

JS

Pregled

- 1 Uvod
- 2 Node.js
- 3 Deklaracija varijabli
- 4 Tipovi podataka
- 5 Operatori
- 7 Objekti
- 8 Funkcije
- 9 Nizovi
- 10 Klase
- 11 Izuzeci
- 12 Asihrono programiranje

1. Uvod

- Programski jezik
 - Nije strogo tipiziran
 - Interpretiran
 - Podržan je od strane svih browser-a
 - Funkcionalan i imperativan
 - Even-driven
 - Objektno-orijentisan
 - Klase su dodate u kasnijim verzijama
 - Sa HTML-om i CSS-om čini osnovu www tehnologije
 - Omogućava da web stranice budu zapravo interaktivne
 - Inicijalno nastao radi dinamičnosti stranica na klijentskoj strani

1. Uvod

- U posljednoj deceniji doživjela veliku popularnost
 - Trenutno je na 7. mjestu po TIOBE indeksu
 - Odmah iza jezika kao što su Java, C, C++ itd
 - Pojavilo se mnoštvo frejmworka i biblioteka za izradu klijentskih aplikacija
 - Nalazi daleko širu primjenu od one za koju je inicijalno kreiran
 - Može se upotrebljavati za server-side programiranje, razvoj mobilnih aplikacija
 - Putem okruženja kao što su NodeJS, ReactNative itd
 - Sintaksa se stalno unapređuje
 - Standardi ES5, ES6

1. Uvod

- Omogućava manipulaciju sadržajem HTML stranica

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>What Can JavaScript Do?</h2>
```

```
<p id="demo">JavaScript can change HTML content.</p>
```

```
<button type="button" onclick='document.getElementById("demo").innerHTML =  
"Hello JavaScript!">Click Me!</button>
```

```
</body>
```

```
</html>
```

2. Node.js

- Node.js
 - Do pojave ovog alata JS je korišćen isključivo na strani klijenta
 - 2009. godine se pojavila prva verzija
 - Omogućava da se JS koristi i za programiranje izvan *browser-a*
 - programski jezik opšte namjene
 - Najveću primjenu je našao u server-side programiranju
 - Zasniva se na V8 JS engine
 - Chrome
 - Glavni principi
 - Asinhronost
 - Single-thread
 - Brzina

2. Node.js

- Instalacija
 - Preuzeti odgovarajuću verziju programa sa sajta <https://nodejs.org/en/>
 - CLI okruženje
 - U komandnom prozoru kucati
 - node -v
 - Provjera koja je verzija instalirana
 - pokrenuti prvi program
 - node hello.js

```
hello.js:  
console.log('Hello world');
```

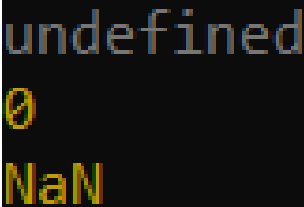
3. Deklaracija varijabli

- Ključne riječi `var`, `let` i `const`
 - `const` znači da varijabla neće mijenjati vrijednost
 - `var x, y, z;` ili `let x, y, z;`
 - Nema tipova
 - Nije strogo tipiziran jezik
 - Dodjelom se često odredi koji tip je u pitanju
 - `a = 5; b = 6; c = a + b;`
 - Dominiraju numerički i tip teksta (`string`)

3. Deklaracija varijabli

- Prilikom deklaracije može se izostaviti vrijednost
 - U tom slučaju vrijednost varijable je `undefined`

```
var a;  
var b = 0;  
var c = a + b;  
console.log(a);  
console.log(b);  
console.log(c);
```



```
undefined  
0  
NaN
```

3. Deklaracija varijabli

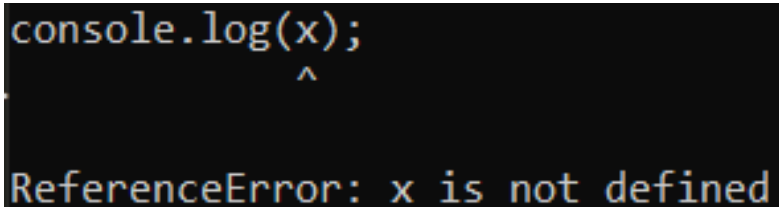
- Razlike u pogledu opsega važenja
 - `var` i `let`
 - Kada deklarirate varijablu sa `var` ona postoji i van bloka u kojem je deklarirana, dok to nije slučaj sa `let`

```
if (true) {  
    var x = 10;  
}  
console.log(x);
```



10

```
if (true) {  
    let x = 10;  
}  
console.log(x);
```



```
console.log(x);  
           ^  
ReferenceError: x is not defined
```

4. Tipovi podataka

- Primitivni tipovi
 - Number
 - Boolean
 - String
 - Tekst se može zadati i sa jednostrukim i dvostrukim znacima navoda
 - Specijalni simboli i Unicode
 - `\r`, `\n`, `\t`, `'\u00A9'` itd
- Objekti
 - Funkcije
 - Nizovi
 - Regularni izrazi
 - Date
- Specijalni literali
 - `null` i `undefined`

4. Tipovi podataka

- Numerički literali
 - Uvijek se svode na 64-bitne realne brojeve
 - ekvivalentno sa `double` u *Javi*

