

Razvoj programske podpore za web

- predavanja -

6. JavaScript

2/3

Creative Commons



- slobodno smijete:
 - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
 - **prerađivati** djelo
- pod sljedećim uvjetima:
 - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
 - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
 - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licenčne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

Klase u JavaScriptu (1)

- Klase u JavaScriptu slične su klasama u drugim programskim jezicima
 - Temeljem klasa stvaramo objekte
 - Objekti se stvaraju sintaksom `new MyClass()`;
- Neke specifičnosti JavaScripta
 - Mogu imati samo jedan konstruktor
 - Nema točke-zareza (;) između metoda klase
 - Svojstva se ne mora deklarirati - korištenje i dodjela vrijednosti stvara samo svojstvo:
 - `this.prop2 = propValue;`

```
class MyClass {  
    // declared property  
    prop1 = "Value";  
    // constructor  
    constructor(prop2Value) {  
        this.prop2 = prop2Value;  
    }  
    // member methods  
    method1() { ... }  
    method2() { ... }  
    //getter method  
    get getterName(...) {}  
    //setter method  
    set setterName(...) {}  
}
```

Klase u JavaScriptu (2)

Konstruktor s jednim argumentom
(nije moguće imati dodatnih
konstruktor)

```
class InsertionSort {  
  constructor(size) {  
    this.size = size;  
    this.array = this.generateRandomBigArray(size);  
  }  
  generateRandomBigArray(size){  
    let array = [];  
    while(size > 0){  
      array.push(Math.random()*1000);  
      size --;  
    }  
    return array;  
  }  
  get Size() { return this.size;}  
  set Size(newSizeValue) {  
    if(newSizeValue > this.size){  
      console.log("Error! This class does not " +  
        "allow increasing size without enlarging the array");  
    }  
    else  
      this.size = newSizeValue;  
  }  
}
```

Svojstva `this.size` i
`this.array` se ne
moraju deklarirati -
mogu se odmah
koristiti/puniti

Članska metoda klase `InsertionSort`
koja se poziva iz konstruktora i koja
generira testno polje

Testno polje se sastoji od nasumično
generiranih elemenata iz raspona [0,
1000]

Getter/setter `Size` služi pristupu i
učahurivanju svojstva `this.size`

Klase u JavaScriptu (3)

Definicija članske metode sort klase
InsertionSort

```
class InsertionSort {  
  sort(){  
    for (let i = 1; i < this.array.length; i++) {  
      let temp = this.array[i];  
      let j = i;  
      for (; j >= 1 && this.array[j - 1] > temp; j--)  
        this.array[j] = this.array[j - 1];  
      this.array[j] = temp;  
    }  
    return this.array;  
  }  
}
```

U metodi se koristi this kao
referenca na trenutni objekt -
pristupa se atributu array

```
let oneSort = new InsertionSort(500000);  
oneSort.sort();
```

Instanciranje objekta oneSort klase
InsertionSort, te poziv metode
sort() nad objektom oneSort

Što stoji iza klasa u JavaScriptu? (1)

- Klase u JavaScriptu su interno funkcije
 - U navedenom se primjeru stvara *funkcija* InsertionSort
 - Kôd za tu funkciju se uzima iz konstruktora
 - *Funkcija* InsertionSort ima svojstvo *prototype* koje sadrži članske funkcije klase InsertionSort, uključujući i konstruktor
 - Kada zovemo člansku funkciju klase nad nekim objektom, zapravo zovemo funkciju koja se smještena unutar svojstva *prototype*

```
class InsertionSort {  
  // code ommited for simplicity  
}  
console.log(typeof InsertionSort);
```

Navedena linija ispisuje "function"

Što stoji iza klasa u JavaScriptu? (2)

- Klase u JavaScriptu su sintaksni šećer (eng. syntactic sugar) i mogu se izvesti i funkcijama

```
class MyClass {  
  // declared property  
  prop1 = "Value";  
  // constructor  
  constructor(prop2Value) {  
    this.prop2 = prop2Value;  
  }  
  // member methods  
  method1() { ... }  
  method2() { ... }  
  //getter method  
  get getterName(...) {}  
  //setter method  
  set setterName(...) {}  
}
```

```
function createNewMyClass(prop2Value) {  
  const obj = {};  
  obj.prop2 = prop2Value;  
  obj.prop1 = "Value";  
  obj.method1 = function() { ... };  
  obj.method2 = function() { ... };  
  Object.defineProperty(obj, 'getterName',  
    { get: function() { ... } } );  
  Object.defineProperty(obj, 'setterName',  
    { set: function() { ... } } );  
  return obj;  
}  
  
const exampleObj =  
  createNewMyClass('sampleValue');
```

Klase i nasljeđivanje (1)

- Nasljeđivanje se vrši ključnom riječi `extends`
 - Članskim metodama i varijablama neke klase pristupa se (iz izvedene klase) putem ključne riječi `super`
 - Nadjačavanje se izvodi tako što se u izvedenoj klasi navede metoda istog imena kao što je ima i metoda u osnovnoj klasi
 - Ako se poziva metoda koje u klasi nema, traži se istoimena metoda više u hijerarhiji (u prvoj nadređenoj osnovnoj klasi, pa u drugoj nadređenoj osnovnoj klasi i tako do vrha hijerarhije

```
class MyClass1{  
    constructor() { ... }  
    method1() { ... }  
}
```

Ako konstruktor u izvedenoj klasi **ne napišemo** automatski se generira prazni konstruktor

```
class MyClass2 extends MyClass1{  
    constructor() {  
        super();  
    }  
    method1() {  
        super.method1();  
    }  
    method2() { ... }  
}
```

Ako želimo konstruktor u izvedenoj klasi **obavezno** moramo pozvati `super()` - što je konstruktor osnovne klase - prije nego se po prvi puta koristi ključna riječ `this`

Poziv `super.method1()` nije nužan, ali može biti koristan

Klase i nasljeđivanje (2)

```
class Sort {
  constructor(size) {
    this.size = size;
    this.array = this.generateRandomBigArray(size);
  }
  generateRandomBigArray(size){
    let array = [];
    while(size > 0){
      array.push(Math.random()*1000);
      size --;
    }
    return array;
  }
  get Size() { return this.size;}
  set Size(newSizeValue) {
    if(newSizeValue > this.size){
      console.log("Error! Cannot enlarge size without enlarging the array");
    }
    else
      this.size = newSizeValue;
  }
  sort(){
    console.log("This is base class. No sorting is done.");
  }
}
```

Klase i nasljeđivanje (3)

```
class InsertionSort extends Sort{
  constructor(size) {
    super(size);
  }
  sort(){
    for (let i = 1; i < this.array.length; i++) {
      let temp = this.array[i];
      let j = i;
      for (; j >= 1 && this.array[j - 1] > temp; j--)
        this.array[j] = this.array[j - 1];
      this.array[j] = temp;
    }
    return this.array;
  }
}
```

