Petlje

Prethodna lekcija ilustrovala je jedan od osnovnih elemenata za postizanje kontrole toka – uslovno izvršavanje, odnosno grananje. Pored grananja, petlje su jedan od osnovnih elemenata za postizanje kontrole toka. Stoga će lekcija pred vama biti posvećena upoznavanju pojma petlji u JavaScript jeziku.

Pojam petlji

Za razliku od uslovnog izvršavanja (grananja), petlje imaju nešto drugačije osobine. Da bi se njihove osobine na najbolji način razumele, biće razmotrena sledeća situacija:

Na koji način se jedna ista naredba JavaScript koda, može izvršiti više puta?

Na primer, potrebno je pet (5) puta ispisati jednu istu poruku unutar konzole.

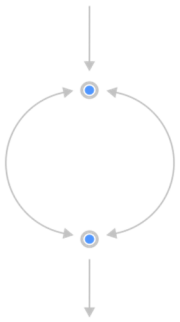
Najjednostavnije rešenje za postizanje opisanog ponašanja jeste navođenje pet sukcesivnih naredbi za ispis:

```
console.log("Hello World");

console.log("Hello World");

console.log("Hello World");

console.log("Hello World");

console.log("Hello World");
```

Željeni rezultat je postignut, ali šta da je bilo potrebno ovakvu poruku ispisati 1000 puta?

Jasno je da je navođenje naredbe za naredbom vrlo nepraktično i zamorno. Ono što je potrebno uraditi kako bi se nastali problem razrešio, jeste iznalaženje načina da se izvršnom okruženju kaže: *Izvrši naredbu za ispis poruke tačno 1000 puta.* Upravo su petlje način da se jedna takva logika formuliše u programski kôd JavaScript jezika.

Osnovna logika petlji ilustrovana je slikom 13.1.



Slika 13.1. Petlja

Sa slike 13.1. se može zaključiti da u svetu programiranja petlje omogućavaju da se određeni deo koda izvrši više puta, tako što se tok izvršavanja ciklično vraća unazad.

U osnovi svake petlje jeste proces ponavljanja. Svako ponavljanje se drugačije naziva **iteracija**. Tako su petlje, zapravo, sačinjene iz iteracija. Petlja započinje prvom iteracijom, a završava se nakon poslednje iteracije. Broj iteracija jedne petlje može biti proizvoljan, od 0 do beskonačno. Petlja sa beskonačnim brojem iteracija nema svoj kraj, pa se drugačije naziva **mrtva petlja**.

Petlje u JavaScriptu

JavaScript poznaje nekoliko naredbi koje omogućavaju kreiranje petlji:

- `while`
- `do…while`
- `for`

Takođe, JavaScript poseduje i dve naredbe za kontrolu izvršavanja petlji:

- `break`
- `continue`

Navedene naredbe biće tema lekcije pred vama.

For petlja

`For` petlja omogućava da se određeni blok koda izvrši tačan broj puta. U uvodnom delu ove lekcije postavljeno je pitanje:

Kako ispisati jednu istu poruku određeni broj puta?

Odgovor na ovo pitanje leži upravo u korišćenju `for` petlje:

```
for (let i = 0; i < 5; i++) {
      console.log("Hello World");
}
```

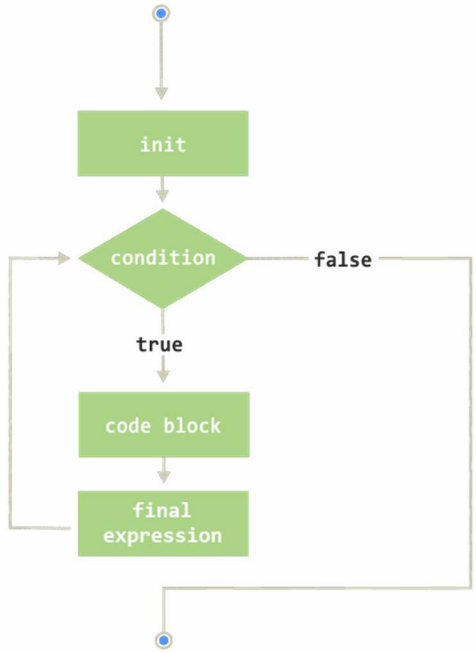
Rezultat prikazanog koda biće:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Da bismo razumeli kako je ovakvo nešto postignuto, neophodno je upoznati se sa sintaksom `for` petlje:

```
for (initialization; condition; final-expression)
  statement
```

Kada se ovakav pseudokôd predstavi korišćenjem algoritma, struktura `for` petlje je kao na slici 13.2.