```
let count = 10;           // integer literal; count is still a double
const blue = 0x0000ff;    // hexadecimal (hex ff = decimal 255)
const umask = 0o0022;     // octal (octal 22 = decimal 18)
const roomTemp = 21.5;    // decimal
const c = 3.0e6;          // exponential ( $3.0 \times 10^6 = 3,000,000$ )
const e = -1.6e-19;       // exponential ( $-1.6 \times 10^{-19} = 0.000000000000000000016$ )
const inf = Infinity;
const ninf = -Infinity;
const nan = NaN;          // "not a number"
```

4. Tipovi podataka

- Stringovi

- Tekst se može zadati i sa jednostrukim i dvostrukim znacima navoda
 - `const dialog = 'Sam looked up, and said "hello, old friend!", as Max walked in.'`;
 - `const imperative = "Don't do that!"`;
- Specijalni simboli i Unicode
 - `\r, \n, \t, '\u00A9'` itd
- Šabloni
 - `let currentTemp = 19.5;`
`const message = `The current temperature is ${currentTemp}\u00b0C``;
 - Backtick
- Moguće je zadati string u više redova
 - *multiline*
 - Na kraju reda treba dodati simbol /
 - `const multiline = "line1\
line2"`;

4. Tipovi podataka

- Literali
 - Specijalni literali
 - `null`
 - Koristi se od strane programera da se označi da je neka niz nedefisana
 - Npr. prazan niz
 - `undefined`
 - Vrijednost je još uvijek nepoznata
 - Varijabla je deklarirana ali joj nije dodijeljena vrijednost
 - Argument prilikom poziva nije definisan
 - Preporučuje se da se ne koristi eksplicitno od strane programera
 - Bolje koristiti `null`

4. Tipovi podataka

- Datumi

- Date

- Predstavlja broj milisekundi od 01.01.1970.

- `var today = new Date();`

- 2020-11-18T23:21:27.194Z

- `Date.now();`

- Dobija se tekući broj proteklih milisekundi

```
var today = new Date();
console.log(today.getDate());
console.log(today.getMonth());
console.log(today.getFullYear());
console.log(today.getHours());
console.log(today.getMinutes());
console.log(today.getSeconds());
//number of milliseconds since January 1, 1970, 00:00:00 UTC
console.log(today.getTime());
console.log(today.getTimezoneOffset()); //-330 Minutes
```

5. Operatori

- Operatori
 - Aritmetički
 - `+`, `-`, `*`, `/`, `%`, `++`, `--`
 - Dodjele
 - `=`, `+=`, `*=` itd
 - Poređenje
 - `==`, `!=`, `>=`, `<=`, `===`
 - Obratiti pažnju na razliku između `==` i `===`
 - Konverzija tipova
 - Logički
 - `&&`, `||`, `!`

6. Kontrolne naredbe

- Kontrolne naredbe
 - Slično kao u programkom jeziku Java
 - if, while, for, case

```
if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

7. Objekti

- Objekti

- Skup parova ključ:vrijednost

- `var car = {type:"Fiat", model:"500", color:"white"};`

- Svojstvima se pristupa koristeći OO notaciju

- *objectName.propertyName*

- *car.model*

- Mogu sadržati i funkcije

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id        : 5566,  
    fullName  : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

7. Objekti

- Objekti

- Članovim objekta može se pristupati i putem operatora []
 - computed member access
 - Npr. `person["firstName"]`
- Dozvoljeno je i ugnježdavanje

```
const sam3 = {  
  name: 'Sam',  
  classification: {  
    kingdom: 'Anamalia',  
    phylum: 'Chordata',  
    class: 'Mamalia',  
    order: 'Carnivoria',  
    family: 'Felidae',  
    subfaimily: 'Felinae',  
    genus: 'Felis',  
    species: 'catus',  
  },  
};
```

8. Funkcije

- Funkcije
 - Glavna organizaciona jedinica programskog koda

```
function name(parameter1, parameter2, parameter3)
{
    code to be executed
}
```

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
```

8. Funkcije

- Funkcije
 - U JS funkcije su zapravo objekti
 - Drugim riječima, može se deklarirati varijabla tipa funkcije
 - Mogu se predavati kao argumenti drugim funkcijama
 - Razlika između poziva i referenciranja
 - `getGreeting()`;
 - Poziv funkcije
 - `getGreeting`;
 - referenca

```
function getGreeting() {  
    return "Hello world!";  
}
```

```
const f = getGreeting;  
f(); // "Hello, World!"
```

8. Funkcije

- Funkcije

- Nema preklapanja funkcija kao u drugim programskim jezicima
 - Preklapanje je koncept u programskom gdje mogu da se definišu dvije funkcije istog imena
 - Razlikuju se na osnovu broja i tipa argumenata
- U JS kada jednom definišete funkciju, npr. `getGreetings`, svaki poziv funkcije pod tim imenom se odnosi na tu jednu funkciju
 - Možete pozvati funkciju sa bilo kojim brojem argumenata
 - Npr. Ako funkcija ima dva argumenta, a prilikom poziva je predat samo jedan, onda će drugi imati vrijednost *undefined*

8. Funkcije

- Funkcije
 - Dozvoljeno je ugnježdavanje
 - Tijelo jedne funkcije definiše su unutar druge funkcije

```
function max3(a, b, c){  
  
    function max2(x, y){  
        if (x > y)  
            return x;  
        else  
            return y;  
    }  
  
    return max2(a, max2(b, c));  
}  
  
console.log(max3(4, 5, 1));
```

8. Funkcije

- Funkcije
 - Argumenti mogu da imaju i svoje podrazumijevane vrijednosti
 - U slučaju da argument nije definisan, onda on uzima svoju podrazumijevani vrijednost navedenu prilikom deklaracije

```
function f(a, b = "default", c = 3) {  
    return a + '-' + b + '-' + c;  
}  
console.log(f(5, 6, 7));  
console.log(f(5, 6));  
console.log(f(5));  
console.log(f());
```