Napomene o OOP-u u JavaScriptu

- Može se koristiti ključna riječ `static` za statičke metode i svojstva (navodi se kao ključna riječ prije naziva metode)
 - Kao i u Javi ne mogu se pozvati nad instancom objekta
- Podrazumijevano svi članovi (svojstva i metode) su javni.
 - Ako se članove želi učiniti privatnim navodi se prefiks `#`. Npr:
 - `#_count = 0;`
 - `#increaseCount() { this.#_count++; }`
 - `static #pi = Math.PI;`
 - Ne postoji varijanta *protected* modifikatora vidljivosti
- Provjera je li objekt obj klase Class se vrši sa ključnom riječi `instanceof` (npr. `obj instanceof Class`)

Polja (eng. arrays) (1)

- Služe pohrani kolekcije vrijednosti ili objekata
- Metode:
 - `push(... items)` - dodaje element(e) na kraj
 - `pop()` - briše i vraća element s kraja
 - `shift()` - briše i vraća element s početka
 - `unshift(...items)` - dodaje element(e) items na početak polja
 - `splice(index, num)` - od pozicije index briše num elemenata
 - `slice(index1, index2)` - vraća novo polje s elementima od pozicija indeks1 do index2

```
let persons = ["Mark", "John", "Joe"];
persons[2] = "Susan";
persons[1] = function() { console.log("Hello from an array!") };
persons[1]();
for (let person of persons) {
  console.log(person);
}
```

Hello from an array!
Mark
[Function]
Susan

Polja (eng. arrays) (2)

```
let persons = ["Mark", "John", "Joe"];

persons.push("Zack");
// persons = ["Mark", "John", "Joe", "Zack"]
console.log(persons.pop()); // "Zack" is removed for array
console.log(persons.shift()); // "Mark" is removed from array
console.log(persons.unshift("Mark", "Zack"));
// output: ["Mark", "Zack", "John", "Joe"]
```

Slice(start, end)
izvlači članove iz
niza te ih vraća
kao rezultat
metode.
Ne mijenja niz na
kojem se izvršava
metoda

```
console.log(persons.slice(1, 3)); // ["Zack", "John"]
console.log(persons.splice(1, 1)); // ["Zack"]
console.log(persons); // ["Mark", "John", "Joe"]
```

Vraća dio niza koji je
izbačen

```
console.log(persons.splice(1, 1, "Travis", "Bill")); // ["John"]
console.log(persons); // ["Mark", "Travis", "Bill", "Joe"]
```

Splice(start, N, ...) služi za brisanje/zamjenu N
članova niza.

Mijenja niz nad kojem se izvršava metoda

John je izbačen, a Travis
i Bill ubačeni na njegovo
mjesto

Polja (eng. arrays) (3)

- Metode:
 - `concat(...items)` – nadodaje element(e) `items` u polje i vraća kopiju novog polja
 - `indexOf/lastIndexOf(item, pos)` – traži element `item`, počevši s pozicijom `pos`
 - `includes(value)` – provjerava je li `value` u polju
 - `find/filter(func)` – vraća sve vrijednosti koje zadovoljavaju uvjet (funkciju) `func`
 - `findIndex(func)` - isto kao i `find`, ali vraća indeks

```
let persons = ["Mark", "John", "Joe"];

console.log(persons.concat("Zack")); // ["Mark", "John", "Joe", "Zack"]
console.log(persons.indexOf("Joe")); // 2
console.log(persons.includes("Zack")); // false
console.log(persons.filter(person => person.includes("o"))); // ["John", "Joe"]
console.log(persons.findIndex(person => person.includes("o"))); // 1
```

Concat **ne mijenja** niz, već vraća spojene nizove kao rezultat

Polja (eng. arrays) (4)

```
let persons = ["Mark", "John", "Joe"];

persons.forEach(person => {
  console.log(person);
});
// Output:
// "Mark"
// "John"
// "Joe"

console.log(persons.map(person => {
  console.log(person.split("o"));
}));
// Output:
// ["Mark"]
// ["J", "hn"]
// ["J", "e"]

console.log(persons.join("=>"));
// "Mark => John => Joe"
```

- `forEach(func)` – poziva funkciju `func` za svaki element polja
- `map(func)` – zove `func` za svaki element polja i stvara novo polje temeljem rezultata te ga vraća
- `sort(func)` – sortira polje, te ga vraća
- `reverse()` – obrće polje, te ga vraća
- `split()/join()` – pretvara *string* u polje i obrnuto
- itd...

Mape (eng. maps) (1)

- Struktura podataka za pohranu elemenata po sistemu *ključ-vrijednost*, s time da ključ može biti bilo što
 - Kod klasičnih objekata ključ/svojstvo može biti jedino tipa *string*
- Metode:
 - `new Map()` – stvara novi objekt - mapu
 - `map.set(key, value)` – pohranjuje vrijednost `value` pod ključ `key`
 - `map.get(key)` – dohvaća vrijednost za dani ključ (`undefined` ako ne postoji ključ)
 - `map.has(key)` – provjerava postoji li ključ `key` u mapi
 - `map.delete(key)` – briše vrijednost za dani ključ `key`
 - `map.clear()` – briše sve iz mape
 - `map.size` – vraća broj parova *ključ-vrijednost* pohranjenih u mapi

Mape (eng. maps) (2)

- Po mapi se iterira pomoću:
 - `map.keys()` – vraća sve ključeve
 - `map.values()` – vraća sve vrijednosti
 - `map.entries()` – vraća niz vrijednosti u obliku `[key, value]`, koristi se u *for..of* petlji

```
let personMap = new Map([
  [1, "Mark"],
  [2, "John"],
  [3, "Joe"]
]);
personMap.set(4, "Mary");
personMap.delete(2);
for (let personKey of personMap.keys()) {
  console.log(personKey);
}
for (let personValue of personMap.values()) {
  console.log(personValue);
}
```



1
3
4
Mark
Joe
Mary

Iznimke u JavaScriptu (1)

- Iznimke u JavaScriptu slične su iznimkama u drugim programskim jezicima
 - u bloku try navodi se kôd koji potencijalno dovodi do iznimke
 - ako u bloku try dođe do iznimke, preskače se ostatak bloka try i prelazi na blok catch
 - ako u bloku try ne dođe do iznimke, ne izvodi se kôd u bloku catch
 - kôd u bloku finally se izvršava uvijek (nakon try ili nakon catch)
- Neke specifičnosti JavaScripta
 - Radi samo sinkrono - greške koje se dogode u kôdu asinkrono pokrenutom iz bloka try se neće uhvatiti u bloku catch koji je nadovezan na taj blok try
 - catch može koristiti informaciju o grešci preko objekta greške koji mu se automatski predaje (u primjeru nazvan err)

```
try {  
    // code potentially generating an exception  
} catch (err) {  
    // code handling an exception  
} finally {  
    // code executes always  
}
```

