JavaScript – callbacks, prototypes…

JavaScript – Callbacks

- U JavaScriptu, funkcije su „first-class“ objekti, odnosno funkcije su tipa **Object** i mogu se koristiti kao bilo koji drugi objekat, pošto su zapravo one objekti sami
- Mogu se smestiti u varijablama,
- proslediti kao argumenti funkcijama,
- kreirati u funkciji,
- vratiti iz funkcije

JavaScript – Callbacks

- Zato što su funkcije **first-class** objekti, možemo proslediti funkciju kao argument nekoj drugoj funkciji i kasnije izvršiti tu prosleđenu funkciju
- Ili je čak možemo vratiti ti da se izvrši kasnije
- Te dve pomenute stavke su suština korišćenja **callback** funkcija u JavaScriptu

Callback funkcije

- Verovatno najraširenija funkcionalna programska tehnika u JavaScriptu
- „Izvedene“ iz programske paradigme funkcionalnog programiranja koje specificira korišćenje funkcija kao argumenata
- Mogu se naći u skoro svakom delu JavaScript i JQuery koda

Callback funkcije

- Callback (**high-order**) funkcije su funkcije koje se prosleđuju drugim funkcijama (nazovimo tu drugu funkciju **drugaFunkcija**) kao parametar
- Onda se callback poziva (ili izvršava) u toj **drugojFunkciji**
- Callback funkcija je u suštini šablon (**pattern**) koji predstavlja rešenje za neki problem
- Stoga korišćenje **callback** funkcije je poznato i kao **callback pattern**

Primer

```
var friends = [“Pera”, “Mika”, “Zika”];

friends.forEach(function (eachName, index) {
    console.log(index + 1 + “.” + eachName);
});
```

Primer

- U primeru je ilustrovana tipična upotreba **callback** funkcije
- **forEach** metodi prosleđena anonimna funkcija (funkcija bez imena) kao parametar

Kako callback funkcije rade?

- Funkcije možemo proslediti kao varijable, možemo ih vratiti u funkcijama i možemo ih koristiti u drugim funkcijama
- Kada prosledimo **callback** funkciju kao argument drugoj funkciji, mi zapravo prosleđujemo definiciju funkcije
- Ne izvršavamo tu funkciju u parametru
- A pošto sadržavajuća funkcija ima callback funkciju kao parametar, može izvršiti **callback** bilo kada

Imenovane ili Anonimne funkcije kao callback

- U dosadašnjim primerima su korištene anonimne funkcije koju su bile definisane u parametrima sadržavajuće funkcije
- Jedan od čestih šablona za korištenje callback funkcija
- Još jedan česti šablon koristi imenova funkcije i prosleđuje ime te funkcije u parametru

Primer

```
var allUserData = [];


function log(userData) {
   if(typeof userData === “string”){
      console.log(userData);
   }else if (typeof userData === “object”){
      console.log(item + “: “+ userData[item]);
   }
   }
```

Primer

```
// Funkcija koja prima dva parametra, drugi je
callback funkcija


function getInput(options, callback) {
    allUserData.push(options);
    callback(options);
}
```

Primer

```
// Kada pozivamo getInput funkciju, prosledjujemo log kao parameter
// Dakle log ce biti funkcija koja je “called back” iz getInput funkcije


getInput({name:”Pera”, specialty:”Java”}, log);
```

Objekti - nastavak

- Objekti se kreiraju na jedan od tri načina
- Objekti imaju svojstva (properties) i funkcije
- Svojstvima se pristupa:
  - objekat.svostvo
  - objekat[svojstvo]
- Funkcije se pozivaju imenom:
  - objekat.funkcija()

Objekti - nastavak

- Korišćenjem objekt literala:

```
var osoba = {ime:"pera", prezime:"peric"};
```

- Korišćenjem ključne reči **new**:

```
var osoba = new Object();
osoba.ime="pera";
osoba.prezime="peric";
```

- Korišćenjem konstruktora:

```
function osoba(ime, prezime) {
   this.ime=ime;
   this.prezime=prezime;
}
var osoba = new osoba("pera", "peric");
```

Objekti - nastavak

- **this** je objekat koji trenutno poseduje JavaScript kod
  - unutar metode se odnosi na objekat koji poseduje metodu
  - unutar konstruktora se odnosi na sam objekat koji se konstruiše
- Ugrađeni konstruktori:
  - Object(), String(), Number(), Boolean(), Array(), RegExp(), Function(), Date()