```
5-6-7  
5-6-3  
5-default-3  
undefined-default-3
```

8. Funkcije

- Funkcije
 - Mogu da budu i članovi (property) objekata
 - U tom slučaju se nazivaju metode

```
const o = {  
  name: 'Wallace',  
  bark: function() { return 'Woof!'; },  
}
```

8. Funkcije

- `this` pokazivač
 - Sličan pojmu `this` pokazivača iz drugih jezika kao što je Java
 - Pokazivač na objekat koji je pozvao funkciju

```
const o = {name: 'Wallace',  
           speak() { return 'My name is ' + this.name; }  
};  
console.log(o.speak());
```

```
My name is Wallace
```

8. Funkcije

- `this` pokazivač
 - U nekim slučajevima moguće je da se izgubi informacija o tome gdje `this` pokazuje
 - Npr. Sačuva se pokazivač na funkciju u neku varijablu

```
const o = {name: 'Wallace',  
           speak() { return 'My name is ' + this.name; }  
};  
const speak = o.speak;  
console.log(speak());
```

```
My name is undefined
```

8. Funkcije

- `this` pokazivač
 - U nekim situacijam zna da se izgubi informacija o tome gdje `this` pokazuje
 - Problem je naročito izražen kod ugnježđenih funkcija

```
const o = {
  name: 'Julie',
  greetBackwards: function() {
    function getReverseName() {
      let nameBackwards = '';
      for(let i=this.name.length-1; i>=0; i--) {
        nameBackwards += this.name[i];
      }
      return nameBackwards;
    }
    return getReverseName() + " si eman ym ,olleH";
  },
};
console.log(o.greetBackwards());
```

8. Funkcije

- Neimenovane funkcije

```
const f = function() {  
  // ...  
};
```

- Dosta se koriste kao argumenti drugih funkcija
 - Moguće je zadati kompletnu definiciju funkcije kao argument
- Članice objekata
- *Callback* kod asinhronog programiranja
- Funkcija može da bude i rezultat druge funkcije

```
const f = function() {  
  return "message in a bottle";  
}
```

```
console(f());
```



8. Funkcije

```
function max(a, b){  
    if (a >= b) return true;  
    else return false;  
}
```

```
function min(a, b){  
    if (a >= b) return false;  
    else return true;  
}
```

```
function sort(arr, comparator){  
    for(let i = 0; i < arr.length; i++){  
        for(let j = 0; j < arr.length - i - 1; j++){  
            if (comparator(arr[j], arr[j+1])){  
                let temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
            }  
        }  
    }  
}  
  
let arr = [3, 2, 1, 5, 4];  
sort(arr, max)  
console.log(arr);  
sort(arr, min)  
console.log(arr);
```



8. Funkcije

```
function sort(arr, comparator){
  for(let i = 0; i < arr.length; i++){
    for(let j = 0; j < arr.length - i - 1; j++){
      if (comparator(arr[j], arr[j+1])){
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

let arr = [3, 2, 1, 5, 4];
sort(arr, function(a, b){
  if (a >= b) return true;
  else return false;
});

console.log(arr);
```

8. Funkcije

- Arrow notation
 - Definisane u standardu ES6
 - Uvijek su anonimne
 - Suštinski je samo sintaksna olakšica
 - Uz jedan bitan izuzetak
 - `this` pokazivač kod ugnježđenih funkcija
 - Pojednostavljuje sintaksu na tri načina
 - Može se izostaviti *function*
 - Ako funkcija ima jedan argument mogu se izostaviti zagrade
 - Ako tijelo funkcije ima samo jednu naredbu mogu se izostaviti vitičaste zagrade

8. Funkcije

- Arrow notation

```
const f1 = function() { return "hello!"; }  
// OR  
const f1 = () => "hello!";
```

```
const f2 = function(name) { return 'Hello, ' + name + '!'; }  
// OR  
const f2 = name => 'Hello, ' + name + '!';
```

```
const f3 = function(a, b) { return a + b; }  
// OR  
const f3 = (a,b) => a + b;
```

```
function sort(arr, comparator){
  for(let i = 0; i < arr.length; i++){
    for(let j = 0; j < arr.length - i - 1; j++){
      if (comparator(arr[j], arr[j+1])){
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

let arr = [3, 2, 1, 5, 4];
sort(arr, (a, b) => a >= b);
console.log(arr);
```



Outline

8. Funkcije

8. Funkcije

- Arrow notation
 - Bitna razlika je `this` pokazivač
 - Automatski povezan

```
const o = {  
  name: 'Julie',  
  greetBackwards: function() {  
    const getReverseName = () => {  
      let nameBackwards = '';  
      for(let i=this.name.length-1; i>=0; i--) {  
        nameBackwards += this.name[i];  
      }  
      return nameBackwards;  
    }  
    return getReverseName() + " si eman ym ,olleH";  
  },  
};  
console.log(o.greetBackwards());
```

eiluJ si eman ym ,olleH

9. Nizovi

- Nizovi
 - `var array_name = [item1, item2, ...];`
 - Elementima se slično pristupa kao i u drugim jezicima
 - Preko indeksa koji počinje od 0
 - Heterogeni
 - Mogu sadržati više elemenata različitog tipa
 - Slično kao u Javi imaju properti `length`
 - Ako se postavi element koji je izvan granica ne dolazi do greške
 - Niz se proširuje
 - Elementi između dobijaju vrijednost `undefined`

9. Nizovi