Iznimke u JavaScriptu (2)

- Objekt za greške tipa Error se sastoji od tri glavna svojstva:
 - name - vrsta greške (npr. SyntaxError)
 - message - poruka o detaljima greške
 - stack - lanac poziva metoda sa sistemskog stoga koji je doveo do greške
- Objekti greške (Error) se po potrebi mogu stvoriti i aktivirati (*baciti*, eng. throw).
 - Može se koristiti klasa Error, ali i SyntaxError, ReferenceError, TypeError, RangeError itd.
- Blok catch tipično služi za *hvatanje* (eng. catch) grešaka koje dolaze iz različitih dijelova programa
 - Ali, blok catch nije nužno pisati - npr. ako želimo da se iznimka ignorira i izvrši finally ako postoji

Iznimke u JavaScriptu (3)

```
class InsertionSort {  
  //class-specific code (removed for brevity)  
  set Size(newSizeValue) {  
    if(newSizeValue > this.size){  
      throw new RangeError("New value cannot be larger than the current size!");  
    }  
    else this.size = newSizeValue;  
  }  
}
```

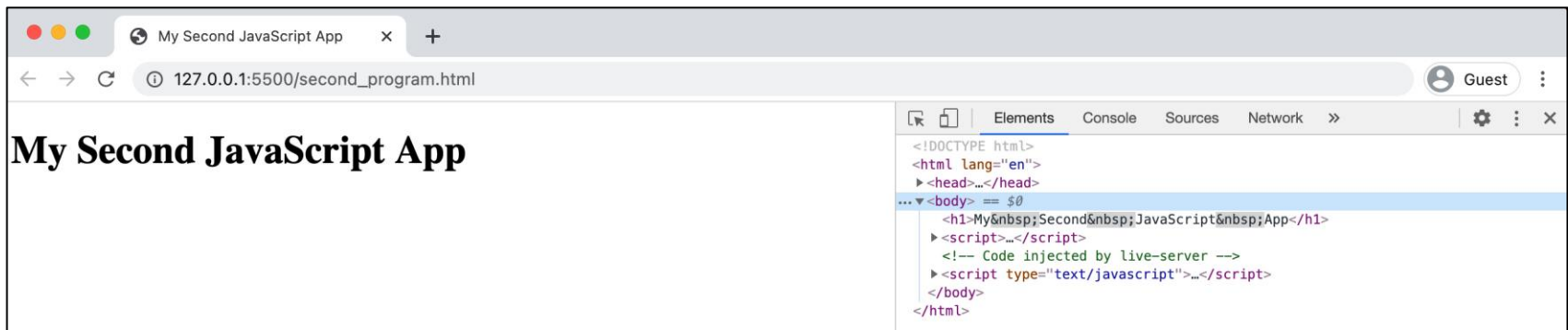
Umjesto ispisa poruke korisniku koristi se greška tipa RangeError. Prekida se izvršavanje settera Size i tijekom programa se preusmjerava u vanjski blok catch.

```
try{  
  let oneSort = new InsertionSort(100);  
  oneSort.sort();  
  oneSort.Size = 101;  
} catch(err){  
  if(err instanceof RangeError)  
    console.log(err.name + ": " + err.message);  
  else console("A sorting error has occurred!");  
}
```

Tipove greška je moguće odrediti provjerom/usporedbom tipa greške err s nekim od postojećih tipova (npr. RangeError).

Podsjetnik: Pozadina svake prikazane HTML stranice

- Internetski preglednici su tzv. okolina domaćina (eng. host environment) za HTML stranice
- Tri načina upravljanja stranicom:
 - **DOM** – Document Object Model (model HTML stranice + manipulacija)
 - **CSSOM** – Cascading Style Sheet Object Model (model CSS-a + manipulacija)
 - **BOM** – Browser Object Model (ugrađene funkcije internetskog preglednika)
- Svaki internetski preglednik omogućava prikaz DOM-a (npr. Google Chrome CTRL+Shift+j)

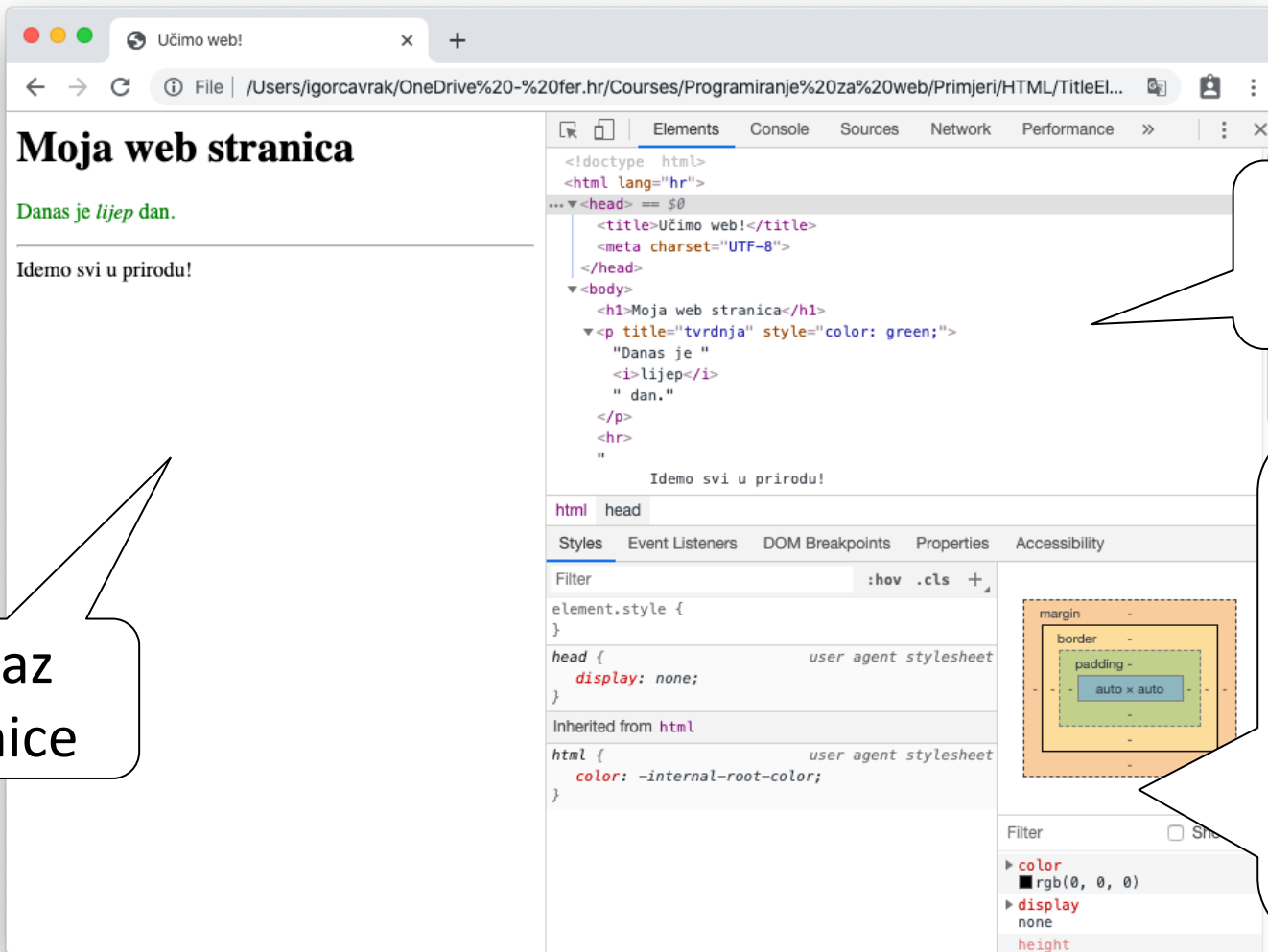


Document Object Model (1)

