

Razvoj programske podpore za web

- predavanja -

5. JavaScript

1/3

Creative Commons



- slobodno smijete:
 - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
 - **prerađivati** djelo
- pod sljedećim uvjetima:
 - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
 - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
 - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licenčne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

Uvod u programski jezik JavaScript (1)

- **JavaScript** je programski jezik koji se tipično koristi u izradi web-aplikacija
 - Iako mu je ime slično Javi, značajno se razlikuje od nje
 - Najčešće se spominje kao programski jezik za klijentsku stranu web-aplikacija
 - U novije vrijeme ga vidamo kao i programski jezik za serversku stranu aplikacije, mobilne aplikacije itd.
- JavaScript je **skriptni programski jezik**
 - Skriptni jezici se ne prevode uz pomoć prevoditelja, već se njihov kôd parsira i odmah izvodi
 - JavaScript se izvodi na tzv. stroju za JavaScript (eng. JavaScript engine), koji se ponekad nalazi i u okviru internetskog preglednika i tada zove tzv. **virtualnim strojem JavaScripta** (eng. JavaScript virtual machine)

Uvod u programski jezik JavaScript (2)

- JavaScript je široj javnosti po prvi puta predstavljen 1995. godine, izvorno se zvao LiveScript te je bio značajno manje popularan od Jave
 - Danas su oba jezika vrlo popularna
- JavaScript je standardiziran tzv. **specifikacijom ECMAScripta**
 - Prva inačica je izrađena 1997. godine
 - Trenutno je aktualno 12. izdanje ECMAScripta iz 2021. godine (tzv. ES2021 ili ES12).
 - Ova specifikacija je važna zato što daje smjernice koje slijede programeri koji izrađuju strojeve koji izvode JavaScript (eng. JavaScript engine)

Uvod u programski jezik JavaScript (3)

- Primjeri tzv. strojeva koji izvode JavaScript (eng. JavaScript engine):

Naziv stroja koji izvodi JavaScript	Internetski preglednici koji koriste stroj
Chakra	Microsoft Edge (stari)
SpiderMonkey	Firefox
Chrome V8	Google Chrome Microsoft Edge (novi) Opera
JavaScriptCore (Nitro)	Safari

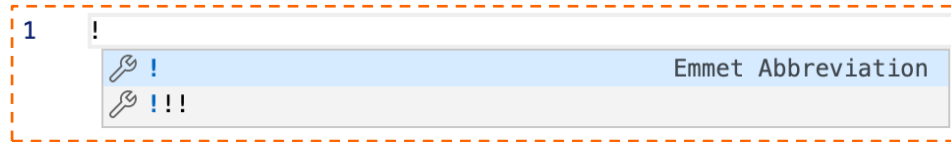
- Npr. Chrome V8 parsira kôd u JavaScriptu, stvara vlastito sintaksno stablo i uz pomoć tzv. interpretera V8 stvara program u vlastitom međujeziku (bytecode) te ga optimizira i prebacuje u strojni kôd platforme, te potom izvršava

Uvod u programski jezik JavaScript (4)

- JavaScript se vrlo često izvodi u internetskom pregledniku što značajno ograničava skup dozvoljenih radnji (uglavnom zbog sigurnosnih razloga)
 - Rad s datotekama je vrlo ograničen
 - Komunikacija dvije stranice (npr. otvorene u dvije različite instance internetskog preglednika) je vrlo ograničena
 - JavaScript kôd može komunicirati sa vlastitim poslužiteljem, ali mu je komunikacija s drugim poslužiteljima ograničena
- Kôd u mnogim jezicima više razine se prevodi (ako je to potrebno) u JavaScript
 - CoffeeScript – zajednica programera u prog. jeziku Ruby
 - TypeScript – Microsoft
 - Flow – Facebook
 - Dart – Google

Podsjetnik: Kako započeti pisanje HTML-a

- Korištenjem predložaka u razvojnoj okolini:
 - Npr. Visual studio code *html template shortcut*
 - **Prečac: ! -> ENTER**

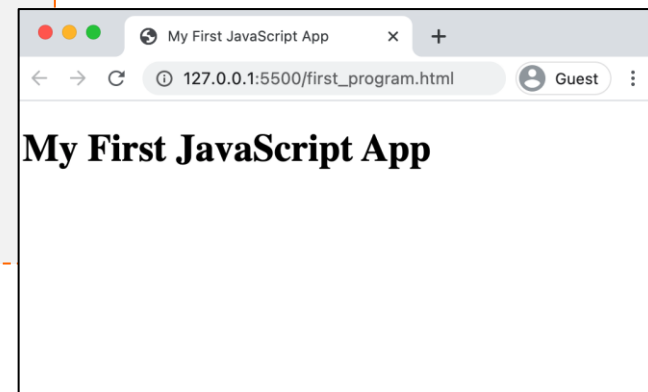
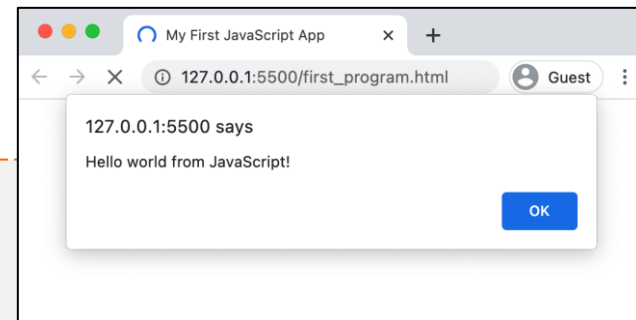


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My First JavaScript App</title>
  </head>
  <body>
  </body>
</html>
```

Prvi program u JavaScriptu

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My First JavaScript App</title>
  </head>
  <body>
    <h1>My First JavaScript App</h1>

    <script>
      alert("Hello world from JavaScript!");
    </script>
  </body>
</html>
```



- Unutar bloka `<script></script>` naveden je kôd u JavaScriptu
- JavaScript funkcija `alert` zadužena je za ispis poruke korisniku
- Nakon potvrde (klika na „U redu”) prikazuje se stranica (HTML)

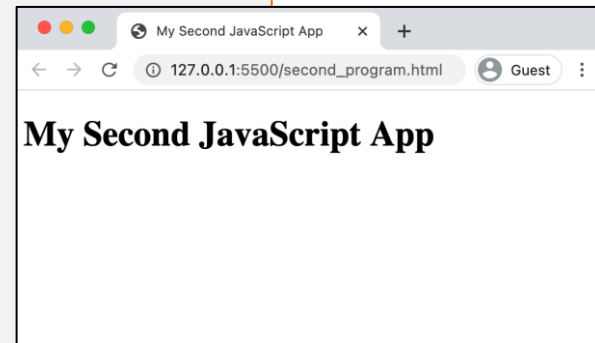
Drugi program u JavaScriptu

- Funkcija za ispis 10 nasumičnih (eng. random) cijelih brojeva u intervalu od 0 do 9

