih varijabli koristi se tip `Undefined`. Ovaj tip ima semo jednu vrednost: `undefined`.

Poseban tip vrednosti u JavaScriptu je `Null` i ovaj tip ima samo jednu vrednost: `null`. Može se desiti da je **varijabla definisana**, ali da **ne znamo koja je njena vrednost**. U tom slučaju, ova varijabla ima vrednost `null`. Razlika između `undefined` i `null` deluje suptilno – `undefined` je varijabla koja nije definisana, a `null` je definisana varijabla kojoj nije dodeljena vrednost – ali videćemo da je razlika između ova dva tipa vrlo značajna.

Dinamički tipovi

U programskom jeziku Java videli smo da prilikom deklaracija varijable navodimo i tip varijable. Tako smo, na primer, mogli deklarisati varijablu `int i`, što je značilo da će varijabla `i` moći da primi jedino celobrojne vrednosti. Varijabla je mogla da promeni vrednost, ali nije mogla da promeni tip. Ovo svojstvo programskog jezika zove se *statički tipovi*.

Programski jezik JavaScript ima *dinamičke tipove* što znači da varijabla nema fiksiran tip. Jednoj varijabli možemo da dodelimo celobrojnu vrednost, a zatim da joj dodelimo string vrednost, kao što je prikazano u listingu ispod.

```
>> x = 123
>> console.log("x:",x,"x.typeof x:",typeof x)
<< x: 123 x.typeof x: number
>> console.log("x:",x,"x.typeof x:",typeof x)
<< x: abc x.typeof x: string
```

Listing – dinamički tipovi. >> označava unos u konzolu, a >> predstavlja rezultat izvršavanja

Provera tipa u JavaScriptu se obavlja operatorom `typeof`, kao što je prikazano na listingu iznad. Obratite pažnju da, iako je varijabla `x` promenila tip, vrednost dodeljena varijabli je uvek imala tip. U prvom slučaju to je bio `Number`, u drugom `String`.

Dinamički tipovi u JavaScriptu su jedno od svojstava koje ovaj jezik čine suštinski različitim od programskog jezika Java. Na primer, prilikom definisanja funkcije, parametri funkcije nemaju tip, tako da nije moguće razlikovati funkcije po tipu parametara.

Važno je da napomenemo da postoje jezici izvedeni iz programskog jezika JavaScript koji podržavaju statičke tipove. Jedan od njih je TypeScript.

Obzirom na dinamičke tipove, relacioni operator == proverava identitet vrednosti neuzimajući u obzir tipove (vršeći implicitnu konverziju). Relacioni operator === (tri jednako, nije štamparska greška) poredi vrednosti uzimajući u obzir i tip. Primer je dat listingom ispod. Analogno su definisani i operatori razlike != i !==.

```
>> 1 == "1"
<< true
>> 1 === "1"
<< false
```

Listing – relacioni operatori == i ===

3. Kontrola toka

Programski jezik JavaScript podržava konstrukte kontrole toka slične onima u programskom jeziku Java.

Uslovno izvršavanje dela programa

U programskom jeziku JavaScript postoje dva konstrukta za uslovno izvršavanje delova programa: if-else i switch-case. Pošto su veoma slični kao u programskom jeziku Java, biće prikazani ukratko pre svega radi podsećanja.

If-else

Uslovno izvršavanje dela programa realizuje se if-else konstruktom. Primer uslovnog izvršavanja dela programa dat je listingom ispod.

```
>> x = 5;
>> if(x<3){
  console.log("x<3");
}else if(x<8){
  console.log("x>=3 and x<8");
}else{
  console.log("x>=8");
}
<< x>=3 and x<8
```

Listing – if-else uslovno izvršavanje

Switch-case

Ukoliko postoji velik broj uslova, uslovno izvršavanje dela programa moguće je realizovati i pomoću `case` konstrukta. Primer je dat u listingu ispod.

```
>> x = 5;
>> switch(x){
    case 4:
      console.log('4');
    case 5:
       console.log('5');
    case 6:
       console.log('6');
}
<< 5
<< 6
```

Listing – switch-case uslovno izvršavanje

Obratite pažnju da su ispisane vrednosti 5 i 6. Prilikom izvršavanja switch-a biće izvšrne i sve case kaluzule switch konstrukta nakon one koja je tačna. Kao i u programskom jeziku Java, postoje i break i default.