```
// array literals
const arr1 = [1, 2, 3]; // array of numbers
const arr2 = ["one", 2, "three"]; // nonhomogeneous array
const arr3 = [[1, 2, 3], ["one", 2, "three"]];
const arr4 = [ { name: "Fred", type: "object", luckyNumbers = [5, 7, 13] },
               [
                 { name: "Susan", type: "object" },
                 { name: "Anthony", type: "object" },
               ],
               1,
               function() { return "arrays can contain functions too"; },
               "three",
             ];
// accessing elements
arr1[0]; // 1
arr1[2]; // 3
arr3[1]; // ["one", 2, "three"]
arr4[1][0]; // { name: "Susan", type: "object" }
// array length
arr1.length; // 3
arr4.length; // 5
arr4[1].length; // 2
// increasing array size
arr1[4] = 5;
arr1; // [1, 2, 3, undefined, 5]
arr1.length; // 5
```

7. Nizovi

- Brojne ugrađene funkcije za manipulaciju
 - Dodavanje i Brisanje
 - `pop`
 - Uklanja element sa kraja
 - `push`
 - Dodaje element na kraj
 - `shift`
 - Briše element sa početka
 - `unshift`
 - Dodaje element na početak
 - `splice`
 - Promjenjiv broj argumenata
 - Prvi označava početak izmjene
 - Drugi broj elementa koje treba brisati
 - Ostali su elementi koji se umeću

7. Nizovi

- Brojne ugrađene funkcije za manipulaciju
 - Spajanje nizova
 - `concat`
 - Dodaje više elemenata na kraj niza i vraća kopiju
 - Niz se ne mijenja
 - Podniz
 - `slice`
 - Vraća sračunati niz i ne mijenja niz nad kojim se primjenjuje ova funkcija
 - Ima dva argumenta
 - Prvi je indeks početka podniza, a drugi indeks kraja podniza (isključno)
 - Moguće je zadavati i negativne brojeve i tu slučaju računanje ide od kraja

7. Nizovi

- Brojne ugrađene funkcije za manipulaciju
 - Sortiranje i okretanje
 - `sort` i `reverse`
 - Za nizove obojekata moguće je definisati uslov poređenja
 - Traženje
 - `indexOf` i `lastIndexOf`
 - Locira prvo i posljednje pojavljivanje elementa
 - `find` i `findIndex`
 - Vraća element ili indeks elementa
 - `some`
 - Da li postoji element koji zadovoljava uslov
 - Filteriranje i mapiranje
 - `map` i `filter`



9. Nizovi

```
const arr = ["b", "c", "d"];
arr.push("e");      // returns 4; arr is now ["b", "c", "d", "e"]
arr.pop();          // returns "e"; arr is now ["b", "c", "d"]
arr.unshift("a");   // returns 4; arr is now ["a", "b", "c", "d"]
arr.shift();        // returns "a"; arr is now ["b", "c", "d"]
```

```
const arr = [1, 5, 7];
arr.splice(1, 0, 2, 3, 4); // returns []; arr is now [1, 2, 3, 4, 5, 7]
arr.splice(5, 0, 6); // returns []; arr is now [1, 2, 3, 4, 5, 6, 7]
arr.splice(1, 2);      // returns [2, 3]; arr is now [1, 4, 5, 6, 7]
arr.splice(2, 1, 'a', 'b'); // returns [5]; arr is now [1, 4, 'a', 'b', 6, 7]
```

```
const arr = [1, 2, 3];
arr.concat(4, 5, 6); // returns [1, 2, 3, 4, 5, 6]; arr unmodified
arr.concat([4, 5, 6]); // returns [1, 2, 3, 4, 5, 6]; arr unmodified
```

```
const arr = [1, 2, 3, 4, 5];
arr.slice(3);      // returns [4, 5]; arr unmodified
arr.slice(2, 4);   // returns [3, 4]; arr unmodified
arr.slice(-2);     // returns [4, 5]; arr unmodified
arr.slice(1, -2);  // returns [2, 3]; arr unmodified
arr.slice(-2, -1); // returns [4]; arr unmodified
```

```
const arr = [1, 2, 3, 4, 5];
arr.reverse(); // arr is now [5, 4, 3, 2, 1]
const arr = [5, 3, 2, 4, 1];
arr.sort(); // arr is now [1, 2, 3, 4, 5]
```



9. Nizovi

```
const arr = [{ name: "Suzanne" }, { name: "Jim" },  
{ name: "Trevor" }, { name: "Amanda" }];  
  
arr.sort(); // arr unchanged  
arr.sort((a, b) => a.name > b.name); // arr sorted alphabetically  
// by name property  
  
const o = { name: "Jerry" };  
const arr = [1, 5, "a", o, true, 5, [1, 2], "9"];  
  
arr.indexOf("a", 5); // returns -1  
arr.indexOf(5);      // returns 1  
arr.lastIndexOf(5); // returns 5  
arr.indexOf(5, 5);   // returns 5  
arr.lastIndexOf(5, 4); // returns 1  
  
const arr = [{ id: 5, name: "Judith" }, { id: 7, name: "Francis" }];  
arr.findIndex(o => o.id === 5); // returns 0  
arr.findIndex(o => o.name === "Francis"); // returns 1  
arr.findIndex(o => o === 3); // returns -1  
  
const arr = [{ id: 5, name: "Judith" }, { id: 7, name: "Francis" }];  
arr.find(o => o.id === 5); // returns object { id: 5, name: "Judith" }  
arr.find(o => o.id === 2); // returns null  
  
const arr = [5, 7, 12, 15, 17];  
arr.some(x => x%2===0); // true; 12 is even  
arr.some(x => Number.isInteger(Math.sqrt(x))); // false; no squares
```



9. Nizovi