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My Second JavaScript App</title>
  </head>
  <body>
    <h1>My Second JavaScript App</h1>
    <script>
      function randomNumbers() {
        for (var i = 0; i < 10; i++) {
          console.log(Math.floor(Math.random() * 10));
        }
      }
      randomNumbers();
    </script>
  </body>
</html>
```

Definicija funkcije

Poziv funkcije



Ispisa nema zbog toga što se ne vidi konzola

Windows, Linux

- Ctrl + Shift + I

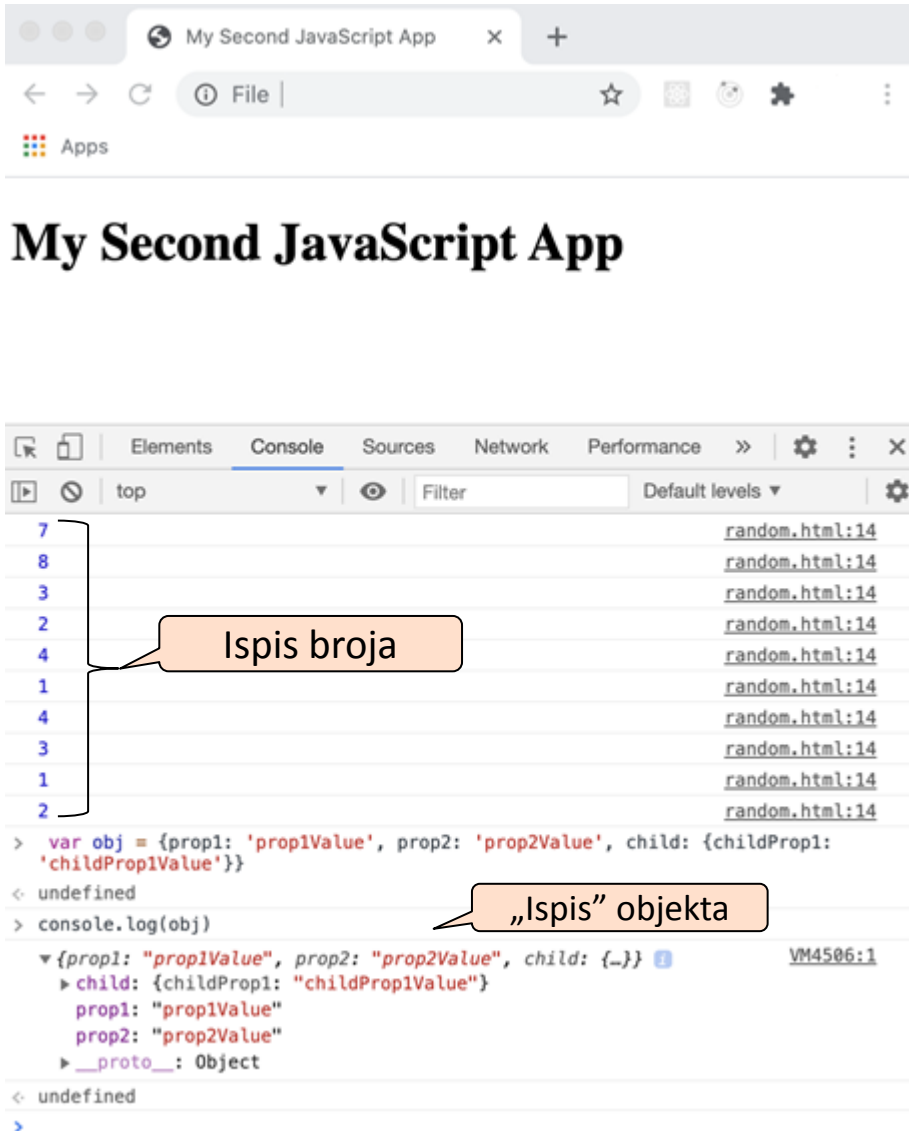
Mac

- Command + Option + J

Iduća prikaznica

- Mogućnost pisanja proizvoljnog kôda unutar tagova `<script></script>`

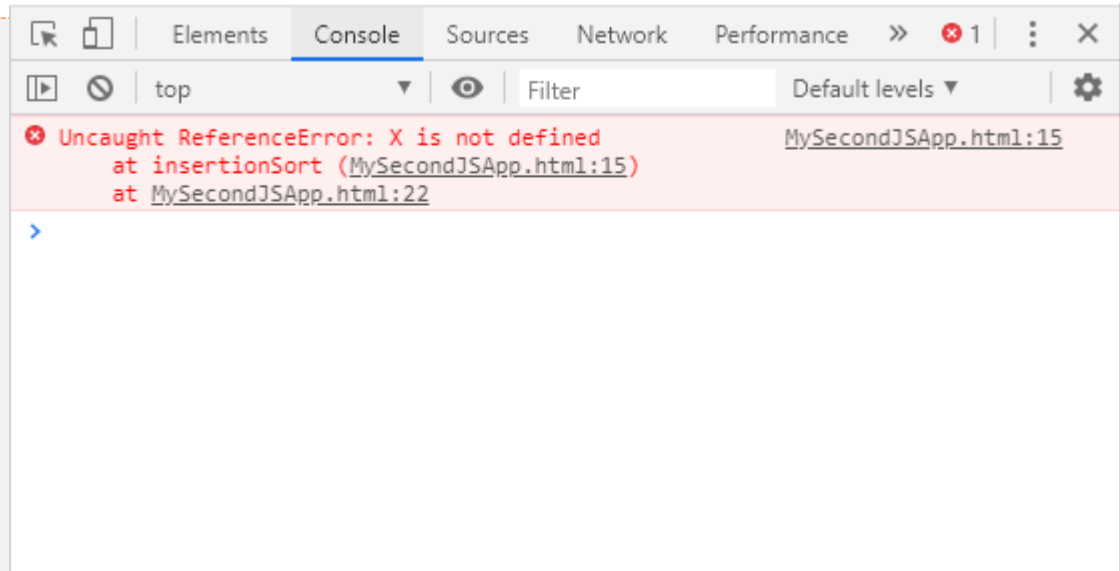
Razvojni alati preglednika (Browser developer tools)



- Kroz razvojne alate preglednika omogućen je pregled **HTML** stranice i rad sa pojedinim elementima (DOM – više riječi kasnije), kao i pristup svim **CSS** svojstvima elemenata
- **Console.log()** ispisuje poruku unutar konzole (kartica *Console*) - može imati N argumenta bilo kojeg tipa (vrijednosti, varijable, čak i funkcije)

Greške pri izvođenju JavaScripta

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <title>My Second JavaScript App</title>
  <meta charset="UTF-8"/>
  <meta name="keywords" content="JavaScript,
    HTML, programming"/>
</head>
<body>
  <h1>My Second JavaScript App</h1>
  <script>
    var array = [8, 5, 6, 2, 1, 9];
    function insertionSort(array){
      var i, j;
      var temp;
      for (i = 1; i<X.length; i++) {
        temp = array[i];
        for (j = i; j >= 1 && array[j - 1] > temp; j--)
          array[j] = array[j - 1];
        array[j] = temp;
      }
    }
    insertionSort(array);
    alert(array);
  </script>
</body>
</html>
```



