

JavaScript osnove

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

- ◆ JavaScript:
- ◆ Princip rada
- ◆ Proceduralni aspekti
 - Operatori
 - Funkcije
 - Varijable
 - Tipovi podataka (templejt stringovi)
 - Nizovi
- ◆ Objektno-orientisani aspekti
 - Metode
 - Funkcije kao objekti
 - Prototipi
 - Klase i nasleđivanje
- ◆ Aspekti funkcionalnog programiranja
 - Anonimne funkcije
 - Arrow funkcije
- ◆ Primeri: codepen.io

JavaScript istorijat

- ◆ Nastao 1995 (Brendan Eich)
- ◆ Ko-osnivač Mozilla-e
- ◆ Nema naročitih veza sa Java jezikom
 - Razlozi isključivo marketinški
- ◆ Prva verzija jezika napisana za 10 dana
- ◆ Glavne odluke o jeziku su donete zbog politike kompanije, ne tehničkih (stručnih) razloga
- ◆ Vremenom je jezik unapređivan
 - Tradicionalni JavaScript – globalni opsezi; nema klasa/modula; mogu biti veće količine koda uređenog sekvencialno
 - Moderni (unapređeni) JavaScript – klase, moduli; unapređene mogućnosti jezika, i izgled i organizacija koda;
 - ECMAScript (TypeScript)

JavaScript programski jezik

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

JavaScript u Web stranici

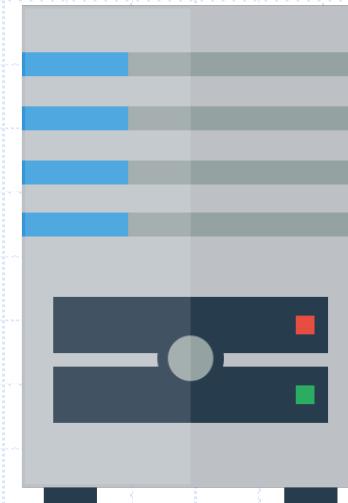
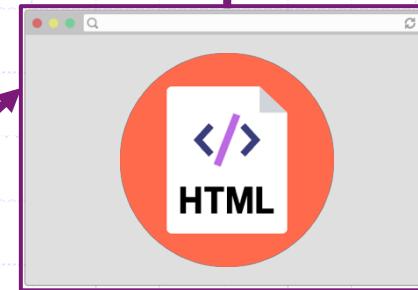
```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>CS 193X</title>  
    <link rel="stylesheet" href="style.css" />  
    <script src="filename.js"></script>  
  </head>  
  <body>  
    ... contents of the page...  
  </body>  
</html>
```

console.log()

- ◆ Ispisivanje log poruka u JavaScript-u

Kako radi JavaScript ?

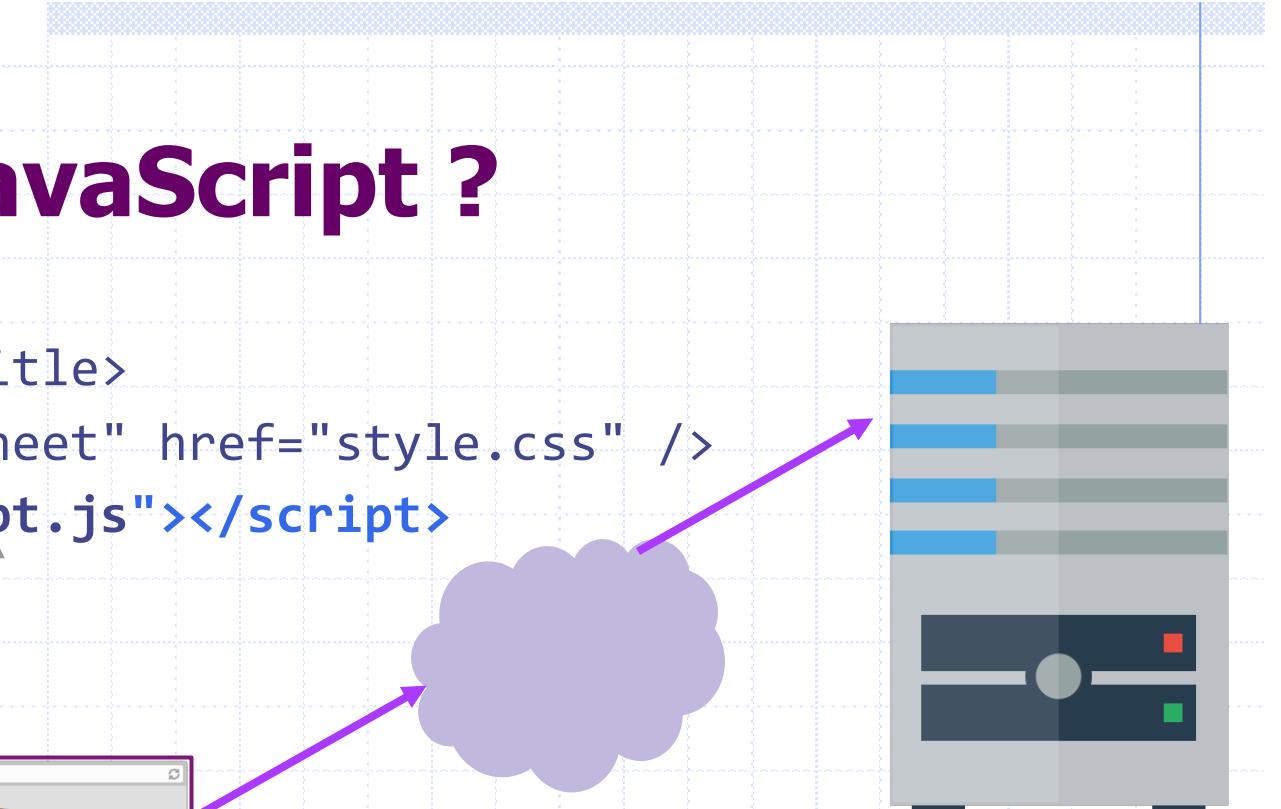
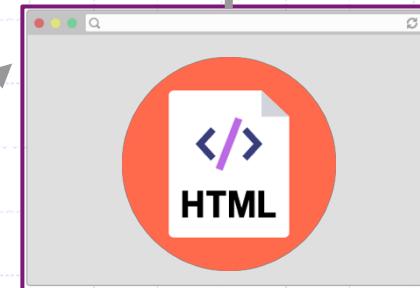
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



