Funkcije

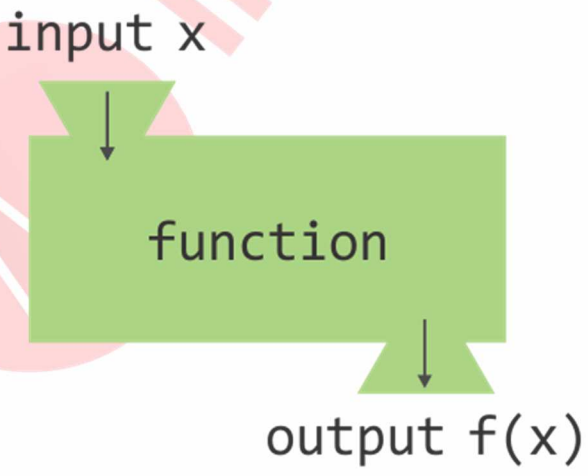
JavaScript je jedan od programskih jezika za koje se kaže da poseduju funkcije prvog reda (*engl. first-class functions*). To praktično znači da je reč o jeziku kod koga su funkcije jedan od najznačajnijih jezičkih elemenata. Kako bi se funkcije unutar nekog programskog jezika tretirale na takav način, neophodno je da jezik obezbedi neke specifične osobine korišćenja funkcija:

- prosleđivanje funkcija kao argumenata drugim funkcijama;
- emitovanje funkcija kao povratnih vrednosti drugih funkcija;
- dodeljivanje funkcija promenljivama;
- kreiranje anonimnih funkcija, odnosno funkcija bez imena;
- smeštanje funkcija unutar drugih funkcija;

Sve ovo su specifični, odnosno napredni primeri korišćenja funkcija koje su podržane od strane jezika JavaScript. U lekcijama modula koji je pred vama bavićemo se ovakvim i sličnim naprednim tehnikama rada sa JavaScript funkcijama. Za početak, prvi deo ove lekcije prezentovaće neke osnovne osobine funkcija u JavaScriptu, kao uvertiru za ilustraciju naprednih pristupa koji slede.

Šta su funkcije?

Funkcija je specijalni blok koda, zadužen za izvršenje određene logike. Pri tome, funkcija prilikom izvršavanja takve logike može da koristi određene vrednosti, koje se nazivaju ulazni parametri, a može i emitovati svoj rezultat kao povratnu vrednost (slika 4.1).

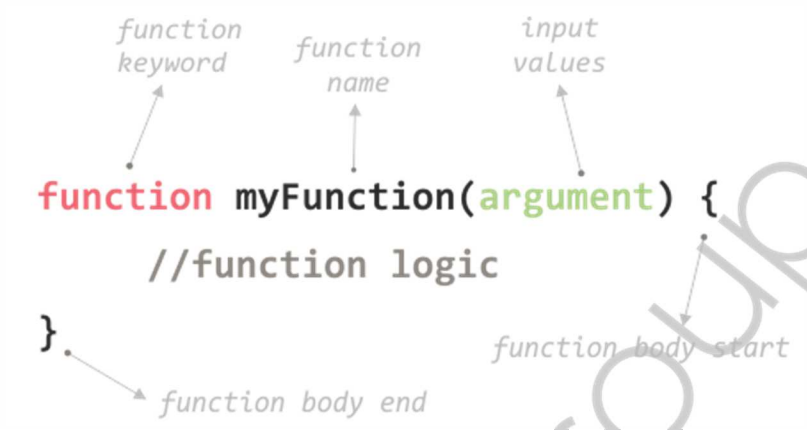


Slika 4.1. Funkcija

Na slici 4.1. slovom x je označen ulazni parametar, odnosno vrednost koju funkcija dobija na obradu. Nakon završene obrade, funkcija emituje rezultat izvršavanja. Ipak, nije obavezno da funkcija ima ulazne i izlazne parametre, što ćete uskoro videti.

Osnovni način za kreiranje i pozivanje funkcija

U programskom jeziku JavaScript funkcija se definiše korišćenjem ključne reči `function`. Nakon ove ključne reči navodi se naziv funkcije, eventualni parametri i blok koda, oivičen uglastim zagradama (slika 4.2).



