Osnove Jezika JavaScript

Mislav Mandarić

Sadržaj

- Uvod
- Programski jezik
  - Karakteristike
  - Sintaksa
  - Ugrađene funkcionalnosti
- Platforme
  - Preglednik
  - Server
  - Samostalne
    - React Native, Cordova, Electron

Uvod

Info

- Jedan od najpopularnijih jezika
- Inicijalno - programski jezik weba
  - Danas - serverske, mobilne i desktop aplikacije
- Razne implementacije
  - V8 (Chrome, Opera, NodeJS, Electron)
  - SpiderMonkey (Firefox)
  - Nitro (Safari)
  - Chakra (Edge)

Povijest

- 1995
  - Objavljena prva verzija JavaScript jezika
- 1997
  - Prva verzija ECMA standarda
- 2009
  - Peta verzija standarda, ES5
- 2015
  - Šesta verzija standarda, ES2015 (ES6)
- 2017
  - Trenutna verzija standarda, ES2017
    - Danas učimo ovo!

Programski jezik

Karakteristike

- “JavaScript [...] is a **high-level**, **dynamic**, **weakly typed**, **prototype-based**, **multi-paradigm**, and **interpreted** programming language.” – Wikipedia
  - High level - visoka razina apstrakcije
  - Dynamic - tipovi podataka se definiraju prilikom izvođenja
  - Weakly typed - tipovi se implicitno konvertiraju prilikom izvođenja
  - Prototype based - ponašanje se može dijeliti prototipiziranjem
  - Multi paradigm - imperativna, funkcijska, event paradigma
  - Interpreted - kod se interpretira, bez kompilacije

Karakteristike

- Jednodretvenost
  - Asinkronost omogućava neblokirajuće pozive
- Okruženje izvršavanja
  - Globalni objekt
  - Dijeljenje funkcionalnosti putem sučelja globalnog objekta
    - Sučelje za pristup ugrađnim objektima jezika
    - Sučelje za pristup dodatnim funkcionalnostima platforme

Demo

Varijable

- Deklaracija
  - `var a = 1; // Old syntax, do not use!`
    - Lokalne ili globalne varijable
    - Doseg je *funkcija* u kojem je varijabla deklarirana ili globalni doseg
    - Uzdizanje deklaracije na vrh dosega
  - `let a = 1;`
    - Isključivo lokalne varijable
    - Doseg je *blok* u kojem je varijabla deklarirana
  - `const a = 1;`
    - Slicnih karakteristika kao let
    - Varijabli se ne može dodijeliti nova vrijednost
    - Ali svojstva joj se mogu mijenjati

Tipovi podataka

- Boolean
- Number
- String
- Symbol
- null
- undefined
- Object
  - Function
  - Date, Error, Array, RegExp, Math, Map, Set, ...

Tipovi podataka

- Literal
  - Boolean `true; false`
  - Number `1; 3.14`
  - String `'a'; "b"; `c``
  - Object `{ a: 1, b: 2 }`
  - Array `[1, 2, 3]`
  - RegExp `/[abc]+/`

`null` i `undefined`

- `undefined`
  - Nedefinirana vrijednost
  - Namjerno ili nenamjerno izostavljena definicija
- `null`
  - Definirana, ali prazna vrijednost
  - Namjerno definirana kao ništavna