```
const cart = [ { name: "Widget", price: 9.95 }, { name: "Gadget",  
price: 22.95 }];  
  
const names = cart.map(x => x.name); // ["Widget", "Gadget"]  
const prices = cart.map(x => x.price); // [9.95, 22.95]  
const discountPrices = prices.map(x => x*0.8); // [7.96, 18.36]  
const lcNames = names.map(String.toLowerCase); // ["widget", "gadget"]  
  
// create a deck of playing cards  
const cards = [];  
for(let suit of ['H', 'C', 'D', 'S']) // hearts, clubs, diamonds,  
spades  
for(let value=1; value<=13; value++)  
cards.push({ suit, value });  
// get all cards with value 2:  
cards.filter(c => c.value === 2); // [  
// { suit: 'H', value: 2 },  
// { suit: 'C', value: 2 },  
// { suit: 'D', value: 2 },  
// { suit: 'S', value: 2 }  
// ]  
// (for the following, we will just list length, for compactness)  
// get all diamonds:  
cards.filter(c => c.suit === 'D'); // length: 13  
// get all face cards  
cards.filter(c => c.value > 10); // length: 12  
// get all face cards that are hearts  
cards.filter(c => c.value > 10 && c.suit === 'H'); // length: 3
```

10. Klase

- Klase
 - Noviji standard (ECMAScript2015)
 - Uvodi se principi OO
 - Više paradigmi programiranja
 - Funkcionalno, OO proramiranje itd

```
class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  present() {  
    return "I have a " + this.carname;  
  }  
}
```

```
mycar = new Car("Ford");  
console.log(mycar.present());
```

I have a Ford

10. Klase

- Klase
 - Propertiji putem get i set metoda

```
class Car {  
    constructor(make, model) {  
        this.make = make;  
        this.model = model;  
        this._userGears = ['P', 'N', 'R', 'D'];  
        this._userGear = this._userGears[0];  
    }  
    get userGear() { return this._userGear; }  
    set userGear(value) {  
        if(this._userGears.indexOf(value) < 0)  
            return;  
        this._userGear = value;  
    }  
    shift(gear) { this.userGear = gear; }  
}
```

10. Klase

- Podržano je i nasljeđivanje

```
class Vehicle {  
    constructor() {  
        this.passengers = [];  
        console.log("Vehicle created");  
    }  
    addPassenger(p) {  
        this.passengers.push(p);  
    }  
}  
  
class Car extends Vehicle {  
    constructor() {  
        super();  
        console.log("Car created");  
    }  
    deployAirbags() {  
        console.log("BWOOSH!");  
    }  
}
```

11. Izuzeci

- Obrada izuzetaka je slična kao u drugim programskim jezicima
 - `try` i `catch` programske konstrukcije
 - U `try` dio se stavalj kritičan kod koji može dovesti do greške
 - Tzv. izuzetak
 - `Exception`
 - U slučaju da dođe do greške izvršavanje funkcije se nastavlja u `catch` dijelu
 - Ima jedan argumet koji opisuje grešku koja se dogodila
 - Može se izazvati izuzetak putem komadne `throw`
 - Objekat klase `Error`



11. Izuzeci

```
function sort(arr, comparator){
  try
  {
    for(let i = 0; i < arr.length; i++){
      for(let j = 0; j < arr.length - i - 1; j++){
        if (comparator(arr[j], arr[j+1])){
          let temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }
  }
  catch(err){
    console.log(err);
  }
}

let arr = [3, 2, 1, 5, 4];
sort(arr); //the second argument is missing
console.log(arr);
```



11. Izuzeci

```
function sort(arr, comparator){
  if (!comparator)
    throw new Error("Comparator function is not defined");

  try
  {
    for(let i = 0; i < arr.length; i++){
      for(let j = 0; j < arr.length - i - 1; j++){
        if (comparator(arr[j], arr[j+1])){
          let temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }
  }
  catch(err){
    console.log(err);
  }
}

let arr = [3, 2, 1, 5, 4];
sort(arr); //the second argument is missing
console.log(arr);
```

12. Asinhrono programiranje

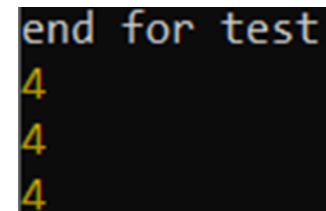
- Nema blokiranja izvršavanja dok traju „dugačke“ operacije
 - Npr. učitavanje podataka iz fajla ili sa udaljenog servera
 - Umjesto čekanja tekuća funkcija nastavlja sa radom
 - Definiše se funkcija koja će biti pozvana kada se okonča izvršenje operacije
 - tzv. *Callback*
 - Analogija sa restoranom
 - Kada je mušterija u restoranu u kojem je gužva ne čeka u redu već ostavi broj telefona da ga pozovu kada se oslobodi sto

```
fs.readFile("./test.txt", {encoding : "utf-8"}, function(err, data){  
    console.log('err', err);  
    console.log('result', data);  
});  
console.log("end for readFile");
```

12. Asinhrono programiranje

- Jedan *thread* izvršavanja
 - Sve funkcije čekaju u redu (*queued*)
 - Nema prekidanja

