

# Funkcionalno programiranje

Školska 2024/25 godina

Letnji semestar

(Blok 1,2,3)

# Tema 1: Osnove jezika JavaScript (JS)

# Master Yuan-Ma, The Book of Programming

Drevni filozofski tekstovi, ali u kontekstu programiranja.

**1. "Ispod površine mašine, program se kreće. Bez napora, širi se i skuplja".**

**"Ispod površine mašine,,?"**

Programi deluju na dubljem nivou lako vidimo samo rezultat (UI, operacije), ispod haube računar izvršava niz složenih operacija.

**"Program se kreće" ?**

Program nije statičan; on se izvršava dinamički, procesira podatke i reaguje na unose.

**"Bez napora" ?**

Kada je dobro napisan, program deluje glatko i efikasno, bez kašnjenja ili problema.

**"Širi se i skuplja" ?**

Programi su fleksibilni. Mogu da se prilagode različitim uslovima i optimizuju za efikasnost.

## **2. U velikoj harmoniji, elektroni se raspršuju i pregrupisavaju.**

### **"U velikoj harmoniji" ?**

Programiranje i rad računara deluju kao skladan proces, iako se u pozadini odvijaju složene operacije.

### **"Elektroni se raspršuju i pregrupisavaju" ?**

Na najnižem nivou, računari rade pomoću električnih impulsa. Elektroni koji se kreću kroz tranzistore procesora omogućavaju izvršavanje instrukcija, skladištenje podataka i sve ostalo...

Ovo može biti metaforičan način da se opiše kako se podaci obrađuju i transformišu, dok programer orkestrira ovu harmoniju kroz kod.

### **3. Forme na monitoru su samo talasi na vodi. Suština ostaje ispod, nevidljiva.”**

#### **Forme na monitoru su samo talasi na vodi?**

Grafički interfejs, vizuelni prikaz programa i sve što korisnik vidi samo su površinski efekti, prolazni i zavisni od dubljih procesa. Kao što su talasi samo posledica kretanja vode, tako su slike na ekranu rezultat složenih operacija u pozadini.

#### **"Suština ostaje ispod, nevidljiva.,,**

Pravi rad programa odvija se ispod površine – u kodu, algoritmima, podacima i mašinskim instrukcijama koje korisnik ne vidi. Kao što se u okeanu ispod talasa krije duboka, nevidljiva masa vode, tako i u programiranju ono što se vidi na ekranu proističe iz nevidljive kompleksnosti iza scene.

#### **Zaključak:**

Ovo je refleksija o iluziji površinskog prikaza – kako u programiranju, tako i u životu. Prava suština stvari često ostaje skrivena, dok mi opažamo samo njen spoljašnji odraz.

- Programer početnik piše svoje programe kao što mrav gradi svoje brdo, jedan po jedan deo, bez razmišljanja o većoj strukturi.?
- Programi će biti poput rastresitog peska.
- Oni mogu stajati neko vreme, ali kada postanu preveliki, raspadaju se.

# Sadržaj

- I deo
  - O jeziku JavaScript
  - Sintaksa jezika JS i struktura koda JS programa
  - Varijable, izrazi, operatori, tipovi
  - Kontrola toka programa
  - Funkcije
- II deo
  - Izvršavanje JS koda
- III deo
  - JS strukture podataka i strukturni tipovi: objekat, niz

Zašto se JavaScript tako zove i kakva mu je veza sa jezikom Java?

- U vreme njegovog nastajanja jezik Java bio veoma popularan, a veza sa jezikom Java je NIKAKVA – Java je objektni jezik, JavaScript je multi-paradigmatski jezik.

Gde se izvršava?

- Prvo je bio napravljen da se izvršava samo u brauzeru, ali od kasnih 2000-tih (recimo 2009) sa pojavom okruženja nodejs ulazi i u programiranje serverske strane.

Sa čim ima integraciju?

- JavaScript je danas najčešće korišćen jezik brauzera koji ima potpunu integraciju sa HTML/CSS.

Da li ima programskih jezika koji se prevode u JS?

- Postoje mnogi jezici koji se “transpiluju” (kovertuju/prevode) u JavaScript i imaju dodatne mogućnosti.



# O jeziku JavaScript

- **Inicijalna namena i prva pojava:**
  - Skriptni jezik?  
u startu namenjen da se podrži programiranje u brauzerima.
- Način nastajanja: sve, samo ne planski i sistematično; prva verzija napravljena za 10 dana
- Brendan Eich, 1995; dodavanje mogućnosti programiranja Netscape Navigator brauzeru:

# Netscape Navigator

- Veoma popularan veb-pregledač na globalnom nivou, 1990-ih.
- 2002. prestalo njegovo korišćenje.
- Zašto?
  - Zbog povećane upotrebe Microsoft-ovog veb-pregledača IE,
  - Netscape (koju je kasnije kupio [AOL](#)) nije pratio tehničke inovacije
- Poslovna propast Netscape-a je bila glavna tema [Majrosoftovog antimonopolskog suđenja](#)?
- MS vezao Internet Explorer-a sa OS Windows - sud presudio kao monopol
- Ta odluka je doneta kasno za Netscape, jer je već tada Internet Explorer bio najdominantniji veb-pregledač na Windows-u.
- Microsoft nije razbijen na više kompanije što je sud zahtevao...

# 2.čas

# O jeziku JavaScript (5 osobina)

## 1. Ime: ?

**Nema nikakve veze sa jezikom Java**, osim slične sintakse.

## 2. Bitna svojstva:

- velika sloboda (u JS-u svašta može da se uradi, često nenamerno)
- **multi paradigmatiski jezik** sa specifičnom podrškom **objektnom** pristupu i mogućnostima da podrži **funkcionalnu** paradigmatu,

**Paradigma** je obrazac za ugled, model po kojem se nešto gradi.

**Multiparadigma?**

Programski jezik ima više modela/platformi...

Da li je Python multiparadigmatiski jezik?

DA: podržava imperativno, objektno-orijentisano i funkcionalno programiranje.

## Primeri programskih paradigmi

**A. Imperativna** – Program se piše kao niz instrukcija koje menjaju stanje programa.

**B. Proceduralna** – Kôd je organizovan u funkcije i procedure (npr. C, Pascal).

**C. Objektno-orijentisana (OOP)** – Bazira se na objektima i klasama (npr. Java, Python, C++).

**D. Funkcionalna** – Fokusira se na funkcije bez promenljivih i stanja (npr. Lisp).

**E. Logička (deklarativna)** – Bazira se na logičkim izrazima i pravilima (npr. Prolog).

**D. Funkcionalno programiranje** je programska paradigma koja tretira program kao izračunavanje matematičkih funkcija i izbegava stanja i promenljive podatke.

- Akcenat je na primeni funkcija, u suprotnosti sa stilom :

**A. Imperativnog programiranja** kod koga je naglasak na promeni stanja.

- Imperativ u govoru ?
- Zapovedni način - Niz naredbi

### 3. Razvoj:

- Promena (proširenje) namene: Od 2004 koristi se van brauzera
  - baze podataka - **Mongo,Couch**;
  - serverska strana **node.js**?

**omogućava korišćenje jedinstvenog jezika između front-end i back-end-a. To znači da kompletnu aplikaciju možete realizovati pomoću jednog programskog jezika.**

- Pokušaj (uspešan) da se stvori standardizuju
  - ECMA-262 standard, 1997
  - [ECMAScript 2022, jun 2022 \(13 verzija standarda\)](#)
  - [ECMA-262 - Ecma International \(ecma-international.org\)](#)

## **4. Relevantnost za tržište?:**

u 2022 rangiran na prvom mestu programskih jezika koje treba učiti (i prethodnih godina je uvek bio među prvih tri do pet)

## **5. Odnos programera?:**

vrlo emotivan - od nekontrolisanog obožavanja do nepatvorenog nipodašatavanja, čak mržnje.