Pretraživač parsira HTML datoteku, i iz script taga čita nazine JavaScript datoteka.

Kako radi JavaScript ?

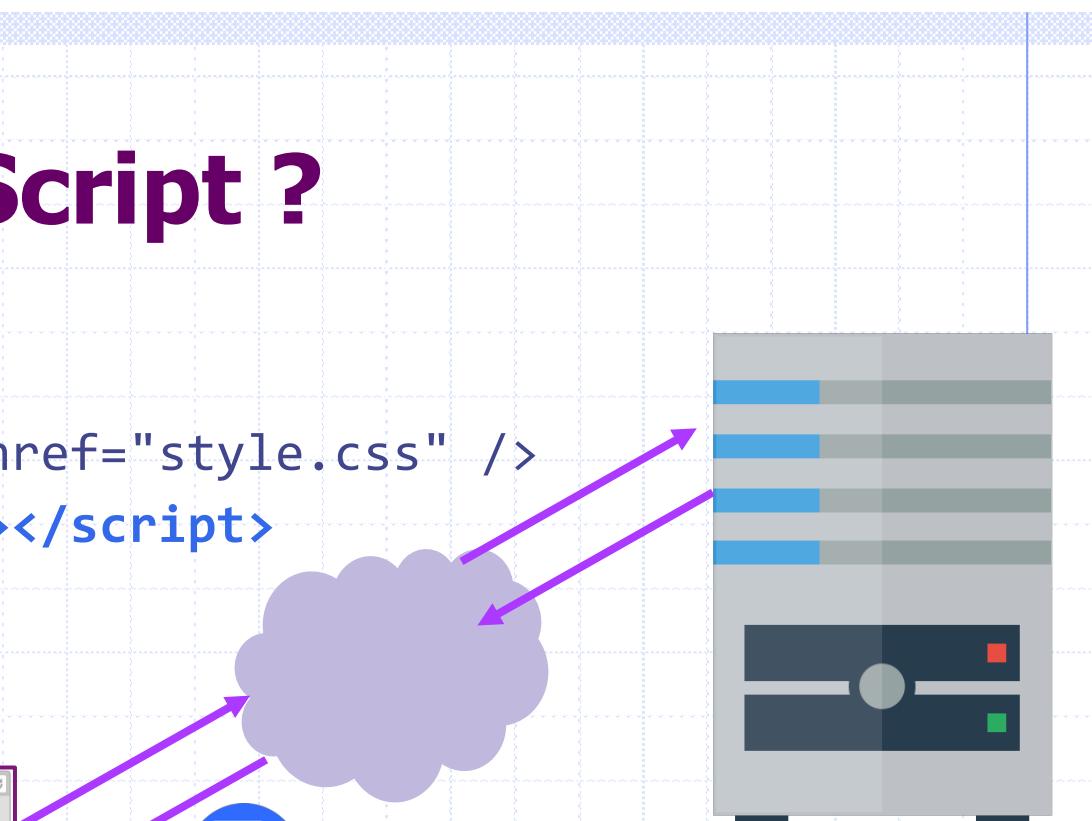
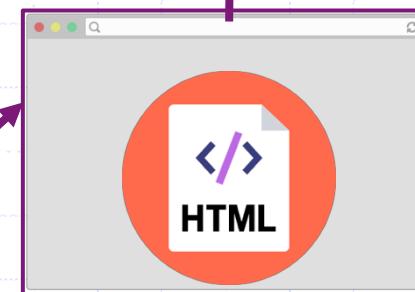
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  → <script src="script.js"></script>
</head>
```



Pretraživač šalje zahtev serveru za datoteku script.js, slično
kao za CSS datoteku ili sliku ...

Kako radi JavaScript ?

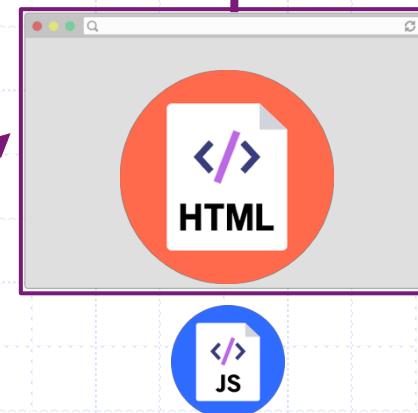
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



I server šalje JavaScript datoteku, kao što bi poslao CSS datoteku ili sliku ...

Kako radi JavaScript ?

