Programiranje u skriptnim jezicima (PJS)

Nositelj: doc. dr. sc. Nikola Tanković
Asistenti:

- Luka Blašković, univ. bacc. inf.
- Alesandro Žužić, univ. bacc. inf.

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

[5] DOM, JSON i Asinkrono programiranje

Prilikom izrade web aplikacija i stranica, često ćete na neki način manipulirati strukturom dokumenata i njihovim sadržajem. U ovom poglavlju upoznat ćemo se s Document Object Model (DOM) standardom, koji predstavlja aplikacijsko programsko sučelje (API) za kontrolu HTML-a koristeći Document objekt. Važno je razumjeti kako funkcionira DOM budući da se svi poznati JavaScript razvojni okviri temelje na njemu (React, VUE, Angular, jQuery...). Dodatno, upoznat ćemo se s JSON formatom (JavaScript Object Notation) koji se koristi za razmjenu podataka između klijenta i servera te predstavlja jedan od najčešćih, ako ne i najčešće korišteni format za razmjenu podataka. Za sam kraj ćemo proći asinkrono programiranje i time postaviti dobre temelje za uvod u svijet programskog inženjerstva i razvoja web aplikacija.

Posljednje ažurirano: 3.6.2024.

Sadržaj

0. Ponavljanje HTML-a i CSS-a

U ovoj sekciji ćemo se osvrnuti na osnove HTML-a i CSS-a. **HTML** (HyperText Markup Language) je markup jezik za označavanje struktura web stranica, dok je CSS (Cascading Style Sheets) jezik za stilizaciju tih struktura. Kombinacija ova dva jezika omogućuje nam da oblikujemo i prikažemo sadržaj web stranica na željeni način.

Kako bi vam HTML i CSS jezici već trebali biti poznati, u nastavku ćemo se osvrnuti na neke osnovne koncepte i primjere korištenja HTML i CSS elemenata. U skripti se dalje nećemo detaljno baviti HTML i CSS jezicima, već ćemo se fokusirati na JavaScript, odnosno kako možemo **manipulirati HTML elementima** pomoću JavaScripta.

> **HTML (HyperText Markup Language)**

HTML je markup jezik (***eng. markup language***) koji se koristi za strukturiranje sadržaja web stranica. Sastoji se od HTML elemenata koji se koriste za označavanje dijelova sadržaja. Svaki HTML element sastoji se od otvarajuće oznake (***eng. start tag***), sadržaja elementa (***eng. content***) i zatvarajuće oznake (***eng. end tag***).

Osnovna struktura HTML dokumenta često se može podijeliti na `head` i `body` dijelove. `head` dio obično sadrži meta informacije o dokumentu, kao što su naslov stranice, veze prema CSS datotekama, skripte itd. `body` dio sadrži glavni sadržaj stranice, poput zaglavlja (**header**), navigacije (**nav**), članaka (**article**) i podnožja (**footer**).

```
<head>
    <title>Moja web stranica</title>
</head>

<body>
    <header>
        <h1>Naslov</h1>
        <nav>
            <ul>
                <li><a href="#">Početna</a></li>
                <li><a href="#">O nama</a></li>
                <li><a href="#">Kontakt</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <article>
            <h2>Članak 1</h2>
            <p>Ovo je prvi članak.</p>
        </article>

        <article>
            <h2>Članak 2</h2>
            <p>Ovo je drugi članak.</p>
        </article>
    </main>
```

```
    <footer>
        <p>&copy; 2022 Web Stranica</p>
    </footer>
</body>
```

> Treba napomenuti da je dolaskom HTML5 standarda uvedeno mnogo novih elemenata i atributa koji omogućuju bolje semantičko označavanje sadržaja web stranica. Semantičko označavanje je važno jer pomaže tražilicama i drugim alatima da bolje razumiju strukturu i značenje sadržaja na web stranici. Svakako je moguće gotovo sve elemente stilizirati pomoću CSS-a i svesti na `div` i `span` elemente, ali korištenje semantičkih elemenata poboljšava pristupačnost i SEO (Search Engine Optimization) web stranice.

U najnovijem HTML standardu za vrijeme pisanja ove skripte (svibanj 2024), postoji 142 HTML elementa. U tablici ispod prikazani su neki od najčešće korištenih HTML elemenata, zajedno s njihovim opisima, atributima i primjerima korištenja. Svakako provjerite i HTML5 specifikaciju.