Petlje

U programskom jeziku JavaScript postoje tri konstrukta za zadavanje ciklusa – for petlja, while petlja i do-while petlja. Zbog sličnosti sa programskim jezikom Java, i oni će biti prikazani ukratko, radi podsećanja.

For petlja

For petlja namenjena je za slučajeve u kojima znamo koliko puta će se telo petlje izvršiti. U zaglavlju petlje obavlja se inicijalizacija, i zadaju uslov završetka petlje i korekcija (izmena vrednosti koja određuje završetak petlje). Primer for petlje dat je listingom ispod.

```
>> for(i=0;i<5;i+=1){
  console.log("i:",i);
}
<< i: 0
<< i: 1
<< i: 2
```

```
<< i: 3
<< i: 4
```

Listing – for petlja

While petlja

Ova vrsta petlji namenjena je za programski kod koji se ponavlja sve dok je ispunjen uslov. U zaglavlju petlje zadaje se samo uslov izvršavanja tela petlje. Korekcija koja će na kraju dovesti do prekida petlje treba da se ostvari u telu petlje – inače imamo beskonačnu petlju. Primer while petlje dat je listingom ispod.

```
>> x = 3;
while(x<8){
  console.log("x:",x);
  x+=2;
}
<< x: 3
<< x: 5
<< x: 7
```

Listing – while petlja

Do-while petlja

Ova vrsta petlje je veoma slična while petlji. Jedina razlika je u tome što se telo petlje izvršava makar jednom. Setimo se da je telo while petlje moglo da se uopšte ne izvrši – ako uslov nije bio ispunjen pre prvog izvršavanja. Primer do-while petlje dat je listingom ispod.

```
>> x = 3;
do{
  console.log("x:",x);
  x+=2;
}while(x<8);
<< x: 3
<< x: 5
<< x: 7
```

Listing – do-while petlja

Komande skoka

Kao i u programskom jeziku Java, i u programskom jeziku JavaScript postoje dve komande skoka: break i continue. Break prekida izvršavanje trenutnog ciklusa petlje i nastavlja izvršavanje programa nakon petlje, a continue prekida izvršavanje trenutnog ciklusa petlje i prelazi na novi ciklus te petlje.

4. Objekti

JavaScript je objektno orijentisani programski jezik iako ne podržava zadavanje klasa[1]. Objekte je moguće napraviti bez posredovanja klasa, direktnim navođenjem svojstava i vrednosti. U listingu ispod dat je prikaz jednog jednostavnog JavaScript objekta koji predstavlja podatke o osobi.

[1] Od verzije ES6 JavaScript ima Class konstrukt, ali to je samo skraćeni naziv za pisanje specijalnih (konstruktorskih) funkcija.

```
osoba = {
  ime:"Pera",
  prezime:"Peric",
  godiste:1980
}
```

Listing – JavaScript objekat

Varijabla osoba je objekat koji za svojstvo `ime` ima string vrednost `"Pera"`, za svojstvo `prezime` ima string vrednost `"Peric"`, a za svojstvo `godiste` ima brojčanu vrednost `1980`. Podsetimo se da smo se u programskom jeziku Java susreli sa sličnom strukturom podataka u kojoj smo vrednosti čuvali pod ključevima – to je bila mapa. Većina programskih jezika strukture podataka deli u dve gurpe: one u kojima se elementima pristupa po *indeksu* (broju) i one u kojima se elementima pristupa po *ključu* (nazivu). Strukture podataka analogne mapi tako postoje i u drugim programskim jezicima: rečnik (dictionary) u programskom jeziku Python, asocijativni niz (associative array) u programskom jeziku C. Međutim, kolekcije u kojima se elementima pristupa po ključu imaju toliko važnu ulogu u programskom jeziku JavaScript, da su svi objekti u stvari ovakve kolekcije.