```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



```
console.log('Hello, world!');
```

U ovom trenutku, JavaScript datoteka se izvršava na klijentskoj strani, tj. u pretraživaču računara korisnika.

Kako radi JavaScript ?

- ◆ Ne postoji **main** metoda:
- ◆ Instrukcije se izvršavaju sekvencijalno od vrha ka dnu
- ◆ Ne postoji prevođenje (kompilacija) koda od strane programera
- ◆ JavaScript datoteke se prevode i izvršavaju kako se učitavaju od strane pretraživača
- ◆ Just-in-time (JIT) prevođenje
- ◆ V8-Ignition je najpopularniji JavaScript engine (Chrome, NodeJS):
 - ◆ Princip je da se kod prevodi i izvršava što je brže moguće:
 - ◆ Izvorni JavaScript kod se prevodi u byte kod
 - ◆ Byte kod se izvršava od strane interpretera
 - ◆ SuperMonkey (Mozilla)



Elementi JavaScript jezika

Operatori

```
i = 3;  
i = i * 10 + 3 + (i / 10);  
while (i >= 0) {  
    sum += i*i;  
    i--;
```

Standardne operacije:

/ % + - ! >= <= > < && || ?:

Naredbe skoka:

continue/break/return

Petlje i naredbe grananja

for petlja:

```
for (let i = 0; i < 5; i++) { ... }
```

while petlja:

```
while (notFinished) { ... }
```

komentari:

```
// comment ili /* comment */
```

uslovne (**if**) naredbe:

```
if (...) {  
  ...  
} else {  
  ...  
}
```

Funkcije

Jedan od načina za definisanje JavaScript funkcije je kao ispod:

```
function name() {  
    statement;  
    statement;  
    ...  
}
```

Funkcije

```
function hello() {  
    console.log('Hello!');  
    console.log('Welcome to JavaScript');  
}  
  
hello();  
hello();
```

Console

"Hello!"

"Welcome to JavaScript"

"Hello!"

"Welcome to JavaScript"

Da li će biti izvršeno ?

```
hello();  
hello();
```

```
function hello() {  
    console.log('Hello!');  
    console.log('Welcome to JavaScript');  
}
```

Hoće, za ovaj način
definisanja funkcija !

Definicija funkcije je *hoisted*
(može se pozivati pre
definisanja)

Console
"Hello!"

"Welcome to JavaScript"

"Hello!"

"Welcome to JavaScript"

JavaScript funkcije

- ◆ Imaju promenjiv broj argumenata:
- ◆ Varijabilan niz argumenata (`arguments[0]` je prvi argument)
- ◆ Nenavedeni argumeti imaju vrednost `undefined`
- ◆ Sve funkcije imaju povratnu vrednost (podrazumevana je `undefined`)
- ◆ Funkcije su objekti prvog reda (*first-class objects*):
- ◆ Mogu biti vrednost varijable ili elementa niza, prosleđeni kao argument funkcije, povratna vrednost funkcije

Funkcija kao objekat prvog reda

```
var aFuncVar = function (x) {  
    console.log('Called with', x);  
    return x+1;  
};  
function myFunc(routine) {  
    console.log('Called with', routine.toString());  
    var retVal = routine(10);  
    console.log('retVal', retVal);  
    return retVal;  
}  
myFunc(aFuncVar);
```

Rezultat ?

```
Called with function (x) {  
    console.log('Called with', x);  
    return x + 1;  
}  
Called with 10  
retVal 11
```

Varijable var, let, const

Varijable se mogu definisati na tri načina:

```
// Opseg vidljivosti je globalan
```

```
var x = 15;
```

```
// Opseg vidljivosti je blok
```

```
let fruit = 'banana';
```

```
// Konstanta
```

```
const isHungry = true;
```