# Vrednost, tip, varijabla, izraz

- **Vrednost** je reprezentacija nekog entiteta kojim računarski program može da manipuliše.
- Unutar računara nema nečega osim podataka zapisanih u obliku 0 ili 1. Da bi se moglo raditi sa velikom količinom bitova, a da se ne pogubimo, oni se moraju razdvojiti u delove koji predstavljaju delove informacija.
- U JavaScript okruženju, ti delovi se nazivaju ***vrednostima***.
- Vrednosti, imaju različite uloge: neki nam služe da manipulišemo sa numeričkim vrednostima, drugi nam služe da predstavimo sliku ili zvuk, treći da predstavimo tekst (i puno drugih stvari tekstom).
- **Uloga vrednosti** određuje način na koji će se sa tom vrednošću manipulirati. **Uloga vrednosti** iskazuje se ***tipom*** vrednosti.

Da bi se vrednostima manipulisalo:

- one moraju biti dostupne (moraju se negde uskladištiti)
- i mora se omogućiti pristup uskladištenom sadržaju; za to se u programiranju koristi koncept ***identifikatora – imena - varijable***.
- Da bi se vrednostima manipulisalo, potrebni su instrumenti za manipulaciju. Šta je to?
- U programiranju se za to koristi koncept ***operatora***. Za složenije manipulacije, ti operatori se kombinuju sa vrednostima dajući ono što se u programiranju zove ***izraz***.

# Varijabla

Šta je varijabla. Koji je drugi naziv?

- **Promenljiva**
- **U matematici ?**
  - veličina označena simbolom koji može da se zameni bilo kojim članom jednog skupa. Kako?
  - Primer:  $y=2x+3$ , X je nezavisna varijabla jer može poprimiti bilo koju vrednost a Y je zavisna?
- **Promenljiva u programiranju —**
  - imenovana memorijska lokacija koja je podobna za skladištenje određenog podatka.
  - Primer: Broj =10                      Šta je varijabla?
  - Broj je varijabla koja čuva celobrojnu vrednost 10.
  - U svakom trenutku izvršenja programa sadrži tačno određenu, konkretnu, vrednost koja se u svakom trenutku može zameniti drugom

Varijabla je *imenovano* skladište za vrednost  
Varijabla se mora kreirati pre korišćenja.

## Izraz

- **Izraz** u programskom jeziku je ?
- kombinacija vrednosti, varijabli, operatora i funkcija, koji računa i *vraća*, drugu vrednost.
- Za izraz se kaže da *evaluira* u tu vrednost.
- **Izraz** je svaki deo koda koji se **evaluira** na **vrednost**
- Evaluacija može da bude **striktna** ili **lenja** (**nestriktna**)

# Varijable i vezivanja<sub>1</sub>

- Primer 1

Da li je ovo je korektan JS program?:

```
1;
```

```
!false;
```

**Jeste korektan ali je beskoristan.** Zašto?

On samo pravi vrednosti ?

- **1** i

- **true**

sa kojima posle niko ništa ne može da uradi jer nema mehanizma da im pristupi.

Naredba stoji sama za sebe, tako da predstavlja nešto samo ako utiče na svet. Može da prikaže nešto na ekranu — što se smatra menjanjem sveta — ili može da promeni unutrašnje stanje mašine na način koji će uticati na naredbe koje dolaze posle nje. Ove promene se nazivaju bočnim (ponekad neželjenim) efektima. Naredbe u prvom primeru samo proizvode vrednosti **1** i **true**, a zatim ih odmah odbacuju. Ovo ne ostavlja nikakav utisak na svet. Kada pokrenete ovaj program, ništa ne možete primetiti.

- Pitanja: Kako program pamti stvari? Kako program održava svoje interno stanje?
- Odgovor: Korišćenjem koncepta zvanog **varijabla** i mehanizma zvanog **vezivanje**:
- `let neka_varijabla = 5 * 5;`

Sada imate varijablu **neka\_varijabla** sa kojom možete da uradite različite stvari, u suštini možete da pristupite uskladištenoj vrednosti.

Na primer:

`neka_varijabla +1;`

`neka_nova_varijabla = neka_varijabla - 20;`

# Kreiranje varijabli

- Varijable se u JS programu **kreiraju** deklaracijama ?
- **let**, **const** i **var**
  - Deklaracija **const** deklariše **konstantu** – vrednost koja **ne može biti promenjena**.
  - Deklaracija **var** je iz ranijih verzija specifikacije jezika
  - Osnovna razlika između **let** i **var** je u *pravilima dosezanja*:
    - **let** i **const**: doseg je **blok** (**{ }**) u kome je deklaracija
    - **var** : doseg je **funkcija** u kojoj je deklaracija

Biće objašnjeno na primeru ..

- **Blokovski dosezanje** (let i const) omogućava bolje i preciznije upravljanje promenljivim unutar blokova koda, što **smanjuje mogućnost grešaka**, jer ne možete slučajno koristiti promenljive van njihovog stvarnog opsega.
- **Funkcionalno dosezanje** (var) ima širi opseg, što znači da **varijable mogu "pobeći" iz blokova, što je često izvor grešaka u kodu**, naročito kada se koristi unutar petlji ili uslova.



## Šta je dozezanje (*scope*) ?

- područje u programu u kojem je određena promenljiva ili funkcija dostupna ili
- *Dosezanje* je deo izvornog koda u kome važi vezivanje imena i vrednosti
  - Nešto kao polica na kojoj su tegle u kojima su različite vrste pekmeza pri čemu su na teglama nalepnice sa slikama voća od koga je pekmez napravljen.
  - Pogrešna slika na nalepnici – tegla sa pogrešnim pekmezom.
- Da bi varijabla imala upotrebnu vrednost u programu, treba je **kreirati i dodeliti joj vrednost**.
- Mogu da **postoje i varijable kojima vrednosti nisu dodeljene** ali se **ne mogu dodeliti vrednosti varijablama koje nisu deklarisanе**.

# Primeri sintakse kreiranja varijable

**// deklarisanje varijable bez vezivanja**

**var** a; // jedna varijabla

**let** a, b; // više varijabli istovremeno

**const** a; // Ovo nije ispravno. Zašto?

**// deklarisanje varijable sa vezivanjem -**

Deklarisanje i vezivanje mogu se raditi istovremeno

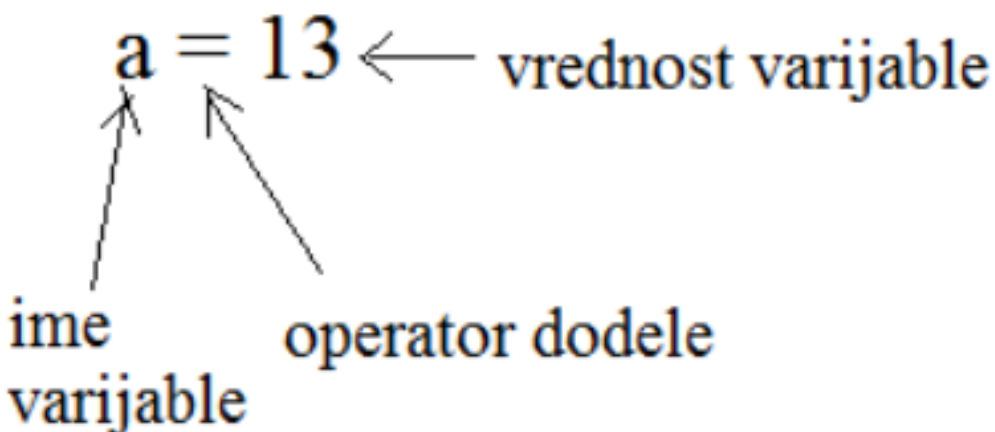
**const** a = 120; // jedna varijabla

**let** a = 120, b = 77; // više varijabli

**const** a=1, b=2; // Ovo jeste ispravno. Zašto?

**const** c=10, d; // A ovo nije ispravno. Zašto?

# Kako se u Python deklarir[e varijabla?

  
The diagram shows the code `a = 13` with three annotations: an arrow from 'ime varijable' (variable name) pointing to 'a', an arrow from 'operator dodele' (assignment operator) pointing to '=', and an arrow from 'vrednost varijable' (variable value) pointing to '13'.

