

Pripremni materijal za laboratorijsku vježbu br. 3: JavaScript uvod

1. JavaScript

1.1. Šta je JavaScript?

JavaScript je prvobitno kreiran da „oživi web stranice“. Programi pisani u ovom jeziku se zovu skripte. Mogu se napisati direktno u HTML-u web stranice i pokrenuti automatski kako se stranica učitava. Za pokretanje skripti im nije potrebna posebna priprema ili kompilacija.

1.2. Šta može raditi JavaScript u browseru?

Mogućnosti JavaScripta uvelike zavise od okruženja u kojem radi. Na primjer, Node.js podržava funkcije koje dozvoljavaju JavaScript-u da čita/piše proizvoljne datoteke, izvršava mrežne zahtjeve itd. JavaScript u pretraživaču može učiniti sve što se odnosi na manipulaciju web stranicom, interakciju s korisnikom i web serverom. Na primjer, JavaScript u pretraživaču može:

- Dodati novi HTML stranici, promijeniti postojeći sadržaj, promijeniti stilove
- Reagovati na radnje korisnika, pokretati akcije klikom miša, pokretima kursomom miša ili pritiskom na tipke
- Slati zahtjeve preko mreže na udaljene servere, preuzimati i postavljati datoteke (tzv. AJAX i COMET tehnologije)
- Preuzeti i postaviti kolačiće (eng. *cookies*)
- Zapamtiti podatke na strani klijenta (tzv. „lokalna memorija“).

1.3. Po čemu je JavaScript jedinstven?

Postoje najmanje tri sjajne karakteristike JavaScript-a:

1. Potpuna integracija sa HTML/CSS
2. Jednostavne stvari se rade jednostavno
3. Podržan je od strane svih glavnih web pretraživača

JavaScript je jedina tehnologija u sklopu web pretraživača koja kombinuje ove tri stvari. To je ono što JavaScript čini jedinstvenim. Zbog toga predstavlja najrasprostranjeniji alat za kreiranje interfejsa web pretraživača. Naravno, JavaScript se može koristiti i za kreiranje servera, mobilnih aplikacija itd.

1.4. Hello, world!

JavaScript programi se mogu ubaciti gotovo bilo gdje u HTML dokument pomoću taga `<script>`. Primjer:

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

</body>

</html>
```

Tag `<script>` sadrži JavaScript kod koji se automatski izvršava kada ga pretraživač obradi. Pokretanjem ovog koda će se u alertu ispisati “Hello, world!”. Inače, `alert` prikazuje poruku i čeka da korisnik pritisne “OK”.

Međutim, mnogo je bolja praksa koristiti vanjske (eksterne) skripte koje se korištenjem `<script>` taga uključuju u HTML kod. Ako imamo dosta JavaScript koda, možemo ga staviti u zasebnu datoteku. Datoteke sa skriptama se uključuju u HTML korištenjem atributa *src*:

```
<script src="/path/to/script.js"></script>
```

Ovdje `/path/to/script.js` predstavlja apsolutnu putanju na kojoj se nalazi skripta. Naravno, moguće je navesti i relativnu putanju kao što je `src="script.js"` ili `src="./script.js"` koje označavaju da se skripta naziva `script.js` nalazi u trenutnom folderu. Za dodavanje više datoteka sa skriptama, potrebno je koristiti više `<script>` tagova:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
```

Jedan `<script>` tag ne može imati i *src* atribut i kod. Ovo neće ispravno raditi:

```
<script src="file.js">
```

```
    alert(1); // sadrzaj se ignorira jer postoji src  
</script>
```

Prethodni primjer je moguće rastaviti na dva `<script>` taga kako bi bio ispravan (što je prikazano u nastavku):

```
<script src="file.js"></script>  
<script>  
    alert(1);  
</script>
```

2. Varijable

Varijable služe za pohranu podataka. Mogu se deklarirati ključnim riječima `var`, `let` i `const`.

Za kreiranje varijable `message` će se koristiti ključna riječ `let`. Naredba u nastavku kreira (tj. deklarira) varijablu s imenom `message`:

```
let message;
```

Sada možemo staviti neke podatke u varijablu koristeći operator dodjeljivanja `=`

```
let message;  
  
message = 'Hello'; // u varijablu message se pohranjuje string 'Hello'
```

String je sada pohranjen na lokaciju u memoriji koja je povezana sa varijablom. Možemo mu pristupiti koristeći ime varijable:

```
let message;  
message = 'Hello!';  
  
alert(message); // prikazuje sadržaj varijable
```

Također, deklaracija varijable i dodjela se mogu kombinirati u jednu liniju na sljedeći način:

```
let message = 'Hello!'; // deklariranje i dodjela vrijednosti  
  
alert(message); // Hello!
```

Koncept varijabli se može lako shvatiti ako je zamislimo kao “kutiju” za pohranu podataka, sa naljepnicom jedinstvenog imena koja se nalazi na njoj. Npr. promjenljiva `message` se može zamisliti kao kutija sa oznakom “message” koja sadrži vrijednost “Hello!” (slika 2.1.)



Slika 2.1. Varijabla message prikazana kao “kutija” u kojoj je pohranjena vrijednost “Hello!”

U kutiju možemo staviti bilo koju vrijednost. Također, vrijednost možemo mijenjati koliko god puta želimo:

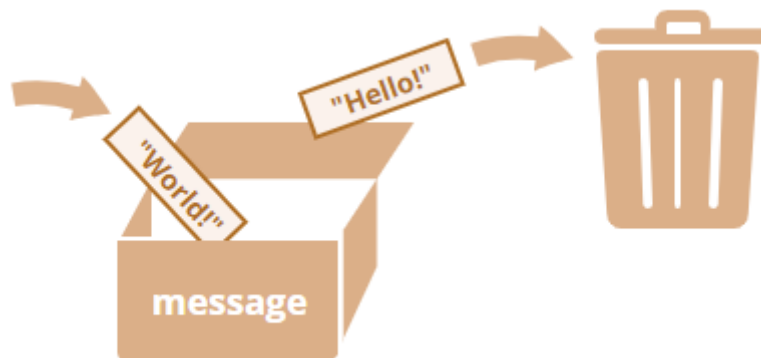
```
let message;

message = 'Hello!';

message = 'World!'; // promjena vrijednosti

alert(message);
```

Kada se promijeni vrijednost varijable, stara vrijednost se uklanja iz te varijable.



Slika 2.2. Uklanjanje vrijednosti “Hello!” iz varijable message koja je predstavljena kao “kutija” i postavljanje nove vrijednosti “World!” u “kutiju”

Dvostruko deklariranje izaziva grešku

Varijablu treba deklarirati samo jednom. Ponovljena deklaracija iste varijable je greška:

```
let message = "This";

// ponovljeno 'let' vodi do greske
let message = "That"; // SyntaxError: 'message' has already been declared
```

Dakle, varijabla se treba deklarirati jednom a zatim se na nju referiramo bez `let`.