Tip varijable se ne navodi pre definisanja

Dinamičko tipiziranje – tip se definiše dinamički, prilikom dodelje vrednosti

Varijable dobijaju tip poslednje dodeljene vrednosti

Parametri funkcije

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
}
```

Parametri funkcije se ne deklarišu sa var, let, ili const

var – opseg vidljivosti

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
    console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Šta se dešava ako želimo da ispišemo vrednost i van petlje ?

var – opseg vidljivosti

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
    console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Console

"hello"

"hello"

"hello"

"Value of i is 3"

Vrednost *i* je vidljiva van petlje zbog toga što varijable deklarisane sa var imaju opseg vidljivosti na nivou funkcije u kojoj su deklarisani

var – opseg vidljivosti

- ◆ Varijable deklarisane sa **var** imaju opseg vidljivosti na nivou **funkcije** u kojoj se definišu, ne bloka naredbi
- ◆ Možete koristiti varijablu i nakon završetka bloka koda (tj. posle petlje ili if naredbe u kojoj su deklarisane)

```
var x = 10;  
if (x > 0) {  
    var y = 10;  
}  
console.log('Value of y is ' + y);
```

"Value of y is 10"

let – opseg vidljivosti

```
function printMessage(message, times)
{
    for (let i = 0; i < times; i++) {
        console.log(message);
    }
    console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

Šta se dešava ako želimo da ispišemo vrednost i van petlje ?

let – opseg vidljivosti

```
function printMessage(message, times)
{
    for (let i = 0; i < times; i++) {
        console.log(message);
    }
    console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

3 hello

✖ ► Uncaught ReferenceError: i is not defined
at printMessage (pen.js:29)
at VM479 pen.js:23

let ima opseg
vidljivosti
bloka !

const – opseg vidljivosti

```
let x = 10;  
if (x > 0) {  
    const y = 10;  
}  
console.log(y);
```

Value of i is 3

Value of y is 10

✖ Uncaught ReferenceError: y is not defined

[console_runner-1df7d...81e5fba982f6af.js:1](#)

[console_runner-1df7d...81e5fba982f6af.js:1](#)

[pen.js:25](#)

const ima opseg vidljivosti **bloka** !

var, let, const

- ◆ Deklracija varijable u JS:

- ◆ Opseg vidljivosti funkcije:

```
var x = 15;
```

- ◆ Opseg vidljivosti bloka:

```
let fruit = 'banana';
```

- ◆ Opseg vidljivost bloka (konstanta):

```
const isHungry = true;
```

Šta je blok ?

- ◆ Skup naredbi ograničen vitičastim zagradama ({})

Java, C#, C++:

```
if (...) {  
    int x = 5;  
    ...  
}
```

// ovde se ne može pristupiti x

Blok i opseg vidljivosti

- Isto važi i u JavaScript-u, pod uslovom da su varijable deklarisane sa **const** i **let**

```
if (...) {  
    let x = 5;  
    ...  
}  
// ovde se x ne vidi
```

- Ali ako koristimo **var**, varijabla se vidi u čitavoj funkciji nezavisno od blokova

```
if (...) {  
    var x = 5;  
    ...  
}  
// ovde se x vidi
```

JavaScript tipovi podataka

- ◆ JavaScript **varijable nemaju** tipove
- ◆ JavaScript **vrednosti imaju** tipove
- ◆ Osnovni tipovi podataka (primitive):
 - ◆ Boolean – true ili false
 - ◆ Number – double (ne postoji integer)
 - ◆ String – 'primer stringa 1' "primer stringa 2"
 - ◆ Symbol – jedinstven simbol kao objekat
 - ◆ Null – nulta vrednost
 - ◆ Undefined – varijabli nije dodeljena vrednost
 - ◆ Postoje i Object (tj. objektni) tipovi, kao što su Array, Date, String (omotač-wrapper primitivnog tipa)

JavaScript brojevi

- ◆ Svi brojevi su tipa realnog broja
- ◆ Operatori su identični kao u Java jeziku
- ◆ Tri simboličke konstante kao vrednosti:
 - ◆ -Infinity (Number.MIN_VALUE)
 - ◆ +Infinity (Number.MAX_VALUE)
 - ◆ NaN – *not-a-number*
- ◆ Postoji Math klasa: Math.floor, Math.ceil

Stringovi

```
let mojString = 'primer';
mojString += 'stringa';
mojString = mojString.toUpperCase();
console.log("Sadržaj stringa: " + mojString);
```

- ◆ Mogu se koristiti jednostruki ili dvostruki navodnici
- ◆ Nepromenjivi (immutable)
 - Jednom kreiran, ne može se menjati kao takav
 - Novi string se kreira kao rezultat operacije nad originalom
- ◆ Ne postoji znakovni tip (char)
- ◆ Konkatenacija (+)
- ◆ Veličina stringa – length atribut
- ◆ Korisne metode: indexOf(), charAt(), match(), search(), replace(), toUpperCase(), toLowerCase(), slice(), substr()

Templejt stringovi (*template strings*)

- ◆ Stringovi koji omogućavaju dinamičko ugrađivanje i izračunavanje izraza pomoću *placeholder-a*
- ◆ String nije ograničen znakovima navoda već back-tick simbolom (`) !