Naziv elementa	Opis	Atributi	Primjer
`<html>`	Korijenski element HTML dokumenta	`lang`	`<html lang="en"> ... </html>`
`<head>`	Sadrži meta-informacije o dokumentu	-	`<head> ... </head>`
`<title>`	Naslov dokumenta, prikazan u naslovnoj traci preglednika	-	`<title>Naslov stranice</title>`
`<meta>`	Definira metapodatke o HTML dokumentu	`charset`, `name`, `content`, `http-equiv`	`<meta charset="UTF-8">`
`<link>`	Povezuje vanjske resurse, poput CSS datoteka	`rel`, `href`, `type`	`<link rel="stylesheet" href="style.css">`
`<style>`	Sadrži CSS stilove za dokument	`type`	`<style> body { background-color: lightblue; } </style>`
`<script>`	Sadrži ili povezuje JavaScript kôd	`src`, `type`	`<script src="script.js"></script>`
`<body>`	Glavni sadržaj HTML dokumenta	-	`<body> ... </body>`
`<h1>` do `<h6>`	Naslovi različitih razina	-	`<h1>Naslov 1</h1>`
`<p>`	Paragraf teksta	-	`<p>Ovo je paragraf.</p>`
`<a>`	Hiperlink (poveznica)	`href`, `target`	`<a href="https://example.com">Link</a>`
`<img>`	Ugrađena slika	`src`, `alt`, `width`, `height`	`<img src="slika.jpg" alt="Opis slike">`
`<ul>`	Neuređena lista (bez numeriranja)	-	`<ul><li>Prva stavka</li><li>Druga stavka</li> </ul>`
`<ol>`	Uređena lista (numerirana)	-	`<ol><li>Prva stavka numerirano</li><li>Druga stavka numerirano</li></ol>`
`<li>`	Stavka liste	-	`<li>Stavka</li>`
`<table>`	Tablica	-	`<table> ... </table>`
`<tr>`	Redak u tablici	-	`<tr> ... </tr>`
`<td>`	Ćelija u tablici	`colspan`, `rowspan`	`<td colspan="2">Ćelija</td>`
`<th>`	Zaglavlje ćelije u tablici	`colspan`, `rowspan`, `scope`	`<th scope="col">Zaglavlje</th>`
`<form>`	Forma za unos podataka	`action`, `method`	`<form action="/submit" method="post"> ... </form>`
`<input>`	Unos podataka	`type`, `name`, `value`, `placeholder`	`<input type="text" name="ime" placeholder="Unesite ime">`
`<button>`	Gumb	`type`	`<button type="submit">Pošalji</button>`
`<div>`	Block element za grupiranje sadržaja	-	`<div> ... </div>`
`<span>`	Inline element za grupiranje sadržaja	-	`<span> ... </span>`
` `	Prijelom linije	-	`Tekst Prelomljena linija`
`<hr>`	Horizontalna linija	-	`<hr>`
`<b>`	Podebljani tekst	-	`<b>Podebljano</b>`
`<i>`	Kurziv tekst	-	`<i>Kurziv</i>`
`<u>`	Podvučeni tekst	-	`<u>Podvučeno</u>`

Primjer upotrebe `id` atributa:

Atribut `id` koristi se za jedinstveno identificiranje gotovo bilo kojeg HTML elementa na stranici. `id` atribut mora biti **jedinstven unutar cijelog dokumenta**.

Sintaksa: `<tag id="jedinstveniID">`.

```
<style>
    #jedinstveniElement {
        color: blue;
        font-size: 20px;
    }
</style>
<body>
    <p id="jedinstveniElement">Ovo je paragraf s jedinstvenim ID atributom.</p>
</body>
</html>
```

Primjer upotrebe `class` atributa:

Atribut `class` koristi se za grupiranje više elemenata koji dijele iste stilove ili ponašanje. Elementi mogu imati **više klasa odvojenih razmakom**.

Sintaksa: `<p class="klasa1 klasa2 klasa3 klasaN...">`.

```
<style>
    .crveniTekst {
        color: red;
    }
    .velikiTekst {
        font-size: 24px;
    }
</style>
<body>
    <p class="crveniTekst">Ovo je paragraf s crvenim tekstom.</p>
    <p class="velikiTekst">Ovo je paragraf s velikim tekstom.</p>
    <p class="crveniTekst velikiTekst">Ovo je paragraf s crvenim i velikim tekstom.</p>
</body>
</html>
```

Primjer kombiniranja `id` i `class` atributa:

U ovom primjeru koristimo oba atributa kako bismo jedinstveno identificirali jedan element, dok ostala dva grupiramo pomoću klase `podnaslov`.

```
<style>
    #glavniNaslov {
        color: green;
        font-size: 28px;
    }
    .podnaslov {
        color: gray;
        font-size: 18px;
    }
</style>
<body>
    <h1 id="glavniNaslov">Ovo je glavni naslov s ID atributom.</h1>
    <h2 class="podnaslov">Ovo je podnaslov s class atributom.</h2>
    <h2 class="podnaslov">Ovo je još jedan podnaslov s class atributom.</h2>
```

```
</body>
```

CSS (Cascading Style Sheets)

CSS je jezik za stilizaciju HTML elemenata. Omogućuje nam definiranje izgleda i rasporeda elemenata na web stranici. CSS pravila sastoje se od selektora (***eng. selectors***) i deklaracija svojstava (***eng. declarations***).

Sljedeća tablica pokriva osnovna CSS svojstva:

Naziv svojstva	Opis	Vrijednosti	Primjer
`color`	Boja teksta	Naziv boje, heksadecimalna, RGB, RGBA, HSL, HSLA	`color: red;`
`background-color`	Boja pozadine	Naziv boje, heksadecimalna, RGB, RGBA, HSL, HSLA	`background-color: #f4f4f4;`
`background-image`	Slika pozadine	URL slike	`background-image: url('slika.jpg');`
`background-size`	Veličina slike pozadine	cover, contain, px, %	`background-size: cover;`
`font-size`	Veličina fonta	px, em, rem, %, vw, vh	`font-size: 16px;`
`font-family`	Obitelj fonta	Naziv fonta, lista fontova	`font-family: Arial, sans-serif;`
`font-weight`	Debljina fonta	normal, bold, bolder, lighter, 100-900	`font-weight: bold;`
`text-align`	Poravnanje teksta	left, right, center, justify	`text-align: center;`
`margin`	Vanjski razmak	px, em, %, auto	`margin: 10px;`
`padding`	Unutarnji razmak	px, em, %, auto	`padding: 10px;`
`border`	Obrub elementa	širina tip stil boje	`border: 1px solid #ccc;`
`border-radius`	Zaobljenost rubova	px, %, em	`border-radius: 5px;`
`width`	Širina elementa	px, em, %, vw	`width: 100%;`
`height`	Visina elementa	px, em, %, vh	`height: 50px;`
`display`	Način prikaza	block, inline, inline-block, flex, grid, none	`display: block;`
`position`	Pozicioniranje elementa	static, relative, absolute, fixed, sticky	`position: absolute;`
`top`, `right`, `bottom`, `left`	Pozicija elementa	px, %, em	`top: 10px;`
`overflow`	Upravljanje preljevom sadržaja	visible, hidden, scroll, auto	`overflow: hidden;`
`z-index`	Redoslijed prikazivanja elemenata	brojčana vrijednost	`z-index: 10;`
`opacity`	Prozirnost elementa	vrijednost od 0 do 1	`opacity: 0.5;`
`box-shadow`	Sjenka okvira	x-offset y-offset blur-radius spread-radius boja	`box-shadow: 0 4px 8px rgba(0,0,0,0.1);`
`text-shadow`	Sjenka teksta	x-offset y-offset blur-radius boja	`text-shadow: 1px 1px 2px black;`
`flex-direction`	Smjer fleksibilnog kontejnera	row, row-reverse, column, column-reverse	`flex-direction: row;`
`justify-content`	Poravnanje stavki duž glavne osi	flex-start, flex-end, center, space-between, space-around	`justify-content: center;`
`align-items`	Poravnanje stavki duž poprečne osi	flex-start, flex-end, center, baseline, stretch	`align-items: center;`
`transition`	Definira glatku animaciju prijelaza između različitih stilova	svojstvo, trajanje, kašnjenje, funkcija	`transition: width 0.3s ease-in-out;`

Postoje i CSS pseudo-klase (***Pseudo-classes***) kao što su `:hover`, `:active`, `:focus` i druge koje se mogu nadodati na CSS klase. Mogu se koristiti za primjenu stilova na elemente u određenim stanjima ili uvjetima.

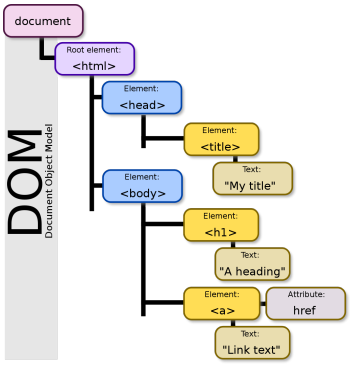
Tablica prikazuje neke od najčešće korištenih CSS pseudo-klasa:

Pseudo-klasa	Opis	Primjer
`:hover`	Primjenjuje stil kada korisnik prelazi mišem preko elementa.	`css a:hover { color: blue; }`
`:active`	Primjenjuje stil kada je element aktiviran (npr. kliknut).	`css button:active { background-color: green; }`
`:focus`	Primjenjuje stil kada je element u fokusu (npr. odabran tipkovnicom).	`css input:focus { border-color: red; }`
`:visited`	Primjenjuje stil na posjećene linkove.	`css a:visited { color: purple; }`
`:link`	Primjenjuje stil na neposjećene linkove.	`css a:link { color: blue; }`
`:first-child`	Primjenjuje stil na prvi dijete elementa.	`css p:first-child { font-weight: bold; }`
`:last-child`	Primjenjuje stil na zadnji dijete elementa.	`css p:last-child { font-style: italic; }`
`:nth-child(n)`	Primjenjuje stil na n-ti dijete elementa.	`css li:nth-child(2) { color: red; }`
`:not(selector)`	Primjenjuje stil na sve elemente koji ne odgovaraju zadanom selektoru.	`css p:not(.highlight) { color: grey; }`
`:checked`	Primjenjuje stil na odabrane (checked) elemente kao što su checkbox ili radio button.	`css input:checked { background-color: yellow; }`
`:disabled`	Primjenjuje stil na onemogućene (disabled) elemente.	`css input:disabled { background-color: lightgrey; }`
`:enabled`	Primjenjuje stil na omogućene (enabled) elemente.	`css input:enabled { background-color: white; }`
`:empty`	Primjenjuje stil na elemente koji nemaju djece (prazne elemente).	`css div:empty { display: none; }`

1. DOM manipulacija

Nakon stjecanja osnovnog razumijevanja JavaScript varijabli, funkcija, struktura i metoda, sada smo spremni za početak manipulacije **Document Object Model**, odnosno DOM-om, što uključuje dinamičko upravljanje HTML elementima i njihovim CSS svojstvima.

Document Object Model (DOM) je standard koji definira strukturu i način pristupa HTML dokumentima. DOM predstavlja HTML dokument kao stablo objekata, gdje svaki HTML element predstavlja objekt, a svaki atribut i sadržaj elementa predstavlja svojstvo tog objekta.



> Izvor: https://en.wikipedia.org/wiki/Document_Object_Model

> **Zašto je važno naučiti DOM manipulaciju?**
>
> - Većina današnjeg digitalnog poslovanja bazirana je upravno na web tehnologijama.
> - JavaScript je jedan od najpopularnijih jezika za web razvoj.
> - Statične web stranice postale su prošlost, danas su gotovo sve web stranice dinamične (interaktivne).
> - DOM manipulacija je ključna za stvaranje interaktivnih web stranica, samim time ih onda počinjemo nazivati i web aplikacijama.
> - Poznavanje DOM-a je ključno za razumijevanja rada svih modernih JavaScript biblioteka i okvira (React, Angular, Vue, jQuery...).
> - Poznavanje DOM-a je ključno za razvoj efikasnih i responzivnih korisničkih sučelja.