U programskom jeziku JavaScript vrednosti objekta je moguće pristupiti na dva načina:

- []-notacijom – kao kada indeksiramo element u nizu u uglastim zagradama navodimo naziv svojstva za koje hoćemo da preuzmemo element, na primer `osoba[godiste]`.
- .-notacijom – kao kada pristupamo polju u objektu, naziv svojstva navodimo nakon tačke, na primer `osoba.godiste`

Vrednost svojstva je moguće izmeniti prostim postavljanjem nove vrednosti za to svojstvo. Svojstvo iz objekta uklanja se operatorom `delete`. Izmena vrednosti prikazana je na listingu ispod.

```
>> osoba = {
ime:"Pera",
prezime:"Peric",
godiste:1984
}

>> osoba.ime //pristup
<< "Pera"
>> osoba.ime="Marko"//izmena
>> osoba
<< { godiste: 1984, ime: "Marko", prezime: "Peric" }
>> delete osoba.prezime //brisanje
>> osoba
<< {ime: "Marko", godiste: 1984}
```

Listing – Operacije nad objektom

Ništa nas ne sprečava da vrednost svojstva u objektu bude drugi objekat. Primer pristupa ugnježdenom objektu prikazan je listingom ispod.

```
>> osoba = {
ime:"Pera",
prezime:"Peric",
adresa:{
  broj:12,
  ulica:{
```

```
    naziv:"Fruskogorska",
    grad:{
       naziv:"Novi Sad",
       drzava:"Srbija"
     }
   }
}
}

>> osoba.adresa.ulica.grad.drzava
<< "Srbija"
```

Listing – pristup ugnježdenoj vrednosti

Nizovi

U programskom jeziku JavaScript nizovi se zadaju navođenjem vrednosti između uglastih zagrada. Vrednostima se pristupa po indeksu, tako što se indeks zadaje između uglastih zagrada. U programskom jeziku Java prilikom deklarisanja niza navodili smo i tip vrednosti koje će biti smeštane u niz. U celobrojni niz nije bilo moguće smestiti string. Ovakve kolekcije (u kojima sve vrednosti moraju da budu istog tipa) zovu se *homogene* kolekcije. Nizovi u JavaScriptu su *heterogene kolekcije* tako da je u njih moguće smeštati vrednosti različitih tipova.

Vrednosti mogu da se menjaju, postavljanjem nove vrednosti na zadatu poziciju u nizu. Vrednosti se uklanjaju iz niza pomoću operatora `delete`. U listingu ispod prikazane su osnovne operacije nad nizovima.

```
>> l = [1,2,3,4]//definisanje niza
>> l
<< [1, 2, 3, 4]
>> l[2]="abc" // izmena vrednosti niza
>> l[2]
<< "abc"
>> l
<< [1, 2, "abc", 4]
>> delete l[2]//brisanje elementa niza
>> l
<< [1, 2, undefined × 1, 4]
>> l[6]=5
>> l
<< [1, 2, undefined × 1, 4, undefined × 2, 5]
```

Listing – osnovne operacije nad nizovima

Obratite pažnju šta se desilo kada smo obrisali element niza na poziciji 2: na mestu tog elementa u nizu je ostala „rupa“. Takođe, kada smo dodali novi element na poziciju 6, u nizu je ostala „rupa“ od dva elementa. Ovo je neobična posledica činjenice da JavaScript nizove tretira kao specijalne *objekte koji kao ključeve imaju cele brojeve*. Kada smo uklonili element na poziciji 2, samo smo svojstvo 2 postavili na `undefined`. Kada smo postavili element na poziciji 6, svojstva 4 i 5 su ostala `undefined`.