```
string text ${expression} string text  
tag `string text ${expression} string text`
```

- ◆ U prvom slučaju, računa se vrednost izraza, konvertuje u string, i spaja sa ostatkom stringa (pre i posle izraza)
- ◆ U drugom slučaju, izračunavanje, konverzija i spajanje se vrše unutar funkcije (tzv. tag funkcije)

Templejt stringovi - primer

```
var a = 5;  
var b = 10;  
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
```

"Fifteen is 15 and not 20."

```
var name = 'Student';  
var age = 22;  
function greet(arr, nameArg, ageArg) {  
    console.log(arr[0] + nameArg + arr[1] + ageArg + arr[2]);  
}  
greet`${name} ima ${age} godine?`;
```

"Student ima 22 godine?"

Logički tip (**boolean**)

- ◆ Dve vrednosti: true i false
- ◆ Operatori: && (logičko i); || (logičko ili); !(negacija)
- ◆ Ne-logičke vrednosti mogu biti korišćene u uslovnim naredbama
- ◆ Konvertuju se u odgovarajuću vrednost:
 - ◆ null, undefined, 0, NaN, "", "" se pretvaraju u false
 - ◆ Sve ostalo se pretvara u true (sadržaji svih tipova koji nisu prazni)

```
if (username) { // username is defined }
```

object tip

- ◆ JavaScript objekat je kolekcija parova **atribut-vrednost** (*property-value*)

```
var obj1 = {};  
var obj2 = {name: "Alice", age: 23, state:"California"};
```

- ◆ Naziv atributa može biti proizvoljan string
- ◆ Atributi se referenciraju kao:
- ◆ struktura - **obj2.name**
- ◆ hash tabela sa kjučevima - **obj2["name"]**
- ◆ Fleksibilno dodavanje i brisanje atributa:

```
var obj1 = {};  
obj1.name = "Alice"; // ili obj1[name] = "Alice"
```

```
var obj1 = {name: "Alice"};  
delete obj1.name;
```

Jednakost – starija pravila (*loose equality*)

JavaScript `==` i `!=` su zapravo dve operacije: implicitna konverzija tipa pre poređenja (tabela: poređenje jednakosti)

		Operand B					
		Undefined	Null	Number	String	Boolean	Object
Operand A	Undefined	true	true	false	false	false	false
	Null	true	true	false	false	false	false
	Number	false	false	A === B ToNumber(B)	A === ToNumber(B)	A === ToNumber(B)	A == ToPrimitive(B)
	String	false	false	ToNumber(A) == B	A === B	ToNumber(A) == ToNumber(B)	A == ToPrimitive(B)
	Boolean	false	false	ToNumber(A) == B	ToNumber(A) == ToNumber(B)	A === B	ToNumber(A) == ToPrimitive(B)
	Object	false	false	ToPrimitive(A) == B	ToPrimitive(A) == B	ToPrimitive(A) == ToNumber(B)	A === B

Jednakost – starija pravila (*loose equality*)

JavaScript == i != su zapravo dve operacije: implicitna konverzija tipa pre poređenja)

```
' ' == '0' // false
' ' == 0 // true
0 == '0' // true
NaN == NaN // false
[' '] == '' // true
false == undefined // false
false == null // false
null == undefined // true
```

Jednakost – novija pravila, ECMA Script (*strict equality*)

Nema implicitne konverzije

Pravila:

Različiti tipovi vrednosti -> \neq

Isti tipovi vrednosti -> poređenje vrednosti

```
'' === '0' // false
'' === 0 // false
0 === '0' // false
[''] === '' // false
false === undefined // false
false === null // false
null === undefined // false
```

Koristiti $==$ i $!=$ umesto $==$ i $!=$

loose (==) i strict (===) equality

x	y	==	===
undefined	undefined	true	true
null	null	true	true
true	true	true	true
false	false	true	true
'foo'	'foo'	true	true
0	0	true	true
+0	-0	true	true
+0	0	true	true
-0	0	true	true
0	false	true	false
""	false	true	false
""	0	true	false
'0'	0	true	false

Null i undefined

- ◆ Razlika ?
- ◆ Null predstavlja nultu vrednost kao takvu (Java null)
- ◆ Undefined znači da varijabli nije dodeljena vrednost (varijabla je samo deklarisana)

```
x = null;  
let y;  
console.log(x);  
console.log(y);
```

```
null  
undefined
```

Nizovi (array)

- ◆ **Objektni** tipovi koji se kreiraju i koriste kao **liste**
- ◆ Objektni: memorijska struktura podrazumeva referencu na element niza i sam element niza, tj, njegovi podaci
- ◆ **Lista:** struktura tipa liste

[element0, element1, ..., elementN]

new Array(element0, element1[, ...[, elementN]])

new Array(arrayLength)

Parametri:

elementN: element niza kada se niz kreira ručno kao lista

arrayLength: ceo broj koji se prosleđuje kada se niz kreira kao objekat tipa Array. Kreira se niz sa praznim elementi (ne undefined)

Dužina niza: atribut length (ne metoda)

Indeks počinje od vrednosti 0

Dužina i tipovi elemenata niza su promenjivi

Operacije sa nizovima

- ◆ Kreiranje niza