1.1 Osnovni DOM element

`Document` objekt je ključna komponenta u JavaScriptu koja predstavlja cijelu web stranicu u trenutnom pregledniku. On omogućava pristupanje i manipulaciju svim elementima na stranici, kao i njihovim svojstvima i sadržaju. Olakšava dinamičko upravljanje sadržajem stranice, što je ključno za stvaranje interaktivnih i responzivnih korisničkih iskustava.

Možemo ga zamisliti kao korijenski čvor HTML dokumenta (slika iznad).

`document` objekt možemo referencirati direktno ili preko `window` objekta.

```
// Referenciranje document objekta
let doc = document;

// ili
let doc = window.document;
```

`document` objekt ima brojna svojstva i metode, mi ćemo u ovoj skripti baviti prvenstveno metodama, no to mogu biti i svojstva. Na primjer, `document.title` vraća naslov stranice, a `document.URL` vraća URL stranice.

```
console.log(document.title); // Ispisuje naslov stranice
console.log(document.URL); // Ispisuje URL stranice
```

1.2 Dohvaćanje DOM elemenata

Kako bismo uopće mogli raditi sa DOM elementima prvo ih moramo "dohvatiti".

Imamo zadan sljedeći HTML gdje želimo izvući vrijednosti iz pojedinih elemenata.

```
<p id="mojID">Ja sam paragraf 1</p> <!-- ID -->
<input type="text" name="ime" id="form_ime" class="mojInput" value="Marko" /> <!-- ID, class
-->
<input type="text" name="prezime" id="form_prezime" class="mojInput" value="Marić" /> <!--
ID, class -->
<p class="mojaKlasa">Ja sam paragraf 2</p> <!-- class -->
<span class="mojaKlasa">Ja sam span</span> <!-- class -->
<p>Ja sam paragraf 3</p>
```

HTML elementi se mogu dohvatiti na sljedeće načine:

Metoda	Objašnjenje	Sintaksa	Primjer
`getElementById(x)`	Vraća prvi element po jedinstvenom `Id`-u.	`document.getElementById(x)`	`document.getElementById("mojID")`
`getElementsByTagName(x)`	Vraća sve elemente po HTML `tagu`-u.	`document.getElementsByTagName(x)`	`document.getElementsByTagName("p")`
`getElementsByClassName(x)`	Vraća sve elemente po klasi/klasama ili kombinaciji HTML `tag`-a i klase (`class`).	`document.getElementsByClassName(x)`	`document.getElementsByClassName("mojaKlasa")`
`getElementsByName(x)`	Vraća sve elemente po imenu.	`document.getElementsByName(x)`	`document.getElementsByName("ime")`

`querySelector(x)`	Vraća **prvi element** koji odgovara određenom selektoru ili grupi selektora. Ako nema pronađenih podudaranja, vraća `null`.	`document.querySelector(x)`	`document.querySelector("#mojID")`
			`document.querySelector(".mojaKlasa")`
			`document.querySelector("p")`
			`document.querySelector("input[name='ime']")`

`querySelectorAll(x)`	Vraća **sve elemente** koji odgovaraju određenom selektoru ili grupi selektora. Ako nema pronađenih podudaranja, vraća `null`.	`document.querySelectorAll(x)`	`document.querySelectorAll("#mojID")`
			`document.querySelectorAll(".mojaKlasa")`
			`document.querySelectorAll("p")`
			`document.querySelectorAll("input[name='ime']")`

U sljedećem JavaScript kôdu možete vidjeti primjere dohvaćanja elemenata koristeći metode iz tablice.

```
// Dohvaćanje prvog DOM elementa po ID-u
```

```
const mojDiv = document.getElementById('mojID');
console.log("Dohvaćen po ID-u: " + mojDiv.innerHTML);

// Dohvaćanje DOM elemenata po tagu
const paragrafi = document.getElementsByTagName('p');
for (let paragraf of paragrafi){
    console.log("Dohvaćen po tagu: " + paragraf.innerHTML);
}

// Dohvaćanje DOM elemenata po klasi
const sveKlase = document.getElementsByClassName('mojaKlasa');
for (let klasa of sveKlase){
    console.log("Dohvaćen po klasi: " + klasa.innerHTML);
}

// Dohvaćanje DOM elemenata po imenu
const svaImena = document.getElementsByName('ime');
for (let ime of svaImena){
    console.log("Dohvaćen po imenu: " + ime.value);
}

// Dohvaćanje prvog DOM elemenata koristeći querySelector
const query1 = document.querySelector('#form_prezime');
console.log("Dohvaćen koristeći querySelector: " + query1.value);
const query2 = document.querySelector('span');
console.log("Dohvaćen koristeći querySelector: " + query2.innerHTML);
const query3 = document.querySelector('.mojaKlasa');
console.log("Dohvaćen koristeći querySelector: " + query3.innerHTML);
const query4 = document.querySelector("input[name='ime']");
console.log("Dohvaćen koristeći querySelector: " + query4.value);

// Dohvaćanje svih DOM elemenata koristeći querySelectorAll
const queryAll = document.querySelectorAll('p');
for (let query of queryAll){
    console.log("Dohvaćen koristeći querySelectorAll: " + query.innerHTML);
}
```

