Anonimne i Arrow funkcije

U prethodnoj lekciji ilustrovani su osnovni načini za kreiranje funkcija u JavaScriptu, njihovim deklarisanjem kao izjava. U JavaScriptu je funkcije moguće kreirati na još neke načine koji će biti prikazani u lekciji koja je pred vama. Tako ćete u ovoj lekciji imati prilike da se upoznate sa anonimnim i Arrow funkcijama. Takve vrste funkcija veoma često se koriste za realizaciju naprednih ponašanja – prosleđivanje jedne funkcije drugoj i emitovanje funkcije kao povratne vrednosti.

Funkcije kao izrazi i anonimne funkcije

JavaScript omogućava da se funkcije kreiraju kao izrazi, odnosno da se funkcija dodeli nekoj promenljivoj. U takvoj situaciji, funkcija se može definisati i bez naziva i tada se govori o takozvanim anonimnim funkcijama. Sledeći primer ilustruje jednu anonimnu funkciju, koja je dodeljena jednoj promenljivoj:

```
var a = function(x){
      return x * x;
}
```

Funkcija je kreirana bez naziva i dodeljena promenljivoj a. Na taj način, promenljiva a služi kao identifikator kreirane funkcije. To znači da se ona sada može pozvati korišćenjem naziva promenljive kojoj je dodeljena:

```
console.log(a(5));
```

Prosleđivanje jedne funkcije drugoj

Još jedna specijalna osobina JavaScripta, kada su funkcije u pitanju, ogleda se u mogućnosti prosleđivanja jedne funkcije drugoj. Mi ćemo se sa takvom osobinom jezika upoznati na sledećem primeru:

```
function sayHello(name){
    console.log("Hello " + name);
}
```

Prikazana funkcija `sayHello()` prihvata jedan parametar koji predstavlja ime osobe. Funkcija obavlja veoma jednostavnu logiku, tako što ispred naziva osobe postavlja tekst *Hello* i kompletnu poruku štampa unutar konzole. Prikazana funkcija se može upotrebiti na sledeći način:

```
sayHello("Ben");
```

Unutar konzole dobija se:

```
Hello Ben
```

Ipak, šta ukoliko je sada potrebno reći *Hello* većem broju osoba odjednom? Imena većeg broja osobina mogu biti definisana jednim nizom:

```
let names = ["John", "Ben", "Ana", "David"];
```

Kako bi se reklo *Hello* svim osobama iz prikazanog niza, može se kreirati jedna funkcija koja će prihvatati dva parametra – upravo kreirani niz, ali i već kreiranu funkciju `sayHello()`:

```
function sayHello(name) {
    console.log("Hello " + name);
}

let names = ["John", "Ben", "Ana", "David"];

function sayHelloToMany(names, sayHelloToOne) {
    for (let i = 0; i < names.length; i++) {
        sayHelloToOne(names[i]);
    }
}
```

Sada je kreirana funkcija `sayHelloToMany()`, koja prihvata dva parametra. Prvi parametar se odnosi na niz imena, dok drugi predstavlja funkciju koja će obavljati pojedinačno *pozdravljanje* svakog imena. Zbog toga se unutar funkcije `sayHelloToMany()` prolazi kroz niz imena i za svako ime se poziva funkcija `sayHelloToOne()` koja se ovoj funkciji prosleđuje kao parametar. Funkcija `sayHelloToMany()` može se upotrebiti na sledeći način:

```
sayHelloToMany(names, sayHello);
```

Unutar konzole dobija se:

```
Hello John
Hello Ben
Hello Ana
Hello David
```

Prilikom prosleđivanja jedne funkcije drugoj, veoma često pribegava se korišćenju anonimnih funkcija, tako što se funkcija koja predstavlja parametar definiše anonimno:

```
let names = ["John", "Ben", "Ana", "David"];

function sayHelloToMany(names, sayHelloToOne) {
    for (let i = 0; i < names.length; i++) {
        sayHelloToOne(names[i]);
    }
}

sayHelloToMany(names, function(name){
    console.log("Hello " + name);
});
```

Sada je funkcija `sayHello()` u potpunosti ukinuta kao zasebna jedinica. Za njom kao nezavisnom jedinicom više nema potrebe, s obzirom na to da se ona sada definiše kao anonimna funkcija direktno prilikom poziva funkcije `sayHelloToMany()`. Reč je o klasičnom primeru korišćenja anonimnih funkcija prilikom njihovog prosleđivanja kao parametara.

