Uslovno izvršavanje

Blok koda je pojam koji je definisan u lekciji o osnovnoj leksičkoj strukturi sa početka ovoga kursa. Blokovi su do sada već korišćeni za praktičnu realizaciju funkcionalnog programiranja. Ipak, blokovi koda u programiranju imaju još jedno veoma značajno mesto primene. Oni se koriste za postizanje kontrole toka, što je tema kojoj će biti posvećen modul pred vama.

Kontrola toka

Izvršavanje JavaScript koda obavlja se sleva nadesno i pri tome se napisane izjave (naredbe) izvršavaju redom, jedna za drugom:

```
var x = 1;
var y;
y = x;
```

U prikazanom primeru definisane su tri JavaScript izjave (naredbe). One će se izvršiti redom kojim su napisane, pa će tako prvo biti deklarisana i inicijalizovana promenljiva x, zatim će biti deklarisana promenljiva y i na kraju će vrednost promenljive x biti dodeljena promenljivoj y. Ilustrativno, tok izvršavanja prikazanog koda je kao na slici 12.1.

Slika 12.1. Linearni tok izvršavanja koda

Zamislite sada situaciju u kojoj biste vi mogli da kontrolišete tok izvršavanja prikazanog koda. Na primer, želite da se nakon izvršavanja prve izjave pređe na izvršavanje treće (slika 12.2).

Slika 12.2. Primer nelinearnog izvršavanja (kontrole toka)

Sem u preskakanju, kontrola toka se može ogledati i u ponavljanju jedne naredbe ili više naredbi. Na primeru tri prikazane naredbe sa početka lekcije takvo ponavljanje može izgledati kao na slici 12.3. Nakon izvršavanja poslednje, treće naredbe, izvršavanje se ponovo vraća na prvu naredbu.

Slika 12.3. Primer nelinearnog izvršavanja (kontrole toka)(2)

Upravo prikazani scenariji spadaju u domen kontrole toka, a jezik JavaScript omogućava različite načine za njeno postizanje:

- grananje (uslovno izvršavanje) i
- petlje (ponavljanje).

U modulu koji je pred vama biće obrađeni osnovni postulati kontrole toka.

Grananje (uslovno izvršavanje)

Grananje omogućava da se određene naredbe izvrše samo u slučaju zadovoljenja nekog uslova. Zbog toga je grananje usko povezano sa logičkim tipovima podataka i logičkim operatorima, što su pojmovi koji su obrađeni u lekcijama za nama. Naime, JavaScript omogućava da se kreira više mogućih grana izvršavanja, a da sam odabir grane izvršavanja bude obavljen u zavisnosti od neke kontrolne logičke vrednosti.

JavaScript poznaje dve vrste kondicionalnih (uslovnih) naredbi:

`if...else` grupa naredbi i
`switch`.

if naredba

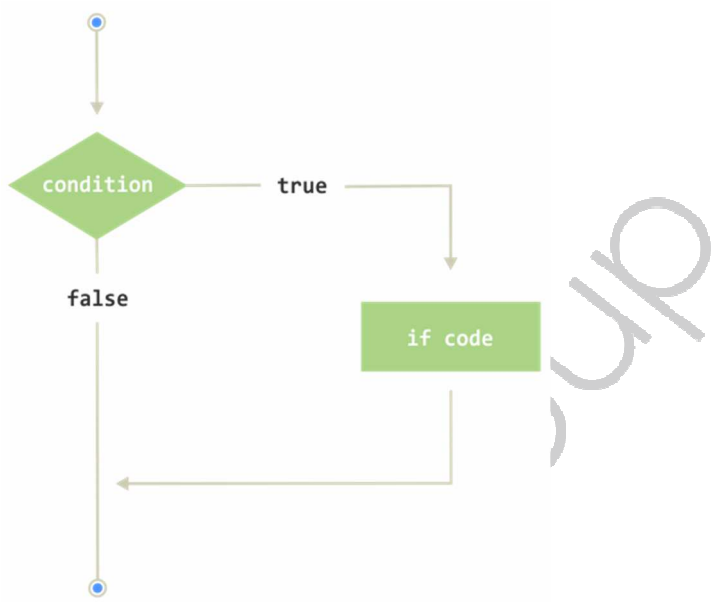
Osnovna naredba grananja u JavaScript jeziku jeste naredba `if`. Ova naredba postoji u praktično svim programskim jezicima, a svrha joj je odslikana u samom nazivu: *ako*.
Reč `if` (*ako*), sama za sebe, nema baš mnogo logike u retorici, a tako je i u programiranju.