2.1. Nazivi varijabli

Postoje dva ograničenja prilikom imenovanja varijabli u JavaScript-u:

1. Naziv mora sadržavati samo slova, cifre ili simbole `$` i `_`
2. Prvi znak ne smije biti cifra

Primjeri validnih naziva varijabli:

```
let userName;
let test123;
```

Kada naziv varijable sadrži više riječi, najčešće se koristi tzv. *camelCase*. To znači da riječi idu jedna za drugom pri čemu svaka riječ, izuzev prve, počinje velikim početnim slovom (npr. `myVeryLongName`).

S obzirom da je moguće koristiti i znak dolara (\$) kao i donju crtu (_) u nazivima varijabli, validni su i sljedeći nazivi:

```
let $ = 1; // deklariranje varijable naziva "$"  
let _ = 2; // deklariranje varijable naziva "_"
```



```
alert($ + _); // ispisuje se 3
```

Primjeri neispravnih naziva varijabli:

```
let 1a; // naziv ne može početi brojem
```



```
let my-name; // crtice '-' nisu dozvoljene u nazivima varijabli
```

Bitno je obratiti pažnju na mala i velika slova u nazivima varijabli. Varijable naziva `apple` i `APPLE` su dvije različite varijable.

Postoji lista rezerviranih riječi, koje se ne mogu koristiti kao nazivi varijabli jer ih koristi sam programski jezik. Na primjer: `let`, `class`, `return` i `function` su rezervirane riječi. Sljedeći isječak koda daje sintaksnu grešku:

```
let let = 5; // varijabla ne može imati naziv "let", greska!  
let return = 5; // varijabla ne može imati naziv "return", greska!
```

2.2. Konstante

Za deklariranje konstantnih (nepromjenljivih) varijabli, koristi se `const` umjesto `let`:

```
const myBirthday = '18.04.1982';
```

Varijable deklarirane ključnom riječju `const` se nazivaju “konstante”. Njima vrijednost ne može biti ponovno dodijeljena. Pokušaj ponovne dodjele vrijednosti konstantnoj varijabli uzrokuje grešku:

```
const myBirthday = '18.04.1982';
```



```
myBirthday = '01.01.2001'; // greska, Assignment to constant variable
```

Kada je programer siguran da varijabla nikada ne treba promijeniti vrijednost, onda je takvu varijablu moguće deklarirati ključnom riječju `const`.

2.3. var i let

U starijim JavaScript kodovima, može se pronaći i ključna riječ `var` koja se koristi umjesto `let`:

```
var message = 'Hello';
```

Ključna riječ `var` je skoro ista kao ključna riječ `let`. I ona deklarira varijablu, ali to čini na malo drugačiji način. Postoje razlike između `var` i `let` ključne riječi, ali u ovoj vježbi nećemo ulaziti u detalje.

3. Tipovi podataka

Vrijednost u JavaScript programskom jeziku je uvijek određenog tipa. Npr. vrijednost može biti string ili broj. Postoji 8 standardnih tipova podataka u JavaScript-u. Varijable u JavaScript-u mogu biti bilo kojeg tipa. Na primjer, varijabla u jednom momentu može sadržavati string a nakon toga (u nekom drugom momentu) u ovu varijablu može biti pohranjen broj.

```
// ovo je ispravno
let message = "hello";
message = 123456;
```

Programski jezici koji ovo podržavaju se nazivaju “dinamički”. JavaScript je jedan od primjera ovih jezika. Kod dinamičkih jezika postoje tipovi podataka, ali varijable nisu vezane ni za jedan od njih.

3.1. Brojevi (eng. *number*)

```
let n = 123;
n = 12.345;
```

Brojčani tip podataka predstavlja i cijele brojeve (eng. *integer*) i brojeve sa pokretnim zarezom (eng. *float*). Postoje mnoge operacije koje su podržane sa brojevima kao što su množenje (operator `*`), dijeljenje (operator `/`), sabiranje (operator `+`), oduzimanje (operator `-`) itd.

Pored standardnih brojeva, postoje i tzv. “posebne brojčane vrijednosti” koje pripadaju ovom tipu podataka. Te vrijednosti su: **Infinity**, **-Infinity** and **NaN**.

- **Infinity** predstavlja matematičku beskonačnost. Može se dobiti kao rezultat dijeljenja nulom:

```
alert( 1 / 0 ); // Infinity
```

Ili se može direktno navesti kao:

```
alert( Infinity ); // Infinity
```

- **NaN** predstavlja grešku prilikom proračuna. Rezultat je neispravne ili nedefinirane matematičke operacije, kao što je:

```
alert( "not a number" / 2 ); // NaN, pogresno dijeljenje
```

Bilo koja dalja operacija sa vrijednošću **NaN** vraća **NaN**:

```
alert( NaN + 1 ); // NaN
alert( 3 * NaN ); // NaN
alert( "not a number" / 2 - 1 ); // NaN
```


Dakle, ako bilo gdje postoji `NaN` u matematičkim izrazima, on se propagira na kompletan rezultat (jedini izuzetak je `NaN ** 0` koji rezultira 1).

Matematičke operacije u JavaScript programskom jeziku su sigurne. Ovo znači da možemo raditi bilo koju matematičku operaciju kao što je dijeljenje nulom, korištenje stringova kao brojeva itd. Skripta nikada neće prestati raditi. U najgorem slučaju se dobije NaN kao rezultat.

3.2. BigInt

U JavaScript-u, tip "broj" se ne može bezbijedno koristiti za predstavljanje cjelobrojnih vrijednosti većih od $(2^{53}-1)$ (tj. `9007199254740991`), ili manjih od $-(2^{53}-1)$ za negativne vrijednosti. `BigInt` tip je nedavno dodan jeziku za predstavljanje cijelih brojeva proizvoljne dužine. `BigInt` vrijednost se kreira dodavanjem `n` na kraj cijelog broja:

```
// "n" na kraju znaci da je rijec o BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

Ovaj tip se rijetko koristi te o njemu neće biti detaljnije diskusije.

3.3. String

String u JavaScript programskom jeziku mora biti okružen navodnicima.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

Postoje tri tipa navodnika:

1. dvostruki navodnici: `"Hello"`
2. jednostruki navodnici: `'Hello'`
3. kosi navodnici: ``Hello``

Dvostruki i jednostruki navodnici su “jednostavni” znakovi navodnika i praktično ne postoji nikakva razlika između njih u JavaScript-u.

Međutim, kosi navodnici omogućavaju ugrađivanje vrijednosti i izraza u stringove na način da ih obilježimo sa `${...}`. Ovo se naziva *template strings*. Na primjer:

```
let name = "John";

// ugradjivanje vrijednosti
alert( `Hello, ${name}!` ); // Hello, John!

