**JavaScript**

Sadrza j

[Uvod u Java Script 4](#_Toc116680518)

[Dodavanje JavaScripta u HTML 5](#_Toc116680519)

[Interno 5](#_Toc116680520)

[Eksterno 5](#_Toc116680521)

[Sintaksa jezika 6](#_Toc116680522)

[DATA TYPES 8](#_Toc116680523)

[Prosti tipovi podataka 8](#_Toc116680524)

[String 8](#_Toc116680525)

[Number 8](#_Toc116680526)

[Null 8](#_Toc116680527)

[Undefined 9](#_Toc116680528)

[Boolean 9](#_Toc116680529)

[Slozeni tipovi podataka 10](#_Toc116680530)

[Slozeni vs prosti tipovi podataka 10](#_Toc116680531)

[Array 10](#_Toc116680532)

[Object 10](#_Toc116680533)

[THIS kljucna rec 12](#_Toc116680534)

[Funkcije 13](#_Toc116680535)

[Scope 14](#_Toc116680536)

[Let, const var 16](#_Toc116680537)

[Arrow function 16](#_Toc116680538)

[IIFE function 17](#_Toc116680539)

[Konverzija tipova podataka 18](#_Toc116680540)

[Loops 19](#_Toc116680541)

[Reference 21](#_Toc116680542)

[Operatori 22](#_Toc116680543)

[DOM 23](#_Toc116680544)

[BOM-Browser Object Model 23](#_Toc116680545)

[Selectori 23](#_Toc116680546)

[document.getElementById(“id”); 23](#_Toc116680547)

[document.getElementsByTagName(“tagname”) 23](#_Toc116680548)

[document.getElementsByClassName(“klasa”) 23](#_Toc116680549)

[document.querySelector(“CSS selector”) 23](#_Toc116680550)

[document.querySelectorAll(“CSS selector”) 23](#_Toc116680551)

[window.getComputedStyle(element) 23](#_Toc116680552)

[innerHTML, innerText, textContent 23](#_Toc116680553)

[Events 25](#_Toc116680554)

[Skidanje eventa 26](#_Toc116680555)

# Uvod u Java Script

Programiranje-instrukcije koje dajemo procesoru da radi nesto  
HTML I CSS nisu programski jezici jer ne dajemo instrukcije procesoru vec web browser.

Procesor razume samo masinski jezik (0 I 1).

Zbog toga je potreban compailer. To je softver koji pretvara programski kod u 0 I 1.

**V8 engine je compailer za JavaScript**. Pisan je u **C++ programskom jeziku.** Browser je isto pisan u C++.

JavaScript je najudaljeniji programski jezik od masinskog jezika**. Njegov compailer je pisan u C++ jeziku I zbog toga je veoma brz.**Njegova “brzina” zavisi I od interneta, odnosno od brzine downloada. .

**Kopmijuteri nemaju intuiciju vec su bukvalni**. Sta im naredis, to ce da urade.  
Za razliku od AI, koja pokusava da nauci nesto na osnovu proslih dogadjaja I da na osnovu toga donosi odluke u odredjenim situacijama. **AI pokusava da ugradi intuiciju** u kompijutere.

JavaScript je nastao zbog validacije formi. Pre su serveri morali da validiraju podatke. Da bi se izbegla preopterecenost servera, validacija se prebacila na Front stranu. JavaScript je validirala formu pre slanja na server.

Da bi se usaglasila pravila i podrska JavaScripta za sve browsere, stvorena je ECMA(organizacija za standardizaciju).

# Dodavanje JavaScripta u HTML

Postoje 2 nacina za dodavanje JavaScript koda:

* Uz pomoc script tagova u HTML **(interno)**

### Interno

<script>  
document.getElementById("demo").innerHTML = "My First JavaScript";  
</script>

* Uz pomoc <script src=main.js></script> **kao eksterni fajl**

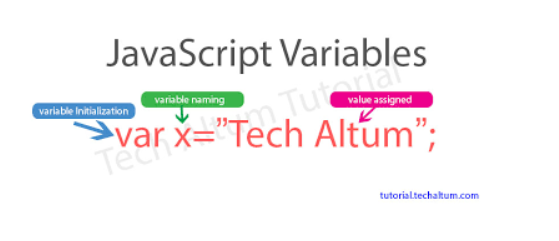
### Eksterno

<script src="myScript.js"></script>

**!!!Kada povezemo eksterno vise fajlova, mozemo koristiti funkcije iz jednog fajla u drugom, bez importovanja!!!**

# Sintaksa jezika

Programski jezici se nazivaju jezici zbog toga sto imaju pravila kao i u nekom drugom jeziku.  
**Ima pravila pisanja, kljucne reci, sintaksu itd.**