Naime, pored ove ključne reči potrebno je postaviti i neki uslov od koga će zavisiti ishod ove naredbe. Tako opšta sintaksa `if` naredbe izgleda ovako:

```
if(condition) {

      //code to execute if condition is met
}
```

Za formiranje `if` naredbe koristi se ključna reč `if`, nakon koje se u zagradama navodi uslov. U slučaju ispunjenja uslova izvršava se blok koda `if` naredbe.

Slika 12.4. ilustruje logiku izvršavanja programa kada se upotrebljava `if` naredba.



Slika 12.4. If naredba

Kao što se sa slike 12.4. može videti, izvršavanje programa dolazi do dela nazvanog *condition* (uslov). U ovom delu se proverava definisani uslov i ukoliko je on ispunjen, izvršava se kôd `if` bloka. U protivnom, `if` blok koda se preskače, a izvršavanje se nastavlja prvom sledećom naredbom. Primer koji ilustruje upotrebu `if` naredbe je sledeći:

```
var speed = 9;

if (speed < 10) {

      console.log("Too slow...");
}
```

Na početku je deklarisana i inicijalizovana promenljiva sa nazivom `speed` i vrednošću `9`.

Nakon ove linije, sledi `if` naredba. Uslov je definisan tako da se proverava da li je vrednost promenljive `speed` manja od 10. Ukoliko je ovakav uslov ispunjen, izvršava se blok koda između vitičastih zagrada, koji u primeru poseduje jednu naredbu.

Da je kojim slučajem vrednost promenljive `speed` bila jednaka broju 10 ili veća, blok koda iz primera se ne bi izvršio.

Logički uslovi i logičke vrednosti

Nešto ranije je rečeno da se unutar zagrada nakon ključne reči `if` navodi uslov koji diktira da li će `if` blok biti izvršen ili neće. Ipak, ukoliko želimo biti u potpunosti precizni, može se reći da se unutar zagrada nakon ključne reči `if` navodi neka logička vrednost (`true` ili `false`):

```
if (true) {
      console.log("Too slow...");
}
```

Naravno, ovakav blok koda uvek će se izvršiti, zato što je unutar zagrada navedena vrednost `true`. Tako se ovakav blok i ne može nazvati uslovnim, jer je verovatnoća njegovog izvršavanja unapred poznata. Stoga se direktno navođenje logičke vrednosti kao uslova grananja vrlo retko koristi. Ipak, bitno je znati da se grananje u svojoj suštini obavlja na osnovi logičkih vrednosti `true` i `false`, pa je i nešto ovako potpuno legitimno napisati. Identično važi i za sve ostale naredbe grananja koje će biti prikazane u nastavku.

if...else naredba

U ovom trenutku se može postaviti jedno pitanje:

Šta ukoliko je potrebno izvršiti određeni blok koda i kada if uslov nije zadovoljen?

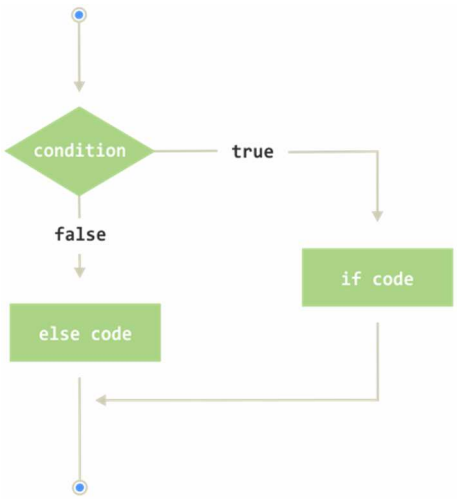
Odgovor na ovo pitanje krije se u upotrebi ključne reči `else`, pa tako naredba `if` postaje naredba `if...else`.