Objekti - nastavak

```
var x1 = {};           // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 = false;        // new primitive boolean
var x5 = [];           // new array object
var x6 = /()/          // new regexp object
var x7 = function(){};  // new function object
```

Reference

- Sve promenljive su reference

```
var x = new osoba("pera", "peric");
var y = x; // x i y ukazuju na isti objekat
y.ime = "mika"; // menja ime i u x i u y
```

Svojstva

- Svojstva se interno čuvaju kao parovi (ime, vrednost)
- Svojstva se mogu dodavati, brisati, čitati i pisati
- Primer dodavanja svojstva:

  osoba.visina = 180;
- Primer uklanjanja svojstva:

  delete osoba.visina;
- Primer iteriranja kroz svojstva:

```
for (x in osoba) {
   txt += osoba[x];
}
```

PRIMER: 1. Svojstva

Svojstva

- Privatna svojstva su vidljiva samo iz opsega objekta
- Deklarišu se uz pomoć ključne reči **var**:

```
function Osoba2(ime, prezime) {
     this.ime = ime;
     this.prezime = prezime;
     var privatna = 3;
     ...
}
var o = new Osoba2("pera", "peric");
console.log(o.privatna); // štampa undefined
```

Funkcije

- Funkcije se mogu dodavati, brisati i pozivati
- Dodavanje funkcije:

```
var osoba = {ime:"pera", prezime:"peric"};
osoba.stampaj = function() {
     document.write(this.ime + ", " + this.prezime);
}
osoba.stampaj();
```

- Brisanje funkcije:

```
delete osoba.stampaj
```

Funkcije

- Privatne funkcije su vidljive samo iz opsega objekta
- Deklarišu se uz pomoć ključne reči **var**:

```
function Osoba2(ime, prezime) {
	this.ime = ime; this.prezime = prezime;
	var privatna = 3;
	var stampaj2 = function() {
		document.write("Osoba2.stampaj2: " + this.ime + ", " + this.prezime + ", privatna: " + privatna + "<br />");
	};
};
var o = new Osoba2("pera", "peric");
o.stampaj2(); // baca izuzetak
```

Funkcije - objasnjenje

- **For in** petlja će izlistati konkretne vrednosti svih svojstava i funkcija:

```
for (x in o) {
     document.write(o[x] + "=>> vrsta: " +
               typeof(o[x]) +  "<br />");
}
```

- Funkcija **Object.keys()** lista sva svojstva i funkcije, a ne njihove vrednosti
- Funkcija **Object.getOwnPropertyNames()** lista sva svojstva i funkcije, ali samo za zadati objekat, a ne i od njegovih "nadklasa" (ako ih ima)

Funkcije - objasnjenje

```
// samo od ovog objekta, ne i od roditelja,
// ako ih ima
// var list = Object.getOwnPropertyNames(o);
var list = Object.keys(o);
for (x in list) {
    document.write(list[x] + "=>> vrsta: "
        + typeof(o[list[x]]) +  "<br />");
}
```

Funkcije - objasnjenje

- Funkcija se može pozvati direktno:

ime_funkcije(...);

- ili ovako:

ime_funkcije.call(thisArgument,arg1, arg2, ...);

  - prvi argument je obavezan, i označava koja će promenljiva biti **this** kada se pozove funkcija
    - ako funkcija bude imala u svom telu **this.nešto**, onda se prvi argument vezuje za **this**
- Tipična primena drugog načina je kod nasleđivanja

Object.create(...)

- Kreira kopiju objekta ili prototipa (kada je u pitanju nasleđivanje):

```
var o = new Osoba("pera", "peric");
o.stampaj();
// napravi kopiju objekta 'o'
var o3 = Object.create(o);
o3.ime = "djura";
o.stampaj();
o3.stampaj();
```

- PRIMER: 2 kreiranje

JavaScript prototipovi

- JavaScript nema klase, već prototipove (Ne vazi od ES6!!)
- Prototip sadrži spisak nasleđenih svojstava i funkcija
  - u neku ruku, to je definicija nadklase
- **prototype** svojstvo postoji u svakom objektu
- Ako koristim nasleđivanje, onda postoji lanac prototipova (počinje sa **null** (prototype od **Object** je **null**), a završava poslednjim prototipom)
- Bitna svojstva:
  - **prototype.constructor** je funkcija koja kreira prototip
- Bitne metode:
  - **prototype.hasOwnProperty(prop)** vraća **true** ako objekat poseduje prosleđeno svojstvo
  - **prototype.isPrototypeOf(obj)** vraća true ako se u lancu "nasleđivanja" nalazi **obj**