Da bi jezik funkcionisao, potrebno je smestiti neki podatak u nesto kako bi on mogao da manipulise sa njim.  
Zbog toga ima pravilo **SACUVAJ PA RADI.  
  
Memorija se deli na klastere.** Klasteri su male kockice(kutije).  
Jedna kockica predstavlja jednu memorisjku lokaciju. Kockice imaju svoje memorisjke adrese  
Memorisjka adresa je predstavljena brojevima. Npr 0x2456  


Podatke u programskim jezicima cuvamo u **VARIJABLAMA.  
Varijabla predstavlja ime memorisjke adrese. Da ne bi pamtili memorisjke adrese koje imaju binarni oblik(** npr 0x2456), mi kreiramo varijablu koja pokazuje na odredjenu memorisjku adresu(lokaciju).

Razlikujemo **deklaraciju i inicijalizaciju varijable.**

**Deklaracija** predstavlja kreiranje varijable.  
Varijabla sadrzi (**var, let ili constant**) **ime\_varijable  
  
Inicijalizacija** predstavlja dodeljivanje vrednosti varijable  
**ime\_varijable=vrednost(value)**

**Ime varijable ne moze pocinjati brojem**. Jedinstvena imena koja dajemo varijablama se zovu identifikatori. Prilikom pisanja identifikatora moramo

postovati odredjena pravila :

1. Identifikatori mogu sadržati slova, brojeve, donju crtu (\_).

2. Identifikatori ne mogu početi sa brojem.

3. Identifikatori su case-sensitive.

4. Identifikatori mogu biti bilo koje dužine.

5. Identifikatori ne mogu biti isti kao JavaScript ključne reči.

6. Da bi napravili razliku možete koristiti znak dolara ($) ispred

identifikatora ($ukupno)

Moguce je u isto vreme deklarisati i inicijalizovati varijablu  
  
**var ime\_varijable=vrednost(value)**

**!!!PROGRAMSKI KOD SE CITA SA LEVA NA DESNO. KAO PISACA MASINA!!!**

**!!!JAVASCRIPT NIJE REAKTIVAN!!!  
To znaci da promena neke varijable vazi samo nakon te promene.**

**let x=”Subota”;**

**console.log(x)---Subota**

**x=”Nedelja”; Vazi samo nakon promene, nakon inicijalizacije. U kodu iznad se ne menja.**

# DATA TYPES

Javascript svoje tipove podataka deli u tri kategorije :

* 1. Primitivni(prosti) tipovi podataka

. brojevi

. text (string)

. boolean

* 2. Slozeni tipovi podataka

. liste (arrays)

. objekti

. funkcije

* 3. Specijalni tipovi podataka

. undefined

. null

## Prosti tipovi podataka

Nakon znaka = imaju samo jednu vrednost

### String

Predstavlja tekst u programiranju.

Zahteva navodnike za definisanje

var text=”Hello world”  
var text2=’Hello world’  
var text3= ‘ Hello “world” ’  
var text4=`Hello world ${text3} ` --Template Literals

Spajanje stringova:

console.log(prezime+‘ ‘+ime)

**!!!SVE SABRANO SA STRINGOM JE STRING!!!**

### Number

Predstavlja brojeve u programskom jeziku.  
**!!!JavaScript NE RAZLIKUJE CELE I DECIMALNE BROJEVE!!!**

parseInt()**—koristimo za pretvaranje u number datatype**var x= +prompt(“Unesi broj);-- moze i ovako da se parsira. Dodavanjem “+” ispred

Null  
Definisemo da nemamo nista u varijabli. Null je ključna reč Javascript-a i predstavlja odsustvo vrednosti : **var** a = null ;  
Tretiramo null kao kontrolisani kod, jer znamo da smo eksplicitno naglasili da varijabla **a** nema vrednost.