`if...else` je naredba za kontrolu toka koja omogućava definisanje logike koja će se izvršiti ukoliko je uslov ispunjen, ali i ukoliko nije. Sintaksa `if...else` naredbe je sledeća:

```
if (condition) {
      statement_1;
} else {
      statement_2;
}
```

U zagradama nakon `if` ključne reči navodi se uslov (`condition`). U slučaju ispunjenja ovakvog uslova izvršava se prva izjava (`statement_1`), a u protivnom druga (`statement_2`).

Sve ovo ilustrovano je slikom 12.5.



Slika 12.5. If…else naredba

Sa slike 12.5. se može videti da u ovoj situaciji postoje dva bloka koda: *if code* i *else code*.

U zavisnosti od ispunjenja uslova, izvršava se jedan od ova dva bloka koda. Bitno je razumeti da će jedan blok koda morati da se izvrši. Koji će to blok biti zavisi od ishoda uslova.

Primer upotrebe `if else` naredbe:

```
if (1 == 1) {

      console.log("true");

} else {

      console.log("false");
}
```

U primeru je definisan uslov `1==1`. Potpuno je jasno da će uslov biti ispunjen, te se stoga izvršava `if` blok, a unutar konzole ispisuje:

```
True
```

Ipak, drugačija situacija će biti ukoliko se uslov promeni:

```
if (10 == 13) {

      console.log("true");

} else {

      console.log("false");

}
```

Sada se unutar konzole dobija:

```
False
```

Jedna `if...else` naredba može imati samo jedan `if` i samo jedan `else` blok. Uslov se uvek definiše samo na `if` bloku.

if...else if...else naredba

U prethodnim redovima je prikazano da `if...else` naredba može imati samo jedan uslov i to na `if` bloku. Ukoliko je on ispunjen, izvršava se `if`, a u protivnom `else` blok. Sada se postavlja novo pitanje:

Šta se dešava ukoliko je potrebno definisati dodatne blokove koda sa sopstvenim uslovima? Odgovor na ovo pitanje krije se u upotrebi ključnih reči `else if`.

`else if` omogućava da se prilikom kreiranja `if...else` naredbe, definišu dodatni uslovi, pa tako naredba `if...else`, postaje `if...else if...else` naredba.

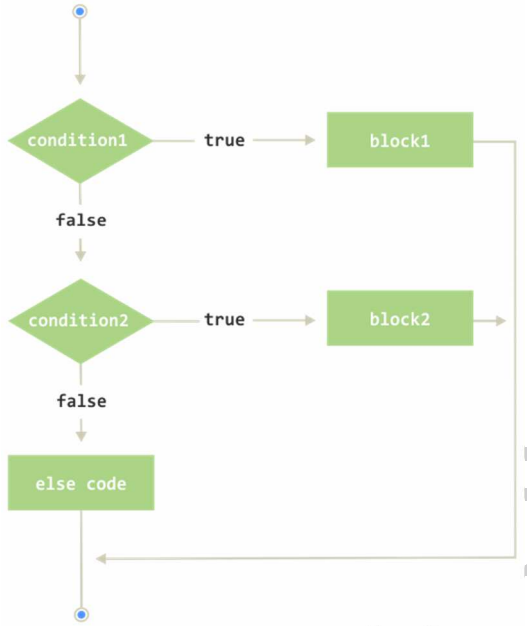
Sintaksa ovakve naredbe je sledeća:

```
if (condition_1) {

      statement_1;

} else if (condition_2) {

      statement_2;

} else if (condition_n) {

      statement_n;

} else {

      statement_last;

}
```

U ovom primeru, pored `if` i `else` blokova, postoje i dva `else if` bloka. Ukoliko je prvi uslov (`condition_1`) ispunjen, izvršiće se izjava `statement_1`. Ukoliko je uslov `condition_2` ispunjen, izvršiće se izjava `statement_2`. Ukoliko je uslov `condition_n` ispunjen, izvršiće se naredba `statement_n`.

Na kraju, ukoliko nijedan uslov nije ispunjen, izvršiće se izjava `statement_last`. Bitno je razumeti da broj `else if` blokova može biti proizvoljan. U primeru ih ima dva, dok je moguće imati bilo koji broj takvih blokova (jedan, dva, tri, četiri...).

Struktura jedne `if...else if...else` naredbe prikazana je slikom 12.6.