Slika 13.2. Logika for petlje

For petlja kreira se korišćenjem ključne reči `for`. Nakon ove ključne reči, u zagradama se navode tri izjave (naredbe):

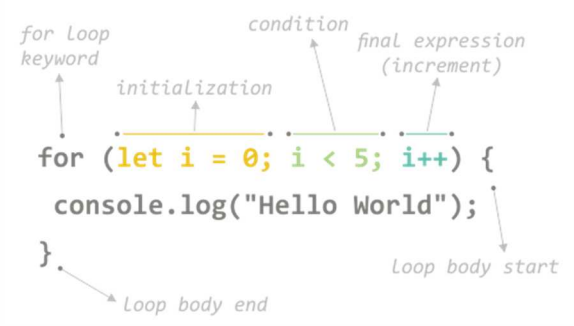
1. **initialization (inicijalizacija)** – početna naredba koja se izvršava samo jednom na početku izvršavanja petlje; koristi se za inicijalizaciju jednog ili više brojača petlje; u prikazanom primeru to je `let i = 0;`

2. **condition (uslov)** – naredba kojom se utvrđuje istinitost uslova; ukoliko je uslov ispunjen, prelazi se na izvršavanje tela petlje; ovo je naredba koja se izvršava pre svake iteracije; u prikazanom primeru je to `i < 5;`

3. **final-expression** – naredba koja se izvršava nakon svake iteracije; uglavnom se koristi za uvećanje ili smanjenje vrednosti brojača; u primeru je to `i++`

Nakon zagrada u kojima se nalaze tri upravo opisane naredbe, sledi telo petlje koje započinje otvorenom, a završava se zatvorenom vitičastom zagradom. Unutar bloka petlje može se naći proizvoljan broj naredbi koje se ponavljaju sa svakom iteracijom. Vitičaste zagrade za formiranje bloka `for` petlje je moguće izostaviti ukoliko telo petlje poseduje samo jednu naredbu.

Kompletna upravo opisana struktura `for` petlje na prvom prikazanom primeru se može ilustrovati kao na slici 13.3.



Slika 13.3. Struktura jedne for petlje

Na kraju ćemo još jednom rezimirati kompletnu logiku prikazanog primera. Kreiranje petlje započinje navođenjem ključne reči `for`. Nakon toga, u zagradama se definiše inicijalizacija, uslov i završna naredba.

Prvo se deklariše i inicijalizuje jedna promenljiva sa nazivom `i`:

```
let i = 0;
```

Ova promenljiva ima početnu vrednost 0, a unutar `for` petlje služi kao **brojač**.
Nakon inicijalizacije, sledi deo kojim se definiše uslov:

```
i < 5;
```

Na ovaj način je rečeno da će se telo petlje izvršavati sve dok je vrednost promenljive `i` manja od 5.

Na kraju, definiše se i završna naredba, koja je u prikazanom primeru realizovana kao inkrementacija brojača:

```
i++
```

Ovakvom naredbom je rečeno da se, nakon svake iteracije, vrednost brojača uvećava za jedan.

Napomena

Deklaracija brojača unutar petlje: `var` ili `let`

U prethodnom primeru ste možda primetili da je za deklaraciju promenljive koja će unutar petlje služiti kao brojač korišćena ključna reč `let`. Umesto ključne reči `let`, bilo je moguće koristiti i `var` ili čak i izostaviti bilo kakvu ključnu reč. Ipak, postoje izvesne razlike između različitih načina za deklarisanje brojača, te na kraju nije slučajno što je za obavljanje takvog posla odabrana ključna reč `let`.

Razlike između opisanih načina za deklarisanje brojača zasnivaju se na razlikama između promenljivih `var` i `let`, o čemu je bilo reči u jednoj od prethodnih lekcija. Oblast važenja jedne `var` promenljive je jedna funkcija, dok je kod `let` promenljivih to bilo koji blok. S obzirom na to da su petlje blokovi koji nisu funkcije, zaključuje se da će `var` brojač ostati vidljiv izvan petlje i nakon njenog završetka, što neće biti slučaj sa brojačem koji se deklariše korišćenjem ključne reči `let`:

```
for (var i = 0; i < 5; i++) {

      console.log("Hello World");
}

console.log(i);
```

Primer je naveden kako bismo se uverili u ono što je upravo napisano. Nakon izvršavanja ovakve petlje, unutar konzole će biti ispisana vrednosti brojača. Takva vrednost će biti `5`, zato što je sa takvom vrednošću petlja završila svoje izvršavanje. Potpuno je drugačija situacija kada se brojač deklariše kao `let` promenljiva:

```
for (let i = 0; i < 5; i++) {

      console.log("Hello World");

}

Console.log(i);
```

Ovoga puta se kao rezultat poslednje naredbe unutar konzole dobija:

```
Uncaught ReferenceError: i is not defined
```

Jasno je da `let` promenljiva `i` nije vidljiva izvan bloka `for` petlje.