// ugradjivanje izraza
alert( `the result is ${1 + 2}` ); // the result is 3
```

Izraz unutar `${...}` se izračunava (tj. evaluira) nakon čega dobiveni rezultat postaje dio stringa. Ovdje možemo ubaciti bilo šta poput varijable kao što je `name`, aritmetičkog izraza kao što je `1+2` ili nešto složenije.

Naravno, ovo je moguće uraditi samo unutar kosih navodnika.

```
// dvostruki navodnici ovdje ne urade nista
alert( "the result is ${1 + 2}" ); // ispis: the result is ${1 + 2}
```

Bitno je naglasiti da u JavaScript-u ne postoji poseban tip koji označava svaki pojedini znak (tj. karakter). U nekim drugim programskim jezicima, kao što su C i Java, postoji tip “char”. U JavaScript-u to nije slučaj, već postoji samo jedan tip i to `string`. String se može sastojati od nula znakova (tj. string je prazan), jednog ili više znakova.

3.4. Logički tip (*Boolean*)

Boolean tip ima samo dvije moguće vrijednosti: `true` i `false`. Najčešće se koristi za pohranjivanje da/ne vrijednosti, pri čemu `true` označava nešto što je ispravno a `false` označava nešto što nije ispravno. Na primjer:

```
let nameFieldChecked = true; // da, polje za naziv je provjereno
let ageFieldChecked = false; // ne, polje za godine nije provjereno
```

Boolean vrijednosti se javljaju i kao rezultat poređenja:

```
let isGreater = 4 > 1;

alert( isGreater ); // true (rezultat poredjenja je “da”)
```

3.5. Null vrijednost

Posebna vrijednost `null` ne pripada niti jednom tipu podataka koji su prethodno opisani. Ova vrijednost formira vlastiti (odvojeni) tip koji sadrži samo `null` kao moguću vrijednost:

```
let age = null;
```

U JavaScript-u, `null` nije referenca na neki nepostojeći objekt ili na null pokazivač kao što je to slučaj u nekim drugim programskim jezicima. Ovo je, ustvari, samo posebna vrijednost koja definira “ništa”, “prazno” ili “nepoznatu vrijednost”. Prethodni isječak koda govori da je dob (`age`) nepoznata.

3.6. Undefined vrijednost

Posebna vrijednost `undefined` se, također, ističe u odnosu na ostale tipove podataka. Slično kao i `null`, ova vrijednost formira vlastiti tip. Značenje vrijednosti `undefined` jeste da “vrijednost nije dodijeljena”.

Ako je neka varijabla deklarirana a nije joj dodijeljena vrijednost, tada je njena vrijednost `undefined`.

```
let age;  
  
alert(age); // ispisuje "undefined"
```

Tehnički, moguće je eksplicitno dodijeliti vrijednost `undefined` nekoj varijabli:

```
let age = 100;  
  
// promjena vrijednosti varijable na undefined  
age = undefined;  
  
alert(age); // ispisuje "undefined"
```

Ovo se ne preporučuje. Obično se koristi vrijednost `null` za dodjelu “prazne” ili “nepoznate” vrijednosti nekoj varijabli, dok je `undefined` rezervirano kao početna (default, incijalna) vrijednost varijablama kojima ništa nije dodijeljeno.

3.7. Objekti i simboli

Objekat (`object`) predstavlja poseban tip. Svi ostali tipovi se nazivaju “primitivni” jer mogu sadržavati samo jednu vrijednost (bilo da je to neki broj, string ili tome slično). Za razliku od njih, objekti se koriste za pohranjivanje kolekcija podataka. U narednoj vježbi će jedna posebna sekcija biti posvećena objektima i načinima na koji se oni koriste u JavaScript-u.

Tip simbol (`symbol`) se koristi za kreiranje jedinstvenih identifikatora za objekte. Kada je riječ o ovom tipu podataka, nećemo ulaziti u detalje.

3.8. Pretvaranje tipova

U većini situacija, operatori i funkcije automatski urade pretvaranje vrijednosti u ispravan tip. Na primjer, `alert` će automatski pretvoriti bilo koju vrijednost u string i prikazati je. Međutim, postoje situacije u kojima je potrebno obaviti eksplicitno pretvaranje tipova.

3.8.1. Pretvaranje vrijednosti u tip string

Pretvaranje vrijednost u tip string se dešava kada je potrebna vrijednost u formi stringa. Na primjer, `alert(value)` će to uraditi kako bi prikazao vrijednost. Moguće je pozvati funkciju `String(value)` kako se obavilo pretvaranje vrijednosti u tip string:

```
let value = true;  
alert(typeof value); // boolean  
  
value = String(value); // sada je vrijednost string "true"  
alert(typeof value); // string
```

Pretvaranje vrijednosti u tip string je očito u većini slučajeva. Vrijednost `false` postaje `"false"`, `null` postaje `"null"`.

3.8.2. Pretvaranje vrijednosti u numerički tip

Pretvaranje vrijednost u numerički tip se automatski dešava u matematičkim funkcijama i izrazima. Npr. kada se operator dijeljenja `/` primijeni nad vrijednostima koje nisu brojčane:

```
alert( "6" / "2" ); // 3, stringovi se pretvaraju u brojeve
```

Moguće je koristiti funkciju `Number(value)` kako bismo obavili eksplicitnu pretvorbu vrijednosti u brojčani tip:

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // postaje broj 123

alert(typeof num); // number
```

Eksplicitno pretvaranje je obično potrebno uraditi kada se učitavaju vrijednosti iz nečega što je bazirano na stringu kao što je tekst forme u kojoj očekujemo unos brojčane vrijednosti.

Ako string nije validan broj, rezultat pretvorbe će biti `NaN`:

```
let age = Number("an arbitrary string instead of a number");

alert(age); // NaN, neuspjela pretvorba tipova
```

Pravila za pretvaranje vrijednosti u numerički tip su:

Vrijednost	Postaje
<code>undefined</code>	<code>NaN</code>
<code>null</code>	<code>0</code>
<code>true</code> i <code>false</code>	<code>1</code> i <code>0</code>
<code>string</code>	Razmaci (tzv. <i>whitespaces</i>) uključujući razmake (eng. <i>spaces</i>), tabove (eng. <i>tabs</i>), nove redove (eng. <i>newlines</i>) itd. sa početka i kraja će biti uklonjeni. Ako je preostali string prazan, tada je rezultat <code>0</code> . U suprotnom, iščitava se broj iz stringa. Ako dođe do greške, rezultat je <code>NaN</code> .

Primjeri:

```
alert( Number(" 123 ") ); // 123
alert( Number("123z") ); // NaN (greska zbog znaka "z")
alert( Number(true) ); // 1
alert( Number(false) ); // 0
```

Bitno je obratiti pažnju da `null` i `undefined` imaju drugačije ponašanje: `null` postaje `0`, dok `undefined` postaje `NaN`.

