JavaScript funkcije

Autori:
Milan Segedinac
Goran Savić

1. Funkcije

U programskom jeziku JavaScript, funkcije su objekti. Kao i svi drugi objekti i funkcije mogu da imaju atribute i vrednosti, imaju prototip (`Function.prototype`), ali imaju i posebna svojstva za predstavljanje konteksta i koda funkcije. Pored toga, činjenica da se funkcije tretiraju kao objekti omogućuje nam da:

- Varijabla može da primi funkciju kao vrednost
- Funkcije mogu da se smeštaju u kolekcije i druge objekte
- Mogu da se proslede drugim funkcijama kao parametri
- Mogu da budu vraćene iz drugih funkcija kao povratna vrednost
- Mogu da imaju svoje atribute sa vrednostima (koje opet mogu da budu i nove funkcije)

Specifičnost funkcija u odnosu na druge objekte je u tome što funkcije mogu da budu pozvane (*invoked*).

Funkcija u programskom jeziku može biti zadata na dva načina:

- Pomoću deklaracije funkcije
- Pomoću funkcijskog izraza

Deklaracija funkcije

U programskom jeziku JavaScript deklaracija funkcije ima oblik dat listingom ispod:

```
function Identifier ( FormalParameterList opt ){ FunctionBody }
```

Listing – deklaracija funkcije

Deklaracija funkcije počinje rezervisanom rečju `function`, zatim sledi identifikator (naziv) funkcije, pa (opciono) lista formalnih parametara u zagradi i, nakon toga, telo funkcije u vitičastim zagradama.

Funkcijska deklaracija

Slično kao i u slučaju deklaracije varijabli, deklaracije funkcija su hoistovane (prebačene na sam početak skripta), tako da je deklarisane funkcije u kodu moguće koristiti pre nego što je data njihova deklaracija. U listingu ispod je dat primer hoistovanja deklarisane funkcije.

```
>> f();

     function f(){
       console.log('Hello world');
     }
<< Hello world
```

Listing – hoistovanje deklarisane funkcije

Činjenica da deklarisane funkcije mogu da se koristi u programskom kodu koji prethodi završetku njihove implementacije određuje i najvažniju primenu deklarisanih funkcija u JavaScriptu: za pisanje rekurzivnih funkcija. Primer rekurzivne funkcije za računanje faktoriela dat je listingom ispod.

```
function faktoriel(x){

  if(x === 0){
    return 1;
  }
  else{
    return x*faktoriel(x-1);
  }
}
```

Listing – rekurzivno računanje faktoriela

Činjenica da smo deklarisali funkciju `faktoriel` omogućuje nam da je pozovemo u njenom telu, odnosno da rekurzivno definišemo računanje faktoriela.

Funkcijski izraz

Zadavanje funkcije pomoću funkcijskog izraza razlikuje se od deklarisanja funkcije jedino u tome što je *identifikator opcioni*. Pomoću funkcijskog izraza moguće je zadati funkciju bez naziva i odmah je dodeliti varijabli. Primer zadavanja funkcije pomoću funkcijskog izraza dat je listingom ispod.

```
>> var x = function(){

     console.log('hello world!');
   }
   x()
<< hello world!
```

Listing – funkcijski izraz

Varijabli x dodelili smo kao vrednost funkciju koja u konzolu ispisuje string `'hello world'`. Zatim smo pozvali `x()`.

Varijabla `x` je hoistovana, odnosno na samom početku skripta postavljeno je da je `undefined`. Evaluacija funkcijskog izraza dešava se u vremenu izvršavanja i evaluirana vrednost se dodeljuje varijabli `x`. Kada se evaluira telo funkcije neimenovanog funkcijskog izraza varijabli `x` još uvek nije dodeljena vrednost (još uvek je undefined), tako da funkcija ne može da pozove sebe samu.

Korišćenje (neimenovanih) funkcijskih izraza u programskom jeziku JavaScript je veoma često, čak češće nego korišćenje deklarisanih funkcija.

Poziv funkcije

U programskom jeziku JavaScript postoje četiri načina invokacije (poziva) funkcije. Funkcija se može pozvati kao

- Metoda
- Funkcija u užem smislu reči
- Konstruktor
- Pomoću `apply` metode

Ključna razlika u načinima invokacije funkcija je u interpretiranju objekta `this`. Dok u programskom jeziku Java `this` uvek referencira objekat nad kojim se izvršava metoda, u programskom jeziku JavaScript, `this` može da referencira i druge objekte.