> `querySelector` uvijek prvo pretražuje po `tag`-u, za pretraživanje po `id`-u treba koristiti oznaku `#`.
> za pretraživanje po klasi treba koristiti `.` dok za pretraživanje po imenu ili drugim atributima prvo treba staviti ime `tag`-a pa unutar uglatih zagrada pretragu `[atribut = vrijednost]`

Primjer 1. - Dohvaćanje elemenata

Za zadani HTML kôd, treba dohvatiti `<input>` s vrijednošću **TOČNO**. Ne smijemo koristiti `id` atribut i naknadno mijenjati HTML.

```
<div class="moja-forma glavni2">
    <span name="ime">KRIVO</span>
    <span name="prezime">KRIVO</span>
</div>
<span class="moja-forma glavni">
    <span name="ime">KRIVO</span>
    <span name="prezime">KRIVO</span>
```

```
</span>
<div class="moja-forma glavni">
    <span name="prezime">KRIVO</span>
    <span name="ime">TOČNO</span>
    <i name="ime">KRIVO</i>
</div>
<div class="druga-forma glavni">
    <span name="prezime">KRIVO</span>
    <span name="ime">KRIVO</span>
</div>
```

Treba dohvatiti prvi `<span>` element s imenom `"ime"`: (`<span name="ime"/>`), a koji se nalazi unutar `<div>` elementa s klasama `"moja-forma glavni"` (`<div class="moja-forma glavni">`)."

> Rješenje:

```
const query = document.querySelector("div.moja-forma.glavni span[name='ime']"); // Pogledati definiciju selektora u tablici
console.log(query.innerHTML)
```

1.3 Svojstva DOM elemenata

DOM Elementi imaju mnogo svojstava, od kojih smo već neka koristili neka za dohvaćanje sadržaja elemenata poput `innerHTML`.

Možemo ih podijeliti u svojstva za:

- dohvaćanje i postavljanje atributa,
- dohvaćanje sadržaja,
- dohvaćanje stilova,
- dohvaćanje djece i susjeda.

U sljedećoj tablici su prikazana neka od bitnijih svojstava:

Svojstvo	Objašnjenje	Sintaksa
`id`	Vraća ili postavlja vrijednost `id` atributa elementa.	`element.id`
`tagName`	Vraća ime `tag`-a elementa velikim slovima.	`element.tagName`
`classList`	Vraća kolekciju klasa elementa.	`element.classList`
`className`	Vraća ili postavlja vrijednost `class` atributa elementa.	`element.className`
`innerHTML`	Vraća ili mijenja HTML sadržaj unutar elementa.	`element.innerHTML`
`outerHTML`	Vraća HTML elementa, uključujući sam element i njegov sadržaj.	`element.outerHTML`
`attributes`	Vraća kolekciju svih atributa elementa.	`element.attributes`
`style`	Vraća `style` atribut elementa.	`element.style`
`childElementCount`	Vraća broj direktne djece elementa.	`element.childElementCount`
`children`	Vraća kolekciju direktne djece elementa.	`element.children`
`firstElementChild`	Vraća prvo direktno dijete elementa.	`element.firstElementChild`
`lastElementChild`	Vraća posljednje direktno dijete elementa.	`element.lastElementChild`
`nextElementSibling`	Vraća sljedeći element nakon `element` u roditeljskom elementu.	`element.nextElementSibling`
`previousElementSibling`	Vraća element prije `element` u roditeljskom elementu.	`element.previousElementSibling`

Primjer 2 - Pristupanje sadržaju, `id`-u, `tag`-u, atributima i klasama elementa

```
<div class="text-5xl">
    Hi!
</div>
<div id="mojID" class="text-5xl h-16 w-36 overflow-scroll">
    Hello, World!
</div>
```

```
const element = document.getElementById('mojID');

console.log(element.innerHTML); // Output: " Hello, World! "
console.log(element.outerHTML); // Output: "<div id="mojID" class="text-5xl h-16 w-36
overflow-scroll"> Hello, World! </div>"

console.log(element.id); // Output: "mojID"
console.log(element.tagName); // Output: "DIV"

for (const attr of element.attributes) {
  console.log(`attr: ${attr.name} -> ${attr.value}`);
  // Output: "attr: id -> mojID"
  // Output: "attr: class -> text-5xl h-16 w-36 overflow-scroll"
}
```

Međutim, kolekciju klasa možemo dohvatiti preko `className` ili `classList` svojstva. `className` vraća `string` svih klasa dok `classList` vraća `DOMTokenList` objekt koji omogućava korištenje `forEach` petlje.

```
// Dohvaćanje klasa preko className
console.log(element.className); // Output: "text-5xl h-16 w-36 overflow-scroll"

// Dohvaćanje klasa preko classList
element.classList.forEach( klasa => {
  console.log(`class: ${klasa}`);
  // Output: class: -> text-5xl
  // Output: class: -> h-16
  // Output: class: -> w-36
  // Output: class: -> overflow-scroll
})
```

> `attributes` svojstvo **ne vraća polje objekata već objekt objekata** kao povratnu vrijednost, tako da je za iteraciju najbolje koristiti `for of` petlju. Međutim, `classList` vraća `DOMTokenList` koja omogućava korištenje `forEach` petlje.
>
> `attributes` vraća kolekciju tipa `NamedNodeMap` koja nema mogućnost korištenja `forEach` petlje.