3.8.3. Pretvaranje vrijednosti u boolean tip

Ovo je najjednostavniji tip pretvaranja vrijednosti. Pretvorba se dešava prilikom obavljanja logičkih operacija ali se može obaviti i eksplicitno pozivom `Boolean(value)`.

Pravila za pretvaranje vrijednosti u boolean tip su:

- Vrijednosti koje su intuitivno “prazne” kao što su `0`, prazan string, `null`, `undefined` i `NaN` postaju `false`
- Ostale vrijednosti postaju `true`

Primjeri:

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("hello") ); // true
alert( Boolean("") ); // false
```

Obratite pažnju da je string sa vrijednošću “0” se evaluira kao `true`. Neki drugi programski jezici tretiraju “0” kao `false`. Međutim, u JavaScript programskom jeziku je svaki neprazni string uvijek `true`.

```
alert( Boolean("0") ); // true
alert( Boolean(" ") ); // razmaci, takodjer true
```

4. Poređenja

Iz matematike su poznati mnogi operatori poređenja. U JavaScript-u ih pišemo na sljedeći način:

- Veće/manje od: `a > b`, `a < b`
- Veće ili jednako/manje ili jednako: `a >= b`, `a <= b`
- Jednako: `a == b`, obratite pažnju da dvostruki znak jednakosti (`==`) označava provjeru na jednakost, dok se jednostruki znak jednakosti (`=`) koristi za dodjelu vrijednosti varijabli
- Nije jednako: U matematici se koristi oznaka \neq , ali JavaScript koristi oznaku `a != b`.

4.1. Rezultat je boolean

Svi operatori poređenja kao rezultat vraćaju boolean vrijednost:

- `true` – označava “da”, “tačno”, “istinito”
- `false` – označava “ne”, “netačno”, “neistinito”

Primjeri:

```
alert( 2 > 1 ); // true (tacno)
alert( 2 == 1 ); // false (netacno)
alert( 2 != 1 ); // true (tacno)
```

Rezultat poređenja se može dodijeliti nekoj varijabli, baš kao i bilo koja druga vrijednost:

```
let result = 5 > 4; // dodjela rezultata poredjenja
alert( result ); // true
```

4.2. Poređenje stringova

Kako bi se provjerilo da li je string “veći” od nekog drugog stringa, JavaScript koristi tzv. “leksikografski” poredak. Dakle, stringovi se porede slovo po slovo.

Primjeri:

```
alert( 'Z' > 'A' ); // true
alert( 'Glow' > 'Glee' ); // true
alert( 'Bee' > 'Be' ); // true
```

Algoritam za poređenje dva stringa je jednostavan. Obavi se poređenje prvih znakova oba stringa. Ako je prvi znak prvog stringa veći (ili manji) od prvog znaka drugog stringa, tada je prvi string veći (ili manji) od drugog i tu se algoritam zaustavlja. U suprotnom, ako su prvi znakovi oba stringa isti, vrši se poređenje drugih znakova oba stringa na isti način kao što je

to urađeno za prve znakove. Ovaj postupak se ponavlja sve do kraja jednog od dva stringa. Ako oba stringa imaju istu dužinu, onda su oni isti. U suprotnom, duži string je veći.

U prvom primjeru na početku ovog potpoglavlja, poređenje 'Z' > 'A' daje rezultat već nakon prvog koraka opisanog postupka.

Drugo poređenje (tj. poređenje stringova 'Glow' i 'Glee') zahtijeva više koraka pošto se stringovi porede slovo po slovo:

- G je isto kao G.
- l je isto kao l.
- o je veće od e. Postupak se ovdje zaustavlja. Prvi string je veći od drugog.

Predstavljeni algoritam poređenja je sličan onome koji se koristi u rječnicima ili telefonskim imenicima, ali nije potpuno isti. Na primjer, bitno je da li je riječ o malom ili velikom slovu. Veliko slovo "A" nije isto kao malo slovo "a". Ali koje je veće? Odgovor je: malo slovo "a". Razlog leži u činjenici da malo slovo ima veći indeks u internoj tabeli kodiranja koju koristi JavaScript (tzv. *Unicode*).

4.3. Poređenje različitih tipova podataka

Prilikom poređenja vrijednosti različitih tipova, JavaScript radi pretvaranje vrijednosti u brojčane vrijednosti.


Primjer:

```
alert( '2' > 1 ); // true, string '2' postaje broj 2
alert( '01' == 1 ); // true, string '01' postaje broj 1
```

Za boolean vrijednosti, `true` postaje `1` a `false` postaje `0`.

Primjeri:

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

 Interesantna posljedica ove činjenice je da su istovremeno:

- Dvije vrijednosti su jednake
- Jedna od njih je `true` kao boolean a druga je `false` kao boolean

Primjer:

```
let a = 0;
alert( Boolean(a) ); // false

let b = "0";
alert( Boolean(b) ); // true
```

```
alert(a == b); // true!
```

Iz perspektive JavaScript programskog jezika, ovaj rezultat je potpuno normalan. Provjera na jednakost vrši pretvaranje vrijednosti u numerički tip zbog čega "0" postaje 0 dok eksplicitno pretvaranje vrijednosti u tip boolean koristi pravila koja su opisana prethodno.

4.4. Stroga jednakost

Standardna provjera jednakosti putem operatora `==` ima problem. Ova provjera nije u stanju razlikovati 0 i `false`.

```
alert( 0 == false ); // true
```

Slično se dešava i sa praznim stringom:

```
alert( '' == false ); // true
```

Ovo se dešava jer se operandi različitih tipova pretvaraju u brojeve prije poređenja operatorom `==`. Prazan string, kao i vrijednost `false` postaju nula.

Operator stroge jednakosti `===` provjerava jednakost operandima bez pretvaranja tipova. To znači da ako su varijable `a` i `b` različitih tipova, tada `a===b` odmah vraća `false` bez da pokuša uraditi pretvaranje tipova.

Primjer:

```
alert( 0 === false ); // false, jer su tipovi razliciti
```

Postoji i strogi operator nejednakosti, tj. operator `!==`.

4.5. Poređenje sa null i undefined

Prilikom poređenja `null` ili `undefined` sa drugim vrijednostima, ponašanje nije toliko intuitivno.

4.5.1. Za strogo poređenje jednakosti ===

Ove vrijednosti su različite jer su različitih tipova:

```
alert( null === undefined ); // false
```


4.5.2. Za standardno poređenje ==

U ovom slučaju postoji posebno pravilo. Ove dvije vrijednosti su jednake jedna drugoj (u smislu operatora jednakosti ==), ali nisu jednake niti jednoj drugoj vrijednosti.