Slika 12.6. if…else if…else naredba

Tok izvršavanja koda sa slike 12.6. je sledeći. Izvršavanje koda dolazi do prvog uslova.

Ukoliko je uslov ispunjen, izvršava se `block1` i nakon toga se izlazi iz `if…else if…else` konstrukcije. Ukoliko prvi uslov (`condition1`) nije ispunjen, ispituje se tačnost drugog uslova (`condition2`). Ukoliko je ispunjen, izvršava se `block2`. Na kraju, ukoliko nijedan od uslova nije ispunjen, izvršava se `else` blok koda.

Primer korišćenja `if…else if…else` naredbe je sledeći:

```
var speed = 50;
if (speed < 10) {
      console.log("Too slow...");
} else if (speed <= 80) {
      console.log("Regular speed.");
}else if (speed < 100) {
      console.log("Too fast!");
}else {
      console.log("Incorrect value");
}
```

Nakon izvršavanja prikazanog koda, unutar konzole se dobija:

```
Regular speed.
```

U narednim redovima biće objašnjen postupak koji je doveo do ispisa ovakve vrednosti unutar konzole. Prvom naredbom deklariše se i inicijalizuje promenljiva `speed` sa vrednošću 50. `if…else if…else` naredba poseduje tri uslova. Ukoliko je vrednost promenljive `speed` manja od 10, izvršava se prvi blok koda. S obzirom na to da to nije slučaj, ispituje se istinitost drugog uslova. Drugi uslov je istinit ukoliko je vrednost promenljive `speed` manja ili jednaka broju 80. Ovaj uslov je ispunjen, s obzirom na to da je 50 manje od 80, pa izvršavanje skripte ulazi u prvi `else if` blok i ispisuje se poruka *regular speed*.

Bitno je razumeti još jednu veoma bitno osobinu prikazane konstrukcije. Drugi `else if` blok poseduje sledeći uslov: `speed < 100`. Ukoliko se analizira ovakav uslov, može se zaključiti da je i on ispunjen, s obzirom na to da je 50 manje od 100. Ipak, unutar naredbe `if…else if…else` izvršava se uvek samo jedan blok koda, i to prvi čiji uslov se pokaže kao istinit.

Pitanje

Unutar kog bloka se ne može definisati uslov?

- if
- **else**
- else if

Objašnjenje:

Else blok ne može imati uslov, dok blokovi if i else if imaju uslove.

Switch

Još jedna naredba za kontrolisanje toka koja postoji unutar JavaScript jezika jeste naredba `switch`.

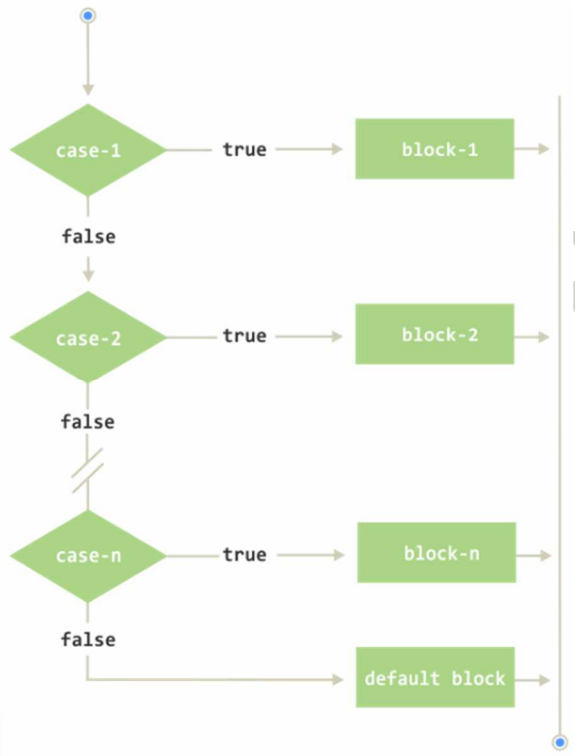
Ova naredba omogućava kontrolisanje toka korišćenjem nešto drugačijeg pristupa:

```
switch (expression) {
      case label_1:
            statements_1
            break;
      case label_2:
            statements_2
            break;
...
      default:
            statements_def
            break;
}
```

`switch` naredba poredi `expression` vrednost sa vrednostima `case` klauzula. U prikazanom primeru, vrednosti `case` klauzula su obeležene se `label_1`, `label_2` itd.