`a = 13` ← vrednost varijable

ime varijable      operator dodele

- Osnovni tipovi podataka u Python-u su:?
- ***int*** – celi brojevi
- ***float*** – decimalni zapis
- ***bool*** – logički tip (?)
- ***string*** – znakovni zapis

# Struktura koda JS programa

- **Kôd** JS programa sastoji se iz *izraza, naredbi, deklaracija i komentara*

**1. Izrazi** su sintaksni konstrukti koji “**prave vrednosti**”.

Recimo: `const a = 120;`

- Izrazi mogu biti funkcijski argumenti, desne strane naredbi dodele, itd – u stvari, **svaki deo koda koji se evaluira na vrednost.**

Šta je izraz u Pythonu?

```
A=2
```

```
B="Ime"
```

```
print(A,B)
```

**2. Naredbe** su sintaksni konstrukti i komande koji “rade stvari” – **rezultuju izvršavanjem neke akcije.**

- **Klasifikuju se na: naredbe dodele; naredbe kontrole toka izvršavanja; naredbe iteriranja; ostale naredbe.**
- u istoj liniji koda može ih se navesti više i tada se obavezno međusobno se razdvajaju terminalnim simbolom tačka-zarez (;), Terminalni simbol se u većini slučajeva (ali ne uvek) jednolinijske naredbe može izostaviti.

• ***Šta je naredba u Python?***

A=2

B=3

if A>B:

    print(A)

else:

    print(B)

Posmatrano iz aspekta jezika generalno:

- **izrazi su mehanizam kojim se prave “delovi rečenice”, a**
- **naredbe su konstrukti kojim se prave “kompletne rečenice”.**

Program je sekvenca naredbi.

**3. Deklaracije** su sintaksni konstrukti jezika koji izjavljuju određene osobine drugih konstrukata (ponekad ih zovu i ***deklarativne naredbe***).

Šta je deklaracija u Pythonu?

Recimo:

A=2

B= [1,2,3]

Def unos();

    a=int(input("Unesi prirodan broj:"))

    return a

#### **4. Komentari** niti “rade stvari”, niti “prave vrednosti” – ne utiču na izvršavanje programa

- Dozvoljeni bilo gde u tekstu
  - Jednolinijski `//` tekst komentara u jednoj liniji
  - Višelinijijski `/*` tekst komentara `*/`
- Dozvoljen samo na početku skripta/modula
  - Hashbang komentar `#!` tekst komentara u jednoj liniji

Kako se pišu komentari u Pythonu?

`#`Program sadrzi jednolinijski i višelinijijski komentar.

`a=5`        `#`Deklaracija promenljive `a` cija je vrednost 5

`b=16.8;`    `"""`Promenljiva

`b` je

`realnog tipa"""`



# Praktičan rad


## 1. Instalacija

	VSCodeUserSetup-x64-1.85.2.exe	Application	92.593 KB
---	--------------------------------	-------------	-----------

## 2. Primer:

- **`alert("pozdrav svima\nHow are you?");`**
- JavaScript ima funkciju `alert` (poruka ) kojom se neka poruka prikazuje u ovom dijalogu.
- Kada pritisnete dugme OK, dijalog će nestati.

- Prvi korak:

 Welcome	JS proba.js X	JS proba1.js
---	---------------	--------------

```
D: > Singidunum > Predmet Funkcionalno programiranje >  
1 alert("pozdrav svima\nHow are you?");
```

- Drugi korak:

**<script src="proba.js"></script>**

Kako se izvršava?

Ovo se smesti u index.html

Pokrene se u browseru:

<file:///d:/Singidunum/Predmet%20Funkcionalno%20programiranje/Predavanja/primeri/index.html>

 file://

pozdrav svima  
How are you?

OK

- Sada se vraćamo na onaj primer sa var, let i const  
VAR:

JS proba.js

<> index.html

D: > Singidunum > Predmet Funkcionalno programiranje > 2024-

```
1  if (true) {  
2      var a = 20;  
3      alert(a); // 20  
4  }  
5  alert(a); // ReferenceError: a is defined
```

file://

20

☐ Don't allow this site to prompt you again

OK

Dva puta se izvrši ovo

- LET:

JS proba.js

<> index.html

D: > Singidunum > Predmet Funkcionalno programiranje > 2024-2025

```
1  if (true) {  
2      let a = 20;  
3      alert(a); // 20  
4  }  
5  alert(a); // ReferenceError: a is NOT defined
```

 file://

20

☐ Don't allow this site to prompt you again

Jednom se izvrši ovo

OK

# IF THEN ELSE u Python-u

A=2

B=3

if A>B:

    print(A)

else:

    print(„Manji broj je:", A)

Manji broj je: 2

# *Naredba i izraz u JS-u*

- **if-then-else** kao naredba

```
var x;  
if (y >= 0)  
{  
  x = y;  
}  
else  
{  
  x = -y;  
}
```

Proba1.js:

```
y=5  
var x;  
if (y >= 0)  
{  
  x = y;  
}  
else  
{  
  x = -y;  
}  
alert(x)
```



# 3.čas



# Da li treba tačka zarez?

y=5

alert ("jj")

(3 + 4).toString();

Situacija	Da li treba ; ?
Naredbe u novim redovima	✗ Nije potrebno (ali može biti)
Linija počinje sa ( ili [	✓ Preporučuje se
return , break , continue na posebnoj liniji	✓ Obavezno
for , while , do...while petlje	✓ Potrebno u nekim slučajevima
Više naredbi u istoj liniji	✓ Obavezno
Definisanje objekata i funkcija	✗ Nije potrebno

# Izraz

- if-then-else kao **izraz**
- Nema if i else (što su naredbe)  
`var x = y >= 0 ? y : -y;`

Primer:

`y=5`

`var x = y >= 0 ? y : -y;`

`alert(x)`

- **if-then-else** kao *izraz* može biti i argument funkcije (a kao *naredba* ne može):  
`myFunction(y >= 0 ? y : -y)`

Iz napred rečenog se može činiti da su naredbe i izrazi u jeziku JS dve potpuno različite stvari. Ali, to i nije baš tako.

**Ista stvar se može zapisati na 2 načina, kao izraz i kao naredba.**

Primeri koji će to bolje razjasniti:

- U jeziku JS **if-then-else** konstrukt (a to je naredba jer “radi odlučivanje” na koju će stranu krenuti izvršavanje) može da se napiše kao *naredba* ili kao *izraz*.
- U stvari, gde god da JavaScript očekuje naredbu, može se koristiti i izraz. Evo primera:
- **foo(7, 1);**
- **kao linija koda je naredba** (takozvana *izrazna naredba*, eng. *expression statement*);
- Međutim, sam **poziv funkcije napraviće vrednost pa je poziv funkcije foo(7, 1) izraz** jer pravi novu vrednost.

# Striktna i nestriktna (lenja) evaluacija

- Koncept se odnosi na način obrade operanada pri evaluaciji izraza.
  - Pri striktnoj evaluaciji vrši se obrada operanada "unapred", bez obzira da li je to potrebno u trenutnoj fazi izvršavanja programa.
  - Pri lenjoj evaluaciji obrada se vrši u trenutku kada je to potrebno za izvršavanje programa.
  - Pri striktnoj evaluaciji, računanje svakog termina koji sadrži podtermin koji otkazuje, takođe otkazuje, pa sledeća naredba otkazuje jer se **1/0** računa  
`print length([2+1, 3*2, 1/0, 5-4])`
  - Pri lenjoj evaluaciji, ova naredba ne otkazuje jer se **1/0** ne računa
- Većina jezika primenjuje striktnu evaluaciju kao pretpostavljenu, ali pruža i mogućnost lenje evaluacije.

# Python

```
a=len([2+1, 3*2, 1/1, 5-4])  
print(a)
```

4

```
a=len([2+1, 3*2, 1/0, 5-4])  
print(a)
```

Traceback (most recent call last):

File "C:/Users/Milan/test.py", line 1, in <module>

a=len([2+1, 3\*2, 1/0, 5-4])

ZeroDivisionError: division by zero

Podsetnik: Šta znači : **||** ?

operand 1	operand 2	and	or
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

			and	or
	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
And je kao množenje	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
Or je kao sabiranje	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

**||** je : or  
odnosno : ili

# Primer:

 file://

```
b=true || false || true || false || true || false  
alert(b);
```

true

Javascript će pročitati samo prvu istinitu vrednost i oceniti da je čitav izraz istinit, jer sve što dolazi nakon veznika ili ne može uticati na evaluaciju . Zašto?