```
function test() {  
    for(var i = 1; i < 4; i++) {  
        setTimeout(function() { console.log(i); }, 0);  
    }  
    console.log('end for test');  
}
```



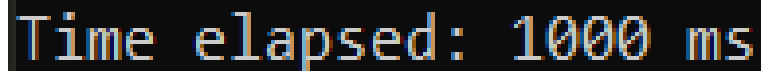
```
end for test  
4  
4  
4
```

12. Asinhrono programiranje

- *Single-threaded*
 - U prethodnom primjeru dobija se „neočekivan“ rezultat
 - Ako se napravila analogija sa nekim drugim programskim jezicima
 - 1, 2, 3 je očekivani rezultat
 - Put `var` deklaracije varijable `i` ima opseg `i` van petlje
 - *Callback*, tj. obrada događaja može da se startuje tek kad tekuća funkcija okonča izvršavanje
 - *Thread* se oslobađa i tek onda se pristupa obradi *callback*-ova
 - Varijabla `i` je i dalje pristupna
 - Ima vrijednost koju je imala na kraju izvršavanja petlje

12. Asinhrono programiranje

```
function test_loop_blocking() {  
    var start = new Date;  
  
    setTimeout(function() {  
        var end = new Date;  
        console.log('Time elapsed:', end - start, 'ms');  
    }, 500);  
  
    while (new Date - start < 1000) {}  
}
```

A terminal window with a black background and white text displaying the output of the code: "Time elapsed: 1000 ms".

Time elapsed: 1000 ms

12. Asinhrono programiranje

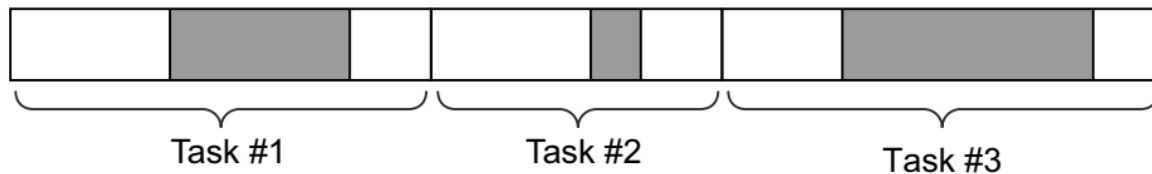
- Koncepti kojima se realizuje su *Callback* i *Promise*
 - *Callback*
 - Zapravo funkcija koja se izvršava kada se steknu uslovi za dalje izvršavanje
 - U primjeru sa prethodnog slajda funkcije *f* je callback
 - Callback hell

```
fs.readFile('a.txt', function(err, dataA) {  
  if(err) console.error(err);  
  fs.readFile('b.txt', function(err, dataB) {  
    if(err) console.error(err);  
    fs.readFile('c.txt', function(err, dataC) {  
      if(err) console.error(err);  
      setTimeout(function() {  
        fs.writeFile('d.txt', dataA+dataB+dataC, function(err) {  
          if(err) console.error(err);  
        });  
      }, 60*1000);  
    });  
  });  
});
```

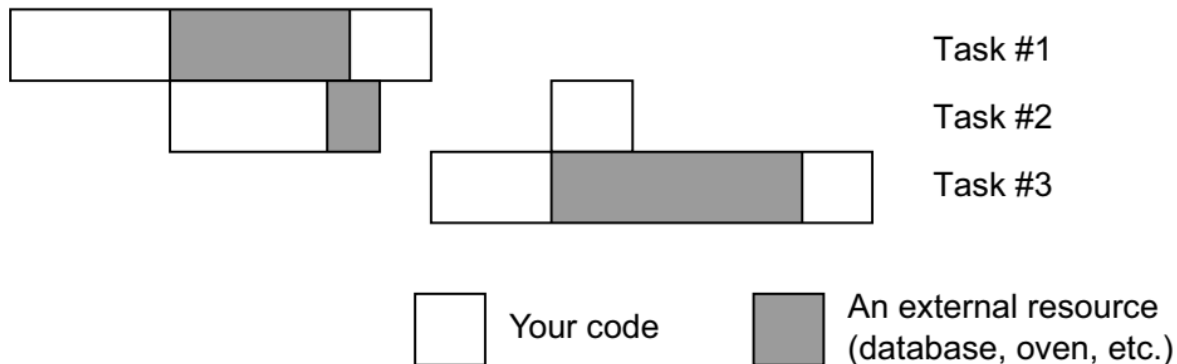
12. Asinhrono programiranje

- Asinhronost poboljšava performanse

A synchronous world



An asynchronous world



12. Asinhrono programiranje

- *Promise*
 - Koncept koji je uveden kako bi se prevazišli problemi koje uzrokuju *callback*
 - U krajnjem to treba da rezultira jasnijim kodom koji je jednostavniji za održavanje
 - *Callback* se i dalje koriste
 - Ali se uvodi red i metodičnost u pozive
 - Ideja je jednostavna
 - Kada se pozove asinhrona funkcija ona treba da vrati objekat tipa *Promise*
 - *fulfilled*
 - *rejected*

6. Asinhrono programiranje

- *Promise*
 - Kreiranje je zapravo instanciranje objekta sa funkcijom koja ima 2 parametra
 - resolve
 - reject

```
function countdown(seconds) {  
  return new Promise((resolve, reject) => {  
    if (seconds < 0) {  
      reject('Invalid number');  
    }  
    setTimeout(() => {  
      resolve('Success!');  
    }, seconds * 1000);  
  }  
);  
}
```

12. Asinhrono programiranje

- *Promise*
 - Kjučna riječ *then*
 - Definiše akciju koja se sprovodi kada se razriješi ili odbije operacija
 - Prvi *callback* ima jedan argument jer prilikom poziva resolve se predaje neka vrijednost

```
countdown(5).then(  
  (data) => {  
    console.log(data);  
  },  
  (err) => {  
    console.log('Error!', err)  
  }  
);
```

6. Asinhrono programiranje

- *Promise*
 - Kod se može dodatno pojednostaviti uvođenjem *catch* handlera