- *Document Object Model* je programska reprezentacija internetske stranice
 - Glavni objekt je *document* i on predstavlja internetsku stranicu koja je učitana u internetski preglednik
- Svaki element internetske stranice je predstavljen objektom koji se nalazi unutar glavnog objekta *document*
 - Struktura navedenih objekata je hijerarhijska i prati strukturu HTML-a

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>Učimo web!</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Moja web stranica</h1>
    <p title="tvrdnja" style="color: green;">
      Danas je <i>lijep</i> dan.</p>
    <hr/>
    Idemo svi u prirodu!
  </body>
</html>
```

Document Object Model (2)



- Prikaz HTML dokumenta i pripadajućeg stabla unutar *Chrome Developer Tools* (View->Developer->Developer Tools) ili desni klik mišem na sadržaj dokumenta -> Inspect)

Document Object Model (3)



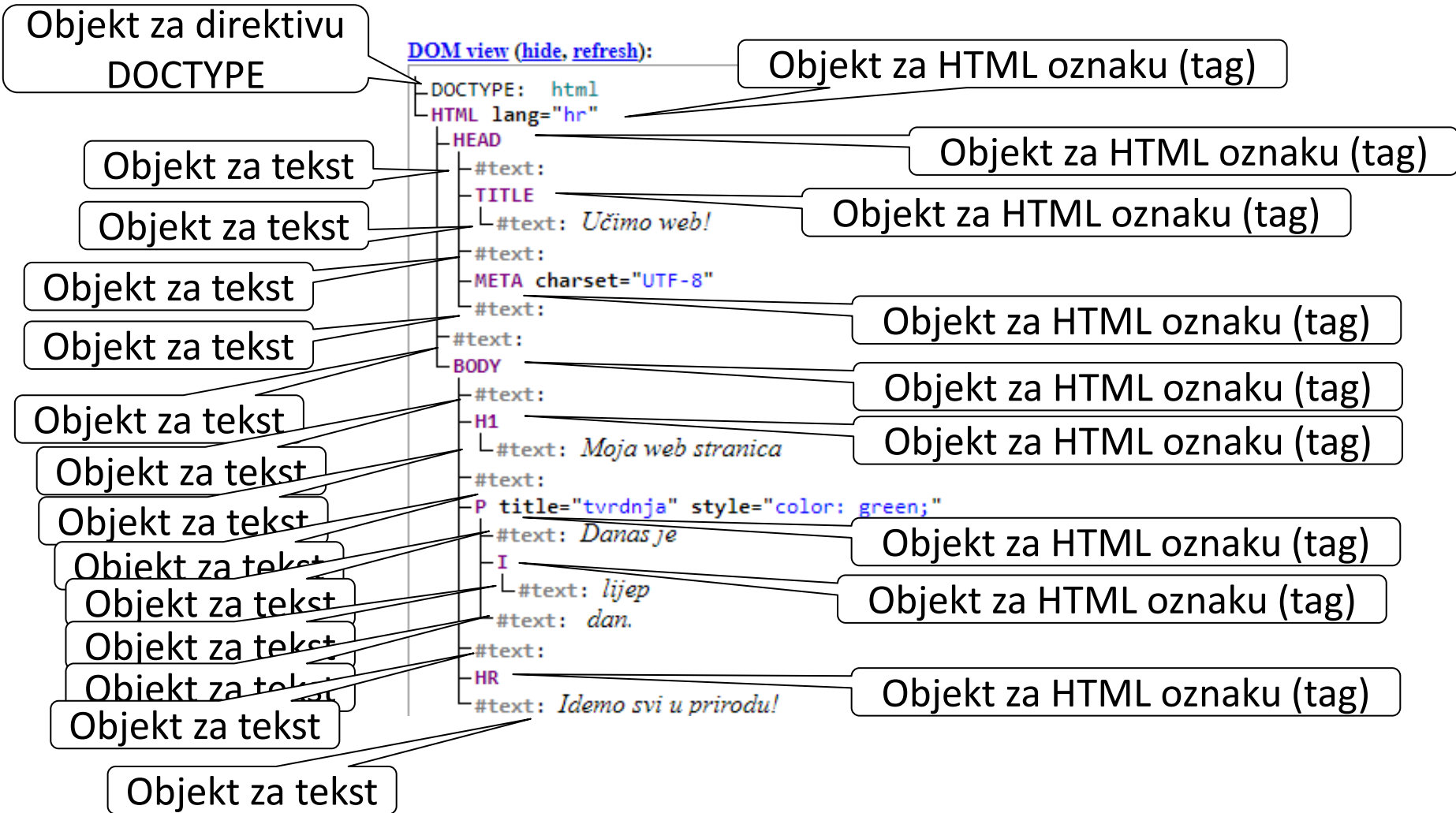
Mogućnost unosa proizvoljnog HTML kôda

Prikaz stabla objekata DOM-a (stablo je dostupno putem objekta document)

Prikaz stranice

- Mogućnost korištenja vanjskih (eng. 3rd party alata) za prikaz DOM-a
 - Npr. Live DOM Viewer <https://software.hixie.ch/utilities/js/live-dom-viewer/>

Document Object Model (4)



- Objekti za tekst u kojima prividno nema teksta koriste se za prikaz prijelaza u novi red ('\n')

Document Object Model (5)

- DOM-u se tipično pristupa iz JavaScripta (npr. manipulira se objektima DOM-a kako bi se napravile promjene internetske stranice)
- Moguć je pristup DOM-u i nakon što je internetska stranica učitana i što je stablo objekata unutar document-a već formirano
 - Primjerice, takav se pristup može ostvariti iz konzole (eng. console) internetskog preglednika

The screenshot shows a web browser window with the address bar displaying "Učimo web!". The page content includes the heading "Moja web stranica", the text "Danas je lijep dan.", and "Idemo svi u prirodu!". The browser's developer console is open, showing the command `> document` and its output, which is a tree view of the document object. Below this, the command `> document.body` is entered, showing the output as the body element. Callouts with arrows point to these elements:

- Prikaz cijele stranice**: Points to the main content area of the web page.
- Pristup konzoli**: Points to the "Console" tab in the developer tools.
- Upisano "document" - rezultat ispod**: Points to the output of the `document` command in the console.
- Upisano "document.body" - rezultat ispod**: Points to the output of the `document.body` command in the console.

Document Object Model (6)

- Glavni objekt DOM-a je `document` i ima niz podobjekata:
 - `document.documentElement`
 - predstavlja oznaku `<html>`
 - `document.body`
 - predstavlja oznaku `<body>`
 - može biti null ako stranica nije u potpunosti učitana
 - `document.head`
 - predstavlja oznaku `<head>`
- Nad objektima DOM-a moguće je pozivati brojne metode i koristiti brojna svojstva. Npr.
 - `.childNodes` (svojstvo koje je kolekcija)
 - `.firstChild` (objekt koji je prvo dijete nekog objekta)
 - `.lastChild` (objekt koji je zadnje dijete nekog objekta)

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>Učimo web!</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Moja web stranica</h1>
    <p title="tvrdnja" style="color: green;">
      Danas je <i>lijep</i> dan.</p>
    <hr/>
    Idemo svi u prirodu!
  </body>
  <script>
    for(node of
      document.documentElement.childNodes){
      alert(node.tagName);
    }
  </script>
</html>
```