```
var fruits = ['Apple', 'Banana'];
```

```
console.log(fruits.length);
```

- ◆ Pristup elementima niza

```
var first = fruits[0];
```

```
var last = fruits[fruits.length - 1];
```

Operacije sa nizovima

◆ Iteracija niza (for petlja)

```
for(let fruit of fruits) {  
    console.log(fruit);  
}
```

"Apple"

"Banana"

◆ Iteracija niza (funkcija)

```
fruits.forEach(function(item, index, array) {  
    console.log(item, index);  
});
```

"Apple" 0

"Banana" 1

Operacije sa nizovima

- ◆ Dodavanje/uklanjanje elementa sa početka niza

```
var first = fruits.shift();
var newLength = fruits.unshift('Strawberry');
```

- ◆ Dodavanje/uklanjanje elementa sa kraja niza

```
var newLength = fruits.push('Orange');
var last = fruits.pop();
```

- ◆ Indeksiranje elementa

```
var pos = fruits.indexOf('Banana');
// 1
```

Operacije sa nizovima

- ◆ Uklanjanje podniza od zadate pozicije (indeksa)

```
var vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot'];
console.log(vegetables);

var pos = 1, n = 2;

var removedItems = vegetables.splice(pos, n);

console.log(vegetables);
console.log(removedItems);
```

["Cabbage", "Carrot"]

["Turnip", "Radish"]

Kopiranje nizova – shallow copy

```
var array1 = [1, 2, 3, 4, 5, 6, 7, 8];
var array2 = array1;
var array3 = array1.slice();
```

const array1 =



const array2 =

Kopiraju se reference na elemente niza, ne i sami element !

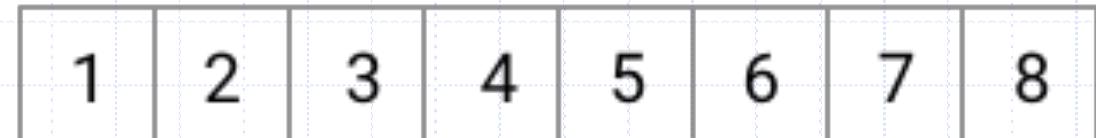
Napomena: načini opisani ispod važe samo za jednodimenzione nizove !

Kopiranje nizova – deep copy

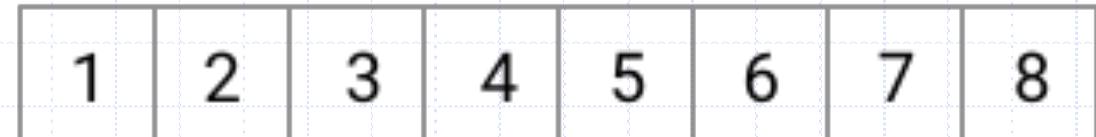
```
var array4 = array1.map(elem => elem);
```

```
var array2 = [];
array1.forEach(elem => {
    array2.push(elem)
});
```

const array1 =



const array2 =



Kopiraju se reference i elementi niza !

Datumi

```
new Date(); //tekuci datum  
new Date(value);  
new Date(dateString);  
new Date(year, monthIndex [, day [, hours [, minutes [,  
seconds [, milliseconds]]]]]);
```

Poseban tip objekta: `typeof date == 'object'`

Broj milisekundi od 1. Januara 1970 (UTC)

Korisne metode (getter/setter):

`Date.now()`; `Date.parse()`
`getDate`/`Time`/`Hours`
`setDate`/`Time`/`Hours`

Datumi - primer

```
var start = Date.now();
doSomethingForALongerTime();
var end = Date.now();
var elapsed = end - start;
```

```
var start = new Date();
doSomethingForALongerTime();
var end = new Date();
var elapsed = end.getTime() - start.getTime();
```

Obrada izuzetaka – try/catch

- ◆ Obrada, tj. prijavljivanje izuzetaka (grešaka) koje nastanu u toku izvršavanja programa

```
try {  
    nonExistentFunction();  
}  
catch(error) {  
    console.error(error);  
}
```

► ReferenceError: nonExistentFunction is not defined

Sadržaj error poruke može varirati u zavisnosti od pretraživača

try/catch – bolji primer

```
try {  
    nonExistentFunction();  
}  
catch (err) {  
    console.log("Error call func", err.name, err.message);  
}
```

```
Error call func ReferenceError  
nonExistentFunction is not defined
```

Varijante:

try...catch

try...finally

try...catch...finally

try/catch – obrada specifičnih grešaka

- ◆ Može se korisititi veći broj clause izjava za specifičan tip grešaka koji može nastati

```
try {  
    myroutine(); // may throw three types of exceptions  
} catch (e if e instanceof TypeError) {  
    // statements to handle TypeError exceptions  
} catch (e if e instanceof RangeError) {  
    // statements to handle RangeError exceptions  
} catch (e if e instanceof EvalError) {  
    // statements to handle EvalError exceptions  
} catch (e) {  
    // statements to handle any unspecified exceptions  
    logMyErrors(e); // pass exception object to error handler  
}
```

try/catch – česta upotreba

- ◆ Pristup skladištu podataka (server-side JS)