Funkcije višeg reda – Higher-Order Functions

Funkcije koje prihvataju jednu ili više drugih funkcija kao ulazne parametre, ili emituju funkciju kao povratnu vrednost, u programiranju veoma često nazivaju se funkcijama višeg reda – *Higher-Order Functions*. Prethodni redovi ilustrovali su jednu takvu funkciju – `sayHelloToMany()`.

Pitanje

Funkcije koje nemaju ime zovu se:

- **anonimne funkcije;**
- funkcije višeg reda;
- funkcije nižeg reda;
- statičke funkcije.

Funkcija bez imena drugačije se naziva anonimna funkcija.

Arrow funkcije

Anonimne funkcije se u JavaScriptu mogu definisati na još kompaktniji način, upotrebom specijalne sintakse koja je predstavljena u ECMAScript 2015 verziji specifikacije. Na taj način se dobija posebna vrsta funkcija – Arrow funkcije.

Nešto ranije je prikazana jedna ovakva funkcija:

```
var a = function(x){
      return x * x;
}
```

Reč je o anonimnoj funkciji za podizanje nekog broja na kvadrat, koja je dodeljena promenljivoj a. Mnogo kompaktnije ova funkcija se može definisati na sledeći način:

```
var a = (x) => { return x * x};
```

Može se videti da Arrow funkcije ne zahtevaju upotrebu `function` ključne reči. Pored toga, za izgradnju Arrow funkcija koristi se specijalni skup karaktera - **=>.** Upravo zbog toga se ovakve funkcije nazivaju Arrow funkcije.

Ipak, Arrow funkcije se u nekim situacijama mogu kreirati u još kompaktnijem obliku. Ukoliko Arrow funkcija prihvata samo jedan ulazni parametar, zagrade unutar kojih se definiše parametar mogu se izostaviti:

```
var a = x => { return x * x};
```

U prikazanoj naredbi se može videti da je sada parametar x definisan bez upotrebe zagrada. Ipak, to nije kraj optimizacijama koje se mogu napraviti prilikom definisanja ovakve Arrow funkcije. Ukoliko telo Arrow funkcije poseduje samo jednu naredbu koja emituje povratnu vrednost, vitičaste zagrade i ključnu reč `return` moguće je izostaviti:

```
var a = x => x * x;
```

Ovo je sada najkompaktniji oblik u kome se može pojaviti jedna Arrow funkcija. Njena povratna vrednost se naziva **implicitna**, s obzirom na to da nigde nije definisana ključna reč `return`.

Napomena

Korišćenje implicitne povratne vrednosti moguće je samo ukoliko telo funkcije ima jednu naredbu i ukoliko se za telo funkcije ne koriste vitičaste zagrade:

```
var a = x => {x * x};
console.log(a());
```

Ovoga puta se unutar konzole dobija vrednost `undefined`, što znači da implicitno emitovanje povratne vrednosti sada ne funkcioniše. Razlog je već naveden – kada se za definisanje tela Arrow funkcije iskoriste vitičaste zagrade, ne funkcioniše implicitno emitovanje povratne vrednosti.

U nastavku će biti prikazani primeri još nekih Arrow funkcija, sa različitim osobinama.

Arrow funkcija bez ulaznih parametara, sa jednom povratnom vrednošću:

```
var a = () => "Hello World";
```

Arrow funkcija sa jednim ulaznim parametrom, bez povratne vrednosti:

```
var a = (name) => {console.log("Hello " + name)};
```

Ili:

```
var a = name => { console.log("Hello " + name) };
```

Arrow funkcija sa dva ulazna parametra i jednom povratnom vrednošću:

```
var a = (x, y) => x * y;
```

Arrow funkcija sa više naredbi unutar tela:

```
var a = (x, y) => {
      x++;
      return y + x;
}
```