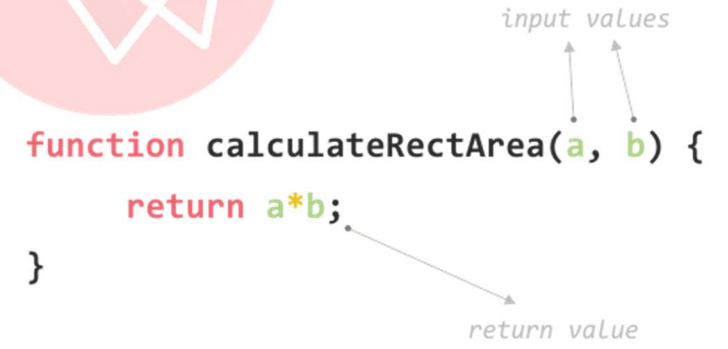
Slika 4.2. Sintaksa funkcije u JavaScriptu

Kreiranje jedne funkcije drugačije se naziva **deklaracija funkcije**. Stoga, funkcija koja je deklarisana na slici 4.2. ima naziv `myFunction`, prihvata jedan ulazni parametar (argument), a njeno telo je prazno. Ključna reč `function`, naziv funkcije i njeni parametri drugačije se nazivaju **potpis funkcije**.

Primer jedne JavaScript funkcije može da izgleda ovako:

```
function calculateRectArea(a, b){
      return a*b;
}
```

Ovakva funkcija ima naziv `calculateRectArea`, prihvata dva ulazna parametra i emituje povratnu vrednost (slika 4.3).



Slika 4.3. Sintaksa funkcije u JavaScriptu

Kako bi se logika funkcije uposlila, nju je potrebno pozvati. Sledeća linija koda ilustruje pozivanje upravo kreirane funkcije:

```
calculateRectArea(5, 10);
```

Na ovaj način, poziva se funkcija sa nazivom `calculateRectArea` i njoj se prosleđuju ulazni parametri `5` i `10`. Tako se aktivira logika funkcije, koja je definisana unutar njenog tela. Funkcija obavlja posao definisan njenom logikom i vraća vrednost izvršavanja. Ipak, u prikazanom primeru ništa nije uređeno sa takvom povratnom vrednošću. Drugim rečima, prikazanom naredbom kojom se poziva funkcija nije precizirano šta uraditi sa rezultatom koji funkcija isporučuje. Stoga će primer biti modifikovan na sledeći način:

```
function calculateRectArea(a, b){
      return a*b;
}

var result = calculateRectArea(5, 10);
document.write(result);
```

Sada je rezultat funkcije smešten unutar promenljive sa nazivom `result`. Da je to stvarno tako potvrđuje ispis koji proizvodi prikazani kod:

```
50
```

Nešto ranije je rečeno da funkcije ne moraju da prihvate ulazne parametre, niti da emituju povratnu vrednost. Ipak, dosad su prikazani samo primeri funkcija koje prihvataju parametre i emituju povratnu vrednost. Stoga, evo još nekih primera funkcija, sa nešto drugačijim osobinama:

Primer funkcije sa jednim ulaznim parametrom, bez povratne vrednosti:

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

Funkcija `sayHello()` prihvata jedan parametar, ali nema izlaznih parametara, zato što unutar njenog tela nije upotrebljena ključna reč `return`. Ova funkcija se može pozvati na sledeći način:

```
sayHello("John");
```

Unutar konzole se dobija:

```
Hello John
```

Primer funkcije bez ulaznih parametara i bez povratne vrednosti

```
function sayHelloWorld() {
    console.log("Hello World");
}
```

Funkcija `sayHelloWorld()` ne prihvata nijedan parametar. Takođe, ona nema ni povratnih vrednosti. Svakim pozivom ove funkcije, unutar konzole se ispisuje tekst *Hello World*:

```
sayHelloWorld(); //Hello World
```

Pitanje

Funkcije se u jeziku JavaScript deklarišu upotrebom ključne reči:

- **`function`**
- `class`
- `object`
- `method`

Objašnjenje:

Kreiranje funkcije drugačije se naziva deklaracija. Deklaracija započinje navođenjem ključne reči `function`, nakon čega se definiše naziv funkcije.