- Greške se prikazuju na kartici *Console*

Skripte u vanjskim .js datotekama

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>Script in file referencing</title>
    <meta charset="UTF-8"/>
    <meta name="keywords" content="JavaScript, HTML, programming"/>
    <script src="s12.ScriptReference.js"></script>
  </head>
  <body>
    <h1>Script in file referencing</h1>
  </body>
</html>
```

- Oznaka `<script></script>` ima atribut **src** kojim se može referencirati vanjsku skriptu u kojoj je napisan kôd u JavaScriptu
 - Ovaj tag se može nalaziti unutar oznaka **head** ili **body**, ali može biti i iza oznake **html**
- Atribut **src** može biti postavljen na:
 - Relativnu lokaciju datoteke (kao u primjeru na ovom slajdu)
 - Na apsolutnu putanju datoteke (npr. C:\imeMape1\ImeMape2\ScriptReference.js)
 - Na URL (npr. <http://www.nekisajt.com/ScriptReference.js>)

Izvršavanje JavaScripta na korisničku akciju

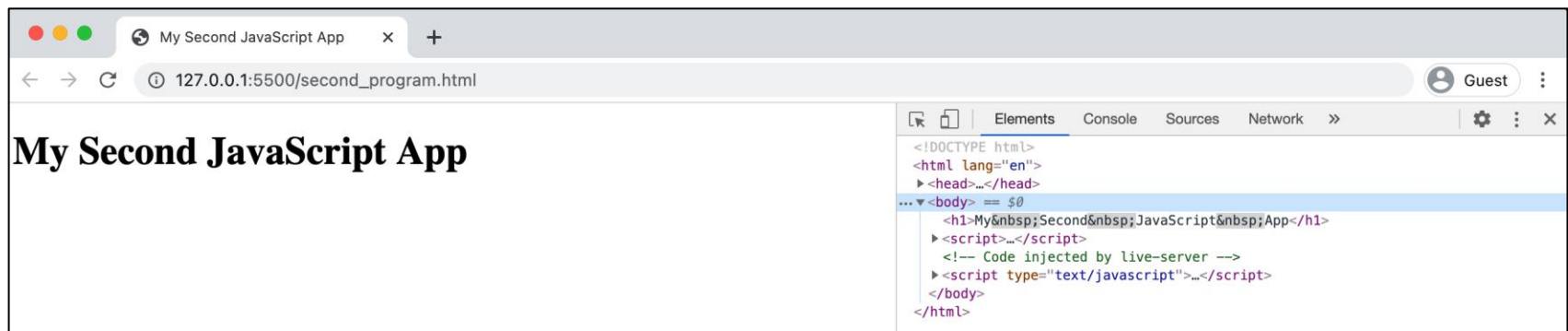
```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>HTML-JS calls</title>
    <meta charset="UTF-8"/>
    <meta name="keywords" content="JavaScript, HTML, programming"/>
    <script src="s14.HTML-JSCalls.js"></script>
  </head>
  <body>
    <h1>HTML-JS calls</h1>
    <button style="height:25px;width:50px"
      onclick="var result = insertionSort([8, 5, 6, 2, 1, 9]); alert(result);">Sort!</button>
  </body>
</html>
```



- Koristi se atribut **onclick** elementa **button** – u taj atribut se postavlja kôd u JavaScriptu koji se izvršava nakon što korisnik klikne na gumb
 - `var result = insertionSort([8, 5, 6, 2, 1, 9]); alert(result);`

Pozadina svake prikazane HTML stranice

- Internetski preglednici su tzv. okolina domaćina (eng. host environment) za HTML stranice
- Tri načina upravljanja stranicom:
 - **DOM** – Document Object Model (model HTML stranice + manipulacija)
 - **CSSOM** – Cascading Style Sheet Object Model (model CSS-a + manipulacija)
 - **BOM** – Browser Object Model (ugrađene funkcije internetskog preglednika)
- Svaki internetski preglednik omogućava prikaz DOM-a (npr. Google Chrome CTRL+Shift+i)



Osnovne značajke programskog jezika JavaScript

- JavaScript je skriptni programski jezik slabog tipa (eng. weakly-typed programming language)
 - Varijable su promjenjivog tipa, tj. tip varijable se može mijenjati prilikom izvođenja programa (eng. run-time)
 - U jednu varijablu je moguće prvo pohraniti vrijednost jednog tipa, pa zatim novu vrijednost drugog tipa
 - Programski jezici poput Jave, C-a, C++-a, C#-a i sl. su jezici strogog tipa (eng. strongly-typed)
 - U njima svaka varijabla ima točno određen tip. On je dodijeljen varijabli prilikom prevođenja programskog kôda i ne može se mijenjati
- Neke specifičnosti JavaScripta:
 - Svaka naredba tipično završava s ; (ali može i bez tog znaka – samo prelaskom u novi red)
 - Jednolinijski komentari počinju s //
 - Višelinijnski komentari su omeđeni s /* i */ (nije dozvoljeno gniježđenje)

Variable i konstante u JavaScriptu

- Variable se deklariraju ključnom riječi **let** (ili **var**)
 - **var** ima doseg tijela funkcije, a **let** bloka u kojoj se nalazi
 - Ime varijable smije sadržavati samo znakove, znamenke i simbole \$ i _
 - Prvi znak imena varijable ne smije biti znamenka
 - Postoje rezervirane ključne riječi
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords
 - Uobičajena konvencija imenovanja - camelCase
 - `isCompleted`, `numPeople`, `avgValue`, `minValue`, `sum`, `personAge` i sl.
- Konstante se deklariraju ključnom riječi **const**
 - Konstante nije moguće mijenjati
 - Uobičajena konvencija imenovanja:
 - Sve riječi naziva konstante pišu se velikim slovima
 - Riječi se međusobno odvajaju donjom crtom (_)
 - Npr. `const MIN_AGE = 16` ili `const RED_COLOR = "#F00"`

Tipovi podataka u JavaScriptu (1)

- JavaScript je jezik slabog tipa – varijablama je moguće mijenjati tip
- Numerički tipovi podataka:
 - *number*
 - IEEE 754, 64 bita, raspon $[-2^{53}, 2^{53}]$, točnost na do 15 znamenki
 - Za varijable cjelobrojnog tipa i za varijable s pomičnim zarezom
 - Podržava mnoštvo operacija poput +, -, * i /
 - Može pohraniti i tzv. specijalne vrijednosti (Infinity, -Infinity i NaN)
 - *bigint*
 - Omogućuje pohranu cijelih brojeva proizvoljne duljine
 - Konstanta tipa BigInt završava s n
 - Podržan u svim preglednicima osim u starijim verzijama preglednika Safari i Edge, te u Internet Exploreru

```
let numPeople = 32;  
numPeople = "Thirty two";
```

```
let numCars = 12;  
let taxRate = 0.25;
```

```
const bigInt = 498758943759  
843754375349874398579384575  
943759843748390754839757439  
87534958n
```

Tipovi podataka u JavaScriptu (2)

■ Znakovni tipovi podataka:

■ *string*

- Vrijednosti tipa *string* moraju biti omeđene navodnicima
- Postoje tri vrste navodnika: dvostruki, jednostruki i ukošeni – svi se koriste ravnopravno
- Ukošeni navodnici omogućavaju da se unutar vrijednosti tipa *string* ugradi proizvoljni izraz koji će se izvršiti

- Ne postoji tip podataka za pohranu znaka – jedan znak se pohranjuje kao *string* duljine jednog znaka

■ *boolean*

- Varijabla tipa *boolean* poprima jednu od dvije moguće vrijednosti: *true*/*false*