### Undefined

U slučaju undefined ključne reči, slučaj je drugačiji jer undefined predstavlja obaveštenje da varijabla nije definisana :  
console.log(ime) *// undefined , varijable ime nema u našem kodu*

Kada **ne inicijalizujemo varijablu**, odnosno kada joj ne dodelimo vrednost, **browser ce automatski da dodeli “undefined”**

Kada neka memorisjka lokacija nema vrednost, bice obrisana automatski. Da bi to izbegli, dodeljuje se UNDEFINED.  
Odnosno rezervisemo varijablu da se ne bi obrisala.   
Treba izbegavati da sami definisemo UNDEFINED

Boolean  
Moze biti:

True-1  
False-0

## Slozeni tipovi podataka

### Slozeni vs prosti tipovi podataka

Prosti tipovi podataka mogu da sadrze samo jednu vrednost, dok slozeni mogu imati vise vrednosti.

### Array

Svaku vrednost u nizu zovemo **element**, a poziciju na kojoj se element nalazi u nizu/listi, **index**. Svi validni tipovi podataka mogu biti elementi niza.

Deklarisanje i inicijalizacija niza:

**var niz=[2,45,”Nemanja”, true, [42,9], “Hello world];**

**Niz moze da sadrzi sve tipove podataka.**

**Niz moze da sadrzi drugi niz u sebi.**

**Elementima niza se pristupa preko indexa.**

**Pocetni index niza je 0.**

**Korisno je zapamtiti da je broj elemenata uvek za jedan veći od indeksa poslednjeg elementa.**

**Npr**niz[0]----rezultat je 2  
niz[4].[0]------rezultat je 42

**Duzina niza** se dobija preko **.length.  
Npr**niz.length----rezultat je 6

### Object

Objekat svih objekata je OBJECT

Globalni objekat je WINDOW.

Svaki put kada se naša stranica učita u brauzeru, Javascript kreira window objekat. Window objekat je globalni objekat za client-side javascript programe. Metode koje su prikačene za window objekat omogućuju pristup određenim radnjama kao sto su pop-up prozori: alert(), prompt(), confirm() i mnogim drugim. Ako kreiramo varijablu koja je vidljiva celom našem kodu, ona će biti prikačena kao property window objekta .  
Varijable koje kreiramo preko VAR deklaracije postaju **property window objekta, dok funkcije postaju metode window objekta. OVO SAMO VAZI ZA JAVASCRIPT U BROWSERu!**

Predstavlja neki objekat sa podacima.

Moze da sadrzi podatke i njihove vrednosti, funkcije i druge objekte.

Podaci se nazivaju **PROPERTY**

Property se sastoji od key(name) i value(Nemanja).

**Key je string tipa.**

**Funkcije** u objektu se nazivaju **METODE**.

var person={

name: ”Nemanja”,

age: 33,

function info(){  
console.log(name + “ “+age)  
},

}

P

**Objekat je slican nizu, samo sto nazivom property-a dajemo KEY-u(indexu) znacenje.**

#### Pristup clanovima objekta:

Clanovima objekta pristupamo preko tacke.

person.name

Pošto je "key" objekta zapravo indeks, možemo se poslužiti i sintaksom kao i u listama  
person[“name”]

**!!!Naziv property-a(key) je string!!!**

**!!!Duzina objekta ne postiji.!!!**  
Ne mozemo da koristimo person.length

delete person.name –brisanje propertya iz objekta

Ako imamo objekat unutar objekta, takodje koristimo tacku za pristup podacima

**var** knjiga = {  
ime : "JavaScript"  
pisac : {  
 ime : "Danilo",  
 prezime : "Vesovic"  
 }  
}  
console.log(knjiga.pisac.ime) *// prikazace se "Danilo"*

### THIS kljucna rec

Kljucna rec THIS ukazuje na trenutni objekat.  
  