Elementu možemo direktno mijenjati ili dodati `id` koristeći `id` svojstvo.

Možemo dohvatiti prvi `div` element s tekstom `"Hi!"` i dodati mu `id`: `"prviDiv"`.

```
const element = document.querySelector('div')
console.log(element.outerHTML); //Output: "<div class="text-5xl"> Hi! </div>"
element.id = "prviDiv" // Dodavanje ID-a
console.log(element.outerHTML); //Output: "<div class="text-5xl" id="prviDiv"> Hi! </div>"
```

Sadržaj mu možemo promijeniti preko `innerHTML` svojstva.

```
element.innerHTML = " Pozdrav! " //Mjenjanje sadržaja
console.log(element.outerHTML); //Output: "<div class="text-5xl" id="prviDiv"> Pozdrav!
</div>"
```

> Mijenjanjem sadržaja preko `outerHTML` svojstva možemo prekrižiti cijeli element pa nije pametno to koristiti.
>
> Međutim, za navedeno postoje metode koje ćemo proći u poglavlju `Dodavanje i brisanje DOM elemenata`

Vježba 1

EduCoder šifra: `funte_u_eure`

Pronašli smo idealni web shop u Engleskoj, međutim sve cijene su prikazane u funtama, a stranica nema ugrađenu konverziju valuta. Želimo da nam se automatski prikažu sve cijene u valuti eura. Zadatak nam je malo "hakirati" ovaj web shop bez da mijenjamo HTML kôd.

```
<div class="item">
    <b>-</b>
    <span> Laptop </span>
    <u>1200</u>
    <span class="symbol">£</span>
```

```
</div>
<div class="item">
    <b>-</b>
    <span> PC </span>
    <u>1800</u>
    <span class="symbol">£</span>
</div>
<div class="item">
    <b>-</b>
    <span> Mouse & Keyboard </span>
    <u>200</u>
    <span class="symbol">£</span>
</div>
```

- Napišite funkciju `azurirajSimbol(klasa, noviSimbol)` koja će za danu klasu promijeniti unutarnji sadržaj svih klasa na novi sadržaj.
- Napišite funkciju `azurirajCijenu(tag)` koja će za dani `tag` napraviti konverziju unutarnjeg sadržaja (cijena) svih `tag`-ova iz cijene u funtama u cijenu u eurima, zaokruženo na dvije decimale.
- Devizni tečaj: `1£ = 1.16547€`

✅Rezultat:

- Laptop 1398.56 €
- PC 2097.85 €
- Mouse & Keyboard 233.09 €

> Rješenje:

```
function azurirajSadrzaj(klasa, noviSimbol) {
  const query = document.querySelectorAll("."+klasa)
  for (let element of query) {
    element.innerHTML=noviSimbol; // Promjena sadržaja elementa
  }
}
function azurirajCijenu(tag) {
  const query = document.querySelectorAll(tag)
  for (let element of query) {
    element.innerHTML=(Number.parseFloat(element.innerHTML)*1.16547).toFixed(2); // Promjena sadržaja elementa (sadržaj dohvaćen s innerHTML je tipa string pa ga treba pretvoriti u broj, zato koristimo parseFloat metodu)
  }
}
azurirajSadrzaj("symbol", "€"); // Promjena simbola svugdje gdje imamo klasu "symbol"
azurirajCijenu("u"); // Ažuriraj cijenu svugdje gdje imamo tag "u"
```

Primjer 3 - Manipulacija klasama

> Ako DOM elementu želimo direktno mijenjati ili dodati klasu `class` onda koristimo `className` svojstvo, ne `classList` svojstvo.

Primjerice, želimo ažurirati klasu elementa s ID-om `prviDiv` iz `text-5xl` u `text-6xl`

```
<div id="prviDiv" class="text-5xl">
    Hi!
</div>
```

```
const element = document.querySelector('#prviDiv')
console.log(element.outerHTML); //Output: "<div class="text-5xl id="prviDiv""> Hi! </div>"
element.className = "text-6xl" // Promjena klase elementa
console.log(element.outerHTML); //Output: "<div class="text-6xl" id="prviDiv"> Hi! </div>"
```

> `className` služi za postavljanje i dohvaćanje cijelog atributa klase odabranog elementa. Za dodavanje dodavanje, brisanje, promjenu i provjeru pojedine klase bolje je koristiti `classList` svojstvo.

Nad svojstvom `classList` mogu se pozvati dodatne metode koje nam olakšavaju manipulaciju klasom (`class`) elementa.

Metoda	Objašnjenje	Sintaksa
`add(className1, className2, ...)`	Dodaje jednu ili više CSS klasa elementu	`element.classList.add(x);`
`contains(className)`	Provjerava sadrži li element određenu CSS klasu	`element.classList.contains(x);`
`remove(className1, className2, ...)`	Uklanja jednu ili više CSS klasa iz elementa	`element.classList.remove(x);`
`replace(oldClassName, newClassName)`	Zamjenjuje postojeću CSS klasu s novom CSS klasom	`element.classList.replace(x, y);`
`toggle(className)`	Dodaje CSS klasu ako ju element nema/uklanja ako ju ima	`element.classList.toggle(x);`

```
<div id="mojID" class="text-6xl">
    Hello, World!
</div>
```

```
const element = document.querySelector('#mojID')
element.classList.add('italic'); // Dodajemo klasu "italic"
console.log(element.className); //Output: "text-6xl italic"

console.log(element.classList.contains('text-6xl')); // Output: true
element.classList.remove('text-6xl'); // Uklanjamo klasu "text-6xl"
console.log(element.className); //Output: "italic"

element.classList.replace('italic', 'underline'); // Zamjenjujemo klasu "italic" s
"underline"
console.log(element.className); //Output: "underline"
element.classList.toggle('font-bold') // Dodajemo klasu "font-bold" jer je nema
console.log(element.className); //Output: "underline font-bold"
```

Vježba 2

EduCoder šifra: `books`

Naišli smo na staru web stranicu s bibliotekom knjiga. Na stranici su prikazane knjige u tablici. Želimo promijeniti stil tablice i ćelija kako bi bila nam bila preglednija.