```
let doubleQuotes = "String 1";  
let singleQuotes = 'String 2';
```

```
let backTickQuotes = `value plus tax  
x amounts to ${1000 * 1.25}`;
```

Varijabla `backTickQuotes` poprima vrijednost ``value plus tax amounts to 1250``

```
let isCompleted = true;  
let hasStopped = false;
```

Tipovi podataka u JavaScriptu (3)

- Specijalni tipovi podataka:
 - *null*
 - Posebni tip podataka čije varijable sadrže jedino null-vrijednost
 - Ovo **nije** null-pokazivač ili referenca na objekt čija je memorija oslobođena / nepostojeći objekt
 - Kada varijabla ima vrijednost null ona je prazna i vrijednost joj nije poznata
 - *undefined*
 - Posebni tip podataka
 - Kada je varijabla deklarirana, ali joj nije pridružena vrijednost
 - Uobičajeno je koristiti null za dodjelu prazne vrijednosti, a ne undefined
 - *object* i *symbol*
 - object služi za grupiranje varijabli različitih tipova podataka
 - symbol služi za definiranje jedinstvenih identifikatora za objekte

```
let currentTime = null;
```

```
let undefinedVariable;
```

```
let currentTemperature = 22;  
currentTemperature = undefined;
```

Pretvorbe tipova podataka (1)

- U JavaScriptu se pretvorbe tipova podataka obavljaju automatski, sukladno potrebama

```
let isCompleted = true;  
alert(isCompleted);
```

Varijabla `isCompleted` se pretvara iz tipa *boolean* u *string* i ispisuje pomoću `alert`-a

```
let firstOperand = "1";  
let secondOperand = "3";  
alert(firstOperand * secondOperand);
```

Varijable `firstOperand` i `secondOperand` su početno tipa *string*. Prije množenja se pretvaraju u *number*, izvršava se množenje, te se zatim rezultat pretvara u tip *string* i ispisuje korisniku

Pretvorbe tipova podataka (2)

- Moguće je napraviti i eksplicitne pretvorbe tipova podataka, ako su potrebne

```
let isCompletedString = "1";  
let isCompletedBoolean =  
  Boolean(isCompletedString);
```

Varijabla `isCompletedBoolean` sada
sadrži vrijednost `true` tipa
boolean

```
let isCompletedNumber =  
  Number(isCompletedString);
```

Varijabla `isCompletedNumber` sada
sadrži vrijednost `1` tipa *number*

```
let isCompletedStringNaN = "1abc";  
let isCompletedNumberNaN =  
  Number(isCompletedStringNaN);
```

Varijabla `isCompletedNumber` sada
sadrži vrijednost *NaN* (greška)
tipa *number*

Pretvorbe tipova podataka (3)

- Pretvorbe u numerički tip podataka (*number*) se obavljaju po ovim pravilima:
 - *undefined* postaje *NaN* (*Not a Number*)
 - *null* postaje 0
 - *true* postaje 1
 - *false* postaje 0
 - varijabla tipa *string* se nastoji pretvoriti u broj (tipa *number*)
 - *string* se pri tome nastoji pročitati kao broj
 - Prilikom čitanja broja se ignoriraju praznine na početku i na kraju (eng. leading and trailing whitespace)
 - Čisti prazni niz znakova se pretvara u broj 0
 - Ako se pojavi greška (npr. niz znakova "1a2" sadrži a u sredini) rezultat je *NaN*.
- Pretvorbe u znakovni tip podataka (*string*) se obavljaju na očekivani način – vrijednost bilo kojeg tipa direktno postaje znakovni niz

Pretvorbe tipova podataka (4)

- Pretvorbe u logički tip podataka (*boolean*) se obavljaju po ovim pravilima:
 - *undefined* postaje *false*
 - *null* postaje *false*
 - *NaN* postaje *false*
 - 0 postaje *false*
 - prazan znakovni niz ("") postaje *false*
 - Sve ostale vrijednosti postaju **true**
 - "0" (znakovni niz s jednim znakom 0) postaje *true*
 - " " (znakovni niz s jednom prazninom) postaje *true*
- Koji izrazi se evaluiraju u laž, a koji u istinu se naziva ***principom truthy/falsy***
 - Često pitanje na intervjuima za posao:
 - „Kako će se navedeni izraz evaluirati?” Primjerice izraz ('0' && 0)

Operatori u JavaScriptu (1)

- U JavaScriptu postoje unarni i binarni operatori
- Operator pridruživanja (=)
 - Uz obavljanje pridruživanja vraća vrijednost koja je pridružena
- Operatori standardnih (matematičkih) operacija (+, -, *, /, %)
 - + i – mogu biti binarni ili unarni
- **Operator potencije (**)**
- Operatori usporedbe (<, >, <=, >=, ==, !=, ===, !==)
- Logički operatori (||, &&, !)
- Operatori Inkrementa i dekrementa (++, --)
- Binarni operatori (&, |, ^, ~, <<, >>, >>>)
 - >>> je operator posmaka uz punjenje s nulama
- Kombinirani operatori (+=, *= itd.)
- Operator zarez (,)
 - Odvaja više izraza koji se svi evaluiraju, ali se vraća rezultat samo zadnjega
 - Primjer korištenja: inicijalizacija varijabli u for petlji
 - Ima vrlo nisku prednost
- Uvjetni operator ?:

Operatori u JavaScriptu (2)

- Izrazi se evaluiraju s lijeva prema desno
- Prednost operatora slična je kao i u ostalim programskim jezicima i dana je tablicom:
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence
- Operatori mogu biti nadjačani
 - Operator + se može koristiti za zbrajanje brojeva i za spajanje znakovnih nizova
 - Za operator + vrijedi: ako je jedan operand tipa *string*, i ostali se pretvaraju u *string*

```
let firstOperandNumAdd = 1;  
let secondOperandNumAdd = 2;  
let resultNumAdd = firstOperandNumAdd + secondOperandNumAdd;
```

Rezultat je tipa *number* i ima vrijednost 3

```
let firstOperandString = "1";  
let secondOperandString = "2";  
let resultString = firstOperandString + secondOperandString;
```

Rezultat je tipa *string* i ima vrijednost "12"

Operatori u JavaScriptu (3)

- Izrazi se evaluiraju s lijeva prema desno
 - Primjenjuju se pravila evaluacije temeljem tipova operanada
- Kod operatora koji nisu nadjačani (npr. -, * i /) se radi automatska konverzija operanada u tip nad kojim ti operatori rade (*number*)

```
let firstOperandMixed = 1;  
let secondOperandMixed = 2;  
let thirdOperandMixed = "3"  
let resultMixed = firstOperandMixed +  
    secondOperandMixed + thirdOperandMixed;
```

Rezultat je tipa *string* i ima vrijednost "33" (prvo se zbrajaju 1 i 2, a zatim se na rezultat nadodaje *string* 3)