**KO JE MOJ VLASNIK?**

**KO ME JE POZVAO?**

**!!!Samo funkcija moze da menja vrednost THIS.!!!  
Pri pozivu funkcije, ona prvo setuje vrednost this-a pa tek onda izvrsava ostali kod.**

**U arrow funkcijama, rec this uvek ukazije na trenutni objekat. Pri pozivu arrow funkcije, ne menja se vrednost this-a.**

windows.volume=20;// Default volume for window object(the entire DOM)

const greenPack={

name: “Frog Pack”,

volume:8,

newVolume:function(volume){

console.log(“This volume before update: “, this volume)

this.volume=volume;

console.log(“Volume after update: ”,this.volume);

(function (){

console.log(“Volume in nested function: “, this.volume);

})();

(()=>{

console.log(“Volume in arrow function: “, this.volume);

})();

}

}

console.log(greenPack.newVolume(5);

Console:

This volume before update: 8

Volume after update:5

Volume in nested function:20--- zato sto je deklarisana funkcija global scope, postaje metoda windows objekta

Volume in arrow function: 5----uzima trenutnu vrednost this, trenutni scope.

## Funkcije

Funkcije u Javascriptu možemo posmatrati kao mini programe koje možemo pozvati da se izvrše

Funkcija moze da se deklarise na 2 nacina:

function ime(argumenti){ ----- function declaration  
// telo funkcije (blok koda)}

Drugi nacin je da funkciju deklarisemo u varijabli:

let sabiranje=function(a,b){ ---- function expression

let c=a+b;

return c

}

**!!!Argumenti funkcije su elementi niza arquments!!!**

**!!! Argumenti su privatne varijable funkcije!!!**

function ime(argument){

var argument= vrednost koju smo prosledili

}

Funkcija po default ima return. **Ako ne navedemo sta vraca, return ce nas vratiti na mesto gde je pozvana funkcija.**

### Scope

Scope predstavlja oblast u kojem je dostupna varijabla.  
Sve sto je deklarisano u funkciji, predstavlja lokalnu varijablu, sve sto je van funkcije predstavlja globalnu varijablu.

Pri pozivu funkcije, funkcija datu varijablu trazi u svom bloku, ako je ne nadje, trazice je u bloku kome pripada.

Npr:

function info(){

var a=2;

function info2(){

console.log(a)

}

info2()

}

info()-------prikazace nam 2, odnosno vrednost varijable a

function info(){

console.log(car);---undifined

var car=”Skoda”;

}

info()---- ispisace undefined zbog toga sto kada se pozove funkcija, prvo se deklarisu varijable. Inicijalizacja vazi samo nakon linije koda gde smo izvirsili inicijalizaciju. U ovom slucaju, funkcija ce doci do console.log, trazice varijablu car, naci ce je da je deklarisana u samoj funkciji ali ne moze da prikaze inicijalizaciju, zbog toga sto je ona u sledecoj liniji koda.

### Let, const var

Kada deklarisemo sa VAR, varijabla moze da bude lokalna ili globalna. Odnosno da bude u lokalnom scope ili u globalnom.

**!!! KADA VARIJABLU DEKLARISEMO SA LET, ONA POSTAJE BLOCK SCOPE I NE POSTAJE PROPERTY WINDOW OBJEKTA !!!**

**MOZE SE KORISTITI SAMO IZMEDJU { } U KOJIMA JE DEKLARISANA**

**{ } proctor izmedju ovih zagrada predstavlja jedan blok!**

**Const**  je takodje block scope i ne mozemo menajti const varijablu kroz program.

### Arrow function

Arrow function je skraceni prikaz standardne funkcije, sa odredjenim ogranicenjima.

// Traditional Anonymous Function

function (a){

return a + 100;

}

// Arrow Function Break Down

// 1. Remove the word "function" and place arrow between the argument and opening body bracket

(a) => {

return a + 100;

}

// 2. Remove the body braces and word "return" -- the return is implied.

(a) => a + 100;

// 3. Remove the argument parentheses

a => a + 100;

Arrow funkcija moze samo da se koristi kao function expression, odnosno moze samo da bude anonymous. Moze se smestiti u varijablu.