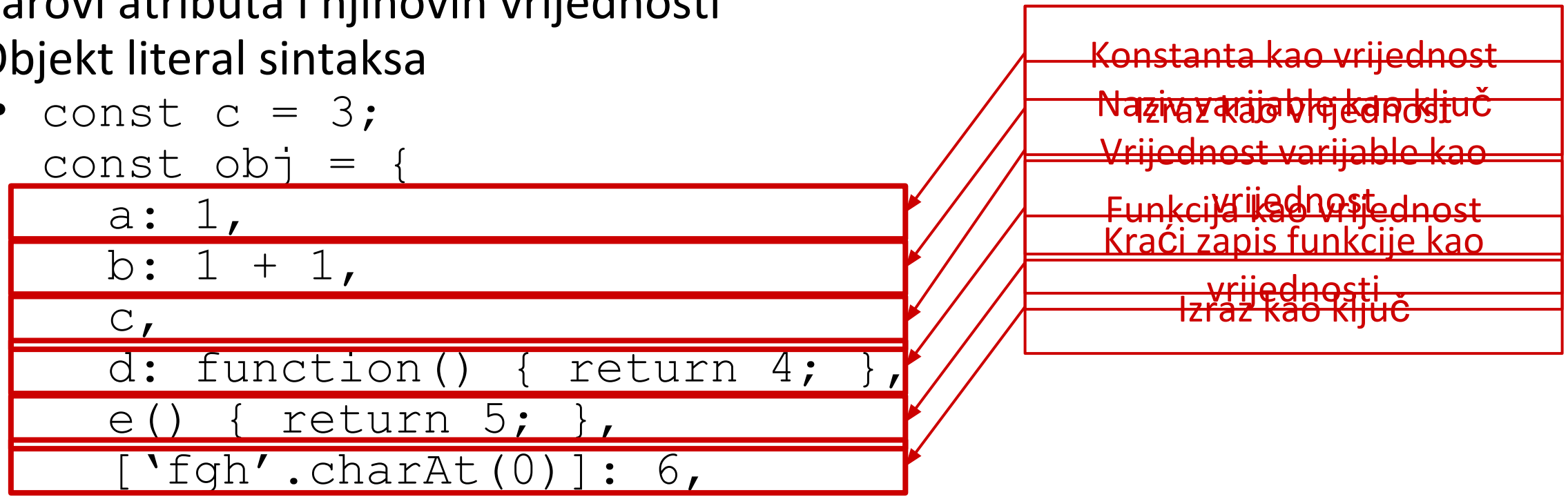
Logički

- `true` ili `false`

Brojevi

- IEEE 754
  - 64-bitni brojevi s dvostrukom preciznošću
- Cijeli
  - Dekadski, oktalni, heksadekadski, binarni
- Decimalni

Stringovi

- Unicode niz znakova
- String literal sintaksa
  - `const a = “Dvostruki navodnici”;`
  - `const b = ‘Jednostruki navodnici’;`
- Template literal sintaksa
  - `const c = \`Backtick navodnici\`;`
  - `const d = \`Formatiranje stringa sa ${a} i ${b}\`;`
  - ```
    const e = `Redak prvi
    Redak drugi
    Redak treci`;
    ```
    ```

Stringovi

- Dohvat duljine stringa
  - `const length = 'abcd'.length;`
- Dohvat N-tog znaka
  - `const c = 'abcd'[2];`
- Iteriranje kroz znakove stringa
  - `for (let char of str) { … }`
- Konkatenacija
  - `const abc = 'a' + 'b' + 'c';`

Objekti

- Parovi atributa i njihovih vrijednosti
- Objekt literal sintaksa
  - ```
    const c = 3;
    const obj = {
        a: 1,
        b: 1 + 1,
        c,
        d: function() { return 4; },
        e() { return 5; },
        [‘fgh’.charAt(0)]: 6,
    };
    ```


    ```

Objekti

- Pristup atributima objekta
  - Sintaksa s točkom
  - Sintaksa s uglatim zagradama
    - ```
      const obj = { a: 1, b: 2 };
      const a = obj.a;
      const b = obj['b'];
      ```
- Brisanje atributa objekta
  - `delete obj.a;`
      ```

Objekti

- Objekti određenog tipa kreiraju se pozivom konstruktora
  - `const date = new Date();`
- Provjera tipa objekta
  - `typeof(date); // “object”`
  - `date instanceof Date; // true`

Polja

- Indeksirani niz elemenata
  - Indeksi startaju s 0
- Objekt tipa Array
  - Atributi su indeksi polja
- Sintaksa za literal polja
  - `const numberArray= [1, 2, 3];`
  - `const stringArray = [‘a’, ‘b’, ‘c’];`
  - `const objectArray = [{a: 1}, {b: 2}];`
  - `const mixedArray = [1, ‘a’, {a: 1}];`