Funkcije u JavaScriptu su objekti

Bitno je reći da su funkcije u JavaScript jeziku objekti koji se u pozadini kreiraju korišćenjem konstruktorske funkcije `Function()`. To ste mogli videti i u prethodnom modulu, u kome je bilo reči o nasleđivanju u JavaScriptu – sve funkcije za svoj prototip imaju `Function.prototype` objekat.

Sve ovo znači da je u JavaScriptu funkciju moguće kreirati i korišćenjem sledeće sintakse:

```
new Function (arg1, arg2, ... argN, functionBody)
```

Prikazana sintaksa podrazumeva kreiranje funkcija korišćenjem konstruktorske funkcije `Function()`. Parametri i telo se ovakvoj funkciji prosleđuju kao argumenti – prvo parametri, a zatim i telo funkcije. Sve ovo znači da se nešto ranije kreirana funkcija sada može definisati na sledeći način:

```
calculateRectArea = new Function("a", "b", "return a*b;");
```

Ovako definisana funkcija sada se može pozvati na već viđeni način:

```
let rectArea = calculateRectArea(5, 10);
console.log(rectArea);
```

Funkcija je pozvana, povratna vrednost smeštena unutar `rectArea` promenljive i na kraju je takva vrednost ispisana unutar konzole.

Nije teško primetiti osnovni nedostatak ovakvog načina definisanja funkcija – nazivi parametara i telo funkcije definišu se u tekstualnom, `string` obliku. Upravo zbog toga se ovakav način definisanja funkcija **ne preporučuje**. Naime, kada se telo funkcije definiše u tekstualnom obliku, JavaScript izvršno okruženje nije u mogućnosti da obavi određene, veoma važne optimizacije.

Ipak, sama činjenica da su funkcije u JavaScriptu objekti omogućava korišćenje nekoliko veoma upotrebljivih pristupa. Naime, s obzirom na to da su funkcije objekti, oni poseduju nekoliko svojstava koje je moguće koristiti unutar tela funkcija. Izdvaja se svojstvo `arguments` (tabela 4.1).

Svojstvo	Opis
• `arguments`	objekat koji sadrži argumente odnosno parametre koji su prosleđeni funkciji
• `arguments.length`	broj parametara koji su prosleđeni funkciji

Tabela 4.1. Svojstvo arguments Function objekta

Svojstvo `arguments` `Function` objekta omogućava dobijanje informacija o prosleđenim parametrima, pa samim tim i kreiranje funkcija koje mogu da prihvate promenljivi broj parametara.

```
function sum() {
    let returnValue = 0;

    for (let i = 0; i < arguments.length; i++) {
        returnValue += arguments[i];
    }

    return returnValue;
}
```

Primer ilustruje funkciju `sum()` kojom se obavlja sabiranje prosleđenih parametara. Ipak, zanimljivo je da prilikom deklaracije funkcije nije naveden nijedan ulazni parametar. Parametri koji se ovakvoj funkciji eventualno proslede obrađuju se unutar tela funkcije, i to upravo korišćenjem nešto ranije spomenutog svojstva `arguments` objekta `Function`. Reč je o svojstvu koje čuva referencu na objekat čija struktura podseća na niz. Upravo zbog toga se kroz `arguments` objekat prolazi kao da je reč o nizu – korišćenjem for petlje i korišćenjem indeksa za pristup svojstvima, odnosno konkretnim parametrima. Unutar for petlje obavlja se sabiranje svih prosleđenih parametara, a zatim se dobijeni rezultat emituje kao povratna vrednost.

Funkcija `sum()` se sada može pozvati i tom prilikom njoj se može proslediti proizvoljan broj parametara. Evo, na primer, poziva sa 3 parametra:

```
let sumResult = sum(10, 15, 15)
console.log(sumResult);
```

Unutar konzole dobija se vrednost 40, što je zbir brojeva 10, 15 i 15.

Funkciju `sum()` je moguće pozvati i bez parametara:

```
let sumResult = sum()
console.log(sumResult);
```

U ovakvoj situaciji se kao povratna vrednost dobija 0, s obzirom na to da nema prosleđenih parametara.