```
alert( null == undefined ); // true
```

4.5.3. Za matematička i ostala poređenja < > <= >=

`null/undefined` se pretvaraju u brojeve: `null` postaje `0`, dok `undefined` postaje `NaN`.

Postoje neke neobične situacije kada primijenimo ova pravila.

4.5.4. Čudan rezultat poređenja `null` sa `0`

Probajmo uporediti `null` sa vrijednošću nula:

```
alert( null > 0 ); // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

Dobiveni rezultati su matematički čudni. Posljednji rezultat označava da je `null` veće ili jednako od `0`, ali i poređenje operatorom veće i operatorom jednako su vratili `false` iako bi barem jedan od njih trebao vratiti `true`.

Razlog ovakvog ponašanja leži u činjenici da provjera na jednakost == i poređenja >, <, >=, <= rade na različite načine. Poređenja vrše pretvaranje `null` u broj, tretirajući ovu vrijednost kao `0`. Zbog toga je rezultat (3) `null>=0` istinit (tj. `true`) a rezultat (1) `null>0` je neistinit (tj. `false`).

Nasuprot ovome, provjera na jednakost == za `undefined` i `null` je takva da, bez ikakve pretvorbe, ove dvije vrijednosti su jednake jedna drugoj, ali nisu jednake ničemu drugom. Zbog toga rezultat (2) `null==0` je neistinit (tj. `false`).

4.5.5. “Neuporedljivi” `undefined`

Vrijednost `undefined` se ne bi trebala porediti sa drugim vrijednostima.

```
alert( undefined > 0 ); // false (1)
alert( undefined < 0 ); // false (2)
alert( undefined == 0 ); // false (3)
```

Poređenjem sa `undefined` su dobiveni prikazani rezultati jer:

- Poređenja (1) i (2) vraćaju `false` jer `undefined` biva pretvoreno u `NaN` a `NaN` je posebna numerička vrijednost koja vraća `false` za bilo koje poređenje
- Provjera jednakosti (3) vraća `false` jer je `undefined` jedino jednako vrijednostima `null`, `undefined` i niti jednoj drugoj vrijednosti

4.6. Kako izbjeći probleme prilikom poređenja?

Bitno je zapamtiti:

- Obratiti posebnu pažnju na svako poređenje sa `undefined/null` vrijednostima izuzev striktne jednakosti
- Ne koristiti poređenja `>=`, `>`, `<`, `<=` sa varijablama koje bi mogle biti `null/undefined`, ako niste sigurni da znate šta radite. Ako varijabla može imati ove vrijednosti, najbolje je provjeru uraditi odvojeno.

5. Uslovi

Ponekad je potrebno obaviti različite akcije na osnovu različitih uslova. U JavaScript-u je to moguće uraditi na različite načine kao što je `if` uslov ili operator `?`.

5.1. If uslov

Uslov `if (...)` evaluira uslov u zagradama i ako je rezultat `true`, onda se izvršava blok koda koji slijedi.

Primjer:

```
let year = prompt('In which year was ECMAScript-2015 specification published?', '');

if (year == 2015) alert( 'You are right!' );
```

U prethodnom primjeru, uslov je jednostavna provjera na jednakost (`year == 2015`). Naravno, uslovi mogu biti kompleksniji od ovoga.

Ako želimo izvršiti više od jedne naredbi, moramo “omotati” kod vitičastim zagradama:

```
if (year == 2015) {
    alert( "That's correct!" );
    alert( "You're so smart!" );
}
```

Preporučuje se korištenje vitičastih zagrada svaki put kada se koristi `if` uslov, bez obzira na to koliko se naredbi nalazi u bloku koda. Na ovaj način kod postaje čitljiviji.

5.1.1. Boolean pretvorbe

Naredba `if (...)` evaluira izraz u zagradama i konvertuje rezultat u boolean. Prisjetimo se pravila pretvaranja tipova iz poglavlja 3.8.3.:

- Broj `0`, prazan string `""`, `null`, `undefined` i `NaN` postaju `false`. Zbog toga ih nazivamo “*falsy*” vrijednostima.
- Ostale vrijednosti postaju `true`, zbog čega ih zovemo “*truthy*” vrijednostima.

Dakle, sljedeći kod unutar uslova se nikada neće izvršiti:

```
if (0) { // 0 je falsy
    ...
}
```

dok će se kod unutar sljedećeg uslova uvijek izvršiti:

```
if (1) { // 1 je truthy
  ...
}
```

Prilikom provjere `if` uslova, moguće je proslijediti i unaprijed izračunatu boolean vrijednost kao:

```
// provjera na jednakost se evaluira sa true ili false
let cond = (year == 2015);

if (cond) {
  ...
}
```

5.2. Else

`if` uslov može sadržavati i opcionalni `else` blok. Ovaj blok će se izvršiti kada je uslov *falsy*.

Primjer:

```
let year = prompt('In which year was the ECMAScript-2015
specification published?', '');

if (year == 2015) {
  alert( 'You guessed it right!' );
} else {
  alert( 'You are wrong!' ); // bilo koja vrijednost osim 2015
}
```

5.3. Višestruki uslovi

Ponekad je potrebno testirati nekoliko varijanti uslova. Klauza `else if` nam omogućava pisanje više različitih uslova.

Primjer:

```
let year = prompt('In which year was the ECMAScript-2015
specification published?', '');

if (year < 2015) {
  alert( 'Too early...' );
} else if (year > 2015) {
  alert( 'Too late' );
} else {
```

```
    alert( 'Exactly!' );  
}
```

U prethodnom isječku koda JavaScript prvo provjerava da li je godina manja od 2015 (`year<2015`). Ako se ovo evaluira kao *falsy*, onda se prelazi na sljedeći uslov, tj. prelazi se na provjeru da li je godina veća od 2015 (`year>2015`). Ako se i ovo evaluira kao *falsy*, onda se prikazuje posljednji `alert`.

Moguće je napisati više `else if` blokova. Također, posljednji `else` blok je opcionalan.

6. Logički operatori

U JavaScript-u postoje četiri logička operatora: `||` (ILI), `&&` (I), `!` (NE), `??` (tzv. *nullish coalescing*). Ovdje će biti opisana prva tri operatora.

Iako se nazivaju „logičkim“, mogu se primijeniti na vrijednosti bilo kojeg tipa, a ne samo na boolean. Njihov rezultat također može biti bilo koje vrste.

6.1. Operator `||` (ILI, OR)

Operator OR se piše sa dvije uspravne linije:

```
result = a || b;
```

U klasičnom programiranju, logički operator OR je namijenjen samo za manipulaciju logičkim vrijednostima. Ako je bilo koji od njegovih argumenata istinit, vraća `true`, u suprotnom vraća `false`.

Međutim, u JavaScript-u je ovaj operator nešto moćniji. Najprije, pogledajmo šta se dešava sa boolean vrijednostima.