```
let firstOperandNumberMixed2 = 1;  
let secondOperandStringMixed2 = "3"  
let resultSubtractionMixed2 =  
    firstOperandNumberMixed2 -  
    secondOperandStringMixed2;
```

Rezultat je -2 i tipa *number* zbog korištenja operatora -. "3" se pretvara iz *stringa* u *number*

Operatori u JavaScriptu (4)

- Usporedba znakovnih nizova se radi temeljem kodne stranice Unicode
- Usporedba pojedinačnih znakova je slična usporedbi u C-u ili C++-u.
- Usporedba znakovnih nizova
 - Uspoređuje se znak po znak s lijeva prema desno dok se ne dođe do razlike, do kraja jednog od nizova ili do kraja oba niza
 - Ako je pronađena razlika, rezultat će biti true ako je znak razlike lijevog niza > znaka razlike desnog niza, inače false
 - Ako se došlo do kraja desnog niza, (a ne i do kraja lijevog) vraća se true, a do kraja lijevog (ne i do kraja desnog) vraća se false
 - Ako se prilikom korištenja > došlo do kraja oba niza u isto vrijeme (i nigdje nema razlike), nizovi su jednaki i rezultat je false

```
let firstOperandStringComp = "a";  
let secondOperandStringComp = "A";  
let resultStringComp = firstOperandStringComp >  
secondOperandStringComp;
```

Rezultat je tipa *boolean* i ima vrijednost true

```
let firstOperandStringComp2 = "Java";  
let secondOperandStringComp2 = "JavaScript";  
let resultStringComp2 = firstOperandString > sec  
ondOperandString;
```

Rezultat je tipa *boolean* i ima vrijednost false

Operatori u JavaScriptu (5)

- Pri usporedbi vrijednosti različitog tipa, JavaScript pretvara vrijednosti koje se uspoređuju u brojeve (tip *number*)
 - Znakovne nizove (*string*) se parsira i pretvara u brojeve
 - Logičke vrijednosti (*boolean*) se pretvara na ovaj način: true u 1, a false u 0

```
let firstOperandMixComp = 1;  
let secondOperandMixComp = "20";  
let resultMixComp =  
    firstOperandMixComp > secondOperandMixComp;
```

Rezultat je tipa *boolean* i ima vrijednost false

```
let firstOperandBoolean = true;  
let secondOperandNumber = 0;  
let resultBoolNumComp =  
    firstOperandBoolean != secondOperandNumber;
```

Rezultat je tipa *boolean* i ima vrijednost true

Operatori u JavaScriptu (6)

- Pri usporedbi vrijednosti različitog tipa, u JavaScriptu je moguće koristiti operatore `===` i `!==` koji uzimaju u obzir informaciju o tipu varijable
- Ako operator pronađe razliku u tipovima operanada, izraz se automatski evaluira u `false`

```
let firstOperand = 1;  
let secondOperand = "1";  
let resultNumString =  
    firstOperand === secondOperand;
```

Rezultat je tipa *boolean* i ima vrijednost `false`

```
let firstOperandBooleanCompOper = true;  
let secondOperandNumberCompOper = 0;  
let resultBoolNumCompOper =  
    firstOperandBooleanCompOper !==  
    secondOperandNumberCompOper;
```

Rezultat je tipa *boolean* i ima vrijednost `true`

Operatori u JavaScriptu (7)

- Logički operatori (`||` i `&&`) ponašaju se slično istovjetnim operatorima u jeziku C/C++
 - U JavaScriptu: ako tipovi operanada nisu *boolean*, pretvaraju se u *boolean*
- Izraz s višestrukim operatorima `&&` vraća vrijednost prvog operanda koji se evaluirao u `false`
 - Ako su svi `true` vraća vrijednost zadnjeg operanda
- Izraz s višestrukim operatorima `||` vraća vrijednost prvog operanda koji se evaluirao u `true`
 - Ako su svi `false` vraća vrijednost zadnjeg operanda
- `&&` je većeg prioriteta od `||`

```
let firstOperandAndOper = 1;  
let secondOperandAndOper = 0;  
let resultAndOper =  
  firstOperandAndOper &&  
  secondOperandAndOper;
```

Rezultat je tipa *number* i ima vrijednost 0

```
let firstOperandOrOper = 1;  
let secondOperandOrOper = 0;  
let resultOrOper = firstOperandOrOper || secondOperandOrOper;
```

Rezultat je tipa *number* i ima vrijednost 1

Operatori u JavaScriptu (8)

- Logički operator `!` je unarni operator
- Pretvara operand u tip *boolean* (vrijednost `true` ili `false`)
- Vraća inverznu vrijednost operanda

```
let firstOperandNegNum = 1;  
let resultNegNum = !firstOperandNegNum;
```

Rezultat je tipa *boolean* i ima vrijednost `false`

```
let firstOperandNegStr = "0";  
let resultNegStr = !firstOperandNegStr;
```

Rezultat je tipa *boolean* i ima vrijednost `false`

Operatori u JavaScriptu (9)

- Uvjetni operator ?:
sličan je uvjetnom operatoru u jeziku C/C++
- Sintaksa:
 - condition ? value1 : value2
 - Ako je ispunjen uvjet condition rezultat izraza je value1, inače value2


```
let isBoilingTemp = (t >= 100) ? true : false;
```

Rezultat je tipa *boolean* i ima vrijednost *true* ako je $t \geq 100$, a inače *false*

Operatori u JavaScriptu (10)

- Usporedbe s *null* i *undefined* su specifične
- Usporedba *null* i *undefined* operatorom `===` uvijek rezultira s `false`, a operatorom `!==` s `true` (različiti tipovi)
- Usporedba *null* i *undefined* **međusobno** operatorom `==` daje **true**, a operatorom `!=` `false`
- **Usporedba *null* ili *undefined* s ostalim vrijednostima operatorom `==` uvijek daje `false`**
- Prilikom usporedbe *null/undefined* i vrijednosti koje nisu *null* ili *undefined* operatorima `<`, `>`, `<=` i `>=` *null* se pretvara u 0, a *undefined* u *NaN*
 - Za operatore `<`, `>`, `<=` i `>=` vrijedi: Zbog pretvaranja *undefined* u *NaN*, sve usporedbe *undefined* sa vrijednostima koje nisu *null* ili *undefined* rezultiraju s `false`

"zanimljivosti" JSa :)

> typeof NaN	> true===1
< "number"	< true
> 999999999999999999	> true===1
< 1000000000000000000	< false
> 0.5+0.1==0.6	> (!+[[]]+[![]]).length
< true	< 9
> 0.1+0.2==0.3	> 9+"1"
< false	< "91"
> Math.max()	> 91-"1"
< -Infinity	< 90
> Math.min()	> []==0
< Infinity	< true
> []+[]	 <p>Thanks for inventing JavaSc</p>
< ""	
> []+{}	
< "[object Object]"	
> {}+[]	
< 0	
> true+true+true===3	
< true	
> true-true	
< 0	



Javascript is weird.