Zato što TRUE or bilo šta (TRUE/FALSE) jeste TRUE

**Lenja evaluacija je tehnika optimizacije koja odlaže evaluaciju izraza sve dok njegova vrednost nije neophodna.**

# Python

```
a=true or false or true or false or true or false  
print(a)
```

**NameError: name 'true' is not defined. Did you mean: 'True'?**

```
a=True or false or true or false or true or false  
print(a)
```

**True**



# Sistem tipova i tipizacija u JS-u

- U programskom jeziku, sistem tipova je skup pravila kojima se dodeljuje svojstvo zvano **tip** koje propisuje (za varijable) dozvoljene vrednosti i dozvoljene operacije.
- Termini kojima je dodeljen određeni tip su **članovi** tog **tipa**.
- **tip** (na primer: integer, float, bulean, string).
- Tip diktira koje se operacije mogu izvršiti nad njim
- Tip u programskom jeziku je koncept koji omogućuje automatsku kontrolu ispravnosti programa: moraju da se zadovolje uslovi propisani njegovim tipom.
- **Svi programski jezici imaju ugrađene tipove, a većina dozvoljava i definisanje novih tipova.**
- **Tipizacija u jeziku može da bude *stroga*** (tip varijable se u potpunosti zna u fazi kompajliranja) **ili *slaba*** (tip varijable ne zna se nužno unapred, već se zna u fazi izvršavanja).

- *JavaScript je slabo tipiziran (dinamički) jezik.* Nekada se to zovu i **dinamički jezici**, gde se **tip varijable ne zna nužno unapred, već se zna u fazi izvršavanja.**
- To povećava fleksibilnost, ali smanjuje mogućnost kontrole.
- Postoje programski jezici sa vrlo razvijenom tipizacijom (sve se deklarise kao tip i sve se može kontrolisati) i oni se zovu **strogo tipizirani jezici.**
- U takvim jezicima se tip varijable u potpunosti zna u fazi kompajliranja (zato se oni nekada zovu i **statički tipizirani jezici**).
- To omogućuje bolju kontrolu, ali smanjuje fleksibilnost.

# Python

Koja komanda definiše TIP u Pythonu?

```
a="tekst"
```

```
b=123
```

```
print(type(a),type(b))
```

Rezultat:

```
<class 'str'> <class 'int'>
```

# JS slaba tipiziranost

- Pošto je JavaScript *slabo tipiziran (dinamički)* jezik, varijabla u JavaScript-u može da skladišti **bilo koju vrstu podataka**, odnosno različite vrste podataka u različitim trenucima.

```
message ="string";  
alert (message)
```

## String

```
message ="string";  
alert (message)  
message =5;  
alert (message)
```

## String

5

# Python

```
a="tekst"
```

```
a=123
```

```
print(type(a))
```

```
<class 'int'>
```

```
a=5
```

```
print(a)
```

```
print(type(a))
```

```
a="Python"
```

```
print(a)
```

```
print(type(a))
```

```
5
```

```
<class 'int'>
```

```
Python
```

```
<class 'str'>
```

- Primer (ubaciti ceo ovaj kod u Visual Studio Code):

```
function test1 () {  
  let message = {};  
  alert (message)}
```

```
message = 123.456;  
alert (message); // Ispis: 123.456  
message = 'Bila sam broj, a sada sam  
string'  
alert (message); // Ispis: Bila sam broj, a  
                  // sada sam string  
test1(); // Ispis:[object Object]
```

 file://

123.456

 file://

Bila sam broj, a sada sam string



Don't allow this site to prompt you again

 file://

[object Object]



Don't allow this site to prompt you again

OK

# Ispis:

[object Object]

☐

Don't allow this site to prompt you again

OK

Kada JavaScript ispiše poruku **[object Object]**, to znači da pokušavate da prikazete objekat kao string, a JavaScript ga automatski konvertuje u njegov podrazumevani format stringa.



# JS tipovi i strukture podataka

- JavaScript **ima tipove podataka**
- Poslednja verzija ECMAScript standarda definiše devet tipova podataka klasifikovanih u 3 grupe
  1. Primitivni tipovi (6 tipova)
  2. Strukturalni tipovi (2 tipa)
  3. Strukturalna korenska primitiva (1 tip)
- Tip podatka može se dobiti operatorom **typeof**:

```
let message = "hello";  
alert(typeof (message));
```



Staviti umesto hello : 123. Šta se dobije?

string

# JS tipovi: 1. primitivni tipovi<sub>1</sub>

- U JavaScript-u, primitiva (primitivna vrednost, primitivni tip podatka) je podatak koji **nije objekat** i **nema metode**. Ima ih 6:
  1. **undefined** : `typeof instance === "undefined"`
  2. **Boolean** : `typeof instance === "boolean"`
  3. **Number** : `typeof instance === "number"`
  4. **String** : `typeof instance === "string"`
  5. **BigInt** : `typeof instance === "bigint"`
  6. **Symbol** : `typeof instance === "symbol"`,
- Prosto, to su podaci “iz jednog komada”, nemaju nikakvih delova.

# 1.undefined

Primer za undefined? Šta je nedefinisano?

```
let a
```

```
message = typeof(a);
```

```
alert(message);
```

 file://

undefined

## 2. BOOLEAN

Primer za Boolean?

```
let message = true;  
alert(typeof (message));
```



boolean

Šta je:

```
let message = 999- 123;  
alert(typeof (message));
```

## Number

Šta je:

```
let message = 999- 123;  
alert(typeof (typeof(message)));
```

## String

## 5. Bigint

- BigInt je ugrađeni objekat u JavaScript-u koji pruža način za predstavljanje celih brojeva većih od  $2^{53}-1$ .
- Najveći broj koji JavaScript može pouzdano da predstavi pomoću primitive Number je  $2^{53}-1$ .
- Taj broj je definisan konstantom `MAKS_SAFE_INTEGER`.
- Kako prikazati `MAKS_SAFE_INTEGER`?

```
alert(Number.MAX_SAFE_INTEGER);
```

 file://

9007199254740991

☐ Don't allow this site to prompt you again

**EXCEL:**

**=2^53-1**

**9,0072E+15**

## 6. SYMBOL

Novi primitivni tip u javascript-u koji je jedinstven i nepromenljiv.

Primer za Symbol?

```
let sym1 = Symbol("a");  
let sym2 = Symbol("a");  
alert(sym1 === sym2);
```

 file://

false

- Rezultat je FALSE jer su simboli jedinstveni

# 1. JS tipovi: primitivni tipovi<sub>2</sub>

- Primitivni tipovi imaju sledeće karakteristike:
  - Porede se po vrednosti. Poredi se “sadržaj”:

`3 === 3 // true`

Kako ćemo to napisati u Javascript?

```
message = (3 === 3)
```

```
alert(message);
```

 file://

A može i bez zagrada da se piše

```
'abc' === 'abc' //true
```

```
3 === 4 // false
```

true

```
message = "abc" === "abc"  
alert(message);
```

true

```
message = "abc" === "abcd"  
alert(message);
```

false



- Sve primitive su imutabilne, t.j., ne mogu se menjati

```
var str = 'abc';  
str.length = 1; // pokušaj promene svojstva `length`  
str.length      // rezultat je 3-pokušaj nema efekta
```

```
var str = 'abc';  
str.length = 1; // pokušaj promene svojstva `length`  
alert(str.length) // rezultat je 3 – pokušaj nema efekta
```

## 2. JS tipovi: Strukturalni tipovi

- **Object** : `typeof instance === "object"`.
  - Specijalni strukturalni (non-data) tip koji se dodeljuje svakoj konstruisanoj instanci objekta koji se koristi i kao struktura podataka: `new Object`, `new Array`, `new Map`, `new Set`, `new WeakMap`, `new WeakSet`, `new Date` i **skoro sve ostalo što se pravi pomoću ključne reči `new`**.
- **Function** : `typeof instance === "function"`.
  - Takođe non-data struktura, iako reaguje na `typeof` operator: to je samo specijalna skraćenica za funkcije, iako je svaki konstruktor funkcije izveden iz konstruktora objekta, dakle **funkcija je neka (posebna) vrsta objekta (callable object)**.
- Ovo su podaci “iz više komada”, imaju delove.
- Što se tiče funkcije, “delovi” bi mogli da budu ime, parametri

# 4.čas

# Primeri za object:

```
message = typeof ['This', 'is', 101];
```

```
alert(message);
```

Rezultat : Object

Ostali primeri:

```
message = typeof new Boolean(true);
```

```
alert(message);
```