```
openMyFile();
try {
    writeMyFile(theData);
}
finally {
    closeMyFile();
}
```



Naredbe koje se izvršavaju posle izvršenja try naredbi. Izvršavaju se uvek, nezavisno od toga da li nastane izuzetak.

JavaScript programski jezik

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

OO JS: metode

- ◆ Atribut (property) objekta može biti i funkcija

```
var o = {count: 0};  
o.increment = function (amount) {  
    if (amount == undefined) {  
        amount = 1;  
    }  
    this.count += amount;  
    return this.count;  
}  
  
console.log(o.increment());  
console.log(o.increment(3));
```

1

4

this

- ◆ Globalni kontekst:
- ◆ Van funkcije, ukazuje na globalni kontekst izvršavanje, na primer window objekat
- ◆ Kontekst funkcije:
- ◆ Funkcija u okviru koje se poziva

```
var o = {oldProp: 'this is an old property'};  
o.aMethod = function() {  
    this.newProp = "this is a new property";  
    return Object.keys(this);  
}  
console.log(o.aMethod());
```

```
["oldProp", "aMethod", "newProp"]
```

Funkcije mogu imati attribute

```
function plus1(value) {  
  if (plus1.invocations == undefined) {  
    plus1.invocations = 0;  
  }  
  plus1.invocations++;  
  return value + 1;  
}  
console.log(plus1());  
console.log(plus1(3));
```

NaN

4

Funkcije se mogu posmatrati kao klase

```
function Rectangle(width, height) {  
    this.width = width;  
    this.height = height;  
    this.area = function() { return this.width*this.height; }  
}  
var r = new Rectangle(26, 14);  
  
console.log(r.constructor.name);  
console.log(r);
```

"Rectangle"

```
Object {  
  area: function () {return this.width * this.height;},  
  height: 14,  
  width: 26  
}
```

Prototipi (prototypes)

- ◆ Object.prototype: svaki JS objekat je instanca Object
- ◆ Svaki JS objekat ima privatni atribut (*property*) koji se zove **prototype** i sadrži link ka drugom objektu (koji naziva svojim prototipom)
 - Svaka kreirana instanca ima svoj prototip objekat
- ◆ Referencirani objekat ima svoj prototype atribut i tako dalje, formirajući lanac prototipa (*prototype chain*)
- ◆ JS objekti predstavljaju dinamički skup atributa
 - Novi atributi se mogu dodeljivati objektima, tj.instancama u bilo kom trenutku
- ◆ Kada pokušamo da pristupimo atributu objekta, atribut neće biti tražen samo u konkretnom objektu, već i u njegovom prototipu, prototipu prototipa, itd. sve dok se atribut ne pronađe, ili ne dođe do kraja lanca prototipa
- ◆ Uopšteno gledano, atributi objekta se mogu posmatrati kao oni koje on sadrži + atributi lanca prototipa
 - Nasleđivanje zasnovano na prototipima

```
function Rectangle(width, height) {  
    this.width = width;  
    this.height = height;  
}  
Rectangle.prototype.area = function()  
{  
    return this.width*this.height;  
}  
var r = new Rectangle(26, 14);  
var v = r.area();  
console.log(r);  
console.log(v);
```

Prototipi primer

Funkcije takođe imaju atribut prototype (zbog toga što su i one instance tipa Object, tj. objekti prvog reda)

```
Object {  
    area: function ()  
    {  
        return this.width * this.height;  
    },  
    height: 14,  
    width: 26  
}
```

Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
console.log(car1.stereo);  
console.log(car2.gps);
```

undefined

undefined

Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
console.log(car1.wheels);  
console.log(car2.wheels);  
console.log(car1.steeringWheel);  
console.log(car2.steeringWheel);
```

```
4
```

```
4
```

```
1
```

```
1
```

Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
car1.wheels=3;  
console.log(car1.wheels);  
console.log(car2.wheels);  
console.log(Car.prototype.wheels);
```

Objekti deca ne mogu menjati atribute roditelja u lancu

Kreira se novi atribut deteta sa istim nazivom

3

4

4

Kako izgleda car3 instanca ?

```
console.log(car1);
```

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  wheels: 3  
}
```

Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
car1.wheels=3;
```

```
Car.prototype.wheels = 5;  
console.log(Car.prototype.wheels);  
console.log(car2.wheels);  
console.log(car1.wheels);
```

Promene u roditelju se automatski reflektuju na decu

Osim za decu u kojoj je promena izvršena eksplicitno

5

5

3

Prototipi primer

```
car1.stereo = 1;  
car2.gps = 1;  
console.log(Car.prototype);  
console.log(car2);  
console.log(car1);
```

Promene u deci se
ne vide u roditelju !