Kada se utvrdi podudaranje `expression` vrednosti sa nekom od `case` klauzula, izvršavanje skripte se preusmerava na taj blok. U slučaju da se ne pronađu podudaranja, izvršava se opcioni `default` blok.

Logika `switch` naredbe ilustrovana je slikom 12.7.



Slika 12.7. Switch naredba

Praktičan primer upotrebe `switch` naredbe:

```
var x = 1;

switch (x) {
      case 0:
            console.log("zero");

            break;
      case 1:
            console.log("one");
            break;
      default:
            console.log("unknown value");

            break;
}
```

U prikazanom primeru kreirana je jedna `switch` naredba koja cifre `0` i `1` pretvara u odgovarajući tekstualni oblik: *zero* i *one*. U zavisnosti od vrednosti promenljive `x`, unutar `switch` naredbe biće obavljeno emitovanje određene vrednosti unutar konzole. Kada je vrednost promenljive `x` jednaka `0`, u konzoli će biti ispisano *zero*. Kada je vrednost promenljive `x` jednaka `1`, u konzoli se ispisuje *one*. Na kraju, ukoliko je vrednost promenljive `x` bilo šta osim brojeva 0 i 1, u konzoli se ispisuje tekst *unknown value*.

Unutar `switch` naredbe obavlja se utvrđivanje jednakosti kontrolne vrednosti (što je u primeru `x`), sa vrednostima pojedinačnih `case` blokova (to su vrednosti nakon ključnih reči `case`). Drugim rečima, `switch` dozvoljava kreiranje logičkih uslova u kojima se proverava jednakost, ali ne i ostali logički uslovi (veće, manje, nejednako...).

Svaki od blokova koda unutar `switch` naredbe na kraju poseduje ključnu reč `break`. Uloga ključne reči `break` je da izvede izvršavanje koda izvan `switch` naredbe. Kada izvršno okruženje naiđe na ovu liniju, završava se izvršavanje `switch` naredbe.

Napomena

Default blok koda se, po konvenciji, postavlja na kraj switch naredbe, ali to ne mora biti pravilo. Ovaj deo koda je moguće postaviti bilo gde.

Višestruko poklapanje kod switch naredbe

Svi primeri u prethodnim redovima odslikavali su ispunjenje samo jednog uslova. Na primer, ukoliko je `x` jednako 1 i samo 1, izvršava se određeni blok koda. Ukoliko je `x` jednako 2 i samo 2, izvršava se drugi blok koda. Ali šta ukoliko je potrebno da se neki blok koda izvrši ukoliko `x` ima bilo koju od vrednosti 1 ili 2? U takvoj situaciji mora se pribeći definisanju višestrukih uslova.

Postavka primera će izgledati ovako:

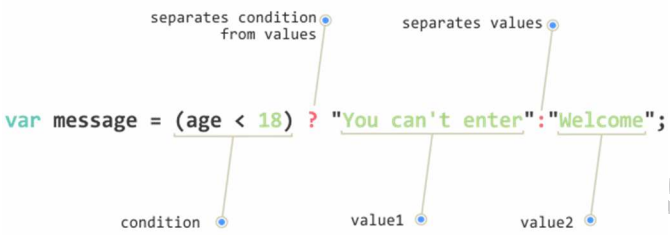
Potrebno je napraviti switch strukturu koja će testirati vrednost promenljive x. Ako x ima vrednost 1 ili 2, na strani napisati „YES", dok za svaku drugu vrednost promenljive x treba ispisati „NO".

```
var x = 1;
switch (x) {
      case 1:

      case 2:
            document.write("YES");
            break;
      default:
            document.write("NO");
            break;
}
```

Na ovaj način, prvi blok koda će se izvršiti kada promenljiva `x` ima vrednost 1 ili 2. Za sve ostale vrednosti izvršava se `default` blok koda.

Ternarni operator

O ternarnom operatoru već je bilo reči u lekciji o logičkim operatorima. Tada je rečeno da je ternarni operator veoma koristan u situacijama kada je potrebno ostvariti kontrolu toka. Da budemo precizniji, korišćenjem ternarnog operatora moguće je promenljivoj dodeliti vrednost na osnovu nekog uslova.