Polja

- Dohvat duljine polja
  - `const length = [‘a’, ’b’, ‘c’].length;`
- Dohvat N-tog elementa
  - `const c = [‘a’, ’b’, ‘c’][2];`
- Iteriranje kroz elemente polja
  - `for (let elem of array) { … }`
- Sortiranje polja
  - `array.sort();`
- Preokretanje redosljeda elemenata polja
  - `array.reverse();`

Polja

- Dodavanje novog elementa u polje
  - `array.push('to_end');`
    `array.unshift('to_start');`
- Uklanjanje elementa iz polja
  - `const from_end = array.pop();`
    `const from_start = array.shift();`

Polja

- Višedimenzionalna polja
  - Polje polja
  - ```
    const multiArray = [
       [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]
    ];
    ```
- Pristup elementima
  - `const six = multiArray[1][2];`
- Iteriranje kroz elemente
  - `for(let inner of multi){ for(let el of inner){…} }`
    ```

JSON

- Tekstualni format za razmjenu podataka
  - Serijalizacija, komunikacija, spremanje
- Mogući tipovi podataka
  - Logička vrijednost - `Boolean`
  - Broj - `Number`
  - Niz znakova - `String`
  - Prazna vrijednost - `null`
  - Neuređena kolekcija ključ/vrijednost - `Object`
  - Uređena lista vrijednosti - `Array`
- Neovisan o jeziku
  - C#, Java, Python, PHP, JavaScript, …

JSON

```
{
  “name”: “Pero”,
  “age”: 25,
  “grade”: 4.5,
  “student”: true,
  “mentor”: null,
  “courses”: [
    {“title”: “JS School”},
    {“title”: “C# School”}
  ]
}
```

JSON

- Konverzije
  - Iz JavaScript vrijednosti u JSON
    - `const jsonString = JSON.stringify(jsObject);`
  - Iz JSON u JavaScript vrijednost
    - `const jsObject = JSON.parse(jsonString);`

Demo

Funkcije

- Blok instrukcija
  - Procedura, subrutina, metoda
- Funkcija se može spremiti u varijablu
  - Parametar funkcije
  - Povratna vrijednost funkcije
  - …
- Različiti oblici definiranja
  - Obične funkcije
  - *Arrow* funkcije

Funkcije - obične

- Izvorno jedini način definiranja funkcije
- Dvije sintakse za definiranje
  - Sintaksa instrukcije
    - ```
      function name(p1, p2) {
        return p1 + p2;
      }
      ```
  - Sintaksa izraza
    - ```
      const name = function (p1, p2) {
        return p1 + p2;
      }
      ```

Funkcije - *arrow*

- Novi način definiranja funkcije
- Samo sintaksa izraza
  - Puna sintaksa
    - ```
      const squared = (a) => {
        return a * a;
      };
      ```
  - Ako je samo jedna instrukcija
    - `const squared = (a) => a * a;`
  - Ako je samo jedan parametar
    - `const squared = a => a * a;`

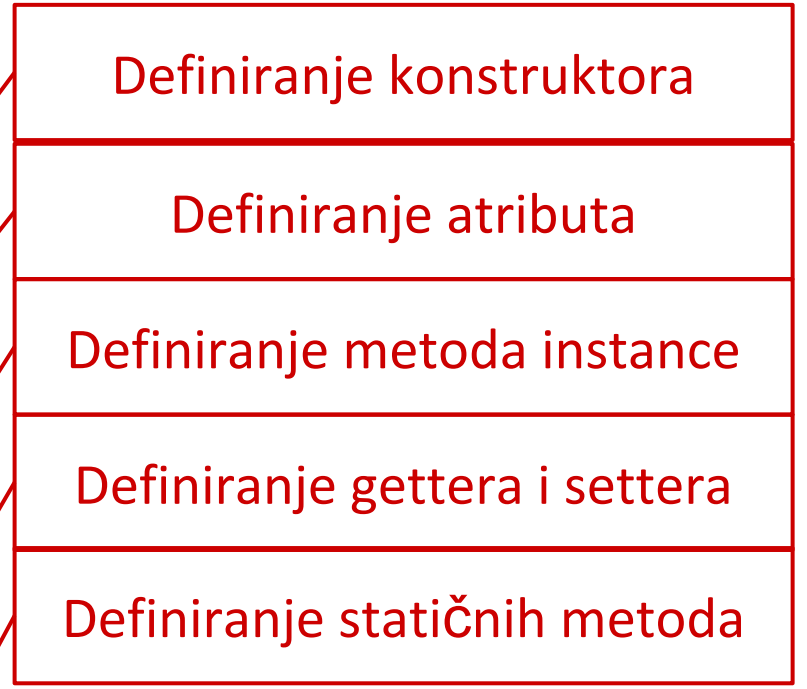

      ```

Funkcije - *arrow*