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  wheels: 5  
}
```

```
Object {  
  color: "metallic",  
  gps: 1,  
  steeringWheel: 1,  
  wheels: 5  
}
```

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  stereo: 1,  
  wheels: 3  
}
```

Šema

Car.prototype

wheels = 4 (5)
steering wheel = 1
color = 'metallic'

car1._proto_

wheels = 3
steering wheel = 1
color = 'metallic'
stereo = 1

car1

car2._proto_

wheels = 5
steering wheel = 1
color = 'metallic'
gps = 1

car2

Prototipi kao JS mehanizam nasleđivanja

- ◆ Svaka funkcija se može posmatrati kao klasa koja ima jednu unikatnu roditeljsku instancu (prototype) i decu primerke instanci (objekti)
- ◆ Instance su međusobno povezane i čine lanac (prototype chain)
 - Sekvencijalan lanac od unikatne instance ka deci po redosledu njihovog kreiranja
- ◆ Promene prototip instance (`Car.prototype`) će se automatski reflektovati na decu
 - Dodavanje/uklanjanje atributa i funkcija
 - Promene vrednosti atributa
- ◆ Obrnuto ne važi !
 - Novi atributi/funkcije ostaju u instanci u kojoj su kreirani

Nasleđivanje (ECMAScript v6 2015)

```
class Rectangle extends Shape {  
    constructor(height, width) {  
        super(height, width);  
        this.height = height;  
        this.width = width;  
    }  
    area() {  
        return this.width * this.height;  
    }  
    static countInstances() {  
        ...  
    }  
}  
var r = new Rectangle(10,20);
```

Definicija i
nasleđivanje

Definisanje
metode

Statička
metoda

Nasleđivanje klase se u
pozadini konvertuje u model
nasleđivanja prototipa !

JavaScript programski jezik

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

Funkcionalno programiranje

- ◆ Zasnovano na ideji da se program tretira kao matematička funkcija
- ◆ Sve u kodu je ili funkcija ili izraz
- ◆ Ne postoji stanje (variable, polja, objekti, ...)
- ◆ Većina modernih programskih jezika nije isključivo zasnovano na funkcionalnoj paradigmi
- ◆ Međutim, neke ideje su korisne i već nalaze primenu u JS jeziku:
 - ◆ Funkcije kao objekti prvog reda (*first-class functions*)
 - ◆ Anonimne funkcije (*Anonymous functions*)
 - ◆ Arrow funkcije (Angular)

Funkcije kao objekti prvog reda

- ◆ Funkcije su JS objekti:
- ◆ Mogu se čuvati u, tj. dodeljivati varijablama
- ◆ Mogu se prosleđivati kao parametri
- ◆ Mogu imati atributе
- ◆ Mogu se definisati bez identifikatora ->

Anonimna funkcija

- ◆ Funkcija koja nema ime i koja se kreira u vreme izvršavanja programa (*runtime*)
- ◆ Mora se definisati pre nego se pozove !
 - Nije hoisted kao standardne JS funkcije

```
//printName("Mlađan");
//ReferenceError: printName is not defined

const printName = function (name){
    console.log(name);
}
printName("Mlađan");
```

"Mlađan"

Arrow funkcija

- ◆ Klasičan (imperativni) stil

```
for (var i=0; i < anArr.length; i++) {  
    newArr[i] = anArr[i]*i;  
}
```

- ◆ Funkcionalni stil:

```
newArr = anArr.map((val, ind) => val*ind);
```

- ◆ Čitav program se može napisati kao lanac funkcija:
`anArr.filter(filterFunc).map(mapFunc).reduce(reduceFunc)`

Linkovi ka primerima i dokumentacijom

JavaScript osnove primer: <https://codepen.io/mljovanovic/pen/BbpjNV>

JavaScript nizovi primer: <https://codepen.io/mljovanovic/pen/XGRroa>

JavaScript objekti i prototipi primer:

<https://codepen.io/mljovanovic/pen/VRWPXR>

JavaScript funkcionalno programiranje primer:

<https://codepen.io/mljovanovic/pen/aMwVjk>

MDN jednakost:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/
Equality comparisons and sameness](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness)

MDN template stringovi:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
Template literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
Template_literals)

MDN arrow funkcije:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
Functions/Arrow functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/
Functions/Arrow_functions)

MDN nasleđivanje:

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/
Inheritance](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/
Inheritance)

Sažetak

- ◆ JavaScript:
- ◆ Princip rada
- ◆ Proceduralni aspekti
 - Operatori
 - Funkcije
 - Varijable
 - Tipovi podataka (templejt stringovi)
 - Nizovi
- ◆ Objektno-orientisani aspekti
 - Metode
 - Funkcije kao objekti
 - Prototipi
 - Klase i nasleđivanje
- ◆ Aspekti funkcionalnog programiranja
 - Anonimne funkcije
 - Arrow funkcije
- ◆ Primeri: codepen.io

JavaScript osnove

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

U pripremi prezentacije korišćene su ilustracije i primeri sa
<https://developer.mozilla.org>