Funkcija koja poziva samu sebe – rekurzija

Prvi specijalan slučaj korišćenja funkcija koji će biti prikazan u okviru ovog izlaganja o funkcijama odnosi se na mogućnost funkcija da pozivaju same sebe. U programiranju se takva pojava naziva rekurzija. Rekurzija se može koristiti za pojednostavljeno rešavanje brojnih problema u programiranju, čime se pretežno izbegava kreiranje veoma komplikovanih petlji. Rekurzija može biti teška za razumevanje, pogotovu za početnike, stoga će u nastavku prvo biti prikazan jedan jednostavan, bazičan primer, a tek zatim i realan primer korišćenja rekurzije.

Prvi primer odnosiće se na realizaciju funkcije koja će obavljati jednostavno odbrojavanje. Funkciji će se prosleđivati početna vrednost, a zatim će ona obavljati pojedinačno štampanje svakog celog broja koji je manji od prosleđenog, uključujući i prosleđeni, sve do broja 0. Osnovni način za realizaciju ovakve logike mogao bi da izgleda ovako:

```
function countdown(start){
    for (let i = start; i >= 0; i--) {
        console.log(i);
 }
}
```

Funkcija nosi naziv `countdown` i ona prihvata jedan parametar koji se odnosi na početnu vrednost odbrojavanja. Unutar tela funkcije definisana je jedna for petlja. Brojač petlje se postavlja na prosleđenu `start` vrednost, a zatim se u svakoj iteraciji umanjuje za jedan. Petlja će se izvršavati sve dok je brojač jednak ili veći od nule. Unutar konzole se štampa vrednost brojača. Stoga, ukoliko se funkciji prosledi, na primer, vrednost 8, unutar konzole biće dobijen sledeći ispis:

```
8
7
6
5
4
3
2
1
0
```

Prikazani kod je u potpunosti regularan i nema mu se šta zameriti. Ipak, identično se može postići i upotrebom rekurzije:

```
function countdown(start) {

    if(start < 0){
        return;
    }

    console.log(start);
```

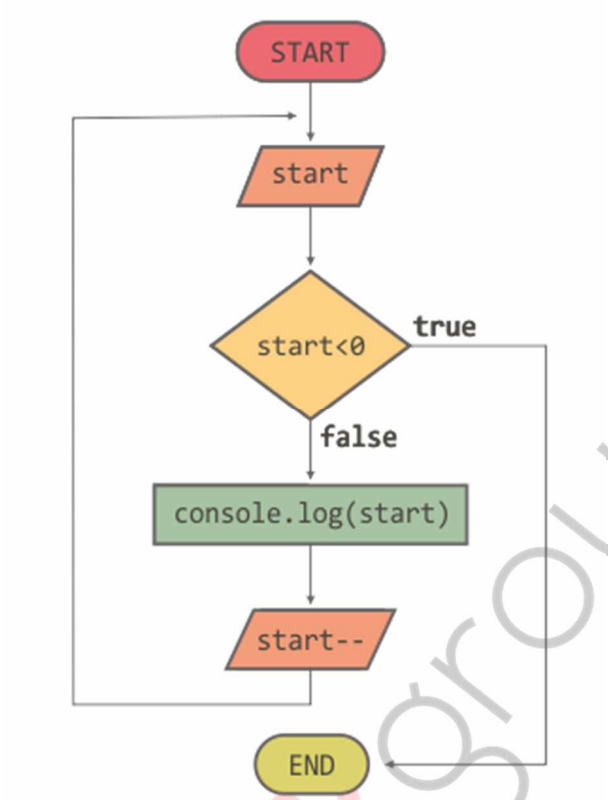
```
        countdown(--start);

    }
```

Logika funkcije `countdown()` sada je izmenjena. Unutar nje više nema for petlje, već funkcija štampa prosleđenu vrednost, zatim je umanjuje, a onda obavlja pozivanje same sebe. Prilikom pozivanja same sebe, metodi se prosleđuje vrednost koja je prethodno umanjena za 1. Sve to rezultuje identičnim efektom:

```
8
7
6
5
4
3
2
1
0
```

Veoma bitno je da primetite da je na početku funkcije `countdown()` definisana jedna provera. Proverava se da li je prosleđena vrednost manja od 0. Ukoliko jeste, prestaje izvršavanje funkcije, pa se na ovaj način zaustavlja rekurzija. Ovaj uslov je zapravo način na koji se zaustavlja ciklično samopozivanje funkcije `countdown()`.Prilikom definisanja svake rekurzije, neophodno je osigurati mehanizam za njeno zaustavljanje. Da ovoga uslova nema, bila bi stvorena beskonačna rekurzija, što je analogno kreiranju beskonačne (mrtve) petlje, odnosno petlje koja nema kraj.