```
function filter(array, condition) {
  const filteredArray = [];
  for (let i=0; i<array.length; i++) {
    if (condition(array[i])) {
      filteredArray.push(array[i]);
    }
  }
  return filteredArray;
}
```

Funkcije - *arrow*

```
const arr = filter(
   [0, 1, -1, 2, -2],
   function (item) {
     return item > 0;
   }
);
```

Funkcije

- Pozivanje funkcije
  - `const result = name(1, 2);`
    `name('a', 'b');`
- Definiranje pretpostavljenih vrijednosti parametara
  - `function sayMyName(name = 'no one') {`
    `  console.log(`I am ${name}.`);`
    `}`

Demo

Zadatak

- Napišite funkciju koja **vraća listu prijava** čija je ocjena veća od zadanog minimuma.
  - ```
    function getTopApplications (
      allApplications,
      minimum,
      gradingFunction
    ) {
      …
    }
    ```
    ```

Klase

- Definiranje tipa objekta, njegovih atributa i metoda
- Sintaksa za definiranje klase
  - `class Person { … }`
- Instanciranje objekta klase
  - `const p = new Person();`

Klase

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.isAlive = true;
  }
  talk() { console.log('Talking…') }
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  }
  set alive(value) { this.isAlive = value; }
  static getSpecies() { return 'Homo Sapiens'; }
}
```



Klase

- Pozivanje konstruktora
  - `const p = new Person('Pero, 'Peric');`
- Pristup atributima i metodama instance
  - `p.firstName; // 'Pero'`
    `p.talk(); // 'Talking…'`
- Pristup getteru i setteru
  - `p.fullName; // 'Pero Peric'`
  - `p.alive = false;`
- Pristup statičnim metodama
  - `Person.getSpecies(); // 'Homo sapiens'`

Klase

- Nasljeđivanje
  - ```
    class Student extends Person {
      constructor(jmbag, firstName, lastName) {
        super(firstName, lastName); // Required!
        this.jmbag = jmbag;
      }
      talk() {
        super.talk();
        console.log('Still talking…');
      }
    }
    ```
    ```

Demo

Zadatak

- Kreirajte klase `Skill` i `ProgrammingLanguage` koje imaju atribute `name` i `value` i metodu `evaluation`. Metoda `evaluation` za baznu klasu `Skill` treba vratiti `value`, a za nasljeđenu klasu `ProgrammingLanguage` treba vratiti vrijednosti po sljedećim pravilima: 2 puta veće od `value` ako je riječ o JavaScriptu, 1.5 veće ako je riječ o .NET-u, a 1.2 veće ako je riječ o bilo čemu ostalom.
- Klasi koja predstavlja prijavu promjenite izračun ocjene tako da koristi sumu evaluacija svih vještina navedenih u prijavi.

Funkcije - *this*

- Svaka obična funkcija dobije novi *this*
  - ***Arrow* funkcije ne**
    - One koriste *this* iz vanjske funkcije
- Definira se dinamički, prilikom izvršavanja
- Moguće vrijednosti
  - Instanca nad kojom se metoda poziva
  - `undefined`

Funkcije - *this*

```
class Person {
  constructor(name) { this.name = name; }
  talk() {
    console.log(this.name);
  }
}
const pero = new Person('Pero');
pero.talk(); // "Pero"

const talking = pero.talk;
talking(); // Error
```

