Specifične osobine JavaScript funkcija

Prethodna lekcija omogućila je uvid u osnove kreiranja i korišćenja funkcija u JavaScript jeziku. Prikazano je kako se funkcije kreiraju i koji su njihovi osnovni elementi. Takođe, u prethodnoj lekciji su predstavljeni i pojmovi korisnički definisanih funkcija i funkcija koje su automatski dostupne, tako što su ugrađene u sam JavaScript jezik.

Pored osnova iznetih u prethodnoj lekciji, za ispravno korišćenje funkcija potrebno je poznavati još neke njihove važne osobine. Stoga će se lekcija pred vama baviti samom suštinom njihove unutrašnje strukture i specifičnim scenarijima korišćenja.

Prosleđivanje parametara po vrednosti i po referenci

Funkcija može imati proizvoljan broj ulaznih parametara (argumenata) koje može koristiti funkcija tokom izvršavanja njene logike. Takođe, funkcija ne mora imati ulazne parametre, pa se u takvom slučaju govori o besparametarskoj funkciji. Ipak, za ispravno korišćenje funkcija potrebno je poznavati još neke važne osobine ulaznih parametara. Parametri se funkciji mogu proslediti na dva načina:

- po vrednosti i
- po referenci.

Svi parametri prostih tipova (`number`, `string`, `boolean`...) uvek se prosleđuju po vrednosti. To praktično znači da se funkciji prosleđuje kopija vrednosti promenljive, a ne sama promenljiva. Tako funkcija dobija vrednost, ali nema mogućnost da izvrši njenu promenu. To ilustruje sledeći primer:

```
var a = 5;
var b = 10;

function calculateRectArea(a, b) {

var c = a * b;
a = 30;
return c;
}

var result = calculateRectArea(5, 10);

console.log('Result is: ' + result);
console.log('a = ' + a);
```

Primer ilustruje definisanje i pozivanje funkcije za računanje površine pravougaonika.

Funkcija dobija dva parametra, odnosno vrednosti `5` i `10` i računa površinu. Ipak, pored računanja površine, unutar funkcije se obavlja i postavljanje vrednosti promenljive `a` na `30`.

Na kraju, unutar konzole se ispisuje rezultat računanja, ali i vrednost promenljive `a`:

```
Result is: 50
a = 5
```

Potpuno je jasno da se vrednost promenljive `a` nije promenila, odnosno da promena vrednosti unutar funkcije nije imala nikakav efekat, s obzirom na to da funkcija rukuje samo kopijom vrednosti stvarne promenljive.

Situacija je znatno drugačija kada se funkciji prosledi neki objekat. Tada se takav parametar prosleđuje po referenci.

```
function calculateRectArea(theObject) {
var result = theObject.number_a * theObject.number_b;
theObject.number_a = 30;
return result;
}

var rectangle = { number_a: 5, number_b: 10 };

var result = calculateRectArea(rectangle);
console.log('Result is: ' + result);
console.log('a = ' + rectangle.number_a);
```

Ovoga puta je funkcija transformisana tako da prihvata jedan parametar. Parametar je objekat koji je kreiran u prvoj naredbi nakon funkcije. Unutar takvog objekta objedinjene su dve vrednosti: dužina stranice a (`number_a`) i dužina stranice b (`number_b`).

Nakon definisanja objekta, obavlja se pozivanje funkcije i njoj se prosleđuje kreirani objekat. Funkcija računa površinu, a nakon toga i menja vrednost `number_a` svojstva.

Na kraju se unutar konzole ispisuje povratna vrednost funkcije, ali i vrednost `number_a` objektnog člana. Rezultat prikazanog koda je:

```
Result is: 50
a = 30
```

Može se primetiti da je vrednost prvog objektnog člana (`number_a`) 30, odakle se zaključuje da je promena izvršena unutar funkcije – promena na originalnoj vrednosti ulaznog parametra.

Pitanje

Vrednosti tipa `number` funkciji se prosleđuju po:

- **vrednosti**
- referenci

Objašnjenje:

Svi prosti tipovi podataka funkcijama se prosleđuju po vrednosti, što znači da funkcije dobijaju kopije vrednosti.

Function scope

Kada se kreira jedna funkcija, s obzirom na to da je reč o bloku koda, takva funkcija postaje jedna leksička celina. Takva leksička celina se drugačije naziva *Function scope* ili oblast

važenja jedne funkcije. Oblast važenja jedne funkcije poseban značaj ima za promenljive koje se deklarišu unutar, ali i izvan takvih funkcija. O ovome je već bilo reči u jednoj od prethodnih lekcija kada je razmatran pojam promenljivih. Tada je rečeno da je oblast važenja `var` promenljivih upravo jedna funkcija (dok je kod `let` promenljivih to bilo koji blok koda).