Na kraju, prilikom deklarisanja brojača petlji uvek je bolje koristiti ključnu reč `let`, s obzirom na to da je brojač u najvećem broju slučajeva namenjen isključivo korišćenju unutar bloka petlje.

Pitanje

Prva naredba u deklaraciji for petlje odnosi se na:

- **definisanje brojača**
- proveru uslova
- korekciju vrednosti brojača
- ništa od navedenog

Objašnjenje:

Unutar prve naredbe for petlje vrši se inicijalizacija jednog ili više brojača petlje.

while petlja

Pored `for` petlje, JavaScript poznaje još nekoliko načina da se određeni kôd izvrši više puta.

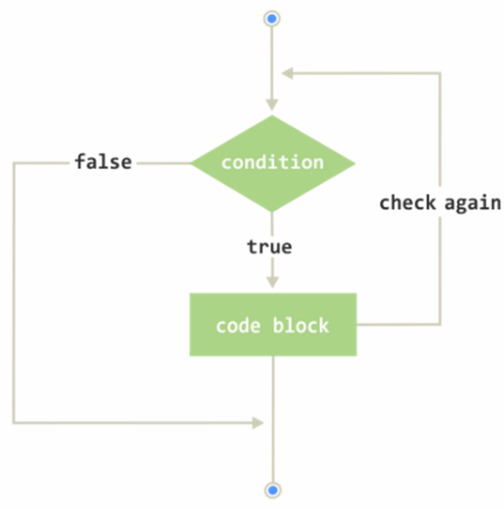
`While` petlja je jedan od tih načina.

`While` petlja izvršava određeni blok koda sve dok je ispunjen uslov za izvršavanje petlje. To praktično znači da ova vrsta petlje nema unapred definisan broj iteracija.

Sintaksa `while` petlje je sledeća:

```
while (condition){
statement
}
```

Sve dok je uslov (`condition`) ispunjen, izvršava se blok koda, odnosno telo petlje. Logika `while` petlje može se ilustrovati slikom 13.4.



Slika 13.4. While petlja

Primer upotrebe `while` petlje je sledeći:

```
var x = 0;

while (x < 5) {

console.log("Hello World");

x = x + 1;

}
```

Na početku se deklariše i inicijalizuje promenljiva `x`, sa vrednošću 0. While petlja se kreira korišćenjem ključne reči `while`, a nakon nje, unutar zagrada se definiše uslov:

```
x<5
```

Ovo praktično znači da će se petlja izvršavati sve dok je vrednost promenljive x manja od 5.

Telo petlje definisano je vitičastim zagradama. Unutar tela petlje vrši se ispisivanje poruke *Hello World*, ali i uvećavanje vrednosti promenljive x za jedan. Da kojim slučajem ova inkrementacija nije navedena, uslov za izvršavanje ove petlje uvek bi bio ispunjen. Na taj način bi se stvorila jedna mrtva petlja, odnosno petlja koja nikada ne bi prestala da se izvršava.

Upravo prikazanim primerom `while` petlje dobija se identičan efekat kao i nešto ranije, korišćenjem `for` petlje:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

do...while petlja

Upravo opisana `while` petlja ni na koji način ne garantuje da će se njeno telo uopšte i izvršiti. Jednostavno, ukoliko uslov nije ispunjen, telo petlje se neće izvršiti nijednom:

```
var x = 6;

while (x < 5) {

console.log("Hello World");

x = x + 1;
}
```

U primeru, vrednost promenljive x je veća od 5, tako da uslov za izvršavanje `while` petlje nije ispunjen, te se kôd unutar tela `while` petlje neće izvršiti nijednom. Ali šta ukoliko je potrebno osigurati da se petlja izvrši makar jednom, bez obzira na ispunjenje uslova?

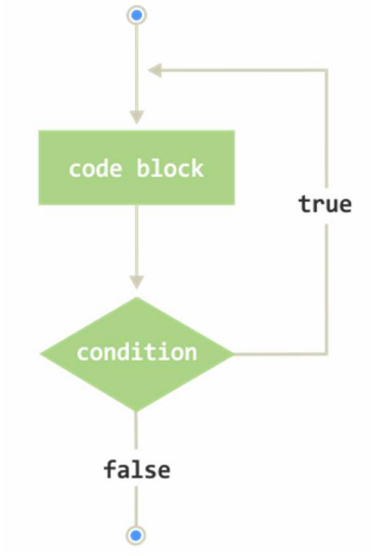
Takvo nešto moglo bi se postići ukoliko bi se provera ispunjenosti uslova smestila nakon bloka koda koji je potrebno izvršiti. Upravo to je osnovna osobina petlje `do…while`.