- `typeof {blog: 'freeCodeCamp', author: 'Tapas A'};`  
`//'object';`
- `typeof new Date();` `//'object'`
- `typeof Array(4);` `//'object'`
- `typeof new Number(101);` `//'object';`
- `typeof new String('freeCodeCamp');` `//'object';`
- `typeof new Object;` `//'object'`

# Primeri za Function

```
message = typeof Math.sqrt;  
alert(message);
```

```
message = typeof alert;  
alert(message);
```

```
message = typeof function () {};  
alert(message);
```

### 3. JS strukturalna korenska primitiva

- Nepostojeće vrednosti su izvor zabune u mnogim programskim jezicima.
- Javascript ima dve nepostojeće vrednosti - null i undefined.
- Vrednost null se koristi da označimo da je varijabla prazna.
- Obično null inicijalno dodelimo varijabli koja kasnije dobija pravu vrednost

# undefined

- **undefined** znači „bez (tipa) vrednosti“ (ni primitiva ni objekat).
- Ako pokušamo da upotrebimo nepostojeću promenljivu, nećemo dobiti ništa:

```
message
```

```
alert(message);
```

- Ali ako upotrebimo `typeof` operator na nepostojeću promenljivu, dobićemo "undefined" (string):

```
message
```

```
alert(typeof(message));
```

 file://

undefined

# JS: Konverzija tipova

- Različiti operatori i funkcije zahtevaju tačno određene tipove podataka
  - Na primer, aritmetičke operacije zahtevaju podatke tipa `number`, operacije ispisa zahtevaju podatke tipa `string`.
  - Zbog toga je potrebno da se izvrši konverzija tipova, na primer:
    - `1+2 = 3 => "3"`
    - `"3", "4" => 3+4`
- Postoje situacije kada se mora izvršiti eksplicitna konverzija tipova, ali u JS-u većina operacija radi automatsku (implicitnu) konverziju tipova po zadatim pravilima koja treba znati da ne bi bilo "iznenađenja"
  - Posebno, treba proučiti pravila za konverziju tipa `object` u primitivne tipove
  - Dobar izvor informacija o konverzijama je <https://javascript.info/type-conversions> (za primitivne tipove) i <https://javascript.info/object-toprimitive> (za konverziju tipa `object` u primitivne tipove).



- `let str = "123";`
- `alert(typeof str); // string`
- `let num = Number(str); // becomes a number`
- `alert(typeof num); // number`

 file://

string

 file://

number

☐ Don't allow this site to prompt you again

# Obrnuta konverzija : iz numerika u string

- `let num = 123;`
- `alert(typeof num); // number`
- `let str = num.toString(); // becomes a string`
- `alert(typeof str); // string`
  
- Probat i

# JS: Operatori i operandi – osnovni pojmovi

- Operator je simbol jezika koji označava specifičnu operaciju nad članovima tipa za koji je operacija legalna.
- Termini nad kojima se primenjuje **operator** zovu se ?
- **operandi**:
- **2 + 3**
- **arnost** operatora je broj operanada koje zahteva operator
  - Unarni (primenjuje se na 1 operand): Primer?
    - **-x**; ( **-** je negacija);
    - **+true** (**+** je konverzija u numerik).

```
let apples = "2";  
let oranges = "3";  
alert( apples + oranges ) //Koliko je ?
```

```
let apples = "2";  
let oranges = "3";  
alert( +apples + +oranges ) //?
```

- Binarni (primenjuje se na 2 operanda):
- let **x** = **1**, **y** = **3**;
- alert( **y** - **x** );
- Koji je rezultat?
- Za operatore je definisan redosled primene:
  - $1 + 2 * 2 \Rightarrow 1 + (2 * 2)$
- Detaljnije o operatorima imate na <https://javascript.info/operators> i <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

# Operatori u JS-u

1. Operatori dodele
2. Aritmetički operatori
3. Operatori poređenja
4. Logički operatori
5. String operatori: operatori poređenja, operator konkatencije.
6. Kondicioni operator: `condition ? val1 : val2`
7. Relacioni operatori: `in`, `instanceOf`.

# 1. Operatori dodele

- Osnovno značenje i simbol: Dodela vrednosti desnog operanda levom operandu, simbol je znak jednakosti (=)
  - **y = 3;**
- Varijacije
  - **Kompozitne** dodele kombinuju operaciju dodele sa aritmetičkom operacijom: **a \*= 3** – množi vrednost varijable a sa 3 i rezultat dodeljuje varijabli a.

a=5

a \*= 3

alert(a)

Rezultat je 15

- **Dodele svojstvima** – dodeljuju vrednosti svojstvu objekta. Dve notacije:
  - dot notacija - **mojAuto.marka = "Ford"**, i
  - notacija srednje zagrade – **mojAuto ["marka"] = "Ford"**

Prvi primer:

- **mojAuto** je objekat koji sadrži svojstvo.
- **marka** je svojstvo objekta **mojAuto**,  
a "Ford" je vrednost koju mu dodeljujemo

• Međutim, da bi ovaj kod radio, objekat **mojAuto** mora biti prethodno definisan. Na primer:

- `let mojAuto = {};` // Prazan objekat
- `mojAuto.marka = "Ford";` // Dodavanje svojstva "marka"

# Python

```
class Auto:
```

```
    def __init__(self, model, boja):
```

```
        self.model = model
```

```
        self.boja = boja
```

```
#self ukazuje na atribut klase
```

```
#Prikaz
```

```
    def prikaz(self):
```

```
        print ("Auto je", self.boja, self.model)
```

```
#Poziv klase
```

```
kola1=Auto("Reno","beli")
```

```
kola1.prikaz()
```

Auto je beli Reno



- **Ulančavanje** – istovremena dodela vrednosti različitim identifikatorima: ?
- let **a=b=c=1**;

- Primer:

```
dosomething();
```

```
function dosomething()
```

```
{
```

```
    let a = b = c = 2;
```

```
    alert( a );
```

```
    alert( b );
```

```
    alert( c );
```

```
}
```

 file://

2

- **Strukturiranje/destrukturiranje** –
- JS izraz koji omogućuje pakovanje pojedinačnih varijabli u niz ili objekat , odnosno raspakivanje vrednosti iz nizova, ili svojstava iz objekta u pojedinačne varijable.
- Operator je oblika `...imeNiza|imeObjekta`:

```
let a, b, rest;  
[a, b] = [10, 20];  
[a, b, ...rest] = [10, 20, 30, 40, 50]  
rest => [10, 20, 30, 40, 50]
```

raspakivanje vrednosti iz nizova

```
dosomething();
```

```
function dosomething()
```

```
{
```

```
    let a, b, rest;
```

```
    [a, b] = [10, 20];
```

```
    [a, b, rest] = [10, 20, 30]
```

```
    alert( a );
```

```
    alert( b );
```

```
    alert( rest );
```

```
}
```

Rezultat:

10

20

30

- let **a=2**, **b=c=a\*= 3**

# Rezultat?

```
dosomething();
```

```
function dosomething()
```

```
{
```

```
  let a=2, b=c=a*= 3
```

```
  alert( a );
```

```
  alert( b );
```

```
  alert( c );
```

```
}
```

 file://

6

## 2. Aritmetički operatori

Simbol operatora	Značenje
+	Sabiranje: $x+y$
-	Oduzimanje: $x-y$
*	Množenje: $x*y$
**	Stepenovanje ( <a href="#">ES2016</a> ): $x**y$
/	Delenje: $x/y$
%	Modul (ostatak): $x\%y$
++	Inkrementiranje: $x++$
--	Dekrementiranje: $x--$

PRIMER:

```
dosomething();
```

```
function dosomething()  
{
```

```
    a=3;
```

```
    c=6;
```

```
    a++;
```

```
    c--;
```

```
    b=a**c;
```

```
    alert(b);
```

```
    alert(4%5);
```

```
}
```

 file://

1024

Zašto je 1024?

 file://

4

# 3. Operatori poređenja

- Operatori
  - Veće/manje od:  $a > b$ ,  $a < b$ .
  - Veće/manje ili jednako od:  $a >= b$ ,  $a <= b$ .
  - Može i kombinujući sa matematičkim operacijama?:  
 **$a+1 > b$**

dosomething();

function dosomething()

{

a=3;

c=6;

a++;

c--;

b=a\*c;

alert(a<c);

alert((b-14)>=c);

}

Rezultat?:

True

true

- Jednako:  $a == b$ ; striktno jednako:  $a === b$
- Provera jednakosti i identičnosti u nekim slučajevima daje različite rezultate:

**2 == "2" Rezultat je?**

**True**

**2 === "2" Rezultat je?**

**False**



Nejednako: ?