Zadan je sljedeći CSS i HTML kôd:

```
<style>
    .slika {
        overflow: hidden;
        border-radius: 100%;
    }
    .tekst {
        color: black;
        font-size: 18x;
    }
    .tablica {
        width: 100%;
        background-color: white;
    }
    .celija {
        border: 1px solid #dddddd;
        text-align: left;
        padding: 8px;
    }
    .naslov {
        background-color: #00000065;
        font-size: 20px;
    }
    .velika {
        background-color: #4cb05065;
    }
    .broj-stranica {
        font-weight: bold;
    }
</style>
<table class="slika tekst">
    <tr>
        <th>Naslov</th>
        <th>Autor</th>
        <th>Broj stranica</th>
    </tr>
    <tr>
        <td>Harry Potter and the Philosopher's Stone</td>
        <td>J. K. Rowling</td>
        <td class="broj-stranica">500</td>
    </tr>
    <tr>
        <td>The Lord of the Rings</td>
        <td>J. R. R. Tolkien</td>
        <td class="broj-stranica">1077</td>
    </tr>
    <tr>
        <td>Don Quixote</td>
        <td>Miguel de Cervantes</td>
        <td class="broj-stranica">120</td>
    </tr>
</table>
```

Koristeći `querySelector` i `classList` metode dodajte "tablica", "celija" i "naslov" na odgovarajuće elemente, ćeliji s najvećim brojem stranica dodajte klasu "velika".

Naslov	Autor	Broj stranica
Harry Potter and the Philosopher's Stone	J. K. Rowling	**500**
The Lord of the Rings	J. R. R. Tolkien	**1077**
Don Quixote	Miguel de Cervantes	**120**

> Rješenje:

```
document.querySelector('table').classList.replace('slika', 'tablica'); // Zamjenjujemo klasu "slika" s "tablica"
document.querySelectorAll('th, td').forEach(element => { // Dodajemo klasu "celija" na sve <th> i <td> elemente
    element.classList.add('celija');
});
document.querySelectorAll('th').forEach(element => { // Dodajemo klasu "naslov" na sve <th> elemente
    element.classList.add('naslov');
});

let maxBrojStranica = 0;
let maxCelijaIndex = -1;
let query = document.querySelectorAll('.broj-stranica'); // Dohvaćamo sve elemente s klasom "broj-stranica"
query.forEach((celija, index) => { // Pronalazimo najveći broj stranica
    const brojStranica = parseInt(celija.innerHTML); // Pretvaramo sadržaj u broj
    if (brojStranica > maxBrojStranica) { // Ako je trenutni broj stranica veći od maksimalnog
        maxBrojStranica = brojStranica; // Postavljamo novi maksimalni broj stranica
        maxCelijaIndex = index; // Postavljamo index najvećeg broja stranica
    }
});
query[maxCelijaIndex].classList.add('velika')  // Dodajemo klasu "velika" na ćeliju s najvećim brojem stranica
```

Primjer 4 - Dohvaćanje `child` i `sibling` elemenata.

Naučili smo dohvaćati elemente koristeći `querySelector` i `getElements` metode. Međutim, ponekad želimo dohvatiti djecu ili susjede određenog elementa. Za to možemo koristiti sljedeća svojstva:
`childElementCount`,
`children`,
`lastElementChild`,
`nextElementSibling`,
`previousElementSibling`

Imamo sljedeći HTML kôd:

```
<div>
    Hi!
</div>
<div id="mojID">
    <span>Hello</span>
    <span>,</span>
    <span>World!</span>
</div>
<div>
    Bye!
</div>
```

Upotrijebit ćemo `querySelector` metodu za dohvaćanje elementa s ID-om `mojID` te potom iskoristiti navedena svojstva za dohvaćanje djece i susjeda tog elementa.

```
const element = document.querySelector('#mojID')
// Dohvaćanje broja djece elementa
console.log(element.childElementCount); // Output: 3

// Dohvaćanje djece elementa
for (const child of element.children) {
  console.log(`child: ${child.outerHTML}`);
  // Output: "child: <span>Hello</span>"
  // Output: "child: <span>,</span>"
  // Output: "child: <span>World</span>"
}

// Ili tradicionalnom for petljom
for (let i = 0; i < element.childElementCount; i++) {
  console.log(`child: ${element.children.item(i).outerHTML}`);
  // Output: "child: <span>Hello</span>"
  // Output: "child: <span>,</span>"
  // Output: "child: <span>World</span>"
}

console.log(element.firstElementChild.outerHTML); // Output: <span>Hello</span>
console.log(element.lastElementChild.outerHTML); // Output: <span>World</span>

console.log(element.nextElementSibling.outerHTML); // Output: "<div class="text-5xl"> Bye!
</div>"
console.log(element.previousElementSibling.outerHTML); // Output: "<div class="text-5xl"> Hi!
</div>"
```