for-of za
iteraciju po
kolekcijama

Ispisuje se:
HEAD
undefined (zbog '\n')
BODY

Document Object Model (7)

- Nad elementima DOM-a moguće je pozivati brojne metode i koristiti brojna svojstva. Npr.
 - `.nextSibling` (sljedeći objekt na istoj razini)
 - `.previousSibling` (prethodni objekt na istoj razini)
 - `.parentNode` (roditeljski objekt)
- Svojstva i metode koje rade samo s elementima (oznakama), a ne i tekстом i ostalim objektima
 - `.children` (elementi djece)
 - `.firstElementChild` (prvo dijete element)
 - `.lastElementChild` (zadnje dijete element)
 - `.previousElementSibling` (prethodni element na istoj razini)
 - `.nextElementSibling` (sljedeći element na istoj razini)
 - `.parentElement` (roditeljski element)

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>Učimo web!</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Moja web stranica</h1>
    <p title="tvrdnja" style="color: green;">
      Danas je <i>lijep</i> dan.</p>
    <hr/>
    Idemo svi u prirodu!
  </body>
  <script>
    for(node of
      document.documentElement.children){
        alert(node.tagName);
      }
  </script>
</html>
```

for-of za
iteraciju po
kolekcijama

Ispisuje se:
HEAD
BODY

Document Object Model (8)

- DOM je moguće pretraživati kako bi se pronašli specifični elementi
 - Takvi elementi moraju imati atribut id postavljen na neku vrijednost
 - Pretraživanje se obavlja koristeći:
 - document.getElementById(element id value)
 - Koristi se id elementa
- document.querySelectorAll(CSS query) i document.querySelector
 - za složenije slučajeve pretraživanja

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <title>Učimo web!</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Moja web stranica</h1>
    <p id="tvrdnja" title="tvrdnja"
      style="color: green;">
      Danas je <i>lijep</i> dan.</p>
    <hr/>
    Idemo svi u prirodu!
    <br/>
    <button style="height:25px;width:200px"
      onclick="changeWeather()">
      Change weather</button>
  </body>
  <script>
    function changeWeather(){
      //change weather code (toggle)
    }
  </script>
</html>
```

changeWeather
se poziva klikom
na gumb

Document Object Model (9)

```
function changeWeather(){  
  let tvrdnjaPara = document.getElementById("tvrdnja");  
  if(tvrdnjaPara.style.color == "green")  
    tvrdnjaPara.style.color = "red";  
  else  
    tvrdnjaPara.style.color = "green";  
  if(tvrdnjaPara.innerHTML.indexOf("nije lijep") != -1)  
    tvrdnjaPara.innerHTML = "Danas je <i>lijep</i> dan.";  
  else  
    tvrdnjaPara.innerHTML = "Danas <i>nije lijep</i> dan.";  
  tvrdnjaParaSecondSibling = tvrdnjaPara.nextSibling.nextSibling.nextSibling;  
  if(tvrdnjaParaSecondSibling.nodeValue == "")  
    tvrdnjaParaSecondSibling.nodeValue = "Idemo svi u prirodu!";  
  else  
    tvrdnjaParaSecondSibling.nodeValue = "";  
}
```

Dohvat elementa po id-u

Promjena stila (boje) elementa-tvrdnje

Svojstvo innerHTML služi pristupu HTML-u unutar elementa <p>

Atributom
nodeValue pristupa
se tekstu objekta
koji je tekstualnog
tipa

Objekt s tekстом
"Idemo svi u
prirodu!" je treći
objekt nakon <p>.
Prvi je '\n', drugi
<hr/>.

Document Object Model (10)

- `querySelector`, `querySelectorAll`
 - Uzimaju referencu na element/elemente iz DOM-a i omogućavaju rad s njima u JavaScriptu
 - Parametar funkcije je selektor elementa (kao i u CSS-u)
 - https://www.w3schools.com/cssref/css_selectors.asp

```
// Select by element type - First element of type body
const body = document.querySelector('body');
// Select by element type - All elements of type div
const div = document.querySelectorAll('div');
// Select by class - All elements with class class
const elements = document.querySelectorAll('.class');
// Select by attribute - All elements with attribute attr set to "12"
const elements = document.querySelectorAll('[attr="12"]');
// All input elements that are not disabled
const inputs = document.querySelectorAll('input:not([disabled])');
```

Document Object Model (11)

- `Document.createElement`
 - Stvara HTML element u JavaScriptu
 - Element se može dodati bilo gdje na stranicu

```
<html>
  <head>...</head>
  <body>
    ...
    <body>
  </html>
```

```
const div = document.createElement("div");

const body = document.querySelector("body");
body.appendChild(div);
```

Novostvoreni element se
mora dodati u DOM
koristeći metodu
`appendChild()`

```
<html>
  <head>...</head>
  <body>
    ...
    <div></div>
  <body>
</html>
```


Document Object Model (12)

- U JavaScriptu možemo uređivati stilove, tekst, slušače događaja, itd.