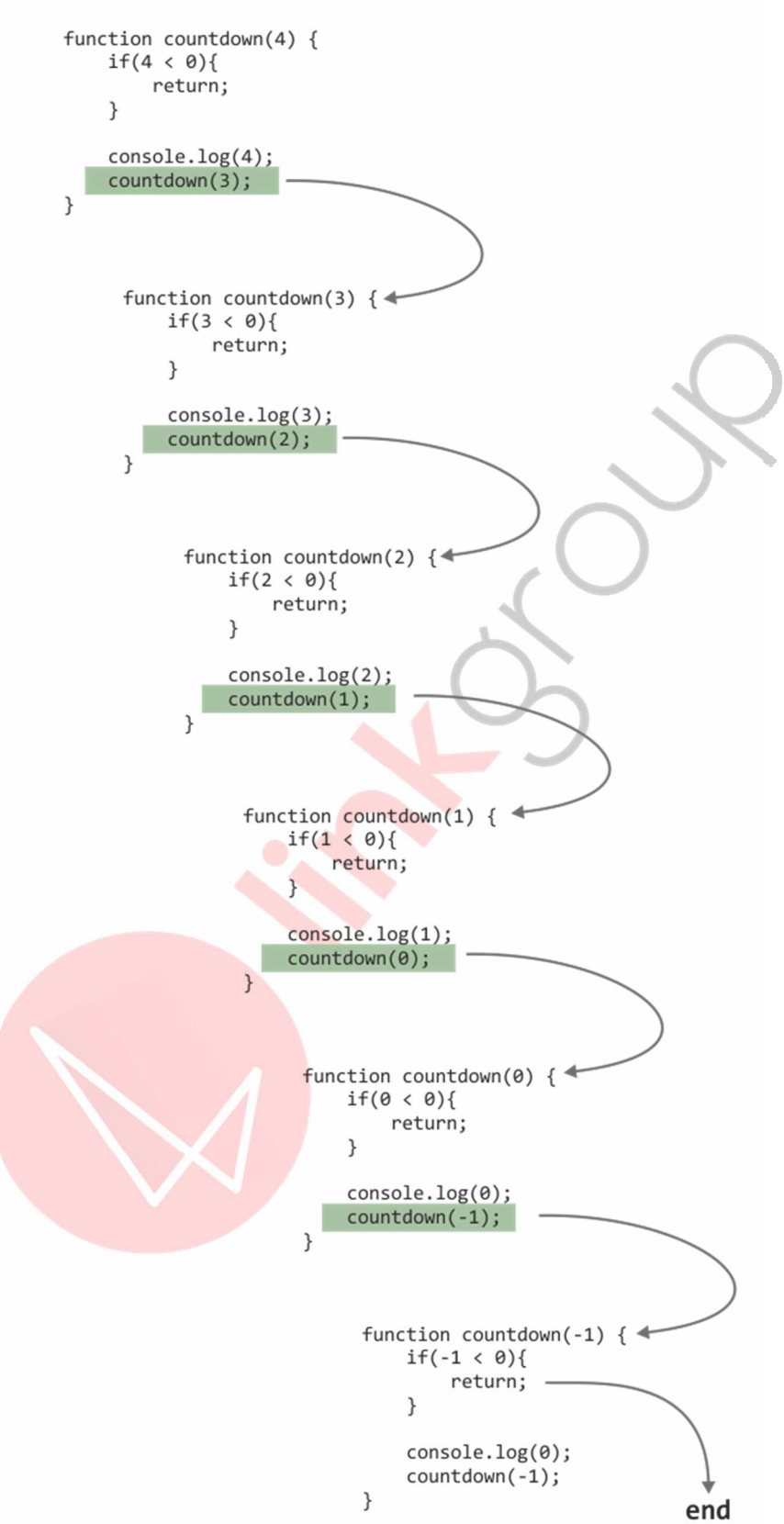
Kako biste sve ovo još bolje razumeli, prikazani primer rekurzije biće ilustrovan algoritmom (slika 4.4).