```
> ('b' + 'a' + + 'a' + 'a').toLowerCase()
< "banana"
```



if-else if-else naredba

- *If-else if-else* naredba slična je istoimenoj naredbi u jeziku C/C++
- Izraz koji se evaluira može biti bilo kojeg tipa, ali se uvijek pretvara u tip *boolean*

```
if (boolean expression 1) {  
    //code in if block  
} else if (boolean expression 2) {  
    //code in else if block  
} else {  
    //code in else block  
}
```

switch naredba (1)

- *switch* naredba slična je istoimenoj naredbi u jeziku C/C++
- Uspoređuje se i tip i vrijednost varijable s tipom i vrijednosti u pojedinim *case* granama (slično usporedbi korištenjem operatora `===`)
 - Prvo *value1*, zatim *value2* itd.
 - Ako se pronađe podudaranje izvršava se sav kôd do prvog *break*-a (ili do kraja *switch*-a, ako nema ni jednog *break*-a prije toga)

Može biti bilo koji izraz koji se evaluira prije izvršavanja *switch*-a

```
switch(any expression) {  
  case 'value1':  
    //code under first case  
    break; //optional  
  case 'value2':  
    //code under second case  
    break; //optional  
  default:  
    //code under default  
    break; //optional  
}
```

switch naredba (2)

```
let switchVal = 1;
switch (switchVal) {
  case 1:
  case 2:
    alert('Executes for 1 or 2');
    break;
  case '3':
    alert('Executes for 3');
    break;
  default:
    alert('Default branch');
}
```

Grana case 1 nema kôda ni break što je dozvoljeno. Zbog nedostatka break-a u case 1, izvršava se kôd u case 2, te zbog break-a u case 2 switch završava

```
let switchVal = 3;
switch (switchVal) {
  case 1:
  case 2:
    alert('Executes for 1 or 2');
    break;
  case '3':
    alert('Executes for 3');
    break;
  default:
    alert('Default branch');
}
```

Izvršava se default grana zbog toga što nije pronađeno podudaranje i po tipu i po vrijednosti ni u jednoj od tri case grane (usporedba 3 i '3' je false zbog razlike u tipovima)

while i do-while petlja

- *while* i *do-while* petlje slične su istoimenim petljama u jeziku C/C++
 - Vitice nisu potrebne ako tijelo petlji ima jednu naredbu
- Izraz koji se evaluira u svakoj iteraciji izvršavanja ovih petlji može biti bilo kojeg tipa, ali se uvijek nakon evaluacije pretvara u tip *boolean*
- Beskonačne petlje koje se nikada ne prekidaju (npr. s `break`) mogu rezultirati zaustavljanjem programa od okoline za izvršavanje programa (npr. od strane internetskog preglednika)

```
while (boolean expression) {  
    // code in the loop body  
}  
  
do {  
    // code in the loop body  
} while (boolean expression);
```

for petlja (1)

- for petlja slična je for petlji u jeziku C/C++
- *initialization command* se izvršava na početku prije izvršavanja petlje (može se sastojati od više naredbi odijeljenih sa ,)
- *boolean expression* se izvršava prije tijela for petlje (u svakom koraku)
- *after step command* se izvršava nakon izvršavanja tijela for petlje u svakom koraku
 - Obično se koristi za povećavanje brojača (npr. i++)

```
for (initialization command; boolean expression; after step command) {  
    // code in the loop body  
}
```

```
for (let k = 0; k < 10; k++) {  
    console.log(k);  
}
```

for petlja (2)

for...of

```
for (variable of iterable) {  
  statement  
}
```

```
const array = ['a', 'b', 'c'];  
  
for(const element of array){  
  console.log(element)  
}  
  
// a  
// b  
// c
```

for...in

```
for (variable in object){  
  statement  
}
```

```
const array = ['a', 'b', 'c'];  
  
for(const element in array){  
  console.log(element)  
}  
  
// 0  
// 1  
// 2
```

Razlika:

- **for...of** iterira po *elementima* (bilo kojeg iterabilnog podatka – npr. string, array)
- **for...in** iterira po *ključevima (indeksima)*

Mijenjanje tijeka petlji

- Naredba *break* služi prekidu petlje
 - Izvršavanje se nastavlja prvom naredbom nakon prekinute petlje
- Naredba *continue* služi prelasku na idući korak petlje
 - Preskaču se naredbe između *continue* i kraja petlje, te se započinje s novom iteracijom petlje
- Podsjetnik: *break* i *continue* se treba adekvatno koristiti unutar if-naredbe i unutar naredbe ?:

```
let text = "";  
  
for (let i = 0; i < 10; i++) {  
  if (i === 3) {  
    continue;  
  }  
  text += i;  
}  
  
console.log(text);  
  
// 012456789
```

```
let text = "";  
  
for (let i = 0; i < 10; i++) {  
  if (i === 3) ? continue : console.log("else")  
  
  text += i;  
}  
  
console.log(text);  
  
// Error: Unexpected token 'continue'
```

Funkcije u JavaScriptu (1)

- Funkcija u JavaScriptu započinje ključnom riječi `function`
- Parametre funkcije se odvaja zarezima
- Funkcija može deklarirati svoje lokalne varijable (u tijelu funkcije)
 - Funkcija može koristiti globalne varijable koje su deklarirane izvan funkcije i u dosegu (eng. scope) su funkcije. Također, svaka varijabla definirana u funkciji bez `let` i `var` je također globalna i vidljiva u toj i ostalim funkcijama
- Funkcija završava:
 - Ključnom riječi `return`
 - Ako tijek izvršavanja funkcije dođe do njene zadnje linije (a tijekom izvršavanja nije bilo `return`-a)
 - `return` može navesti vrijednost koja se vraća u pozivajući program
 - Ako `return` ne navodi ništa, vraća se `undefined`
 - `return` treba završiti s `;`

```
function name(parameters) {  
    // code in the function body  
}
```

Funkcije u JavaScriptu (2)

- Parametri funkcije nemaju naveden tip, već se samo navode nazivi parametara
- Primitivne varijable se iz pozivajućeg programa predaju po vrijednosti (dolazi do stvaranja kopije)
 - Funkcije mijenjajući parametre mijenjaju samo lokalnu kopiju primitivnih varijabli (promjene nisu vidljive u pozivajućem programu)
- Ako se u pozivajućem programu ne navede vrijednost parametra, automatski se pri pozivu funkcije postavlja na undefined

```
function doubleErr(value) {  
  value *= 2;  
}
```

Funkcija imena doubleErr preko parametra prima vrijednost primitivne varijable value i udvostručuje tu vrijednost. No, nakon završetka funkcije parametar value se skida sa sistemskog stoga i udvostručena vrijednost nije vidljiva u pozivajućem programu

```
function add(value1, value2) {  
  let result = value1 + value2;  
  return result;  
}  
add(1);
```

Rezultat izvršavanja ovog poziva je *NaN*. Parametar value2 nije specificiran u pozivu, te je postavljen na undefined, a ukupno rezultat je postao *NaN*.

Funkcije u JavaScriptu (3)

- Moguće je navesti i podrazumijevane vrijednosti parametara funkcija (eng. default values)
- Podrazumijevana vrijednost može biti direktno navedena ili može biti rezultat izvođenja funkcije

```
function addDefault(value1, value2=0) {  
  let result = value1 + value2;  
  return result;  
}  
add(1);
```

Rezultat funkcije je 1.