Funkcije - *this*

```
class Person {
  constructor(name) { this.name = name; }
  talk() {
    delay(function() {
      console.log(this.name);
    }, 1000);
  }
}
const pero = new Person('Pero');
pero.talk(); // Error
```

Funkcije - *this*

```
class Person {
  constructor(name) { this.name = name; }
  talk() {
    delay(() => {
      console.log(this.name);
    }, 1000);
  }
}
const pero = new Person('Pero');
pero.talk(); // "Pero"
```

Funkcije - *closure*

- Ugnježđena funkcija
  - Funkcija definirana unutar druge funkcije
  - Može pristupiti varijablama vanjske funkcije
    - Vanjska funkcija ne može pristupiti varijablama ugnježđene
- *Closure*
  - Kombinacija funkcije i varijabli iz vanjskog dosega

Funkcije - *closure*

```
function getCounter() {
  let value = 0;
  function getValue() { return value; }
  function increment() { value += 1; }
  return { getValue, increment };
}
const counter = getCounter();
counter.increment();
const value = counter.getValue();
```

Demo

Operatori - dodjeljivanje vrijednosti

- Operatori dodjeljivanja vrijednosti
  - `const a = 2;`
  - `const a += 2; const a -= 2;`
  - `const a *= 2; const a /= 2;`
  - `const a %= 2; const a **= 2;`

Operatori - dodjeljivanje vrijednosti

- Operatori destrukturiranja
  - `const [first, second, ...rest] = [1, 2, 3, 4];`
  - `const {x, y, ...rest} = {x: 1, y: 2, z: 3, w: 4};`
  - Destrukturiranje parametara funkcije
    - ```
      function getFullName({first, last}) {
        return `${first} ${last}`;
      };
      ```


      ```

Operatori - aritmetički

- Operatori aritmetike
  - `x + y, x - y`
  - `x * y, x / y`
  - `x % y`
  - `x++, x--`
  - `x ** y`
    - Implicitna konverzija tipova

Operatori - usporedbe

- Operatori usporedbe
  - `x == y, x != y`
  - `x > y, x >= y, x < y, x <= y`
    - Implicitna konverzija tipova
  - `x === y, x !== y`
    - **Bez implicitne konverzije tipova**
    - Različiti tipovi -> različite vrijednosti

Operatori - logički

- Logički operatori
  - `x && y`
  - `x || y`
  - `!x`
    - Implicitna konverzija tipova
    - Skraćena provjera

Operatori - logički

- Implicitna konverzija u logičke izraze
  - Vrijednosti koje se evaluiraju kao neistina - *falsy*
    - `false`
    - `null`
    - `undefined`
    - `0`
    - `NaN`
    - `''`
  - Sve ostale vrijednosti se evaluiraju kao istina - *truthy*
    - `true, {}, [], 1, 3.14, 'string', …, …`

Operatori - logički

- Skraćena provjera logičkih izraza
  - `x && y`
    - Ukoliko je x vrijednost koja se evaluira kao neistina
      - Rezultat je x
    - Ukoliko je x vrijednost koja se evaluira kao istina
      - Rezultat je y
  - `x || y`
    - Ukoliko je x vrijednost koja se evaluira kao istina
      - Rezultat je x
    - Ukoliko je x vrijednost koja se evaluira kao neistina
      - Rezultat je y

Operatori - logički

- `const x = true && true;`
  `const x = false && true;`
  `const x = true && false;`
  `const x = false && false;`
  `const x = null && 'a';`
  `const x = {} && 'a';`
  `const x = '' && 'a';`
  `const x = 'b' && 'a';`
  `const x = 0 && 'a';`
  `const x = 1 && 'a';`

- `const x = true || true;`
  `const x = false || true;`
  `const x = true || false;`
  `const x = false || false;`
  `const x = null || 'a';`
  `const x = {} || 'a';`
  `const x = '' || 'a';`
  `const x = 'b' || 'a';`
  `const x = 0 || 'a';`
  `const x = 1 || 'a';`