- a **!=** b;
- Striktno nejednako: ?
- a **!==** b;

alert(2!="2")      ?

false

alert(2!== "2")      ?

true

- Rezultat poređenja je uvek bulovski tip

- Šta je rezultat?

- `alert("Ena" < "Ana")`
- `alert("Dobar" > "Dobra")`

```
dosomething();
```

```
function dosomething()  
{  
    alert("Ena" < "Ana")  
    alert("Dobar" > "Dobra")  
}
```

- False
- false

- Stringovi se porede “slovo”-po-“slovo” leksikografskim redosledom:

- Detaljno o operatorima poređenja na <https://javascript.info/comparison> i <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- Slovo **E** ima ASCII kod **69**
- Slovo **A** ima ASCII kod **65**
- Komanda u JS:
  - `alert(("E".charCodeAt(0)));`
  - Rezultat je 69

## 4. Logički operatori

- **||** (OR), **&&** (AND), **!** (NOT)
- **logičko** || odgovara uniji ( $\cup$ ) a logičko **odgovara** preseku ( $\cap$ )

a	b	c = a    b	c = a && b	c = !a / c = !b	Napomena
true	true	true	true	false/false	Prioritet operatora && je viši od prioriteta operatora
true	false	true	false	false/true	
false	true	true	false	true/false	
false	false	false	false	true/true	

- Detalji o logičkim operatorima na <https://javascript.info/logical-operators> i <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- Šta je rezultat?

dosomething();

```
function dosomething()
```

True

```
{
```

```
    a=true;
```

True

```
    b=false;
```

```
    alert(a || b);
```

True

```
    alert(a&&!b);
```

```
    alert(a || b&&b);
```

```
}
```

## 5. String operator poredjenja

```
dosomething();
```

```
function dosomething()  
{  
    var a = "Doživljaji ";  
    var b = "Sojera";  
    var c = a + "Toma " + b + "!";  
    alert(c);  
}
```

 file://

Doživljaji Toma Sojera!

## 6. Kondicionalni operator “?”

- Uslovna dodela vrednosti:

```
let rezultat = uslov ? vrednost1 : vrednost2;
```

- Ako je uslov **true**, **rezultat** = **vrednost1**; ako je uslov **false**, **rezultat** = **vrednost2**

- Primer 1:

```
dosomething();
```

```
function dosomething()
```

```
{
```

```
    let uzrast = prompt("Vaše godine, molim?");
```

```
    let message = (uzrast < 30) ? 'Ćao, mladiću!' : 'Poštovanje!'
```

```
    alert( message );
```

```
}
```

# 5.čas



- Primer 2:

```
let uzrast = prompt("Vaše godine, molim?");

let message = (uzrast < 3) ? 'Ćao, bebice!' :
  (uzrast < 18) ? 'Zdravo, drugar!' :
  (uzrast < 60) ? 'Poštovanje!' :
  'Vaše godine su zaista za poštovanje!';

alert( message );
```

# 7. Relacioni operatori: operator **in**

- Semantika: **Vraća true ako specificirano svojstvo (PROPERTY) postoji u objektu ili u prototipskom lancu objekta (OBJECT).**
- Sintaksa:  
`prop in object`  
**prop** je ime svojstva čije se postojanje proverava  
**Object** je ime objekat u kome se traži svojstvo

# Python

- *for i in range(0,-10,-1): print(i)*

0  
-1  
-2  
-3  
-4  
-5  
-6  
-7  
-8  
-9  
|

# Python

```
class Player():  
    def __init__(self, name):  
        self.name=name  
    def __contains__(self, substring):  
        if substring in self.name:  
            return True  
        else:  
            return False  
obj1=Player("Sam")  
print ('am' in obj1)  
print ('ami' in obj1)
```

True  
False

# Python

- **Klasa**: Auto
  - **Podklase**: Limuzina, Terenac, Sportski, Karavan
  - **Objekat** : NašAuto , poslovni auto, porodicni auto, licni auto
  - **Atributi** : reno, bela boja, 4 vrata.
  - **Metode**: kreni, zaustavi, ubrzaj
- 
- **Klase** definišu objekat.
  - Da bi se kreirao objekat, potrebno je prethodno imati klasu.
  - **Klase mogu imati više objekata.**
  - **Klase mogu imati podklase, na primer prema vrsti...**
  - **Atributi predstavljaju osobine objekta**
  - **Metode predstavljaju operacije koje objekat izvršava**

```
const auto = { marka: 'Honda', model: 'Accord', godina: 2008 };
```

```
a=('marka' in auto);
```

```
alert(a);
```

true

```
delete auto.marka;
```

```
b=('marka' in auto);
```

```
alert(b);
```

False

```
if ('marka' in auto === false) {
```

```
    auto.marka = 'Suzuki';
```

```
}
```

true

```
c=('marka' in auto);
```

```
alert(c);
```

# Relacioni operatori: operator `instanceof`

- Semantika: proverava da li se svojstvo property pojavljuje bilo gde u prototipskom lancu objekta.
- Povratna vrednost je `true` ako se pojavljuje, `false` ako se ne pojavljuje.
- Sintaksa:

`object instanceof constructor`

`object` – ime objekta u kome se traži

`constructor` – ime konstruktora čije se property svojstvo traži

print(result)  
• Pošto je numbers zaista lista, isinstance(numbers, list) vraća **True**.

# Python

Sintaksa:

`isinstance(object, classinfo)`

`isinstance()` vraća:

- Tačno ako je objekat instanca ili podklasa klase
- Netačno ....

```
numbers = [1, 2, 3, 4, 2, 5]
```

```
result = isinstance(numbers, list)
```

```
print(result)
```

True

Funkcija `isinstance(obj, type)` proverava da li je objekat **obj** određenog tipa **type**. U ovom slučaju, proverava se da li je `numbers` tipa `list`.

• **print(result)**

• Pošto je `numbers` zaista lista, `isinstance(numbers, list)` vraća **True**.



- Da li je ovo lista?

```
numbers = [1, 2, 3, 4, 2, 5]  
result = isinstance(numbers, list)  
print(result)
```

True  
Zašto?

Kada neće biti lista?

```
numbers = (1, 2, 3, 4, 2, 0.5)  
result = isinstance(numbers, list)  
print(result)
```

False  
Zašto?  
Ovo je sada  
Tuple-zagrade...

# Python

- Još dva primera:

```
result = isinstance(5, float)  
print(result)
```

False

```
result = isinstance(5, int)  
print(result)
```

True

# JS

## • Primer

```
function Auto(marka, model, godina) { // Ovo je  
konstruktor
```

```
    this.marka = marka;
```

```
    this.model = model;
```

```
    this.godina = godina;
```

```
}
```

```
const auto = new Auto('Honda', 'Accord', 1998);  
//ovo je instanca
```

```
alert(auto instanceof Auto);
```

 file://

true

# Poja šnje nje:

```
function Auto(marka, model, godina) {  
    this.marka = marka;  
    this.model = model;  
    this.godina = godina;  
}
```

- Ova funkcija Auto služi kao **konstruktor**/šablon za pravljenje objekata.
- Postavlja vrednosti svojstava za objekat koji će biti napravljen.

```
const auto = new Auto('Honda', 'Accord', 1998);
```

- Ovim pravimo **novi objekat auto** na osnovu konstruktora Auto.
- Prosleđujemo tri vrednosti:
  - "Honda" → dodeljuje se svojstvu marka
  - "Accord" → dodeljuje se svojstvu model
  - 1998 → dodeljuje se svojstvu godina

# Kontrola toka programa

U programima ne može sve da se odvija linearno

Najčešće algoritmi zahtevaju da se tok izvršavanja dešava u zavisnosti od nekih uslova.

Zato svaki programski jezik mora da ima naredbe za kontrolu toka programa. Ima ih i JS.