Slika 4.4. Algoritam rekurzivnog izvršavanja funkcije countdown()

Algoritam oslikava logiku kreirane rekurzije unutar funkcije `countdown()`. Započinje se ulaznom vrednošću `start`. Zatim se proverava da li je ulazna vrednost manja od nule. Ukoliko jeste, završava se izvršavanje. Ukoliko nije, obavlja se štampanje vrednosti. Zatim se vrednost umanjuje za jedan i tako umanjena vrednost prosleđuje funkciji `countdown()` prilikom ponovnog poziva.

Ilustrovan prikaz izvršavanja rekurzivne funkcije `countdown()` izgleda kao na slici 4.5.



Slika 4.5. Rekurzivni tok izvršavanja funkcije countdown()

Primer – funkcija za sortiranje niza koja se zasniva na rekurziji

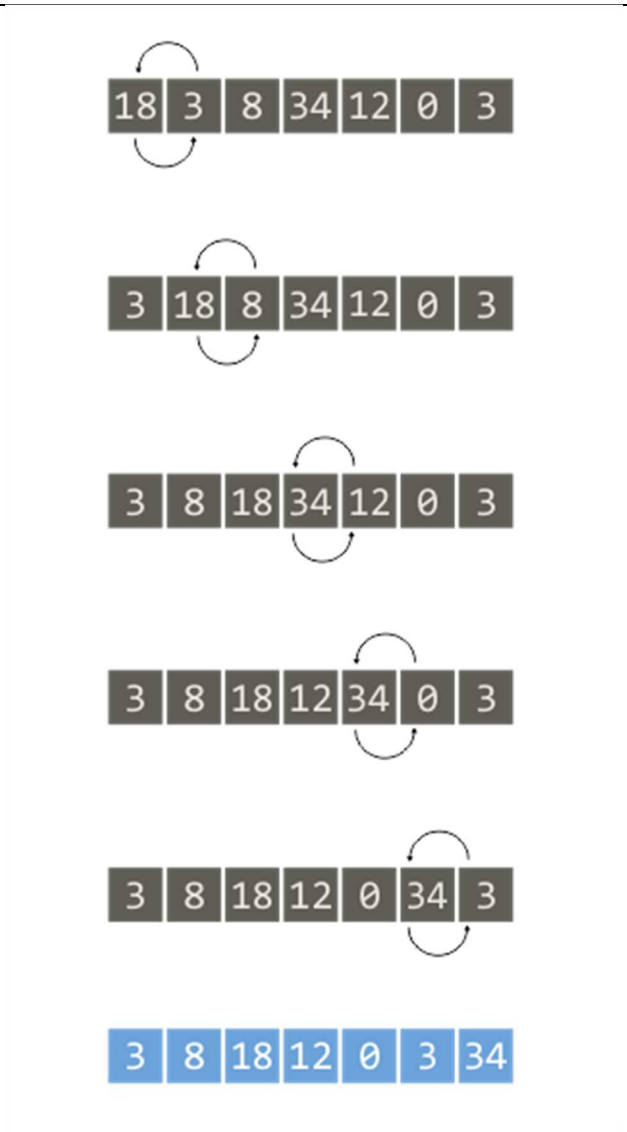
Realan primer korišćenja rekurzije biće prikazan na problemu sortiranja numeričkih nizova. Sortiranje će se obavljati unutar jedne funkcije i njene for petlje. Prolaskom kroz niz koji treba sortirati vršiće se poređenje susednih elemenata i rotiranje njihovih pozicija, ukoliko je prethodni element veći od narednog. Korišćenjem takvog pristupa, nije moguće sortirati niz samo jednim prolaskom kroz njega. Ipak, nakon završetka for petlje, funkcija za sortiranje će iznova pozivati samu sebe, sa parametrom koji predstavlja niz koji će svakim novim pozivanjem biti sve bliži u potpunosti sortiranom nizu. Evo kako će takva funkcija izgledati:

```
function sort(array) {
    let temp;
    let sorted = true;
    for (let i = 0; i < array.length-1; i++) {
        if (array[i] > array[i + 1]) {
            temp = array[i];
            array[i] = array[i + 1];
            array[i + 1] = temp;
            sorted = false;
        }
    }
    if (!sorted) {
        sort(array);
    }

    return array;
}
```

Unutar funkcije `sort()` nalazi se for petlja, kojom se prolazi kroz niz koji je potrebno sortirati. Petlja za prolazak ima jednu iteraciju manje od ukupnog broja elemenata niza. Razlog je vrlo jednostavan. Unutar petlje se porede susedni elementi, pa kako u poslednjoj iteraciji ne bi došlo do situacije da se poslednji element poredi sa nepostojećim elementom, odnosno elementom čiji indeks prevazilazi granice niza, broj iteracija je za jedan manji od dužine niza.