Operatori - ostali

- Ternarni operator
  - `condition ? positive : negative;`
- Operatori provjere tipova
  - `typeof obj;`
  - `obj instanceof ClassName;`
- Operator proširivanja
  - ```
    const first = [2, 3];
    const last = [1, ...first, 4];
    ```
  - ```
    const first = {y: 2, z: 3};
    const last = {x: 1, ...first, w: 4};
    ```

Demo

Kontrola toka - grananja

- Uvjetno izvršavanje
  - ```
    if (condition1) {
      …
    } else if (condition2) {
      …
    } else {
      …
    }
    ```


    ```

Kontrola toka - grananja

- Grananje
  - ```
    switch (condition) {
       case label1:
          …
          break;
       …
       default:
          …
          break;
    }
    ```


    ```

Kontrola toka - petlje

- Standardne petlje
  - `for (let i = 0; i < x; i++) {`
    `…`
    `}`
  - `while (condition) {`
    `…`
    `}`
  - `do {`
    `…`
    `} while (condition);`

Kontrola toka - petlje

- Iteracijske petlje
  - `for (let property in object) {`
    `…`
    `}`
    - Iteriranje kroz sve atribute bilo kojeg objekta
  - `for (let item of iterable) {`
    `…`
    `}`
    - Iteriranje kroz sve elemente posebnih iteracijskih objekata
      - String, Array, Map, Set

Kontrola toka - petlje

```
const x = [‘a’, ‘b’, ‘c’];
for (let prop in x) {
  console.log(prop); // 0, 1, 2
}
for (let item of x) {
  console.log(item); // “a”, “b”, “c”
}
```

Kontrola toka - iznimke

- Bacanje iznimke
  - `throw new Error('Message');`
- Hvatanje iznimke
  -
```
try {
  …
} catch (e) {
  …
} finally {
  …
}
```

Demo

Kontrola toka - asinkronost

- Sinkronost
  - Izvršavanje instrukcija redosljedom nailaska
  - Glavni program kontrolira tok izvršavanja
- Asinkronost
  - Mogućnost izvršavanja instrukcija reagiranjem na događaje
  - Vanjski događaji kontroliraju tok izvršavanja
  - Asinkronost uzrokuju funkcije platforme
    - Dohvat
    - Spremanje
    - …

Kontrola toka - asinkronost

- Učenje primjerom :)
  - Dohvat informacije je li neka stranica trenutno dostupna
  - https://downforeveryoneorjustme.com/
    - Dohvati stranicu
    - Vrati podatke
    - Nakon što se podaci vrate, provjeri je li status kod dobar

Kontrola toka - asinkronost

```
function isUp(url) {
  const page = fetch(url);
  const ok =
    page.status >= 200 &&
    page.status < 400;
  return ok;
}
const up = isUp('https://jsschool.axilis.com');
console.log(up); // true
```

Kontrola toka - asinkronost

- Sinkrono izvršavanje
  - Dok se stranica dohvaća, ništa drugo se ne izvršava
    - Blokirajući poziv
  - Nakon što se stranica dohvati, provjeri se status kod

- ```
  function isUp(url) {
    const page = fetch(url);
    const ok =
      page.status >= 200 &&
      page.status < 400;
    return ok;
  }
  const up = isUp(
    'https://jsschool.axilis.com'
  );
  console.log(up);
  ```
  ```

Kontrola toka - asinkronost

```
function isUp(url) {
  const page = fetch(url);
  const ok =
    page.status >= 200 &&
    page.status < 400;
  return ok;
}
const up = isUp('https://jsschool.axilis.com');
console.log(up); // true
```

Kontrola toka - asinkronost

```
function isUp(url) {
  fetch(url, page => {
    const ok =
      page.status >= 200 &&
      page.status < 400;
    return ok; // ????
  });
}
const up = isUp('https://jsschool.axilis.com');
console.log(up); // undefined
```

Kontrola toka - asinkronost

```
function isUp(url) {
  let ok;
  fetch(url, page => {
    ok =
      page.status >= 200 &&
      page.status < 400;
  });
  return ok; // ????
}
const up = isUp('https://jsschool.axilis.com');
console.log(up); // undefined
```

Kontrola toka - asinkronost