```
countdown(2).then(  
  (data) => {  
    console.log(data);  
  })  
  .catch((err) => {  
    console.log('Error!', err)  
  });
```

12. Asinhrono programiranje

- *Promise*

- Ulančavanje može da bude korisno kad imamo više vezanih asinhronih akcija
 - Greška se obrađuje na jednom mjestu

```
function launch() {
  return new Promise(function(resolve, reject) {
    console.log("Lift off!");
    setTimeout(function() {
      resolve("In orbit!");
    }, 2*1000);    // a very fast rocket indeed
  });
}

c.go()
  .then(launch)
  .then(function(msg) {
    console.log(msg);
  })
  .catch(function(err) {
    console.error("Houston, we have a problem....");
  })
```

12. Asinhrono programiranje

- *async/await*
 - Uvode se nove komande kako bi se dodatno pojednostavilo pisanje koda
 - Kod postaje čitljiviji
 - Sekvencijalan
 - Izbjegava se eksplicitno pisanje *Promise*-a i njegovo razrešavanje

12. Asinhrono programiranje

- *async/await*
 - *async*
 - Implicitno se postavlja da funkcija vraća *Promise*
 - tj. da je funkcija asinhrona

```
function f() {  
  return Promise.resolve(1);  
}
```

```
f().then((data) => console.log(data));
```

```
async function f() {  
  return 1;  
}
```

```
f().then((data) => console.log(data));
```

12. Asinhrono programiranje

- *async/await*
 - *await*
 - može se pozvati samo unutar *async* funkcije
 - pauzira izvršavanje funkcije sve dok se ne razriješi *Promise*
 - poslije razrešenja se nastavlja dalje izvršavanje funkcije

```
function resolveAfter2Seconds()
{
    return new Promise(resolve => {
        setTimeout(() => {resolve('resolved');}, 2000);
    });
}

async function asyncCall()
{
    console.log('calling');
    var result = await resolveAfter2Seconds();
    console.log(result); // expected output: 'resolved'
}
```

13. Datumi

- Datumi su „problematičan“ dio JS
 - Nedostatke nadomještaju posebne biblioteke
 - `moment.js`
 - Gregorijanski kalendar je komplikovan
 - Prestupne godine, vremenske zove, ljetnje računanje vremena itd
 - Datum i vrijeme se cuvaju kao broj milisekundi koji je prošao od trenutka koji se naziva *Unix Epoch*
 - *January 1, 1970, 00:00:00 UTC*
 - Date objekat
 - Može se konstruisati na više načina



13. Datumi

```
new Date(2015, 0); // 12:00 A.M., Jan 1, 2015
new Date(2015, 1); // 12:00 A.M., Feb 1, 2015
new Date(2015, 1, 14); // 12:00 A.M., Feb 14, 2015
new Date(2015, 1, 14, 13); // 3:00 P.M., Feb 14, 2015
new Date(2015, 1, 14, 13, 30); // 3:30 P.M., Feb 14, 2015
new Date(2015, 1, 14, 13, 30, 5); // 3:30:05 P.M., Feb 14, 2015
new Date(2015, 1, 14, 13, 30, 5, 500); // 3:30:05.5 P.M., Feb
14, 2015
```

```
// creates dates from Unix Epoch timestamps
new Date(0); // 12:00 A.M., Jan 1, 1970 UTC
new Date(1000); // 12:00:01 A.M., Jan 1, 1970 UTC
new Date(1463443200000); // 5:00 P.M., May 16, 2016 UTC
```

```
// use negative dates to get dates prior to the Unix Epoch
new Date(-365*24*60*60*1000); // 12:00 A.M., Jan 1, 1969 UTC
```

```
// parsing date strings (defaults to local time)
new Date('June 14, 1903'); // 12:00 A.M., Jun 14, 1903 local
time
new Date('June 14, 1903 GMT-0000'); // 12:00 A.M., Jun 14, 1903
UTC
```

13. Datumi

- `moment.js`
 - Biblioteka koja nije sastavni dio JS
 - U širokoj upotrebi u svrhu manipulacije sa datumima
 - Jedna od manjkavosti ugrađenih funkcija za prikaz jeste nekonzistentnost
 - Funkcija `format`
 - Ima podršku za vremenske zone
 - Moguće je prikazivati datume u brojnim formatima
 - Prevazilazi nedostatke ugrađenih funkcija za prikaz



13. Datumi

```
// passing an array to Moment.js uses the same parameters as JavaScript's
Date
// constructor, including zero-based months (0=Jan, 1=Feb, etc.). toDate()
// converts back to a JavaScript Date object.
const d = moment.tz([2016, 3, 27, 9, 19], 'America/Los_Angeles').toDate();

const d = new Date(Date.UTC(1930, 4, 10));
// these show output for someone in Los Angeles
d.toLocaleDateString() // "5/9/1930"
d.toLocaleFormat() // "5/9/1930 4:00:00 PM"

d.toLocaleTimeString() // "4:00:00 PM"
d.toTimeString() // "17:00:00 GMT-0700 (Pacific Daylight Time)"
d.toUTCString() // "Sat, 10 May 1930, 00:00:00 GMT"
moment(d).format("YYYY-MM-DD"); // "1930-05-09"
moment(d).format("YYYY-MM-DD HH:mm"); // "1930-05-09 17:00"
moment(d).format("YYYY-MM-DD HH:mm Z"); // "1930-05-09 17:00 -07:00"
moment(d).format("YYYY-MM-DD HH:mm [UTC]Z"); // "1930-05-09 17:00 UTC-
07:00"
moment(d).format("dddd, MMMM [the] Do, YYYY"); // "Friday, May the 9th,
1930"
moment(d).format("h:mm a"); // "5:00 pm,,
```

13. Datumi

- Operacije
 - Poređenje
 - Putem operatora $< i >$
 - Aritmetika datuma
 - Sabiranje oduzimanje itd
 - Svodi se operacije nad milisekundama
 - Moguće je tražiti datum koji se dobija kada se na tekući datum doda neki vremenski interval
 - Biblioteka `moment.js`



13. Datumi