Uređivanje stila

Dodavanje slušača
na klik

Uređivanje
unutarnjeg teksta
elementa

```
▼ const div = document.createElement('div');  
  
div.style.width = '100px';  
div.style.height = '200px';  
div.style.backgroundColor = 'red';  
  
div.onclick = (event) => console.log('KLIK');  
  
div.innerText = 'INNER TEXT';  
  
console.log(div);  
► <div style="width: 100px; height: 200px; background-color: red;">
```

Browser Object Model (BOM)

- Sadržava globalne objekte preglednika koji su dostupni u cijeloj web-aplikaciji
 - Objekti omogućavaju korištenje ugrađenih funkcija preglednika
- Dostupni objekti su: window, location, screen, history, navigator, cookie
 - window je roditelj svim ostalim objektima
 - https://www.w3schools.com/js/js_ex_browser.asp

Browser Object Model (BOM) - window

- Objekt window sadržava metode za upravljanje internetskim preglednikom
- Neke od funkcija su:
 - Otvaranje novog prozora
 - Promjena dimenzija i položaja prozora
 - Zamagljivanje (eng. blur)
 - Promjena fokusa
 - Zatvaranje prozora
 - ...

```
const newWindow =  
window.open('https://www.fer.unizg.hr/');
```

```
newWindow.blur();  
newWindow.focus();
```

Blur() – oduzima fokus trenutnom prozoru

Focus() – daje fokus trenutnom prozoru

```
newWindow.moveTo(200, 300);  
newWindow.resizeTo(200, 200);
```

ResizeTo()/MoveTo() – mijenja poziciju/veličinu prozora na zadane koordinate/dimenzije

```
newWindow.close();
```

Close() – zatvara trenutni prozor ("tab")

Browser Object Model (BOM) - location

- Sadrži informacije o lokaciji prozora
- Lokacija = trenutni URL prozora
- Lokacija != pozicija prozora

```
// Extracting username from URL
const params = location.search.substr(1).split('&');
const username = params[0].split('=')[1];
```

Trenutni URL

Dohvati parametre iz URL-a, podijeli ih po & te izvuci korisničko ime iz prvog parametra

```
▼ // https://www.fer.unizg.hr/?username=USERNAME&firstname=FIRSTNAME
console.log(location.search);

const params = location.search.substr(1).split('&');
const username = params[0].split('=')[1];

console.log(username);
```

?username=USERNAME&firstname=FIRSTNAME
USERNAME

Browser Object Model (BOM) - history

- Sadrži informacije o povijesti lokacija prozora
- Koristi se za navigaciju na druge lokacije
- Prati stanje lokacije i omogućuje dodavanje novog stanja (*pushState*), zamjenu stanja (*replaceState*) te vraćanje na prethodno posjećena stanja (*back*, *go*)
- <https://developer.mozilla.org/en-US/docs/Web/API/History>

Komunikacija s korisnikom (1)

- JavaScript može komunicirati s korisnikom metodama `alert`, `prompt` i `confirm`
- `alert(message);`
 - Prikazuje poruku korisniku i očekuje potvrdu s "U redu"
- `prompt(message, default answer);`
 - Postavlja pitanje korisniku i nudi podrazumijevani odgovor (`default answer`)
 - Podrazumijevani odgovor je opcionalan
 - Vraća uneseni odgovor ili `null` ako je korisnik pritisnuo ESC
- `confirm(question);`
 - Postavlja pitanje korisniku i očekuje od njega odgovor "OK" ili "Cancel"
 - Vraća `true` za odabir "OK", a `false` za "Cancel" ili ESC

```
alert("Hello from JavaScript");  
let promptAnswer = prompt("JavaScript needs an answer", "default");  
let confirmResponse = confirm("A question from JavaScript?");
```

Komunikacija s korisnikom (2)

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JavaScript example -> Prompt</h2>

    <button onclick="promptExample()">Try it</button>

    <p id="res"></p>

    <script>
      function promptExample() {
        var txt;
        var name = prompt("Please enter your name:", "Fran");
        if (name == null) {
          txt = "User cancelled the prompt.";
        } else if (name == "") {
          txt = "Please try again";
        } else {
          txt = "Hello " + name + "! How are you today?";
        }
        document.getElementById("res").innerHTML = txt;
      }
    </script>
  </body>
</html>
```

JavaScript example -> Prompt

Try it



This page says

Please enter your name:

Fran

Cancel

OK



JavaScript example -> Prompt

Try it

Hello Fran! How are you today?

JSON (1)

- JSON (*JavaScript Object Notation*) je standard koji se koristi za predstavljanje vrijednosti i objekata
 - Opisan je standardom RFC 4627
 - Inicijalno je napravljen za potrebe JavaScripta, ali je postao vrlo popularan i koristi se i u drugim jezicima
 - U današnje se vrijeme često koristi za razmjenu podataka klijenta i poslužitelja
- U JavaScriptu postoje dvije glavne metode:
 - `JSON.stringify` - pretvara objekte u JSON format
 - `JSON.parse` - učitava objekte iz JSON formata
- Pretvorba objekta u znakovni niz u JSON formatu naziva se serijalizacijom (eng. *serialization, marshalling*)

JSON (2)

```
let person = {  
  OIB: "12345678912345",  
  name: "Pero",  
  surname: "Perić",  
  "home city": "Zagreb"  
};  
  
personJSON = JSON.stringify(person);  
console.log(personJSON);  
let personFromJSON = JSON.parse(personJSON);
```

Poziv metode `JSON.stringify` kao rezultat vraća *string* kojim se objekt predstavlja u JSON formatu (vraćeni *string* se pohranjuje u varijablu `personJSON`)

I za svojstva i za vrijednosti su obavezni dvostruki navodnici (jednostruki se ne prihvataju)

```
"{"OIB":"12345678912345","name":"Pero","surname":"Perić","home city":"Zagreb"}"
```

Objekt koji je pretvoren u JSON sastoji se od parova "*naziv svojstva*" : "*vrijednost svojstva*" odvojenih zarezom. Cijeli JSON je omeđen viticama { i }

JSON (3)

- `JSON.stringify` prihvaća i može serijalizirati:
 - objekte (omeđene s `{ i }`)
 - polja (omeđena s `[i]`)
 - stringove
 - brojeve
 - logičke vrijednosti
 - `null`
- `JSON.stringify` prilikom serijalizacije preskače:
 - funkcije
 - simbole
 - sva svojstva u kojima je pohranjen `undefined`
- `JSON.stringify` generira grešku ako dođe do pojave cirkularnih referenci (npr. objekt A referencira objekt B, koji referencira objekt A)