Primer prosleđivanja Arrow funkcije drugoj funkciji

Nešto ranije je prikazana mogućnost JavaScripta koja obezbeđuje da jedna funkcija nekoj drugoj bude prosleđena kao parametar. Takav primer će sada biti modifikovan, tako da se anonimna funkcija transformiše u jednu Arrow funkciju:

```
let names = ["John", "Ben", "Ana", "David"];

function sayHelloToMany(names, sayHelloToOne) {
      for (let i = 0; i < names.length; i++) {
            sayHelloToOne(names[i]);
      }
}

sayHelloToMany(names, name => { console.log("Hello " + name) });
```

Anonimna funkcija koja se prosleđuje funkciji `sayHelloToMany()`, sada je konvertovana u Arrow funkciju. S obzirom na to da ona prihvata samo jedan ulazni parametar (`name`), on je definisan bez upotrebe zagrada. Pošto funkcija ne emituje povratnu vrednost, već obavlja štampanje direktno unutar konzole, naredba za ispis je postavljena unutar vitičastih zagrada, kako ne bi došlo do implicitnog emitovanja povratne vrednosti.

Ključna reč this unutar regularnih i Arrow funkcija

Još jedna značajna osobina Arrow funkcija odnosi se na ključnu reč `this` unutar njih. Stoga, u narednim redovima biće razmotren jedan uporedni primer ponašanja ključne reči `this` unutar regularnih i Arrow funkcija.

Regularne funkcije, odnosno one koje nisu Arrow funkcije, vrednost ključne reči `this` dobijaju na nekoliko različitih načina, i to u zavisnosti od konteksta u kome se pozivaju. To praktično znači da jedna ista funkcija u različitim situacijama može imati različite vrednosti `this` ključne reči, a sve to u zavisnosti od toga kako se pozove. U nastavku će biti navedeno nekoliko najznačajnijih situacija pozivanja regularnih funkcija:

- kada se funkcija pozove unutar globalnog konteksta, `this` uvek ima vrednost globalnog objekta Window;
- objektne metode, odnosno funkcije koje se pozivaju nad objektima, za vrednost ključne reči `this` uvek imaju objekat nad kojim se pozivaju;
- kada se funkcija poziva mimo konteksta, odnosno kada se ne poziva ni nad jednim objektom, vrednost ključne reči `this` unutar nje je globalni objekat Window;
- unutar konstruktorskih funkcija, `this` se odnosi na tekući objekat koji se kreira korišćenjem takve funkcije;
- unutar funkcija za obradu događaja (*engl. event handlers*), `this` se odnosi na element za koji se vrši pretplata na događaj.

Ovo su bila neka od najznačajnijih pravila na osnovu kojih regularne funkcije dobijaju vrednost `this` ključne reči.

Kod Arrow funkcija situacija je mnogo jednostavnija – one ne poseduju sopstvenu vrednost ključne reči `this`. Drugim rečima, za vrednost ključne reči `this` one uvek dobijaju `this` iz sklopa u kome se nalaze. Tako kod njih nije bitan način na koji se pozivaju, već sklop, odnosno blok koda unutar koga se nalaze.

Kako biste najbolje razumeli razlike između regularnih i Arrow funkcija koje se odnose na `this` vrednost, biće razmotren sledeći primer:

```
function Student() {

    this.courses = [];

    this.addCourses = function (courses) {
        courses.forEach(function(value, index){
            this.courses[index] = value;
        });
    }
}

let s = new Student();
s.addCourses(["Math", "Geo", "History"]);

console.log(s.courses);
```

U primeru je kreirana jedna konstruktorska funkcija `Student()`. Logika konstruktorske funkcije je uprošćena tako da se unutar nje nalaze samo elementi koji su u ovom trenutku značajni za realizaciju primera. Tako konstruktorska funkcija `Student()` poseduje jedno svojstvo `courses` koje će čuvati vrednosti kurseva koje student pohađa. Za postavljanje vrednosti ovoga svojstva definisana je metoda `addCourses()`. Ona prihvata jedan parametar koji se odnosi na niz kurseva. Unutar metode se koristi ugrađena funkcija `forEach()` za prolazak kroz prosleđeni niz kurseva. Ugrađena funkcija `forEach()` kao parametar zahteva još jednu funkciju, koja će se aktivirati za svaki element niza kroz koji se prolazi. U primeru je takva funkcija definisana kao klasična anonimna funkcija, unutar koje se svaki element prosleđenog niza kurseva upisuje u niz unutar objekta `Student`.