Postoje četiri moguće logičke kombinacije:

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false
```

Kao što vidimo, rezultat je uvijek `true` osim u slučaju kada su oba operanda `false`. Ako operand nije logičkog tipa, tada se pretvara u boolean za evaluaciju. Na primjer, broj `1` se tretira kao `true`, a broj `0` kao `false`:

```
if (1 || 0) { // ponasa se isto kao da pise ( true || false )
  alert( 'truthy!' );
}
```

Većinom se operator OR koristi prilikom testiranja u sklopu `if` uslova kako bi se provjerilo da li je barem jedan od iskaza tačan:

```
let hour = 9;

if (hour < 10 || hour > 18) {
  alert( 'The office is closed.' );
}
```

Naravno, može se proslijediti više uslova za provjeru:

```
let hour = 12;
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert( 'The office is closed.' ); // vikend je
}
```

6.1.1. OR || pronalazi prvu vrijednost koja je *truthy*

Do sada opisani način na koji radi ovaj operator je uobičajen. U nastavku će biti opisane neke specifičnosti kada se dodaju određene karakteristike JavaScript programskog jezika.

Proširena varijanta algoritma radi na način koji će biti opisan u nastavku. Neka postoji više vrijednosti između kojih je operator OR:

```
result = value1 || value2 || value3;
```

Operator OR radi sljedeće:

- Evaluira operande s lijeva na desno
- Svaki operand pretvara u boolean. Ako je rezultat `true`, zaustavlja se i vraća originalnu vrijednost tog operanda
- Ako su svi operandi evaluirani (tj. svi su bili *falsy*), vraća posljednji operand

Vrijednost se vraća u izvornom obliku, bez pretvaranja tipova. Drugim riječima, lanac OR operatora vraća prvu *truthy* vrijednost ili vraća posljednju vrijednost ako nije pronađena niti jedna *truthy* vrijednost.

Primjeri:

```
alert( 1 || 0 ); // 1 (1 je truthy)

alert( null || 1 ); // 1 (1 je prva truthy vrijednost)
alert( null || 0 || 1 ); // 1 (prva truthy vrijednost)

alert( undefined || null || 0 );
// 0 (sve je falsy, vraća se posljednja vrijednost)
```

Ovo kreira razne mogućnosti upotrebe operatora OR izuzev onih koje su uobičajene za druge programske jezike:

1. Dobivanje prve *truthy* vrijednosti iz liste varijabli ili izraza:

Na primjer, neka imamo varijable *firstName*, *lastName* i *nickName*. Možemo koristiti OR da se odabere ona varijabla koja sadrži neke podatke ili da se prikaže “Anonymous” ako nijedna od prethodnih varijabli nema podešenu vrijednost:

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";

// ispis ce biti: SuperCoder
alert( firstName || lastName || nickName || "Anonymous");
```

Da su sve varijable bile *falsy*, ispis bi bio “Anonymous”.

2. “Kratkospojna” evaluacija:

Još jedna karakteristika OR operatora je takozvana “kratkospojna” evaluacija. To znači da OR obrađuje svoje argumente dok se ne dostigne prva *truthy* vrijednost, a zatim se vrijednost odmah vraća, čak i bez provjeravanja drugog argumenta. Važnost ove karakteristike postaje očigledna ako operand nije samo vrijednost, već izraz sa nekom “nuspojavom”, kao što je dodjela varijabli ili poziv funkcije.

U primjeru ispod, ispisuje se samo druga poruka:

```
true || alert("not printed");
false || alert("printed");
```

U prvoj liniji koda, OR operator zaustavlja evaluaciju odmah nakon što dođe do *truthy* vrijednosti, tako da se upozorenje ne prikazuje. Ovo se ponekad koristi kako bi se izvršile neke naredbe samo kada je uslov sa lijeve strane OR operatora *falsy*.

6.2. Operator && (I, AND)

Operator AND se piše sa dva znaka “ampersand” &&:

```
result = a && b;
```

U klasičnom programiranju, AND vraća *true* ako su oba operanda *truthy* a *false* u suprotnom:

```
alert( true && true );    // true
alert( false && true );   // false
alert( true && false );   // false
alert( false && false );  // false
```


Primjer sa `if` uslovom:

```
let hour = 12;
let minute = 30;

if (hour == 12 && minute == 30) {
  alert( 'The time is 12:30' );
}
```

Baš kao i OR, bilo koja vrijednost je dozvoljena kao operand logičkog operatora AND:

```
if (1 && 0) { // evaluira se kao true && false
  alert( "won't work, because the result is falsy" );
}
```

6.2.1. AND `&&` pronalazi prvu *falsy* vrijednost

Neka postoji više vrijednosti između kojih je operator AND:

```
result = value1 && value2 && value3;
```

Operator AND radi sljedeće:

- Evaluira operande s lijeva na desno
- Svaki operand pretvara u boolean. Ako je rezultat `false`, zaustavlja se i vraća originalnu vrijednost tog operanda
- Ako su svi operandi evaluirani (tj. svi su bili *truthy*), vraća posljednji operand

Drugim riječima, AND vraća prvu *falsy* vrijednost ili posljednju vrijednost ako nije pronađena niti jedna *falsy* vrijednost. Dakle, pravila su slična operatoru OR s tim da operator AND vraća prvu *falsy* vrijednost dok OR vraća prvu *truthy* vrijednost.

Primjeri:

```
// ako je prvi operand truthy,
// AND vraća drugi operand:
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5

// ako je prvi operand falsy,
// AND vraća taj operand. Drugi operand se ignorise
alert( null && 5 ); // null
alert( 0 && "no matter what" ); // 0
```

Naravno, moguće je zaredom proslijediti više vrijednosti. U narednom primjeru se vraća prva *falsy* vrijednost:

```
alert( 1 && 2 && null && 3 ); // null
```

Kada su sve vrijednosti *truthy*, vraća se posljednja:

```
alert( 1 && 2 && 3 ); // 3, jer je to posljednja vrijednost
```

6.2.2. Operator AND && ima veću prednost od operatora OR ||

Kod

```
a && b || c && d
```

se, u suštini, tretira isto kao kad su izrazi sa AND operatorom bili navedeni u zagradama:

```
(a && b) || (c && d)
```

Obratite pažnju da nije preporučljivo koristiti OR i AND operatore umjesto `if` uslova. Dakle, tamo gdje je potrebno provjeravati neke uslove koristite `if` a ako želite saznati rezultate logičkih operacija onda koristite operatore OR i AND.