```
function isUp(url, callback) {
  fetch(url, page => {
    const ok =
      page.status >= 200 &&
      page.status < 400;
    callback(ok);
  });
}
isUp('https://jsschool.axilis.com',
  up => console.log(up) // true
);
```

Kontrola toka - asinkronost

- Asinkrono izvršavanje
  - Potrebno je prilagoditi potpis svih metoda da primaju *callback*
  - Otežano korištenje primitivnih instrukcija za kontrolu toka
    - Obrada iznimki
  - Ugnježđivanje višestrukih callbackova čini kod teško čitljivim

```
function isUp(url, callback) {
  fetch(url, page => {
    const ok =
      page.status >= 200 &&
      page.status < 400;
    callback(ok);
  });
}
isUp(
  'https://jsschool.axilis.com',
  up => console.log(up) // true
);
```

Kontrola toka - asinkronost

```
function isUp(url, callback) {
  fetch(url, page => {
    const ok =
      page.status >= 200 &&
      page.status < 400;
    callback(ok);
  });
}
isUp('https://jsschool.axilis.com',
  up => console.log(up) // true
);
```

Kontrola toka - asinkronost

```
function isUp(url) {
  const promise1 = fetch(url);
  const promise2 = promise1.then(page => {
    const ok =
      page.status >= 200 &&
      page.status < 400;
    return ok;
  });
  return promise2;
}
const promise3 = isUp('https://jsschool.axilis.com');
promise3.then(up => console.log(up)); // true
```

Kontrola toka - asinkronost

```
function isUp(url) {
  return fetch(url)
    .then(page => {
      const ok =
        page.status >= 200 &&
        page.status < 400;
      return ok;
    });
}
isUp('https://jsschool.axilis.com');
  .then(up => console.log(up)); // true
```

Kontrola toka - asinkronost

- Asinkrono izvršavanje
  - Potrebno je prolagoditi samo povratne vrijednosti metoda da vraćaju *Promise*
  - Posebna sintaksa za obradu iznimki
  - Ulančavanje metoda koje vraćaju *Promise*

```
function isUp(url) {
  return fetch(url)
    .then(page => {
      const ok =
        page.status >= 200 &&
        page.status < 400;
      return ok;
    });
}
isUp(
  'https://jsschool.axilis.com'
)
  .then(up =>
    console.log(up)
  ); // true
```

Kontrola toka - asinkronost

```
function isUp(url) {
  return fetch(url)
    .then(page => {
      const ok =
        page.status >= 200 &&
        page.status < 400;
      return ok;
    });
}
isUp('https://jsschool.axilis.com');
  .then(up => console.log(up)); // true
```

Kontrola toka - asinkronost

```
async function isUp(url) {
  const page = await fetch(url);
  const ok =
    page.status >= 200 &&
    page.status < 400;
  return ok;
}
const up = await
  isUp('https://jsschool.axilis.com');
console.log(up); // true
```

Kontrola toka - asinkronost

```
function isUp(url) {
  const page = fetch(url);
  const ok =
    page.status >= 200 &&
    page.status < 400;
  return ok;
}
const up = isUp('https://jsschool.axilis.com');
console.log(up); // true
```

Kontrola toka - asinkronost

- Asinkrono izvršavanje
  - Potpisu metode je potrebno samo dodati ključnu riječ *async*
  - Primitivne instrukcije za kontrolu toka rade bez modifikacija
  - Višestruki pozivi asinkronih metoda ne zahtjevaju niti ulančavanje niti ugnježđivanje