Ukoliko se unutar for petlje utvrdi da je prethodni element veći od narednog, obavlja se njihovo rotiranje. Za posao rotiranja koristi se i pomoćna promenljiva `temp`, kako se tokom rotiranja ne bi izgubila vrednost jednog od članova niza.

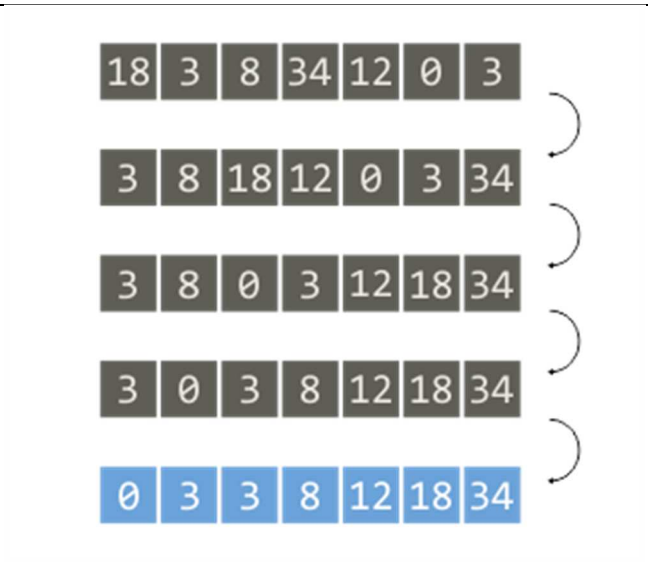
Na primer, za niz od sedam članova prikazana for petlja će imati šest iteracija. Takve iteracije na primeru niza sa članovima **[18, 3, 8, 34, 12, 0, 3]** izgledaju kao na slici 4.6.



Slika 4.6. Tok izvršavanja for petlje, funkcije za sortiranje

Slika 4.6. ilustruje sve iteracije prikazane for petlje. Jasno je da se jednim prolaskom kroz niz obavlja njegovo delimično sortiranje. Stoga, nakon završetka for petlje, funkcija `sort()` poziva samu sebe, ali sada sa delimično sortiranim nizom. Logika se tako ponavlja ukrug, sve dok se niz u potpunosti ne sortira. Kako bi se znalo kada je niz sortiran, unutar funkcije se koristi kontrolna `boolean` promenljiva `sorted`. Njena vrednost je inicijalno postavljena na `true`, a postaje `false` ukoliko je makar jednu intervenciju potrebno sprovesti nad elementima niza. Tek kada funkcija `sort()` dobije u potpunosti sortiran niz, vrednost promenljive `sorted` ostaje `true`, pa se rekurzija zaustavlja i finalni, sortirani niz emituje kao povratna vrednost.

Slika 4.6. je ilustrovala sve iteracije jednog izvršavanja for petlje, a sada će za kraj biti prikazani izgledi nizova nakon završetka svakog pojedinačnog izvršavanja `sort()` funkcije (slika 4.7).



Slika 4.7. Tok rekurzije funkcije sort()

Rezime

- u JavaScriptu funkcije su jedan od najznačajnijih jezičkih elemenata;
- funkcija je specijalni blok koda, zadužen za izvršenje određene logike;
- funkcija se deklariše korišćenjem ključne reči `function`
- funkcija može imati ulazne parametre, odnosno argumente i povratnu vrednost;
- naziv i parametri funkcije drugačije se nazivaju potpis funkcije;
- povratna vrednost se iz funkcije emituje upotrebom ključne reči `return`
- u JavaScriptu funkcije su objekti koji se kreiraju korišćenjem konstruktorske funkcije `Function()`
- svaka funkcija poseduje svojstvo `arguments` koje sadrži argumente, odnosno parametre koji se prosleđuju funkciji;
- rekurzija je pojam koji se odnosi na mogućnost funkcija da pozivaju same sebe.