// Traditional Function

function bob (a){

return a + 100;

}

// Arrow Function

let bob = a => a + 100;

**!!!Arrow funkcija se ne moze koristiti kao metoda objekta zbog toga sto ne razlikuje this scope!!!**

**!!!AUTOMATSKI IMA RETURN. NE TREBA DA SE NAVODI!!!**

### IIFE function

Immediately Invoked Function Expression-funkcija koja se poziva odmah pri deklaraciji

(function () {

/\* ... \*/

})();

Arrow IIFE function sintaksa

(() => {

/\* ... \*/

})();

# Konverzija tipova podataka

JavaScript varijable mogu biti konvertovane u neku novu varijablu i neki drugi tip podataka:  
• korišćenjem neke JavaScript funkcije  
• automatski od strane JavaScript-a  
Operator typeof  
Za otkrivanje tipa podataka neke JavaScript varijable koristi se operator **typeof**.  
**typeof** "Jovan" *// vraća "string"***typeof** 3.14 *// vraća "number"***typeof** NaN *// vraća "number"***typeof** false *// vraća "boolean"*  
**typeof** [1,2,3,4] *// vraća "object"***typeof** {name:'Jovan', age:34} *// vraća "object"***typeof new Date**() *// vraća "object"***typeof function** () {} *// vraća "function"***typeof** myCar *// vraća "undefined"***typeof** null *// vraća "object"*Pretvaranje brojeva u stringove  
Globalna metoda **String()** može konvertovati brojeve u stringove.  
Ona može biti korišćena na bilo kojem tipu brojeva, literala, varijabli ili izraza.  
**String**(x) *// vraća string od brojčane varijable x***String**(123) *// vraća string od brojčanog literala 123***String**(100 + 23) *// vraća string od broja iz izraza*  
Number metoda **toString()** radi istu stvar.  
x.toString()  
(123).toString()  
(100 + 23).toString()  
Pretvaranje boolean-a u stringove  
Globalna metoda **String()** može konvertovati boolean-e u stringove.  
**String**(false) *// vraća "false"***String**(true) *// vraća "true"*Boolean metoda **toString()** radi istu stvar.  
false.toString() *// vraća "false"*true.toString() *// vraća "true"*Pretvaranje stringova u brojeve  
Globalna metoda **Number()** može konvertovati stringove u brojeve.  
Stringovi koji sadrže brojeve (poput "3.14") konvertuju se u brojeve (poput 3.14).  
Prazni stringovi se konvertuju u 0.  
Sve ostalo se konvertuje u NaN (Not a Number).  
**Number**("3.14") *// vraća 3.14***Number**(" ") *// vraća 0***Number**("") *// vraća 0***Number**("99 88") *// vraća NaN*Metoda **parseFloat()** parsuje string i vraća decimalni broj.  
Metoda **parseInt()** parsuje string i vraća integer.  
Pretvaranje boolean-a u brojeve  
Globalna metoda **Number()** može konvertovati boolean-e u brojeve.  
**Number**(false) *// vraća 0***Number**(true) *// vraća 1*

# Loops

Loop-ovi su izrazi u Javascriptu uz pomoć kojih omogućavamo izvršavanje bloka koda više puta. Postoje 4

načina da postignemo loop u našem programu :

**while loop**

Kao što je If statement osnovni kondicional, tako je while statement primarni loop u Javascriptu :

var count = 0;

while (count < 10) {

console.log(count);

count++;

}

Sve dok je expression unutar zagrada true izvrsavaće se ponovo blok koda koji se nalazi unutar krivih zagrada. Kod while loop-a moramo da pazimo da stalno povećavamo test varijablu (count) da ne bi upali u beskonačni loop.

**do/while**

do/while loop je sličan while statement-u, razlika je u tome što se na dnu naseg loop-a nalazi expression

do{

statement

}

while (expression);

do/while se retko koristi, ali ako želite da osigurate da će se kod izvršiti barem jednom, rešenje je ovaj način loop-a.

**for loop**

Najpogodniji i najpopularniji nacin loop-a je for loop :

for(var count = 0; count < 10; count++){console.log(count);

}

Razlika između for loopa i while loop-a je u tome da for loop-u nije potrebna varijabla koju testiramo, već ćemo istu kreirati i testirati unutar zagrada posle reči if.