Metoda

Činjenica da je funkcija u JavaScriptu objekat omogućuje nam, između ostalog, i da funkciju postavimo kao vrednost atributa drugog objekta. Takve funkcije se nazivaju *metode*. Prilikom poziva metode `this` referencira objekat kom metoda pripada. U listingu ispod dat je primer metode.

```
>> var brojac = {

     vrednost: 0,
     uvecaj: function () {
       this.vrednost += 1;
     }
   };
   brojac.uvecaj();
<< console.log(brojac.vrednost);
```

Listing - metoda

Definisali smo objekat `brojac` koji ima atribute `vrednost` i `uvecaj`. Atribut `vrednost` je inicijalno `0`, a atribut `uvecaj` je funkcija koja `this.vrednost` uvećava za `1`. Obzirom da se funkcija `uvecaj` poziva nad objektom `brojac`, `this` će biti upravo taj objekat, tako da će uvećavanje rezultovati inkrementiranjem `vrednosti`.

Ovako definisan objekat sa metodama veoma podseća na objekte kreirane u programskom jeziku Java. Ipak, obratite pažnju na vema važnu činjenicu: sve tako definisane metode su javne! U programskom jeziku JavaScript postoje načini i da se definišu privatne metode, o čemu će biti reči u narednim lekcijama.

Funkcija u užem smislu reči

Prilikom poziva funkcije koja nije metoda objekta, `this` referencira globalni objekat. U pregledačima je to `Window` objekat.

Konstruktorska funkcija

Kada se funkcija poziva kao konstruktor, odnosno kada pozivu funkcije prethodi rezervisana reč `new`, this referencira novokreirani objekat. Listingom ispod dat je primer ovakvog poziva funkcije.

```
>> var Brojac = function(startnaVrednost){

     this.vrednost = startnaVrednost;
   }
   Brojac.prototype.uvecaj=function(){
     this.vrednost += 1;
   }
   var a = new Brojac(5);
   a.uvecaj();
   console.log(a.vrednost);
<< 6
```

Listing - `this` vrednost u konstruktorskoj funkciji

Kreirali smo konstruktorsku funkciju `Brojac` koja novokreiranom objektu postavi startnu vrednost. Zatim smo u `prototype` atribut postavili funkciju `uvecaj` koja inkrementira vrednost. Kreirali smo novi objekat `a` pozivom konstruktora `Brojac` sa startnom vrednošću `5`. Poziv funkcije `uvecaj` je inkrementirao `this.vrednost`, što je u ovom slučaju kreirani objekat `a`, pa je ispis `a.vrednost` u konzolu `6`.

Metoda `apply()`

U programskom jeziku JavaScript moguće je i proslediti objekat this prilikom poziva funkcije korišćenjem `apply` metode. Primer je dat listingom ispod.

```
>> var brojac = {

     vrednost: 0
   };
   var uvecaj = function (x){
     this.vrednost += x;
   };
   uvecaj.apply(brojac, [5]);
   console.log(brojac.vrednost);
<< 5
```

Listing - `apply` metoda

Kreirali smo objekat `brojac` i za atribut vrednost postavili `0`. Zatim smo kreirali funkciju `uvecaj` koja prima parametar `x` i `this.vrednost` uvećava za `x`. Zatim smo pozvali metodu `apply` funkcije `uvecaj`. Prvi parametar ove metode je `this` objekat, a drugi parametar lista argumenata koji se prosleđuju funkciji nad kojom se poziva.

Opseg vidljivosti varijabli

JavaScript je jezik čija sintaksa je inspirisana programskim jezikom C, slično kao i programski jezik Java. Međutim, pored ove, relativno površne sličnosti, postoje duboke razlike između ova dva jezika. Već smo videli jednu veliku razliku: dok programski jezik Java ima statičke tipove, programski jezik JavaScript ima dinamičke tipove. Druga velika razlika je opseg vidljivosti varijabli.

U programskom jeziku Java opseg vidljivosti varijable je *blok koda* u kom je ta varijabla deklarisana. Kažemo da u programskom jeziku Java varijable imaju *blokovski opseg vidljivosti*. Međutim, u programskom jeziku JavaScript varijabli je moguće pristupiti i izvan bloka koda u kom je deklarisana, kao što je prikazano listingom ispod.

```
>> if(true){

     var x = 5;
   }
   console.log(x);

<< 5
```

Listing – pristup varijabli izvan bloka u kom je definisana

U stvari, u programskom jeziku JavaScript, varijabli je moguće pristupiti u okviru *čitave funkcije* u kojoj je varijabla deklarisana. Kažemo da varijable u programskom jeziku JavaScript imaju *funkcijski opseg vidljivosti*. Shodno tome, varijabli ne možemo pristupiti izvan funkcije u kojoj je deklarisana, kao što vidimo na listingu ispod.

```
>> function f(){

     var x = 5;
     console.log('x u funkciji: ',x);
   }
   f();
   console.log('x izvan funkcije: ',x);
<< x u funkciji:  5
<< Uncaught ReferenceError: x is not defined
    at <anonymous>:6:34
```

Listing – funkcijski opseg

Definisali smo funkciju `f` i u njoj deklarisali varijablu `x` kojoj smo dodelili vrednost `5`. U funkciji smo ispisali vrednost varijable u konzolu. Zatim smo pozvali funkciju i nakon toga pokušali da pristupimo varijabli `x` izvan funkcije. Naravno, varijabla nije dostupna pa je pristup rezultovao izuzetkom.