6.3. Operator ! (NEGACIJA, NOT)

Logički operator NOT se piše sa znakom uzvika `!`. Sintaksa je prilično jednostavna:

```
result = !value;
```

Operator prihvata jedan argument i radi sljedeće:

1. Konvertuje operand u logički tip: `true/false`
2. Vraća inverznu vrijednost

Primjer:

```
alert( !true ); // false  
alert( !0 ); // true
```

Dvostruki operator NOT se ponekad koristi za pretvaranje vrijednosti u logički tip:

```
alert( !! "non-empty string" ); // true  
alert( !! null ); // false
```

Prvi operator NOT pretvara vrijednost u boolean i vraća inverznu vrijednost, a drugi NOT radi ponovnu inverziju. Na kraju, rezultat ovoga je obična konverzija vrijednosti u boolean.

Postoji malo opširniji način da se postigne isti rezultat - korištenje ugrađene Boolean funkcije:

```
alert( Boolean("non-empty string") ); // true  
alert( Boolean(null) ); // false
```

Operator NOT ima najveću prednost od svih drugih logičkih operatora, tako da se on uvijek izvršava prije operatora OR i AND.

7. Petlje: for i while

Često je potrebno ponavljati radnje. Na primjer, nekada je potrebno jednostavno pokretanje istog koda za svaki broj od 1 do 10. Petlje su način da se isti kod ponovi više puta.

7.1. while petlja

`while` petlja ima sljedeću sintaksu:

```
while (uslov) {  
    // kod  
    // tzv. "tijelo petlje"  
}
```

Dok je uslov *truthy*, izvršava se kod iz tijela petlje. Na primjer, petlja ispod daje vrijednosti `i` dok je `i < 3`:

```
let i = 0;  
while (i < 3) { // prikazuje se 0, pa 1, pa 2  
    alert( i );  
    i++; i = i+1;  
}
```

Jedno izvođenje tijela petlje se naziva *iteracija*. Petlja u gornjem primjeru ima tri iteracije. Da prethodni primjer ne sadrži `i++`, petlja bi se ponavljala (u teoriji) zauvijek. U praksi, web pretraživač pruža načine za zaustavljanje takvih petlji, a u JavaScript-u na serverskoj strani možemo zaustaviti ("ubiti") proces.

Bilo koji izraz ili varijabla može biti uslov petlje, tj. ne moraju biti samo poređenja. Uslov se procjenjuje i pretvara u boolean u sklopu iskaza `while`. Na primjer, kraći način za pisanje `while (i != 0)` je `while (i)`:

```
let i = 3;  
// kada i postane 0, uslov postane falsy i petlja se zaustavlja  
while (i) {  
    alert( i );  
    i--;  
}
```

7.2. for petlja

`for` petlja je složenija, ali je i najčešće korištena petlja. Ima sljedeći izgled:

```
for (pocetak; uslov; kraj) {  
    // ... tijelo petlje ...  
}
```

Značenje dijelova petlje će biti demonstrirano na jednom primjeru. Petlja ispod pokreće `alert(i)` za `i` od `0` do `3` (ali ne uključujući `3`):

```
for (let i = 0; i < 3; i++) { // prikazuje se 0, pa 1, pa 2  
    alert(i);  
}
```

Analizirajmo `for` iskaz dio po dio:

početak	<code>let i = 0</code>	Izvršava se jednom nakon ulaska u petlju.
uslov	<code>i < 3</code>	Provjerava se prije svake iteracije petlje. Ako je netačno, petlja se zaustavlja.
tijelo	<code>alert(i)</code>	Pokreće se iznova i iznova dok je stanje <i>truthy</i> .
korak	<code>i++</code>	Izvršava se nakon tijela u svakoj iteraciji petlje.

U nastavku je prikazano šta se tačno dešava u prethodnom primjeru:

```
// for (let i = 0; i < 3; i++) alert(i)  
  
// pocetak  
let i = 0  
// if uslov → pokreni tijelo petlje i korak  
if (i < 3) { alert(i); i++ }  
// if uslov → pokreni tijelo petlje i korak  
if (i < 3) { alert(i); i++ }  
// if uslov → pokreni tijelo petlje i korak  
if (i < 3) { alert(i); i++ }  
// ...kraj, jer je sada i == 3
```

7.3. Prekid petlje (eng. *break*)

Petlja se završava kada njeno stanje postane *falsy*. Ali moguće je prisilno uraditi izlaz iz petlje u bilo kojem trenutku koristeći specijalnu `break` direktivu.

Na primjer, petlja ispod traži od korisnika unos niza brojeva, koja se "prekida" kada se ne unese broj:

```

let sum = 0;

while (true) {

    let value = +prompt("Enter a number", '');

    if (!value) break; // (*)

    sum += value;

}
alert( 'Sum: ' + sum );

```

Direktiva `break` se aktivira na liniji (*) ako korisnik unese prazan red ili otkaže unos. Odmah zaustavlja petlju, prebacujući kontrolu programa na prvu liniju nakon petlje. U ovom slučaju je to linija koja sadrži `alert`. Kombinacija beskonačne petlje i prekida po potrebi je odlična za situacije kada se stanje petlje mora provjeriti ne na početku ili na kraju petlje, već u sredini ili čak na nekoliko mjesta u sklopu njenog tijela.

7.4. Nastavak u sljedeću iteraciju petlje (eng. *continue*)

Direktiva `continue` je "blaga verzija" direktive `break`. Ne zaustavlja čitavu petlju. Umjesto toga, zaustavlja trenutnu iteraciju i prisiljava petlju da započne novu (ako uslov to dozvoljava). Ovu direktivu možemo koristiti ako smo završili s trenutnom iteracijom i želimo ići na sljedeću. Petlja ispod koristi `continue` za ispis samo neparnih vrijednosti:

```

for (let i = 0; i < 10; i++) {

    // ako je true, preskoci preostali dio tijela petlje
    if (i % 2 == 0) continue;

    alert(i); // ispisuje se 1, pa 3, 5, 7, 9
}

```

Za parne vrijednosti `i`, `continue` direktiva zaustavlja izvršavanje tijela i prebacuje kontrolu na sljedeću iteraciju `for` petlje (tj. prelazi se na iteraciju sa sljedećim brojem). Dakle, `alert` se poziva samo za neparne vrijednosti.

8. Funkcije

Često je potrebno izvesti sličnu radnju na mnogim mjestima unutar skripte. Na primjer, potrebno je prikazati poruku određenog izgleda kada se posjetitelj prijavi, odjavi a možda i na još nekom drugom mjestu. Funkcije su glavni "gradivni blokovi" programa. One dozvoljavaju da se kod poziva mnogo puta bez ponavljanja koda. U sklopu vježbe su već prikazani neki primjeri ugrađenih funkcija, kao što su `alert(poruka)`, `prompt(poruka, default)` i `confirm(pitanje)`. Međutim, možemo kreirati i vlastite funkcije.