Provera uslova: ?

**If naredba**

- Uslovna dodela vrednosti ?

**switch** naredba

- **Petlje**

- while,
- do ... while
- for petlja

# Provere uslova: 1. If naredba

- **If** naredba:

```
If (uslov) {  
    //kod koji se izvršava, ako uslov ima vrednost true  
}  
else {  
    //kod koji se izvršava, ako uslov ima vrednost  
    false}  
// kod koji se izvršava nakon izvršenja If bloka.
```

- **If** (**uslov**) naredba sračunava **uslov** i rezultat automatski konvertuje u bulovsku vrednost (**true** ili **false**).

# If naredba: primeri

- Bez **else**:

```
let year = prompt('Koje godine je publikovana ECMAScript-  
2015 specifikacija?', '');
```

```
if (year == 2015) {  
    alert("Ispravno!" );  
    alert("Baš ste pametni!" );  
}
```

## Šta će se desiti?

Ako unesemo 2015 ispiše se prvi alert pa drugi alert

Ako unesemo nešto drugo, ništa se ne desi

- Sintaksa (mogućnost izostavljanja ključne reči **else**) dozvoljava da se ista stvar zapiše kompaknije
- Sa (više) **else**:

```
let year = prompt('Koje godine je publikovana ECMAScript-2015  
specifikacija?', '');
```

```
if (year < 2015) {  
    alert('Prerano...' );  
} else if (year > 2015) {  
    alert('Prekasno ...' );  
} else {  
    alert('Ispravno!' );  
}
```



# Naredba `switch`: sintaksa

```
switch(x) {  
    case 'value1': // ako je (x === 'value1')  
        ...  
        [break]  
  
    case 'value2': // ako je (x === 'value2')  
        ...  
        [break]  
  
    default:  
        ...  
        [break]  
}
```

- Situacije u kojima se program može granati na više (konačno mnogo) pravaca izvršavanja, mogu se realizovati sa mnogo if-ova. Ali je to veoma nečitko pa postoji posebna naredba **switch**.
- Zapazite da je ovde ključna reč **break** opcionalna.
- Ako je ima, izlazi se iz **switch** bloka nakon što se izvrši slučaj za koji **x** zadovoljava uslov.
- Ako je nema, nakon što **x** zadovolji uslov, izvršavanje se nastavlja u sekvenci i ne proverava se dalje šta je **x**.
- Ako **x** ne zadovoljava ni jedan **case** a postoji ključna reč **default:**, biće izvršeno ono što kaže **default:**.
- Ako ključne reči **default;** nema, neće biti izvršeno ništa.

# Naredba `switch` : opcija `break`

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert('Premalo' );
    break;
  case 4:
    alert(' Tačno!' );
    break;
  case 5:
    alert(' Preveliko' );
    break;
  default:
    alert(' Ne znam ' );
}
```

Tačno

Šta će sada ispisati?  
let a = 2 + 2;

```
switch (a) {
  case 3:
    alert('Premalo' );
  case 4:
    alert('Tačno!' );
  case 5:
    alert('Preveliko' );
  default:
    alert("Ne znam" );
}
```

Tačno  
Preveliko  
Ne znam

# Petlje: **while** i **do ... while**

- **while** petlja: prvo se proverava uslov, pa se izvršava telo

```
let i = 0;  
while (i < 3) { // ispiši  
    alert( i );  
    i++;  
}
```

 file://

0

 file://

1

 file://

2

- **do ... while** petlja: prvo se izvršava telo, pa se proverava uslov

```
let i = 0;  
do {  
    alert( i );  
    i++;  
} while (i < 3);
```

Rezultat?

Isto

# Python:while

```
broj = 10
```

```
while broj > 0:
```

```
    print("Broj=",broj)
```

```
    broj = broj - 3
```

```
Broj= 10
```

```
Broj= 7
```

```
Broj= 4
```

```
Broj= 1
```

## Python:do?

**Python nema eksplicitnu do-while petlju kao neki drugi jezici (npr. C, JavaScript, Java).**

- Ali se može simulirati sa While/True i Break

while True:

```
    broj = int(input("Unesite pozitivan broj: "))
```

```
    if broj > 0:
```

```
        break # Izađi iz petlje ako je uslov zadovoljen
```

```
    print("Pokušajte ponovo.") # Ako nije, ponovi unos
```

- **while True** osigurava da se kod unutar petlje izvrši bar jednom.
- Unosimo broj od korisnika.
- Ako je broj pozitivan ( $\text{broj} > 0$ ), koristimo break da izađemo iz petlje.
- Ako broj nije pozitivan, prikazuje se poruka i petlja se ponavlja.

```
Unesite pozitivan broj: -1
Pokušajte ponovo.
Unesite pozitivan broj: 0
Pokušajte ponovo.
Unesite pozitivan broj: 3
```

# Petlje: for petlja

- Petlja sa “brojačem” – varijablom koja kontroliše izvršavanje tela petlje

```
for (begin; condition; step) {  
    // ... telo petlje ...  
}
```

begin ?

– početna vrednost brojača

condition ?

– uslov završavanja petlje (terminalna vrednost brojača)

step – ?

korak sa kojim se povećava vrednost brojača, najčešće 1

- Jednostavan primer (sa korakom 1-da li će ispisati 0?):

```
for (let i = 0; i < 3; i++) { // ispisuje ...  
    alert(i);  
}
```

Ispisuje: 0 pa 1 pa 2

- Jednostavan primer (sa korakom različitim od 1-kako to napisati?):

```
var i = 0;  
for (i=5; i <= 20; i += 5){  
    alert (i); // ispisuje ...  
}
```

Ispisuje: 5 pa 10 pa 15 pa 20

# Python

*for n in range(2, 9): print(n)*

Rezultat:?

2

3

4

5

6

7

8



# Vidljivost brojačke varijable

- Ako je brojačka varijabla deklarirana u petlji, ona je vidljiva samo u petlji:

```
for (let i = 0; i < 3; i++) {  
    alert(i); // ispisaće : 0, 1, 2  
}  
alert(„i=„)  
alert(i);
```

Ispisuje: 0 pa 1 pa 2

Šta se onda dešava?

Posle toga je greška (alert(i))

Uncaught ReferenceError:

i is not defined

# Python

```
for n in range(2, 9):  
    print(n)  
print(n)  
#Šta će ispisati?
```

2  
3  
4  
5  
6  
7  
8  
8

Traceback (most recent call last):

File

"C:/Users/Milan/AppData/Local/Programs/Python/Python311/test.py", line 3, in <module>

```
    print(i)
```

NameError: name 'i' is not defined. Did you mean: 'id'?

# Python

```
for n in range(2, 9):
```

```
    print(n)
```

```
print(1)
```

```
#Šta će ispisati?
```

2

3

4

5

6

7

8

Traceback (most recent call last):

File

"C:/Users/Milan/AppData/Local/Programs/Python/Python311/test.py", line 3, in <module>

print(i)

NameError: name 'i' is not defined. Did you mean: 'id'?

- Ako je brojačka varijabla deklarisan ranije ona je vidljiva i van petlje:

```
let i = 0;
```

```
for (i = 0; i < 3; i++) { // koristi se već  
    deklarisan varijabla  
    alert(i); // ispisuje 0, zatim 1, zatim 2  
}
```

```
alert(i);
```

Ispisuje: 0 pa 1 pa 2

Pa ispiše :3

# Python

```
i=0
```

```
for n in range(2, 9):
```

```
    print(n)
```

```
print(i)
```

Šta će ispisati?

2  
3  
4  
5  
6  
7  
8  
0

# Sažeta sintaksa

- Svaki deo for naredbe koji se nalazi u maloj zagradi može se izostaviti

- Izostavljen **begin**

```
let i = 0; // izvršena deklaracija i dodela  
vrednosti
```

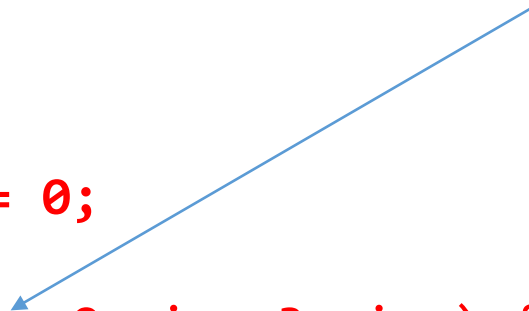
```
for (; i < 3; i++) { // ne treba BEGIN, BRIŠEMO i=0  
    alert( i );  
}
```

Ispisuje: 0 pa 1 pa 2

PRE:

```
let i = 0;
```

```
for (i = 0; i < 3; i++) {  
    alert(i); // ispisuje 0,1,2  
}
```



- Izostavljen **begin** i **step**

```
let i = 0;  
for (; i < 3;) { // premeštamo i++ na dole  
    alert( i++ );  
}
```

Isti rezultat

- Izostavljeno sve – beskonačna petlja

```
for (;;) {  
    // ponavlja se bez ograničenja. Zašto?  
}
```

PRE:

```
let i = 0;
```

```
for (i = 0; i < 3; i++) {  
    alert(i); // ispisuje 0,1,2  
}
```

# 6.čas



# for petlja: “Iskakanje” iz petlje

- Normalno se iz petlje izlazi kada je terminalni uslov zadovoljen.
- Petlja se može prisilno prekinuti direktivom **break**:

```
let sum = 0;  
while (true) {  
    let value = +prompt("Unesite broj", '');  
    if (!value) break; // (*)  
    sum += value;  
}  
alert( 'Sum: ' + sum );
```

Šta, u stvari, ovde uradi ključna reč **break**?

Ona napravi da **while uslov** postane **false**

**Analiza red po red-sledeći slajd:**

```
let sum = 0;                                     #Inicijalizacija promenljive sum
while (true) {                                   #Ovo je beskonačna petlja (while (true)),
                                                # koja će se izvršavati dok se ne prekine break
                                                # naredbom.

    let value = +prompt("Unesite broj", '');

    # +prompt(...) konvertuje unos u broj (+ ispred prompt radi eksplicitnu konverziju u Number).
    # Ako korisnik unese "5", pretvara se u 5 (broj).
    # Ako korisnik samo pritisne Enter ili unese nešto što nije broj, vrednost postaje NaN.

    if (!value) break; // (*)
    # Ova linija proverava da li je value falsy vrednost:
    # Ako korisnik unese 0, "" (prazan string), → !value postaje true i petlja se prekida (break).
    # Ako unese bilo koji broj osim 0, petlja se nastavlja.

    sum += value;                                #dodeljuje uneti broj u sum
}
alert( 'Sum: ' + sum );                         #ispis na ekran
```

# Direktiva **continue**: zaustavljanje tekuće iteracije

- Ne prekida celu petlju, zaustavlja tekuću iteraciju i prisilno počinje sledeću (ako uslov dozvoljava):

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

```
    // ako je true, preskoči ispis
```

```
    if (i % 2 == 0) continue;
```

```
    alert(i); //
```

```
}
```

Šta će se ispisati?

1,3,5,7,9

- Može isto i bez **continue**:

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

```
    if (i % 2) {  
        alert( i );
```

```
    }
```

```
}
```

A ovo ne može (**continue** nije dozvoljeno desno od **?**):

```
(i > 5) ? alert(i) : continue; // continue  
ovde nije dozvoljeno
```

# Labeliranje za **break/continue**

- Koristi se za kada je potrebno istovremeno “iskočiti” iz više petlji:

```
for (let i = 0; i < 3; i++) {  
    for (let j = 0; j < 3; j++) {  
        let input = prompt(`Vrednost u tački  
(${i},${j})`, '');  
        // Šta ako želimo da iskočimo na Done (iz petlje  
        i i petlje j?)  
    }  
}  
alert('Done!');
```

- Za to se koristi labela kojom se označava petlja (label)

# Analiza:

```
let input = prompt(`Vrednost u tački ( $\{i\}$ ,  $\{j\}$ )`, "");
```

prompt() traži unos od korisnika za svaku kombinaciju (i, j), tj. za svaku **ćeliju** u mreži 3x3.

Kako prekinuti petlje?

Pošto imamo **dve ugneždene petlje**, obični break; **prekida samo unutrašnju (j)**, ali **spoljašnja (i) nastavlja dalje**.

Ako korisnik klikne "**Cancel**" ili ostavi unos prazan, obe petlje se odmah prekidaju.

# Labeliranje za break/continue: primer

Labela

```
outer: for (let i = 0; i < 3; i++) {
```

```
  for (let j = 0; j < 3; j++) {
```

```
    let input = prompt(`Vrednost u tački (${i},${j})`, '');
```

```
    // ako se unese prazan string ili kontrolni  
    karakter, iskače se iz obe petlje
```

```
    if (!input) break outer; // (*)
```

```
    // uradi nešto sa vrednošću...
```

```
  }
```

```
}
```

```
alert('Gotovo!');
```

Direktiva break/continue se može pozvati samo iz unutrašnjosti petlje a labela mora da bude negde iznad direktive. Znači, ne može:

```
break label; // ne može da skače na  
label ispod sebe  
label: for (...)
```

# Pitanja

1. Šta će ispisati sledeći kod:

```
let i = 3;  
while (i) {  
  alert(i--);  
}
```

Šta znači: `while (i)`

`while (i)` znači **"izvršavaj petlju dok je i različito od 0"**.

U JavaScriptu, broj 0 je **falsy**, dok su svi ostali brojevi **truthy**

2. Da li će sledeća dva snipeta rezultovati istim ispisom:

**Prvi snipet:**

```
let i = 0;  
while (++i < 5) alert( i );
```

**Drugi snipet**

```
let i = 0;  
while (i++ < 5) alert( i );
```

Gde je razlika?

**++i i++**

Rezultat 1:

**1,2,3,4**

Rezultat 2:

**1,2,3,4,5**



Razlika između **++i (pre-increment)** i **i++ (post-increment)** u JavaScriptu je u trenutku kada se promenljiva povećava.

### Kako radi i++?

- **Prvo** vraća trenutnu vrednost i.
- **Zatim** povećava i za 1.

### Kako radi ++i?

- **Prvo** povećava i za 1.
- **Zatim** vraća novu, povećanu vrednost i

### 3. Šta će da uradi sledeći kod:

```
let n = 10;
labela:
for (let i = 2; i <= n; i++) {
    for (let j = 2; j < i; j++) {
        if (i % j == 0) continue labela;
    }
    alert( i );
}
```

**Oznaka labela** se koristi za izlazak iz spoljašnje petlje pomoću continue labela.

i	Prolazi kroz j (2 → i-1)	Da li je i deljiv?	continue labela ?	Prikazan broj?
2	---	✗ Ne	✗ Ne	✓ 2
3	2	✗ Ne	✗ Ne	✓ 3
4	2, 3	✓ Da (4 % 2 == 0)	✓ Da	✗ (preskače)
5	2, 3, 4	✗ Ne	✗ Ne	✓ 5
6	2, 3, 4, 5	✓ Da (6 % 2 == 0)	✓ Da	✗ (preskače)
7	2, 3, 4, 5, 6	✗ Ne	✗ Ne	✓ 7
8	2, 3, 4, 5, 6, 7	✓ Da (8 % 2 == 0)	✓ Da	✗ (preskače)
9	2, 3, 4, 5, 6, 7, 8	✓ Da (9 % 3 == 0)	✓ Da	✗ (preskače)

4. Šta će da uradi sledeći ako se unese slovo **a**:

```
let num;
```

```
do {
```

```
    num = prompt("Unesite broj veći od 100?", 0);
```

```
}
```

```
while (num <= 100 && num);
```

**Let num:**

- Deklarišemo promenljivu **num**, ali joj ne dodeljujemo vrednost.

```
do {
```

```
    num = prompt("Unesite broj veći od 100?", 0);
```

```
}
```

- **prompt()** prikazuje korisniku dijalog za unos broja. Podrazumevana vrednost je 0. Rezultat **prompt()** je uvek **string** (osim ako korisnik klikne "Cancel", tada je null). Vrednost koju korisnik unese dodeljujemo promenljivoj **num**.

- `while (num <= 100 && num);`
- `num <= 100` → Ako korisnik unese broj **manji ili jednak 100**, petlja se ponavlja.
- `&& num` → Ako korisnik unese null (klikne "**Cancel**") ili prazan string (`""`), uslov postaje false i petlja prestaje.