Kada se promenljiva deklariše izvan funkcije, takva promenljiva postaje globalna promenljiva. Sa druge strane, kada se promenljiva deklariše unutar jedne funkcije, ona je vidljiva samo unutar takve funkcije, a drugačije se naziva lokalna promenljiva.

Sledeći primer bavi se ovom problematikom:

```
function myFunction() {
var x = 4;
}
console.log(x);
```

Unutar funkcije `myFunction()` deklariše se i inicijalizuje promenljiva sa nazivom `x` i vrednošću `4`. Izvan funkcije pokušava se ispisivanje vrednosti promenljive `x`. Primer proizvodi sledeći rezultat:

```
ReferenceError: x is not defined
```

Naravno, kôd proizvodi grešku, zato što promenljiva `x` nije vidljiva na globalnom nivou, već samo unutar funkcije `myFunction()`.

Sa druge strane, globalna promenljiva je vidljiva unutar bilo koje funkcije ili bloka:

```
var x = 4;
function myFunction() {
console.log(x)
}
myFunction();
```

Promenljiva x je globalna promenljiva i kao takva dostupna je i unutar funkcije `myFunction()`. Zato primer bez problema ispisuje vrednost promenljive x:

```
4
```

Svi do sada prikazani primeri stvorili bi identičan efekat i kada bi promenljive bile deklarisane korišćenjem ključne reči `let`.

Podizanje funkcija na početak dokumenta

U svim dosadašnjim primerima definicije funkcija uvek su prethodile pozivima takvih funkcija.

Drugim rečima, funkcije su uvek prvo bile definisane, a tek onda i pozivane:

```
function myFunction(x) {
return x * x;
}
```

```
myFunction(6);
```

Ovo je potpuno logičan sled, jer je očekivano da se ne može pozvati nešto što prethodno nije deklarisano. Da li je stvarno tako? Odgovor na ovo pitanje daje sledeći primer:

```
myFunction(6);

function myFunction(x) {
return x * x;
}
```

Ovoga puta funkcija je pozvana pre nego što je definisana. Potpuno neočekivano, ovakav kôd proizvodi sledeći rezultat:

```
36
```

Iako se deklaracija nalazi ispod poziva, kôd funkcioniše bez problema. Zašto je to tako?

Odgovor na ovo pitanje leži u specijalnoj funkcionalnosti JavaScript prevodioca, o kojoj je već bilo reči – *hoisting*. *Hoisting* predstavlja pomeranje svih deklaracija na početak tekućeg opsega. Iz upravo prikazanog primera se može videti da se ovakva osobina ne odnosi samo na promenljive i konstante, već i na funkcije.

Primer – Kreiranje skripte za sabiranje dva broja definisanjem funkcije

U jednoj od prethodnih lekcija prikazan je primer koji obavlja sabiranje dva broja. Sada se takav primer može obaviti na nešto elegantniji način, upotrebom funkcije. Funkcija za sabiranje dva broja mogla bi da izgleda ovako:

```
function sum(a, b) {
return parseInt(a) + parseInt(b);
}
```

Funkcija ima naziv `sum`. Prihvata dva ulazna parametra i emituje jednu povratnu vrednost.

Kompletan primer u kome se upotrebljava ovakva funkcija izgleda ovako:

```
let numberA = prompt("Please enter number A:");
let numberB = prompt("Please enter number B:");

let result = sum(numberA, numberB);

alert("Sum is: " + result);

function sum(a, b) {
return parseInt(a) + parseInt(b);
}
```

Rezime

- Parametri se funkciji mogu proslediti na dva načina: po vrednosti i po referenci.
- Svi parametri prostih tipova (`number`, `string`, `boolean`...) uvek se prosleđuju po vrednosti.
- Funkcija koja dobija parametre koji se prosleđuju po vrednosti nema mogućnost da vrši njihovu promenu.
- Prosleđivanje objekata kao ulaznih parametara obavlja se po referenci.
- Prosleđivanje parametara po referenci znači da funkcija dobija stvarne vrednosti, pa su sve promene unutar funkcije – promene nad stvarnim vrednostima.
- Svaka funkcija je jedna leksička celina, *function scope*, odnosno oblast važenja.
- Oblast važenja jedne funkcije poseban značaj ima za promenljive koje se deklarišu unutar, ali i izvan takvih funkcija.
- Promenljiva deklarisana izvan funkcije drugačije se naziva globalna promenljiva.
- Promenljiva deklarisana unutar funkcije vidljiva je samo unutar takve funkcije, a drugačije se naziva lokalna promenljiva.
- *Hoisting* je osobina JavaScript jezika, ne primenjuje se samo na deklaracije promenljivih, već i na funkcije.