8.1. Deklaracija funkcija

Za kreiranje funkcije se može koristiti deklaracija funkcije koja ima sljedeći izgled:

```
function showMessage() {  
    alert( 'Hello everyone!' );  
}
```

Najprije se navodi ključna riječ `function` nakon koje ide naziv funkcije, zatim lista parametara unutar zagrada (parametri su inače odvojeni zarezima, ali u primjeru iznad lista parametara je prazna) dok na kraju ide kod funkcije, koji se naziva "tijelo funkcije" i navodi se između vitičastih zagrada.

```
function nazivFunkcije(parametar1, parametar2, ... parametarN) {  
    // tijelo  
}
```

Ovako kreirana funkcija se može pozvati korištenjem njenog imena, tj. `showMessage()`.

Na primjer:

```
function showMessage() {  
    alert( 'Hello everyone!' );  
}  
  
showMessage();  
showMessage();
```

Poziv `showMessage()` izvršava kod funkcije. Pošto je poziv naveden dva puta, poruka će se ispisati dva puta. Ovaj primjer jasno pokazuje jednu od glavnih svrha funkcija: izbjegavanje dupliciranja koda. Ako ikada bude potrebno promijeniti poruku ili način na koji je prikazana, dovoljno je samo izmijeniti kod na jednom mjestu, tj. u funkciji koja ga šalje.

8.2. Lokalne varijable

Varijabla deklarirana unutar funkcije vidljiva je samo unutar te funkcije. Primjer:

```
function showMessage() {
    let message = "Hello, I'm JavaScript!"; // lokalna varijabla

    alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Greska! Varijabla je lokalna za funkciju
```

8.3. Vanjske varijable

Funkcija može pristupiti i vanjskim varijablama. Primjer:

```
let userName = 'John';

function showMessage() {
    let message = 'Hello, ' + userName;
    alert(message);
}

showMessage(); // Hello, John
```

Funkcija ima potpuni pristup vanjskoj varijabli. Dakle, funkcija može modificirati vanjsku varijablu:

```
let userName = 'John';

function showMessage() {
    userName = "Bob"; // (1) promjena vanjske varijable

    let message = 'Hello, ' + userName;
    alert(message);
}

alert( userName ); // ispis je John prije poziva funkcije

showMessage();

// ispis je Bob, jer je funkcija promijenila vrijednost varijable
// userName
alert( userName );
```


Vanjska varijabla se koristi samo ako ne postoji lokalna. Ako je varijabla istog imena deklarirana unutar funkcije onda ona “zasjenjuje” vanjsku. Na primjer, u kodu ispod funkcija koristi lokalnu varijablu `userName`. Ovdje se vanjska varijabla `userName` zanemaruje:

```
let userName = 'John';

function showMessage() {
  let userName = "Bob"; // deklariranje lokalne varijable

  let message = 'Hello, ' + userName; // Bob
  alert(message);
}

// funkcija ce kreirati i koristiti vlastitu varijablu userName
showMessage();

// John, nepromijenjeno, jer funkcija
// nije pristupala vanjskoj varijabli
alert( userName );
```

8.4. Parametri funkcije

Putem parametara je moguće proslijediti proizvoljne podatke funkcijama. U primjeru ispod, funkcija ima dva parametra: `from` i `text`:

```
function showMessage(from, text) { // parametri su: from, text
  alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // ispis je Ann: Hello! (*)
showMessage('Ann', "What's up?"); // ispis je Ann: What's up? (**)
```

Kada se funkcija pozove u redovima `(*)` i `(**)`, date vrijednosti se kopiraju u lokalne varijable `from` i `text`. Nakon toga ih funkcija koristi.

Još jedan primjer je prikazan u nastavku. Neka imamo varijablu `from` i prosljeđujemo je funkciji. Obratite pažnju da funkcija mijenja varijablu `from`, ali promjena se ne vidi izvana, jer funkcija uvijek dobiva kopiju vrijednosti.

```
function showMessage(from, text) {

  from = '*' + from + '*'; // “uljepšavanje” varijable "from"

  alert( from + ': ' + text );
```

```

}

let from = "Ann";

showMessage(from, "Hello"); // ispis je *Ann*: Hello

// vrijednost varijable "from" je ista,
// funkcija je modificirala lokalnu kopiju
alert( from ); // ispis je Ann

```

Kada se vrijednost proslijeđuje kao parametar funkcije, naziva se *argument*. Drugim riječima, ispravno je koristiti sljedeće termine:

- Parametar je varijabla navedena unutar zagrada u deklaraciji funkcije (to je termin koji se koristi za vrijeme deklaracije funkcije).
- Argument je vrijednost koja se proslijeđuje funkciji kada je pozvana (to je termin koji se koristi za vrijeme poziva funkcije).

Funkcije deklariramo navodeći njihove parametre, a zatim ih pozivamo proslijeđivanjem argumenata. U gornjem primjeru, moglo bi se reći: "funkcija `showMessage` je deklarirana sa dva parametra, a zatim pozvana sa dva argumenta: `from` i `"Hello"`".

8.5. Povratne vrijednosti funkcije (eng. *return value*)

Funkcija može vratiti vrijednost nazad u kod koji je pozvao, kao rezultat. Najjednostavniji primjer bi bila funkcija koja sabira dvije vrijednosti:

```

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

let result = sum(1, 2);
alert( result ); // 3

```

Direktiva `return` se može nalaziti na bilo kojem mjestu unutar funkcije. Kada izvršavanje funkcije dođe do ove direktive, funkcija se zaustavlja, a vrijednost se vraća kodu koji je pozvao funkciju (u prethodnom primjeru se povratna vrijednost dodjeljuje varijabli `result`).

U jednoj funkciji može postojati više mjesta na kojima se javlja direktiva `return`. Primjer:

```

function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
    return confirm('Do you have permission from your parents?');
  }
}

```

```

    }
}

let age = prompt('How old are you?', 18);

if ( checkAge(age) ) {
    alert( 'Access granted' );
} else {
    alert( 'Access denied' );
}

```

Moguće je koristiti direktivu `return` bez navođenja povratne vrijednosti. Ovo uzrokuje da se odmah izađe iz funkcije. Primjer:

```

function showMovie(age) {
    if ( !checkAge(age) ) {
        return;
    }

    alert( "Showing you the movie" ); // (*)
    // ...
}

```

U kodu iznad, ako `checkAge(age)` vrati `false`, funkcija `showMovie` neće nastaviti do dijela koda gdje se nalazi `alert`.

Funkcija sa praznom `return` direktivom kao i funkcija bez `return` direktive vraća `undefined`. Dakle, ako funkcija ne vrati vrijednost, to je isto kao da vraća `undefined`:

```

function doNothing() { /* prazno */ }

alert( doNothing() === undefined ); // true

```

Prazna `return` direktiva je isto kao da piše `return undefined`:

```

function doNothing() {
    return;
}

alert( doNothing() === undefined ); // true

```