Jedan od načina korišćenja for loop-a je da listamo sve elemente array-a :

var a = [1,2,3,4,5,6,7,8];

for(var i = 0; i < a.length; i++) {

console.log(a[i]);

}

**for/in loop**

for/in statement isto koristi for ključnu reč, ali je potpuno drugačiji od for loopa. Koristimo ga da listamo sve properties objekta :

var obj = { name: "Samsung",model: "Note7" }

for(var key in obj) {

console.log(obj[key]); // "Samsung", "Note7"

}

For in loop mozemo koristiti i za nizove. U tom slucaju vraca ce indeks kao

let objekat={

    ime: "Nemanja",

    broj: 2

}

let niz=["Automobil", 32];

for(let key in niz){

    console.log(key)///////////// ispisuje indekse(0 i 1)

    console.log(typeof(key)) //////// ispisace STRING za obe vrednosti

}

for(let key in objekat){

    console.log(key)//// Vraca naziv propertya

    console.log(typeof(key))///vratice String za oba elementa niza}

# Reference

**Prosti tipovi** se prosledjuju **po vrednosti**.

let x=10;

let y=x;

console.log("Y je: "+y);//Y je 10

y=32;

console.log("X je:"+x);//X je 10--Ostalo je 10, nije se promenilo pri promeni Y

**Slozeni tipovi** se prosledjuju *po referenci*(prosledjuje se memorijska lokacija).

let niz1=[1,2,3];

let niz2=niz1;

console.log("Niz 2: "+niz2);//Niz 2: 1,2,3

niz2.push(4)//dodace se i u niz1 4

console.log("Niz 1: "+niz1);//Niz 1: 1,2,3,4

console.log("Niz 2: "+niz2);//Niz 2: 1,2,3,4

# Operatori

Najjaci operator u JavaScriptu su ()- zagrade

&& - i

||- ili

!== - negacija

# DOM

Document Object Model je osnovni API (Application Program Interface) za predstavljanje i manipulisanje HTML elementima. Pri ucitavanju web stranice, kreira se dokument “document” koji sadrzi sve HTML elemente u obliku objekta(Nodes list). Od svakog HTML elementa kreira se pojedinacan objekat.

window.document

## BOM-Browser Object Model

window.navigator---nas browser preko kojeg gledamo stranicu

window.history---putanje kojima smo se kretali (back, forward)

## Selectori

HTML Collection nema foreach metodu zato sto je Array like.

### document.getElementById(“id”);

Vraca null ako ne postoji takav element.

**!!!Vraca element kao objekat ako postoji!!!**

### document.getElementsByTagName(“tagname”)

Vraca HTML collection svih elemenata sa zadatim tag name.

The getElementsByTagName() property is read-bttaonly.

### document.getElementsByClassName(“klasa”)

Vraca HTML collection svih elemenata sa zadatom klasom.

### document.querySelector(“CSS selector”)

Vraca **PRVI** element kao object koji ispunjava zadati css selector(id, class, tag name itd)

### document.querySelectorAll(“CSS selector”)

Vraca NodeList svih elemenata koji ispunjavaju zadati CSS selector

## window.getComputedStyle(element)

**!!!Ono sto zadamo u style.css ne ulazi u element.style property!!!**

Sintaksa:  
*window*.getComputedStyle(element, pseudoElement(optional))

**window.getComputedStyle(element)** vraca CSSStyleDeclaration Object sa svim CSS propertys za zadati element.

## innerHTML, innerText, textContent

HTML Example

<p id="myP">   This element has extra spacing     and contains <span>a span element</span>.</p>

JavaScript Examples

let text = document.getElementById("myP").innerText;  
  
let text = document.getElementById("myP").innerHTML;  
  
let text = document.getElementById("demo").textContent;

|  |
| --- |
| The innerText property returns: This element has extra spacing and contains a span element. |
| The innerHTML property returns:    This element has extra spacing    and contains <span>a span element</span>. |
| The textContent property returns:    This element has extra spacing    and contains a span element. |

className  
classList

attributes

# Events

Javascript programi koriste "event-driven" programski model. To znači da web brauzer generiše event  
(dešavanje) kada se nešto desi sa našom web stranicom ili samim brauzerom. Na primer, brauzer emituje event  
kada završi učitavanje stranice, kada korisnik pređe mišem preko linka ili pritisne neko dugme na tastaturi.  
Ako nam je od značaja neki od tih evenata mi možemo registrovati sam event i odgovoriti na njega putem  
nekog koda. Ovakav model programiranja nije jedinstven za Javascript već ga koriste mnogi programski jezici.  
Photoshop je, recimo, program koji koristi takav model.  
Event handler ili event listener je funkcija koju pišemo kao odgovor na neki event.