Kada se obavi kreiranje objekta studenta i pozove metoda `addCourses()`, unutar konzole se dobija sledeći rezultat:

```
Uncaught TypeError: Cannot set property '0' of undefined
```

Izuzetak je produkt linije u kojoj se pokušava postaviti jedan član niza `this.courses` svojstva. Razlog je više nego jednostavan – takva linija nalazi se unutar anonimne funkcije koja se ne poziva ni nad jednim posebnim objektom, pa je stoga vrednost njenog `this` svojstva `Window` objekat. `Window` objekat, naravno, ne poseduje svojstvo `courses`, te otuda izuzetak koji se dobija unutar konzole.

Slične situacije se veoma često sreću u realnom radu, a pre uvođenja Arrow funkcija, one su bile rešavane uvođenjem alternativnog svojstva, kojim bi se ugnežđenim funkcijama na raspolaganje stavila `this` vrednost konteksta u kome se nalaze:

```
function Student() {

    this.courses = [];

    this.addCourses = function (courses) {

        var self = this;

        courses.forEach(function(value, index) {
            self.courses[index] = value;
        });
```

```
    }
}

let s = new Student();
s.addCourses(["Math", "Geo", "History"]);

console.log(s.courses);
```

Sada je u primeru definisan alternativni način za očuvanje konteksta konstruktorske funkcije `Student()`. Unutar metode `addCourses()` kreirana je promenljiva `self` i unutar nje je spakovana referenca na `this` vrednost. Naime, metoda `addCourses()` je poslednja metoda u hijerarhiji metoda, čija vrednost `this` ukazuje na tekući `Student` objekat. Naravno, to je tako zato što se ova metoda direktno poziva nad instancom objekta. Na kraju, promenljiva `self` se koristi unutar ugnežđene anonimne funkcije, umesto ključne reči `this`. Na taj način, sve funkcioniše bez greške, pa se unutar konzole dobija:

```
["Math", "Geo", "History"]
```

Prikazani pristup dugo vremena koristio se kao zaobilazno rešenje za problem koji je nastajao zbog nedostatka ugrađenih mehanizama jezika za njegovo rešavanje. Ipak, sa pojavom Arrow funkcija za tim više nema potrebe, s obzirom na to da Arrow funkcije automatski nasleđuju `this` vrednost iz konteksta u kome se nalaze. Tako je primer moguće preurediti na sledeći način:

```
function Student() {

    this.courses = [];

    this.addCourses = function (courses) {

        courses.forEach((value, index) =>{
            this.courses[index] = value;
        });
    }
}

let s = new Student();
s.addCourses(["Math", "Geo", "History"]);

console.log(s.courses);
```

Sada je metodi `forEach()`, umesto regularne, prosleđena Arrow funkcija. S obzirom na to da `this` ključna reč unutar takvih funkcija dobija vrednost na osnovu konteksta u kome se nalazi, primer funkcioniše na način identičan prethodnom, pa se i sada unutar konzole dobija:

```
["Math", "Geo", "History"]
```

Rezime

- JavaScript omogućava da se funkcije kreiraju kao izrazi, odnosno da se funkcija dodeli nekoj promenljivoj;
- funkcija bez imena naziva se anonimna funkcija;
- JavaScript omogućava da se jedna funkcija prosledi drugoj;
- *Higher-Order Functions* jeste drugi naziv za funkcije koje kao ulazni parametar prihvataju neku drugu funkciju ili funkciju emituju kao povratnu vrednost;
- ECMAScript 2015 verzija specifikacije uvela je još jedan način za definisanje anonimnih funkcija – Arrow funkcije;
- Arrow funkcije obezbeđuju sintaksu za kompaktno definisanje funkcija;
- prilikom kreiranja Arrow funkcija ne koristi se ključna reč `function`;
- ključna reč `this` unutar Arrow funkcija dobija vrednost iz sklopa u kome nalazi.