```
function getDefault(){  
  return 0;  
}
```

```
function add(value1, value2=getDefault()) {  
  let result = value1 + value2;  
  return result;  
}  
add(1);
```

Rezultat funkcije je 1.

Funkcije u JavaScriptu (4)

- Funkcije u JavaScriptu je moguće stvoriti na dva načina:
 - Deklaracijom funkcije (eng. function declaration)
 - Funkcijskim izrazom (eng. function expression)
- Funkcije u JavaScriptu su ravnopravni objekti (eng. first-class objects)
- Moguće ih je pohraniti u varijable, pozivati ih po potrebi te predavati drugim funkcijama

```
function double(value) {  
    return value * 2;  
}  
  
let square = function(value) {  
    return value * value;  
}  
  
function changeArray(array, changeFunction) {  
    for(let i = 0; i < array.length; i++){  
        array[i] = changeFunction(array[i]);  
    }  
    return array;  
}  
  
let squareArray = changeArray([1,2,3,4],  
    square);
```

Deklaracija funkcije

Funkcijski izraz

Poziv funkcije changeArray kojoj se predaje polje i druga funkcija square

Funkcije u JavaScriptu (5)

- Funkcijske izraze je moguće koristiti za definiciju tzv. anonimnih funkcija (eng. anonymous functions)
 - Nemaju naziv
 - Koriste se jednokratno
 - Obično su kratke i pojednostavljaju pisanje kôda

```
function changeArray(array, changeFunction) {  
  for(let i = 0; i<array.length; i++){  
    array[i] = changeFunction(array[i]);  
  }  
  return array;  
}
```

Korištenje anonimnih funkcija
kao drugog parametra
funkcije changeArray

```
let doubleArrayAnn = changeArray([1,2,3,4], function(value) {return value*2;});  
let squareArrayAnn = changeArray([1,2,3,4], function(value) {return value**2;});
```

Funkcije u JavaScriptu (6)

- Lambda-funkcije su poseban oblik funkcija
 - Jednostavan način zapisivanja
 - Izbacivanje nepotrebnih sintaksnih elemenata (return-a i vitičastih zagrada)
 - Uvodi se operator strelice (eng. arrow) (\Rightarrow)
- Prilikom stvaranja funkcije, stroj koji izvodi kôd u JavaScriptu dodaje svojstvo naziva prototype i veže ga uz funkciju
 - prototype je objekt koji je veza na funkciju na način da pokazuje na tu funkciju
 - Pristupa mu se koristeći sintaksu `functionName.prototype`

```
let square = function (value)
{
    return value * value;
}
```

Deklaracija funkcije
korištenjem funkcijskog izraza

```
let squareLambda = (value) =>
value * value;
```

Deklaracija lambda-funkcije

Globalni i lokalni doseg (eng. global and local scope)

```
//Global scope

function firstFunction(){
  //Local scope #1
  function secondFunction(){
    //Local scope #1.2
  }
}

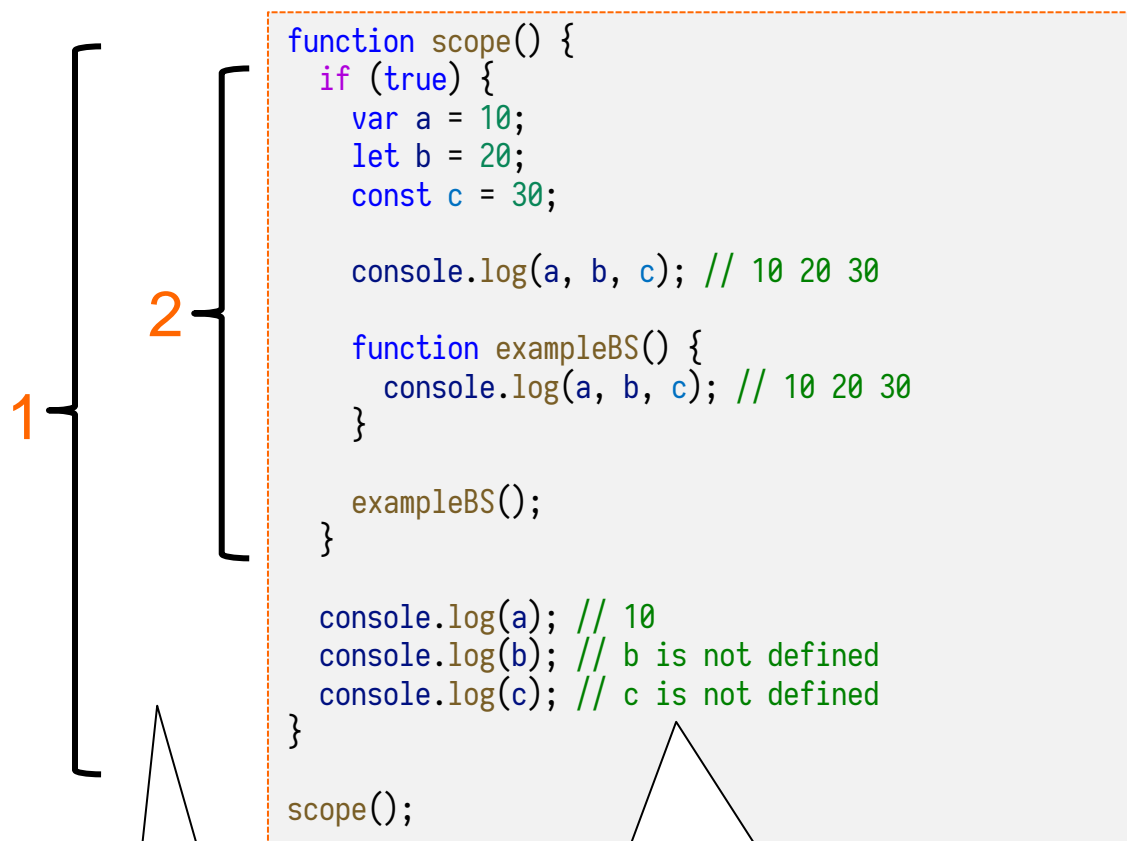
//Global scope

function thirdFunction(){
  //Local scope #2
}

//Global scope
```

- Svaki doseg ima pristup svim dosezima *iznad* sebe
- Primjer:
 - Doseg #1.2 ima pristup #1 i globalnom
 - Doseg #1 nema pristup #1.2

Doseg: var vs let vs const



Funkcija s dva bloka kôda. Blok 2 je ugniježđen u blok 1

Ispisuje se 10 zbog `var a` koji daje vidljivost van `if`-a. S druge strane, `b` i `c` su `let` i `const` pa nisu vidljivi izvan `if`-a (zato se ispisuje `not defined`)

- `let` i `const` su **dosega bloka** (eng. **block scoped**)
 - Doseg varijabli deklariranih na ovaj način je unutar vitičastih zagrada (bloka kôda omeđenog viticama)
 - Ugniježđeni blokovi („podblokovi”) imaju pristup varijablama blokova-roditelja („nadblokovima”)

Objekti u JavaScriptu (1)

- Objekti u JavaScriptu se mogu stvoriti na ovaj način:
 - konstruktorom (=new Object())
 - literalom (={})
- Sastoje se od parova *ključ: vrijednost* odijeljenih zarezom (,)
- Objektu je u bilo kojem trenu moguće dodavati svojstva ili ih brisati
 - korištenjem operatora `delete`
- Dozvoljeno je korištenje bilo kojeg imena za svojstva (čak i while, return i sl.) **osim** `__proto__`