**!!!Dogadjaji su property od elementa. Vec definisan property koji ima vrednost NULL !!!**

Posotji tri nacina setovanja dogadjaja:

* Inline

<div onclick=”console.log(‘Radi’)”><div>

* Preko propertya

btn.onclick=function(){

console.log(“Radi”);

}

* uz pomoc addEventListener

btn.addEventListener(“onlick”, function(){  
console.log(“Radi”);

})

addEventListener ima i treci argument:

btn.addEventListener(“onlick”, function(){  
console.log(“Radi”);

},false/true)

Ovaj argument definise Event flow, odnosno da li ce da pradi ceo event flow od parent elemenata kad child elementima.

Ovo se odnosi na Event bubbling i Event capturing

**!!!Event je string!!!**

**!Funkcija koja se okida na neki dogadjaj se naziva EVENT HANDLER!**

**!!!!Event handler kao prvi argument prihvata EVENT OBJECT!!!!**

**function makeItRed(e){**

**console.log(e)**

**}**

**e-event object**

## Skidanje eventa

btn.removeEventListener(<koji event>, <koju funkciju>)

# Function closure

Closure daje pristup skope-u Outer funkcije iz Inner funkcije. **Closures se kreira svaki put kada je funkcija kreirana!**();

}

init();

**function init() {**

**var name = 'Mozilla';** // name is a local variable created by init

**function displayName() {**

**// displayName() is the inner function, a closure**

**console.log(name); //** use variable declared in the parent function

**}**

**displayName();**

**}**

**init();----rezultat “Mozilla”**

**DisplayName** funkcija je imala pristup **name** varijabli iz Outter funkcije **init.**

Medjutim, ako hocemo da displayName funkciju iskoristimo vise puta onda je ne pozivamo unutar init funkcije nego je vracamo preko return.

function makeFunc() {

const name = 'Mozilla';

function displayName() {

console.log(name);

}

return displayName;

}

const myFunc = makeFunc();

myFunc();

Na ovaj nacin mozemo da pozivamo vise puta displayName funkciju.  
Ono sto je bitno kod ove situacije je da kada kreiramo name varijablu MORAMO da koristimo const ili let. Tako stvaramo **lexical environment. Pri pozivanju funkcije makeFunc, stvara se closure i name cuva vrednost koju je imala u tom trenutku(u ovom slucaju “Mozilla”).  
Kada pozovemo funkciju myFunc, mi ustvari pozivamo funkciju displayName koja prikazuje vrednost varijable name.**

function makeSizer(size) {

return function () {

document.body.style.fontSize = `${size}px`;

};

}

const size12 = makeSizer(12);

const size14 = makeSizer(14);

const size16 = makeSizer(16);

<button id="size-12">12</button>

<button id="size-14">14</button>

<button id="size-16">16</button>

document.getElementById('size-12').onclick = size12;

document.getElementById('size-14').onclick = size14;

document.getElementById('size-16').onclick = size16;

U primeru iznad, kreiramo 3 buttons za promenu fontsize.  
Da se ne bi ponavljali, kreiramo jednu funkciju kojoj prosledjujemo velicinu za fontsize.  
Kreiramo funkciju makeSizer kojoj prosledjujemo velicinu a koja vraca funkciju koja menja fontsize za ceo body. Ovo je moguce jer imamo closure. Pri svakom pozivu makeSizer funkcije, size argument je razlicit i tu vrednost koju prosledimo, vracena funkcija pamti i koristi za promenu fontsize **JER SE PRI SVAKOM KREIRANJU FUNKCIJE KREIRA I CLOSURE.**

# TRICKS AND TIPS

## Intl.RelativeTimeFormat

const f=new Intl.RelativeTimeFormat(“en-us”, {style: “long”})

## new Set()

Izbacivanje duplikata iz Arraya

const orgArray=[1,2,3,4,5, “true”, “Nemanja”, 78, 2, 3, 4, “true”]

const uniqueArray= […new Set(orgArray)];

console.log(uniqueArray)

// [1,2,3,4,5, “true”, “Nemanja”, 78]

## map()

Konvertovanje stringa u integer kroz map metodu

const arr= [ “4”, “67”, “98”];

const numbers= arr.map(Number);

console.log(numbers);

// [4, 67, 98]-bice type Number za sve elemente niza