```
let skip = { undef: undefined, func: () => {} };  
console.log(JSON.stringify(skip));  
{}
```

Ispis

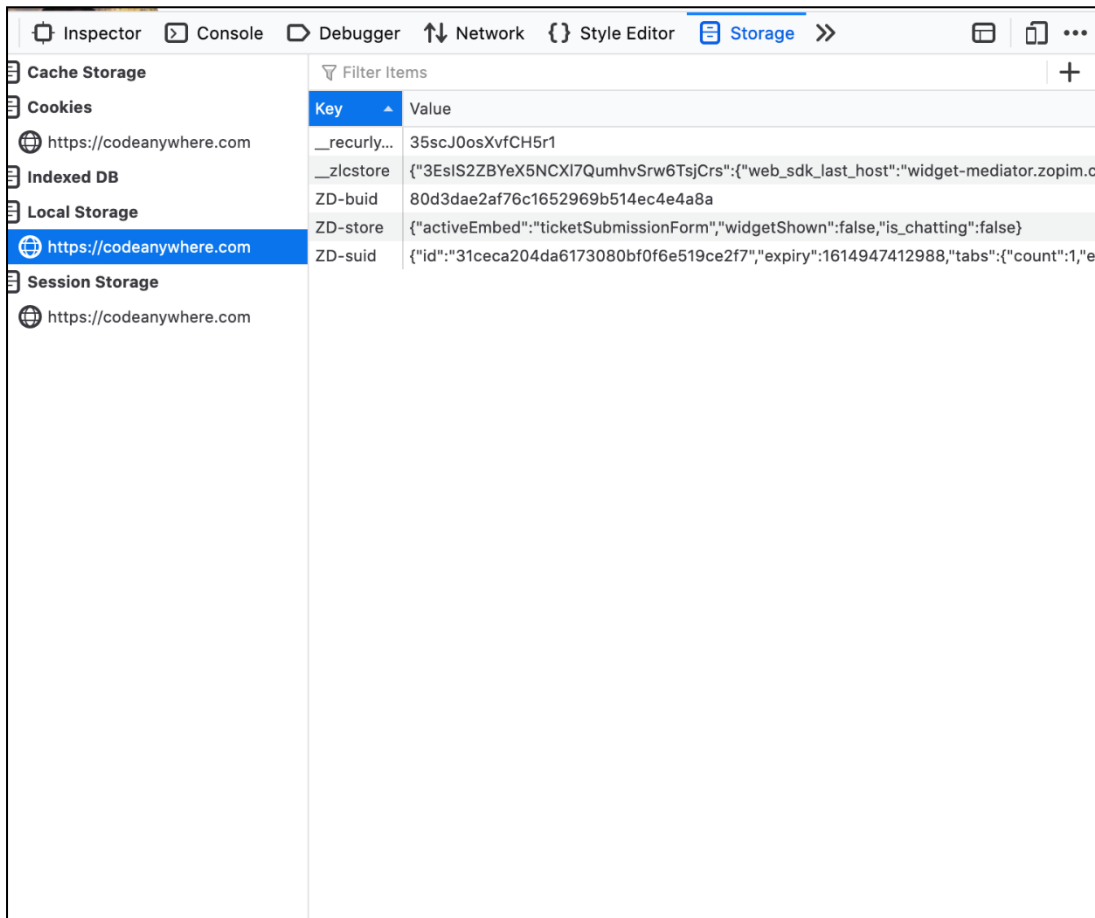
```
let a = { a: "a" };  
let b = { a: a };  
a.b = b;  
  
console.log(JSON.stringify(a))  
► Uncaught TypeError: cyclic object value  
  <anonymous> debugger eval code:5  
  [Learn More]
```

Lokalni spremnik (eng. local storage) (1)

- Lokalni spremnik omogućava pohranu podataka na klijentskoj strani u internetskom pregledniku (eng. browser)
- Koristi se kada je potrebno očuvati podatke uslijed osvježavanja (eng. refresh) ili ponovnog pokretanja internetskog preglednika
- Većina preglednika dozvoljava pohranu barem 2 MB podataka u lokalnom spremniku
- Poslužitelj ne može direktno mijenjati podatke pohranjene u lokalnom spremniku, već se promjene izvode samo na klijentu korištenjem JavaScripta
- Lokalni spremnici su vezani na kombinaciju domena/protokol/port, te stoga nije moguće dijeliti lokalne spremnike među domenama ili među protokolima/portovima

Lokalni spremnik (eng. local storage) (2)

- U razvojnim alatima (eng. developer tools) preglednika (npr. u Application prozoru u Chrome-u) može se vidjeti stanje lokalnog spremnika



Svaki zapis lokalnog spremnika se sastoji od ključa i vrijednosti

Zapise lokalnog spremnika moguće je brisati putem razvojnog alata internetskog preglednika, ali i programski

Lokalni spremnik (eng. local storage) (3)

- Lokalni spremnik ima šest metoda:
 - `setItem(key, value)` – pohranjuje par *ključ-vrijednost*
 - `getItem(key)` – dohvaća vrijednost po ključu
 - `removeItem(key)` – briše ključ i njegovu vrijednost
 - `clear()` – briše cijeli spremnik
 - `key(index)` – dohvaća naziv ključa na danoj poziciji `index`
 - `length` – dohvaća broj pohranjenih parova *ključ-vrijednost* u lokalnom spremniku

```
localStorage.setItem('oneInt', 42);  
alert(localStorage.getItem('oneInt'));  
localStorage.oneInt = 43;  
alert(localStorage.oneInt);  
delete localStorage.oneInt;
```

Lokalni spremnik i JSON

- U lokalni spremnik se mogu pohranjivati samo stringovi
 - Objekte prije pohrane treba pretvoriti u stringove

```
class StorageEntry{  
  constructor(val){  
    this.val = val;  
  }  
}  
  
let entry = new StorageEntry(1);  
let entryJSON = JSON.stringify(entry);  
localStorage.entry = entryJSON;  
let entryJSONFromStorage = JSON.parse(localStorage.entry);  
alert("Storage value: " + entryJSONFromStorage.val);
```

Klasa čiji će se objekti spremati u lokalni spremnik

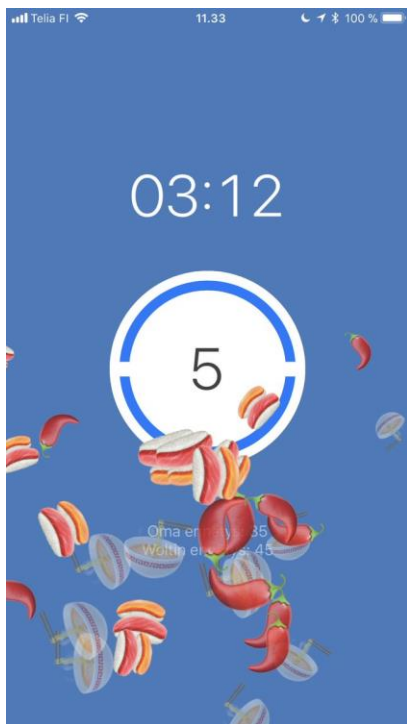
Pretvorba objekta u JSON format metodom `JSON.stringify`. Objekt je sada predstavljen stringom:

```
{"val":1}
```

Nakon čitanja iz lokalnog spremnika potrebno je napraviti pretvorbu iz JSON-a u objekt (`JSON.parse`)

Igra CookieClicker v1 (1)

- Igra koja broji klikove na keksić u vremenu od 10 sekundi i čuva rang ljestvicu uspješnosti igrača
- Inspiracija je Wolt-ova igra (prilikom narudžbe hrane)



Wolt



Igra CookieClicker v1 (2)

HTML