```
async function isUp(url) {
  const page = await fetch(url);
  const ok =
    page.status >= 200 &&
    page.status < 400;
  return ok;
}
const up = await isUp(
  'https://jsschool.axilis.com'
);
console.log(up); // true
```

Kontrola toka - asinkronost

- *Promise*
  - Jedan od standardnih, ugrađenih tipova
  - Metode instance za kontrolu vrijednosti
    - `promise.then`
    - `promise.catch`
  - Statičke metode za orkestraciju
    - `Promise.all`
    - `Promise.resolve`
- *fetch*
  - Funkcija dostupna u pregledniku
  - Dohvat resursa sa zadanog URL-a

Demo

Zadatak

- Napišite funkciju `getApplications` koja će pozvati API i dobiti `Response` servera. Potrebno je na konzolu ispisati status kod tog odgovora.

Standardne metode - String

- `const str = 'JS School';`
- `str.includes('JS'); // true`
  - Nalazi li se predani string kao dio trenutnog
- `str.indexOf('S'); // 1`
  - Vraća index lokacije prvog pojavljivanja predanog stringa
  - Vraća -1 ako se predani string ne pojavljuje
- `str.split(' '); // ['JS', 'School']`
  - Razbija string na polje stringova
  - Predani argument služi kao točka prekida

Standardne metode - Object

- `const obj = {name: 'Pero', age: 25};`
- `obj.toString(); // [object Object]`
  - Konvertira objekt u string
- `Object.keys(obj); // ["name", "age"]`
  - Dohvaća sve nazive atributa objekta
- `Object.assign({}, obj, {student: true});`
  `// {name: 'Pero', age: 25, student: true}`
  - Objektu predanom kao prvi argument dodjeljuje atribute svih ostalih parametara
  - Slično kao `{...obj, student: true}`

Standardne metode - Array

- `const arr = [1, 2, 3, 4];`
- `arr.includes(2); // true`
  - Nalazi li se predani element u polju
- `arr.indexOf(2); // 1`
  - Vraća index lokacije prvog pojavljivanja predanog elementa
  - Vraća -1 ako se predani element ne pojavljuje
- `arr.join(', '); // "1, 2, 3, 4"`
  - Spaja polje u string
  - Predani argument služi kao vezivni string

Standardne metode - Array

- `arr.filter(x => x > 2); // [3, 4]`
  - Vrati novo polje sa elementima koji zadovoljavaju uvjet
- `arr.forEach(x => console.log(x * 2)); // 2 4 6 8`
  - Izvrši funkciju za svaki element polja
- `arr.map(x => x ** 2); // [1, 4, 9, 16]`
  - Vrati novo polje sa elementima kreiranim pomoću funkcije
- ```
  arr.reduce(
     (prev, x) => `${prev}${x ** 2}`,
     ''
  ); // "14916"
  ```
  - Vrati agregiranu vrijednost izvršavanjem funkcije za svaki element
  ```

Demo

Zadatak

- Napišite funkciju koja prima listu prijava, odbaci one koje imaju manje od dvije vještine, mapira listu prijava u oblik `{fullName, grade}`, sortira ih od najveće ocjene prema najmanjoj i na kraju ispiše jednu po jednu na konzolu, sa odmakom od pola sekunde između ispisa svake prijave. Puno ime mora biti ispisano velikim slovima. Implementaciju napraviti bez korištenja for petlje.

Platforme

Preglednik

- Preglednik kao okruženje izvršavanja
- Globalni objekt je *window*
  - Ugrađeni objekti jezika
    - `Promise, Date, Math, Object, …`
  - Funkcionalnosti preglednika (DOM)
    - `console, fetch, setTimeout, …, …`
    - `document, history, localStorage, …, …`

Preglednik

- DOM
  - Sučelje prema funkcionalnostima preglednika
  - Dostupnost ovisi o pregledniku
    - *document* - trenutni dokument stranice
    - *history* - povijest trenutne sesije
    - *localStorage* - jednostavna klijentska baza podataka
    - *fetch* - metoda za dohvat resursa preko mreže
  - Dohvat preko globalnog objekta
    - `const api = window.api;`

Server

- NodeJS kao okruženje izvršavanja
- Globalni objekt je *global*
  - Ugrađeni objekti jezika
    - `Promise, Date, Math, Object, …`
  - Funkcionalnosti Nodea
    - `console, setTimeout, setInterval`
- Standardni, ugrađeni moduli

Server

- Standardni moduli
  - Dostupni kao dio platforme
    - *crypto* - kriptografija
    - *fs* - pristup datotečnom sustavu
    - *http* - HTTP server
    - *path* - putanja
    - …
  - Moraju se eksplicitno dohvatiti
    - `const module = require('module');`

Hvala na pažnji