```
let object1 = new Object();
```

```
let object2 = {};
```

Stvaranje praznog objekta

```
let person = {  
  OIB: "12345678912345",  
  name: "Pero",  
  surname: "Perić",  
  "home city": "Zagreb"  
};
```

Stvaranje objekta i specifikacija atributa objekta po sistemu *ključ: vrijednost*

```
person.age = 20;  
delete person.age;
```

Dodavanje/brisanje svojstava i vrijednosti

```
person["home city"] = "Osijek";  
let homeCityProp = "home city";  
person[homeCityProp] = "Split";
```

Objekti u JavaScriptu (2)

- Attribute je objektu moguće dodavati koristeći postojeće varijable
 - U tome slučaju novi atribut objekta automatski dobiva naziv, tip i vrijednost postojeće varijable
- Provjera postojanja atributa radi se:
 - usporedbom sa *undefined*
 - ključnom riječi *in* (**preporučuje se koristiti**)

```
let nationality = "Croatian";  
let personExtended = {  
  OIB: "12345678912345",  
  name: "Pero",  
  surname: "Perić",  
  "home city": "Zagreb",  
  nationality  
};
```

Dodavanje
novog atributa
po naziva
nationality i
vrijednosti
"Croatian"

Vrijednost oba
rezultata
(varijabli) je
true.

```
let existsNationality = personExtended.nationality !== undefined;  
let existsNationality2 = "nationality" in personExtended;
```

Objekti u JavaScriptu (3)

- Popis svih svojstava objekta, njihovih naziva, vrijednosti i tipova moguće je dobiti iteracijom petljom *for-in*
 - *for-in* petlja prolazi po svojstvima objekta redom kako su navedeni
- operator *typeof* se koristi za identifikaciju tipa varijable
 - Nema zagrada kao u C/C++-u, već se navodi direktno ispred varijable kako bi se odredio njen tip

```
let nationality = "Croatian";
let personExtended = {
  OIB: "12345678912345",
  name: "Pero",
  surname: "Perić",
  "home city": "Zagreb",
  nationality
};

for (let key in personExtended) {
  alert("Key: " + key +
    " Value: " + personExtended[key] +
    " Type: " + typeof personExtended[key]
  );
}
```

Key: OIB Value: 12345678912345 Type: String
Key: name Value: Pero Type: String
Key: surname Value: Perić Type: String
Key: home city Value: Zagreb Type: String
Key: nationality Value: Croatian Type: String

Objekti u JavaScriptu (4)

- Objekti se u memoriji pohranjuju tako da se uz sam objekt čuva i referenca na njega
- Objekte se može uspoređivati sa standardnim usporedbenim operatorima (==, !=, ===, !== itd.)
 - Uspoređuje se referenca, a ne vrijednosti svojstava objekata
- Operator pridruživanja (=) između objekata kopira samo referencu, a ne i svojstva
 - Za izradu kopije svojstava objekta koristi se metoda *Object.assign*

```
let personClone = Object.assign({}, person);
```

Odredišni objekt koji ima svojstva kao i objekt person (iste nazive, tipove i vrijednosti)

Prazni odredišni objekt u koji će se kopirati sva svojstva (i njihove vrijednosti) objekta person

Objekti u JavaScriptu (5)

- Okoline za izvođenje JavaScripta koriste sakupljač smeća (eng. garbage collector) kako bi oslobodile memoriju objekata na koje ne postoji niti jedna referenca

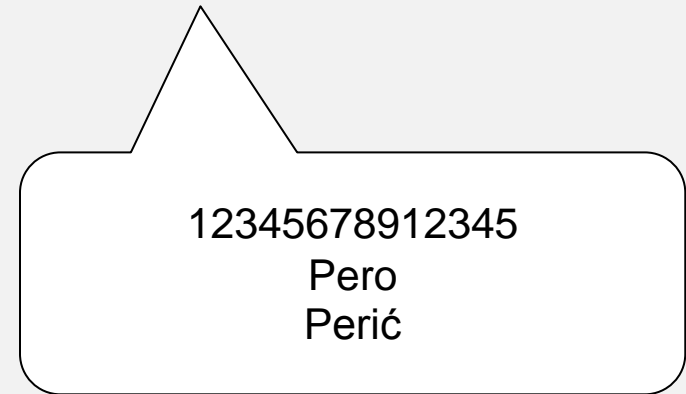
Za oslobađanje memorije svih objekata potrebno je postaviti i allPersons na null s obzirom da čuva kopiju referenci person1 i person2.

```
let person1 = {  
  OIB: "12345678912345",  
  name: "Pero",  
  surname: "Perić"  
};  
let person2 = {  
  OIB: "12345678912346",  
  name: "Krešo",  
  surname: "Kumek"  
};  
  
let allPersons = {  
  firstPerson: person1,  
  secondPerson: person2  
};  
person1 = null;  
person2 = null;  
allPersons = null;
```

Objekti u JavaScriptu (6)

- Atributima i vrijednostima objekata se pristupa ovim metodama:
 - `Object.keys(obj)` – vraća polje atributa/ključeva objekta `obj`
 - `Object.values(obj)` – vraća polje vrijednosti objekta `obj`
 - `Object.entries(obj)` – vraća polje parova [ključ, vrijednost]

```
let person1 = {  
  OIB: "12345678912345",  
  name: "Pero",  
  surname: "Perić"  
};  
  
for (let value of Object.values(person1)){  
  console.log(value);  
}
```



Const vs let

const

```
> const a = {pero: 10, slavko: 12}
< undefined
> a.pero = 13
< 13
> a
< ▶ {pero: 13, slavko: 12}
> a.ivica = 999
< 999
> a
< ▶ {pero: 13, slavko: 12, ivica: 999}
> a = {}
✖ ▶ Uncaught TypeError: Assignment to constant variable.
  at <anonymous>:1:3
> const b = []
< undefined
> b.push(1)
< 1
> b
< ▶ [1]
```

let

```
> let a = {pero: 10, slavko: 12}
< undefined
> a
< ▶ {pero: 10, slavko: 12}
> a.pero = 13
< 13
> a.ivica = 999
< 999
> a
< ▶ {pero: 13, slavko: 12, ivica: 999}
> a = {}
< ▶ {}
```

Varijabli označenoj s **const** se ne može promijeniti referenca!
(eng. cannot be reassigned)

Primjer: objekt + funkcija + lambda-funkcija

```
var obj = {  
  count: 10,  
  countAfterThreeSeconds: function () {  
    setTimeout(() => {  
      this.count++;  
      console.log(this.count);  
    }, 3000);  
  },  
};  
  
obj.countAfterThreeSeconds();
```

- Lambda-funkcije nemaju vlastiti **this**
- S druge strane, klasične funkcije uvijek imaju pristup vlastitom **this-u**
- Primjer: Ako se koristi **this.count** u lambdi od `setTimeout`, to se zapravo odnosi na **this** od funkcije, a taj **this** pristupa varijabli `count` iz `obj`

*prototip funkcije `setTimeout`:
`setTimeout(function, milliseconds)`