Prototipovi

- Pristup prototipu preko ključne reči **prototype:**

```
// dodajemo jos jednu metodu u 'o', a ne u 'Osoba2'
o.stampajBold = function() {
  document.write("o.stampajBold: " + "<b>" + this.ime + ", "
              + this.prezime + "</b><br />");
}
// dodajemo jos jednu funkciju u 'Osoba2', pa ce biti vidljiva i u
  drugim objektima
Osoba2.prototype.stampajBold2 = function() {
  document.write("Osoba2.stampajBold: " + "<b>" + this.ime +
        ", " + this.prezime + "</b><br />");
};
PRIMER: 3.1 Nasledjivanje
```

Prototipovi

- Dodavanje svojstva ili funkcije u već kreiran objekat se ne reflektuje na druge objekte:

```
//dodajemo jos jednu metodu u 'o', a ne u 'Osoba2'
o.stampajBold = function() {
     document.write("o.stampajBold: " + "<b>" +
this.ime + ", "
                + this.prezime + "</b><br />");
}
o.stampajBold();
var oo = new Osoba2("pera", "peric");
oo.stampajBold(); // NE RADI!
oo.stampajBold2(); // RADI!
```

Dodavanje funkcija u prototip

- Ako dodamo svojstvo ili funkciju u prototip,
  - napravi se jedno telo funkcije i u svakom kreiranom objektu se poziva ta funkcija
    - zauzima manje memorije
  - onda se vide u naslednicima

```
Osoba.prototype.stampaj = function() {
  document.write(this.ime + ", " +
                 this.prezime + "<br />");
};
```

PRIMER: 3.2 Nasledjivanje

Nasleđivanje

- Nasleđivanje se svodi na dva koraka:
  - pozivanje konstruktora roditeljskog prototipa (ako je potrebno)
  - podešavanje prototipa kod naslednika

Primer konstruktora

```
function Osoba(ime, prezime) {
     this.ime = ime;
     this.prezime = prezime;
};
function Radnik(ime, prezime, radnoMesto) {
     // poziv konstruktora Osoba
     // Radnik jos ne 'nasledjuje' Osobu
     Osoba.call(this, ime, prezime);
     this.radnoMesto = radnoMesto;
};
```

Definisanje prototipa

```
// ovim podesavamo da radnik 'nasledjuje' osobu
Radnik.prototype =
Object.create(Osoba.prototype);
// Ovim podesavamo da je konstruktor za radnika
// funkcija Radnik
Radnik.prototype.constructor = Radnik;
```

Važno

- Uočiti razliku izmedju ovoga:

```
function Osoba(ime) {
        this.ime = ime;
        this.stampaj = function() {
                document.write(this.ime);
        };
};
```

- i ovoga:

```
function Osoba(ime) {
        this.ime = ime;
};
Osoba.prototype.stampaj = function() {
        document.write(this.ime);
};
```

Razlike

- Razlika je u tome što kod **gornjeg** primera ne postoji funkcija **štampaj** u prototipu (odn. opisu objekta), već u svakom napravljenom objektu
- U **donjem** primeru funkcija **stampaj** postoji i na nivou prototipa i u svakom napravljenom objektu.
- Gde bismo videli razliku?
- Kod nasleđivanja
  - prilikom nasleđivanja, povezujemo prototip naslednika na roditelja

Method override

- Ako je potrebno iz naslednice pozvati roditeljsku metodu
  - potrebno je da se u roditelju ta metoda doda u **prototype**
  - da se u naslednici pozove roditeljska metoda preko **call** funkcije

Method override

```
Osoba.prototype.stampaj = function() {
	document.write(this.ime + ", " + this.prezime + "<br />");
};
Radnik.prototype.stampaj = function() {
	// pozovemo metodu nadklase
	Osoba.prototype.stampaj.call(this);
	// ispod stampa "undefined, undefined" zato nema
	// this
	//Osoba.prototype.stampaj();
	document.write("Radno mesto: " + this.radnoMesto + "<br />");
};
```

PRIMER: 3.2 Nasledjivanje