```
<!DOCTYPE html>
<html lang="hr">
  <head>
    <meta charset="UTF-8">
    <title>Učimo web!</title>

    <link rel="stylesheet" href="style.css" />

    <script src="CookieGame.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

CSS

```
html {
  height: 100%;
}

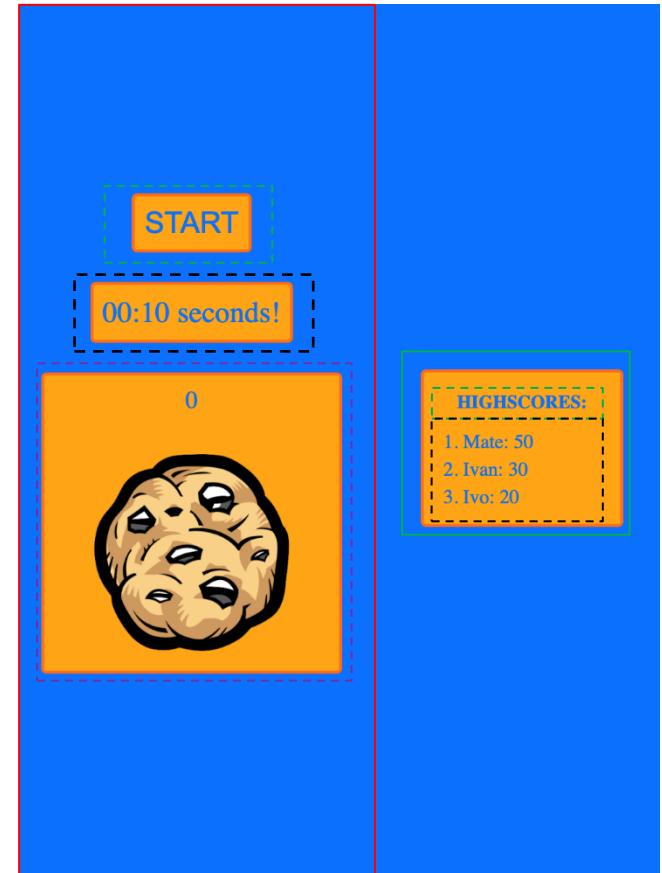
body {
  background-color: #0070FF;
  height: 100%;
  display: flex;
  justify-content: space-evenly;
  align-items: center;
  color: #0070FF;
}
...
```

JS

```
class CookieGame { ... };
class HighScoreManager { ... };
class Utils { ... };
...
```


Igra CookieClicker v1 (3)

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="cookie--wrapper">
      <div class="cookie--wrapper-row">
        <button class="btn--start box-style">START</button>
      </div>
      <div class="cookie--wrapper-row">
        <div class="timer--wrapper box-style">
          <span id="timer">00:10</span> seconds!
        </div>
      </div>
      <div class="cookie--wrapper-row">
        <div class="cookie-image--wrapper box-style">
          <span id="click-count">0</span>
          
        </div>
      </div>
    </div>
    <div class="highscore--wrapper box-style">
      <div class="highscore--wrapper-row">
        <h3>HIGHSCORES:</h3>
      </div>
      <div class="highscore--list-wrapper">
        <ol class="highscore--list"></ol>
      </div>
    </div>
  </body>
</html>
```



Igra CookieClicker v1 (4)

Glavne boje koje koristimo na stranici

```
1  /*
2    primary: #FFA500
3    secondary: #FF6B0D
4    complement: #0070FF
5  */
```

Vertikalno centriramo sve elemente te ih horizontalno razmičemo da budu jednako udaljeni

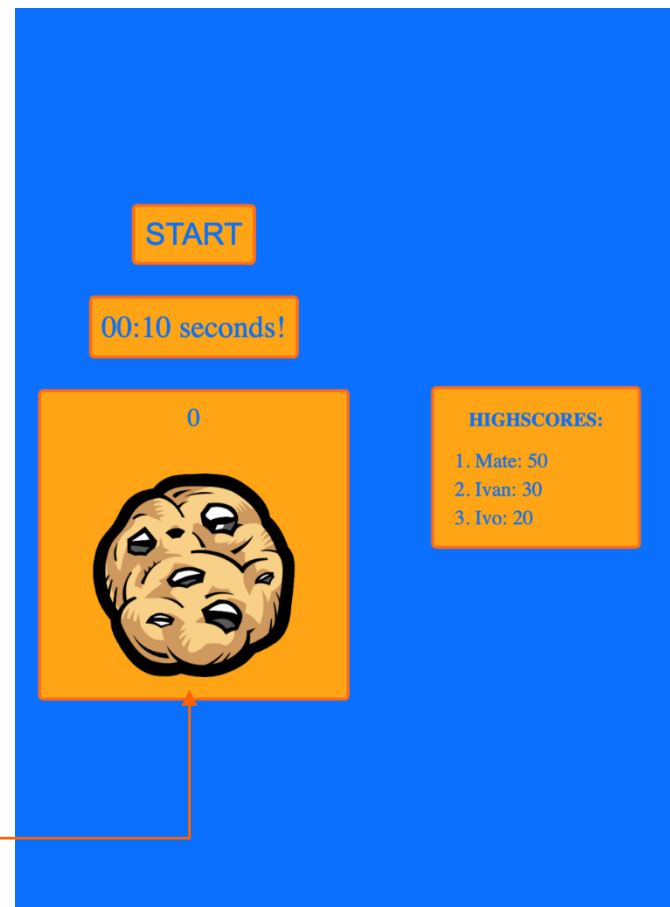
```
body {
  background-color: #0070FF;
  height: 100%;
  display: flex;
  justify-content: space-evenly;
  align-items: center;
  color: #0070FF;
}
```

Univerzalni stil pozadine i obruba elemenata










```
.box-style {
  background-color: #FFA500;
  border: 3px solid #FF6B0D;
  border-radius: 5px;
}
```

Apsolutno pozicioniramo keks na točnu poziciju




```
.cookie-image--wrapper > img {
  position: absolute;
  width: 200px;
  height: 200px;
  top: 60%;
  left: 50%;
  margin-right: -50%;
  transform: translate(-50%, -50%);
  cursor: pointer;
}
```







Igra CookieClicker v1 (5)

- ✓  CookieGame
 -  cookieCount
 -  gameIntervalTimer
 -  updateTimerUI
 -  updateHighScoreListUI
 -  resetGameUI
 -  constructor
 - >  resetGame
 - >  startGame

CookieGame - glavna klasa koja upravlja igrom

- ✓  HighScoreManager
 - >  getHighScores
 - >  updateHighScore

HighScoreManager – klasa s funkcijama za upravljanje najboljim rezultatima

- ✓  Utils
 - >  setDummyHighscores
 -  formatTwoDigits
 - >  formatTime

Utils - klasa s pomoćnim funkcijama