Slika 12.8. Ternarni operator

S obzirom na to da je ternarni operator već obrađen, način njegovog funkcionisanja je jasan. Uslov je `age < 18`, potencijalne vrednosti koje bi mogle da budu dodeljene promenljivoj `message`, su `You can't enter` i `Welcome`. Ukoliko je vrednost promenljive `age` manja od 18, `message` dobija vrednost `You can't enter`. Ukoliko je vrednost promenljive `age` 18 ili više, `message` dobija vrednost `Welcome`.

Ternarni operator je veoma koristan kada je potrebno postaviti podrazumevane vrednosti nekih promenljivih, u slučaju da njihove vrednosti nisu nigde drugde navedene. Na primer, neka vrednost može doći sa korisničke forme na stranici.

Korisnik može da takvu vrednost ne unese, a ternarni operator će se pobrinuti da takva promenljiva dobije svoju podrazumevanu vrednost:

```
var b = (a != null) ? a : 120;
```

Ukoliko `a` ima vrednost `null`, `b` dobija podrazumevanu vrednost 120. Ukoliko je `a` različito od `null`, `b` dobija vrednost `a`.

Ternarni operator se uvek može transformisati u `if...else` naredbu. Tako se logika sa slike 12.8. može transformisati na sledeći način:

```
var message = '';
var age = 19;

if (age < 18) {
      message = "You can't enter";
} else {
      message = "Welcome";
}

console.log(message);
```

Rezultat prikazanog primera biće ispis vrednosti *Welcome* unutar konzole, zato što je vrednost promenljive `age` veća od 18.

Primer – Program za pretvaranje brojeva u nazive dana

U nastavku će biti prikazan primer JavaScript programa koji za prosleđenu numeričku vrednost između 1 i 7 vraća naziv dana u sedmici. Program izgleda ovako:

```
let day = prompt("Please enter number between 1 and 7: ");

switch (day) {
	case "1":
		alert("Monday");
		break;
	case "2":
		alert("Tuesday");
		break;
	case "3":
		alert("Wednesday");
		break;
	case "4":
		alert("Thursday");
		break;
	case "5":
		alert("Friday");
		break;
	case "6":
		alert("Saturday");
		break;
	case "7":
		alert("Sunday");
		break;
	default:
		alert("Unknown value");
		break;
}
```

Program započinje preuzimanjem vrednosti od korisnika. Za obavljanje takvog posla koristi se metoda `prompt()`. Nakon preuzimanja vrednosti, poruka za korisnika se formira unutar jedne `switch` uslovne naredbe. Kontrolna vrednost jeste promenljiva `day`, unutar koje je smeštena vrednost dobijena od korisnika. Uslovna naredba `switch` poseduje 7 pojedinačnih slučajeva (`case`) i jedan podrazumevani (`default`). Za svaku od vrednosti od 1 do 7 postoji po jedan `case`. Unutar takvih slučajeva se obavlja prikaz odgovarajuće vrednosti.

Podrazumevani (`default`) slučaj će se aktivirati kada korisnik unese neku vrednost koja nije u rasponu od 1 do 7. U programu se operiše `string` vrednostima. `string` podatak se dobija od `prompt()` metode, a i unutar `switch` naredbe se obavlja poređenje `string` vrednosti.

Rezime

- Blok koda je osnovni gradivni element mnogih jezičkih konstrukcija u JavaScriptu, a predstavlja više izjava grupisanih u jednu logičku celinu.
- Kontrola toka omogućava uticanje na linearnost izvršavanja koda.
- Grananje omogućava da se određene naredbe izvrše samo u slučaju zadovoljenja nekog uslova.
- Osnovna naredba grananja u JavaScript jeziku jeste naredba `if`.
- `if...else` naredba koristi se za kontrolu toka tako što omogućava da se definiše logika koja će se izvršiti i ukoliko je uslov ispunjen, ali i ukoliko nije.
- Naredba `if…else if…else` omogućava definisanje većeg broja uslova.
- `switch` naredba omogućava postizanje kontrole toka, na osnovu podudarnosti vrednosti.
- Ternarnim operatorom je takođe moguće postići kontrolu toka na sličan način kao i `if…else` blokom.