Za složenije operacije nad nizovima koristi se metoda `splice`. Prvi parametar ove metode označava na kom indeksu operacija treba da se izvrši. Drugi parametar predstavlja broj elemenata koji će biti uklonjeni. Ukoliko je vrednost ovog parametra 0, neće biti uklonjen ni jedan element. Nakon toga

sledi proizvoljno mnogo objekata koji će biti umetnuti na zadatu poziciju u nizu. Primer korišćenja `splice` metode prikazan je na listingu ispod.

```
>> l=[1,2,3,4]
>> l.splice(2,1)//brisanje jednog elementa sa indeksa 2
>> l
<< [1, 2, 4]
>> l.splice(2,0,'a','b','c')//dodavanja elemenata 'a','b' i 'c' na
                            //indeks 2
>> l
<< [1, 2, "a", "b", "c", 4]
```

Listing – splice metoda

Funkcije

Deklaracija funkcije u JavaScriptu je oblika

```
function nazivFunkcije(parametar1,parametar2,...){
  komanda 1;
  komanda 2;
  ...
}
```

Listing – deklaracija funkcije

Prilikom poziva funkcije potrebno je navesti njen naziv i konkretne argumente. Već sada uočavamo nekoliko razlika u odnosu na programski jezik Java:

- U programskom jeziku Java bilo je moguće zadati jedino metode (funkcije koje su vezane bilo za klasu (static), bilo za objekat). U programskom jeziku JavaScript funkcije mogu da se zadaju nezavisno od objekata.
- Prilikom deklarisanja metode u programskom jeziku Java navodili smo tipove vrednosti parametara. Obzirom da JavaScript ima dinamičke tipove, tipovi parametara se ne navode.
- U JavaScriptu se ne navodi ni tip povratne vrednosti funkcije.

U programskom jeziku JavaScript funkcije su *građani prve klase* (*functions as first-class citizens*, termin koji je uveo *Christopher Strachey* sredinom šezdesetih). To znači da su funkcije ravrnopravne sa drugim objektima: *mogu da se proslede kao argumenti prilikom poziva drugih funkcija*, *mogu da budu vraćene iz funkcije*, *mogu da se dodele varijabli* i *smeste u strukturu podataka*. Ova karakteristika programskog jezika JavaScript otvara ogroman prostor za različite tehnike programiranja. Sada ćemo pažnju posvetiti samo činjenici da *funkciju možemo da prosledimo drugoj funkciji kao argument*. Primer prosleđivanja funkcije kao argumenta dat je listingom ispod.

```
>> function process(f){
      console.log("preprocess");
      f();
      console.log("postprocess");
   }

>> function sayHelloWorld(){
      console.log("Hello world");
```

```
  }

>> process(sayHelloWorld)
<< preprocess
<< Hello world
<< postprocess
```

Listing – funkcija kao argument

Čest slučaju u programiranju je da isti programski kod trebamo da izvršimo neposredno pre ili neposredno nakon izvršavanja već napisanih funkcija. To je slučaj kada, na primer, trebamo da u log unesemo informaciju o svakom pozivu funkcije. U listingu iznad prikazan je jednostavan primer kako bi se takva funkcionalnost mogla implementirati u JavaScriptu. Funkcija `process` prima jedan parametar – funkciju koja treba da se izvrši. Neposredno pre izvršavanja prosleđene funkcije ispisuje se tekst `"preprocess"` pozivom `console.log("preprocess");`. Zatim se poziva prosleđena funkcija `f` i poziva `console.log("postprocess");`. U ovom primeru kreirali smo funkciju `sayHelloWorld` i prosledili je kao parametar funkciji `process`.