Sintaksa `do…while` petlje je sledeća:

```
do {

// statements

} while (condition);
```

Na početku se postavlja ključna reč `do`, a nakon nje blok koda koji je oivičen vitičastim zagradama. Nakon bloka, koristi se ključna reč `while` za definisanje uslova izvršenja petlje.

Logika `do...while` petlje se algoritmom može predstaviti kao na slici 13.5.



Slika 13.5. do...while petlja

Sada se prethodni primer može transformisati na sledeći način:

```
var x = 6;

do {

      console.log("Hello World");

      x = x + 1;

} while (x < 5);
```

Rezultat je sledeći:

```
Hello World
```

Zaključak je da se telo petlje izvršilo jednom i pored toga što uslov nije ispunjen. Provera ispunjenosti uslova se obavlja nakon izvršavanja svake iteracije, te stoga `do...while` tip petlje uvek poseduje makar jednu iteraciju.

Naredbe break i continue

Ponekad je potrebno kontrolisati izvršavanje petlje na osnovu nekog uslova i to unutar tela petlje. U takvim situacijama je moguće koristiti naredbe `break` i `continue`.

Naredba `continue` prekida aktuelnu iteraciju i prelazi na sledeću.

Naredba `break` napušta petlju.

Sledeći primer ilustruje upotrebu naredbe `continue`:

```
for (var i = 0; i < 100; i++) {

if (i > 50) {

continue;
}

console.log(i);

}

console.log(i);
```

U primeru je kreirana jedna `for` petlja, koja će imati 100 iteracija (od 0 do 99). Ipak unutar `for` petlje je definisan jedan uslov, kojim se proverava da li je brojač veći od 50. Ukoliko je brojač veći od 50, poziva se naredba `continue`. S obzirom na to da ova naredba odmah prelazi na izvršavanje sledeće iteracije, linija za ispis vrednosti brojača se neće izvršiti u slučaju da je brojač veći od 50. Sve ovo proizvodi sledeći rezultat:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
100
```

Analizom izlaza može se utvrditi da se linija za ispis vrednosti brojača unutar tela petlje izvršava do vrednosti 50. Petlja nastavlja da se izvršava do kraja, ali ne ispisuje vrednosti, u šta se možemo uveriti poslednjim ispisanim brojem. To je broj 100, koji je ispisan od linije koja se nalazi izvan petlje. Ovo praktično znači da je vrednost brojača nakon završetka petlje 100, što dokazuje da se petlja izvršila u celosti.

Za razliku od naredbe `continue`, koja prekida trenutnu i prelazi na sledeću iteraciju, naredba `break` kao svoj efekat ima izlazak iz petlje. Drugim rečima, naredba `break` završava petlju.

Sledeći primer ilustruje upotrebu naredbe `break`:

```
for (var i = 0; i < 100; i++) {

if (i > 50) {

break;

}

console.log(i);

}

console.log(i);
```

Ovoga puta, kôd proizvodi sledeći izlaz:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51
```

Jasno se može videti razlika između naredbi `continue` i `break`. Kada brojač dostigne vrednost 51, aktivira se `if` uslovni blok, unutar koga se nalazi naredba `break`. Izvršavanjem naredbe `break` dolazi do trenutnog izlaska iz petlje, te se prelazi na izvršavanje sledeće naredbe. Zbog toga je poslednja vrednost koja se štampa unutar `for` petlje 50. Nakon izvršavanja `for` petlje, štampa se vrednost 51.

Napomena

U primerima efekata naredbi `break` i `continue` brojači su namerno deklarisani korišćenjem ključne reči `var`, kako bi se njihova vrednost očuvala i nakon završetka petlji. U situacijama u kojima ovakve vrednosti nisu potrebne izvan petlji uvek je bolje koristiti ključnu reč `let`.

for...in i for...of petlje

JavaScript poseduje i dve posebne petlje, koje se mogu koristiti za prolazak kroz nizove. U dosadašnjem toku kursa spomenut je pojam nizova i rečeno je da su nizovi kolekcije više vrednosti. Ipak, nizovi su pojam kome će biti posvećen kompletan naredni modul ovoga kursa, pa će tada biti obrađene i `for...in` i `for...of` petlje.

Rezime

- Petlje, pored grananja, predstavljaju osnovne elemente za postizanje kontrole toka.
- Petlje omogućavaju da se određeni kôd izvrši više puta.
- `for` petlja omogućava da se određeni blok koda izvrši tačan broj puta.
- `while` petlja izvršava određeni blok koda sve dok je ispunjen uslov za izvršavanje petlje.
- `do…while` je specijalna vrsta petlje kod koje se prvo izvršava blok koda, a tek onda proverava i uslov.
- Naredba `continue` prekida aktuelnu iteraciju i prelazi na sledeću.
- Naredba `break` napušta petlju.
- Petlje `for…in` i `for…of` su specijalne petlje namenjene za prolazak kroz nizove i objekte.