Ugnježdene funkcije

Varijable su deklarišu ključnom rečju `var`. Kada nazivu varijable u funkciji prethodi ključna reč `var`, kreira se lokalna varijabla vidljiva jedino u toj funkciji. Svaki sledeći put kada pristupamo varijabli navodimo samo njen naziv. Ali šta bi se desilo da koristimo varijablu, a da je nismo deklarisali?

Kada u funkciji koristimo varijablu koja nije deklarisana, pretpostavlja se da ta varijabla postoji u širem opsegu. Ukoliko funkcija nije ugnježdena u neku drugu funkciju nego je zadata direktno u skriptu, taj širi opseg je *globalni opseg*. (U pregledaču je to objekat `window`.) Ukoliko pokušamo da pristupimo varijabli koja ne postoji u globalnom opsegu, *biće deklarisana u globalnom opsegu*, što je prikazano listingom ispod.

```
>> function f(){

     x = 5;
     console.log('x u funkciji: ',x);
   }
   f();
   console.log('x izvan funkcije: ',x);
<< x u funkciji:  5
<< x izvan funkcije:  5
```

Listing – korišćenje nedeklarisane varijable

Na listingu iznad koristili smo varijablu `x` koja nije deklarisana. Obzirom da varijabla nije pronađena u globalnom opsegu, deklarisana je u globalnom opsegu, pa smo mogli da joj pristupimo i izvan funkcije. Ovo je čest izvor ozbiljnih bugova: varijabla za koju mislimo da je dostupna samo u našoj funkciji dostupna je globalno i neki drugi deo programskog koda može da je izmeni. Stoga je važno voditi računa o deklarisanju lokalnih varijabli funkcija.

Šta bi se desilo da je posmatrana funkcija bila ugnježdena u drugu funkciju? Primer je dat listingom ispod.

```
>> function f1(){

     var x = 5;
     function  f(){
       console.log('x u ugnjezdenoj funkciji: ',x);
     }
     f();
     console.log('x u spoljasnjoj funkciji: ',x);
   }
   f1();
   console.log(x);
<< x u ugnjezdenoj funkciji:  5
<< x u spoljasnjoj funkciji:  5
<< Uncaught ReferenceError: x is not defined
    at <anonymous>:10:13
```

Listing – pristup lokalnoj varijabli spoljašnje funkcije iz ugnježdene funkcije

Varijabla `x` je deklarisana u spoljašnjoj funkciji `f1`. Kao što vidimo na listingu, ta varijabla je dostupna u ugnježdenoj funkciji `f`. Naravno, varijabla `x` nije dostupna izvan funkcije `f1`. Zato *kažemo da funkcija ima pristup lokalnim varijablama spoljašnje funkcije*.

Obzirom da JavaScript ima *first-class* podršku za funkcije, funkcija može biti i povratna vrednost druge funkcije. Šta se u tom slučaju dešava sa opsegom vidljivosti varijabli? Primer je dat listingom ispod.

```
>> function f1(){

     var x = 5;
     return function(){
       console.log(x);
     }
   }
   var f = f1();
   f();
<< 5
```

Listing – pristup varijablama spoljašnje funkcije

Funkcija `f1` definiše varijablu `x` dodelivši joj vrednost `5` i vrati funkciju koja ispisuje varijablu `x`. Za tu ugnježdenu funkciju varijabla `x` dostupna je kao lokalna varijabla spoljašnje funkcije. Nakon definisanja funkcije `f1` pozovemo funkciju `f1` i povratnu vrednost dodelimo varijabli `f`. (Povratna vrednost funkcije `f1` je njena ugnježdena funkcija.) Iako je završeno izvršavanje funkcije `f1`, njena ugnježdena funkcija dodeljena varijabli `f` i dalje ima pristup lokalnim varijablama funkcije `f1`. Zato kažemo da *funkcija ima pristup lokalnim varijablama spoljašnje funkcije čak i nakon što je završeno izvršavanje spoljašnje funkcije*. Ovo svojstvo se zove *zatvaranje opsega vidljivosti varijabli* (*closure*) i omogućuje implementaciju brojnih dizajn šablona, kao što je definisanje privatnih svojstava objekata, o čemu će biti više reči u narednoj lekciji.