```
const d1 = new Date(1996, 2, 1);  
const d2 = new Date(2009, 4, 27);  
d1 > d2 // false  
d1 < d2 // true
```

```
const msDiff = d2 - d1; // 417740400000 ms  
const daysDiff = msDiff/1000/60/60/24; // 4834.96 days
```

```
const m = moment(); // now  
m.add(3, 'days'); // m is now three days in the future  
m.subtract(2, 'years'); // m is now two years minus three days in the past
```

```
m = moment(); // reset  
m.startOf('year'); // m is now Jan 1 of this year  
m.endOf('month'); // m is now Jan 31 of this year
```

```
moment().subtract(10, 'seconds').fromNow(); // a few seconds ago  
moment().subtract(44, 'seconds').fromNow(); // a few seconds ago  
moment().subtract(45, 'seconds').fromNow(); // a minute ago  
moment().subtract(5, 'minutes').fromNow(); // 5 minutes ago  
moment().subtract(44, 'minutes').fromNow(); // 44 minutes ago  
moment().subtract(45, 'minutes').fromNow(); // an hour ago  
moment().subtract(5, 'hours').fromNow(); // 4 hours ago  
moment().subtract(21, 'hours').fromNow(); // 21 hours ago  
moment().subtract(22, 'hours').fromNow(); // a day ago  
moment().subtract(344, 'days').fromNow(); // 344 days ago  
moment().subtract(345, 'days').fromNow(); // a year ago
```

13. Regularni izrazi

- Obrasci

- Vrše provjeru da li se zadati string uklapa po obrascu
- Moguće je i vršiti zamjenu podstringa
 - Transformacije
- Predstavljaju se klasom `RegExp`
 - Često su u upotrebi da je i sintaksa jezika prilagođena da je moguće regularni izraz zadati kao literal
 - Koristi se forward slash
 - Deklaracije `const re2 = new RegExp("going");` i `const re1 = /going/;` su ekvivalentne
 - Provjerava da li string sadrži podstring *going*

13. Regularni izrazi

- Metode
 - `match`
 - Vraća sve podstringove koji zadovoljavaju reg. izraz
 - `search`
 - Vraća prvi podniz koji zadovoljava uslov
 - `test`
 - provjerava da li postoji poklapanje i vraća boolean vrijednost
 - Koristi se forward slash
 - `exec`
 - Nalazi tekuće pojavljivanje
 - Kada se pozove prvi put vraća prvo pojavljivanje
 - `replace`
 - Vrší zamjenu

13. Datumi

```
const input = "As I was going to Saint Ives";
const re = /\w{3,}/ig;
// starting with the string (input)
input.match(re); // ["was", "going", "Saint", "Ives"]
input.search(re); // 5 (the first three-letter word starts at
index 5)

// starting with the regex (re)
re.test(input); // true (input contains at least one three-
letter word)
re.exec(input); // ["was"] (first match)
re.exec(input); // ["going"] (exec "remembers" where it is)
re.exec(input); // ["Saint"]
re.exec(input); // ["Ives"]
re.exec(input); // null -- no more matches

// note that any of these methods can be used directly with a
regex literal
input.match(/\w{3,}/ig);
/\w{3,}/ig.test(input);

const input = "As I was going to Saint Ives";
const output = input.replace(/\w{4,}/ig, '****'); // "As I was
****

// to **** ****"
```

13. Regularni izrazi

- Formiranje izraza
 - Alternative (ili) |
 - /ab|cd/ig
 - ^
 - Negacija, tj. sve osim
 - Cifre se predstavljaju sa [0-9]
 - Može i \d
 - Bjeline (\n \t itd) se predstavljaju sa \s
 - Slova se predstavljaju sa [a-zA-Z_]
 - Može i \w

13. Regularni izrazi

- Kardinalitet

- Moguće ga je zadati u $\{ \}$ zagradama
 - $\{ n \}$ – tačno n pojavljivanja
 - Npr. $\wedge d\{5\}$ / *match*-uje brojeve od tačno 5 cifara
 - $\{ n, \}$ – n ili više pojavljivanja
 - $\{ n, m \}$ – između n i m pojavljivanja
- $?$
 - Opciono, tj. 0 ili 1 pojavljivanje
- $*$
 - Nula ili više pojavljivanja
- $+$
 - Barem jedno pojavljivanje



13. Datumi

```
const stuff =  
'hight: 9\n' +  
'medium: 5\n' +  
'low: 2\n';  
const levels = stuff.match(/:\s*[0-9]/g);  
  
const input = "Address: 333 Main St., Anywhere, NY, 55532.  
Phone: 555-555-2525.";   
const match = input.match(/\d{5}.*\/);  
  
const beer99 = "99 bottles of beer on the wall " +  
"take 1 down and pass it around -- " +  
"98 bottles of beer on the wall.";   
  
const match = beer99.match(/[0-9]+/);  
  
const input = "Address: 333 Main St., Anywhere, NY, 55532.  
Phone: 555-555-2525.";   
Const match = input.match(/\d{5}.*\/);  
  
const equation = "(2 + 3.5) * 7";  
const match = equation.match(/\((\d \+ \d\.\d\) \* \d/);
```