> `children` svojstvo vraća `HTMLCollection` nad kojime se ne može pozvati `forEach` metoda. Međutim, imamo svojstvo `childElementCount` koje nam vraća broj djece elementa te omogućava iteraciju kroz djecu koristeći tradicionalnu `for` petlju i pristupanje pojedinom djetetu preko indexa metodom `item()`

Vježba 3

EduCoder šifra: `web_scraping`

Radimo na projektu analize podataka, gdje trebamo prikupiti informacije s više web stranica škola. Nažalost, te stranice nemaju svoj API za dohvaćanje podataka.
Možemo koristiti web scraping, tehniku koja se koristi za ekstrakciju podataka s web stranica. U te svrhe moramo dobro poznavati HTML strukturu stranice, kao i manipulaciju DOM elementima.

Zadan je sljedeći HTML kôd:

```
<div id="studenti">
    <div>
        <b>Ivo</b>
        <b>Ivić</b>
        <u class="email">ivoivic@skole.hr</u>
        <span>3</span>
    </div>
    <div>
        <b>Ana</b>
        <b>Anić</b>
        <span>5</span>
    </div>
    <div>
        <b>Maja</b>
        <b>Majić</b>
        <u class="email">majamajic@skole.hr</u>
        <span>none</span>
    </div>
    <div>
        <b>Marko</b>
        <b>Marić</b>
        <u class="email">markomaric@skole.hr</u>
        <span>1</span>
    </div>
</div>
```

Zadan je konstruktor `Student`:

```
function Student(ime, prezime, email, ocjena, opisnaOcjena) {
    this.ime = ime;
    this.prezime = prezime;
    this.email = email;
    this.ocjena = ocjena;
    this.opisnaOcjena = opisnaOcjena;
    this.oStudentu = () => console.log(`${this.ime} ${this.prezime} s emailom ${this.email}
ima ocjenu ${this.opisnaOcjena}`)
}
```

1. Napišite funkciju `dodajStudente(id, poljeStudenata)` koja:

- Za svako dijete elementa s danim `id`-em
    - Ekstrahira podatke o *imenu*, *prezimenu*, *emailu* i *ocjeni* koristeći samo `firstElementChild`, `lastElementChild`, `nextElementSibling`, `previousElementSibling` i `classList` metode
    - Ako nedostaje *email*, postavlja ga na `"nema podatka"`

    - Pretvara ocjenu u format: od "odličan" do "nedovoljan" za ocjene od `5` do `1`, ili "nema ocjenu" za ostalo
    - Dodaje svakog studenta u polje studenti i vraća novoizgrađeno polje
2. Spremite sve studente koji imaju ocjenu u polje `filtriraniStudenti`.

- Za svakog studenta koji ima ocjenu, pozovite metodu `oStudentu()` (metoda već definirana u konstruktoru `Student`)

Napišite funkciju `prosjekStudenata(poljeStudenata)` koja vraća prosjek studenata:

- spremite u `sumaOcjena` varijablu sumu svih ocjena studenata koristeći `reduce()` metodu
- spremite u `prosjek` varijablu prosjek ocjena zaokruženo na dvije decimale

Konstruktor `Student` možete ažurirat ako je potrebno s dodatnim metodama ili ažurirat metodu `oStudentu()`.

✅Rezultat:

```
dodajStudente('studenti');

// Output: "Ivo Ivić s emailom "ivoivic@gmail.com" ima ocjenu dobar"
// Output: "Ana Anić s emailom "nema podatka" ima ocjenu odličan"
// Output: "Marko Marić s emailom "markomaric@gmail.com" ima ocjenu nedovoljan"

console.log(`Prosjek ocjena studenata: ${prosjekStudenata(filtriraniStudenti)}`);

// Output: "Prosjek ocjena studenata: 3.00"
```

> Rješenje:

```
function dodajStudente(html_element_id) {
    const elementiStudenata = document.getElementById(html_element_id).children;
    const poljeStudenata = [];
    // Iteriramo tradicionalnom for petljom kroz svu djecu elementa s danim ID-em
(html_element_id)
    // Kôd počiva na pretpostavci da su svi elementi u polju redom: ime, prezime, email,
ocjena
    for (let i = 0; i < elementiStudenata.length; i++) {
        const student = elementiStudenata[i];

        const imeElement= student.firstElementChild;
        const ime = imeElement.innerHTML;
        const prezime = imeElement.nextElementSibling.innerHTML;

        const ocjenaElement = student.lastElementChild;
        const ocjena = student.lastElementChild.innerHTML;

        let opisnaOcjena = "nema ocjenu";
        let email = "nema podatka";

        if (ocjenaElement.previousElementSibling.classList.contains('email'))
          email = ocjenaElement.previousElementSibling.innerHTML;

        switch (ocjena) {
```

```
            case "5":
                opisnaOcjena = 'odličan';
                break;
            case "4":
                opisnaOcjena = 'vrlo dobar';
                break;
            case "3":
                opisnaOcjena = 'dobar';
                break;
            case "2":
                opisnaOcjena = 'dovoljan';
                break;
            case "1":
                opisnaOcjena = 'nedovoljan';
                break;
            default:
                opisnaOcjena = 'nema ocjenu';
                break;
        }
        // Pozivanjem konstruktora stvaramo novi objekt s ekstrahiranim podacima
        poljeStudenata.push(new Student(ime, prezime, email, ocjena, opisnaOcjena));
    }
    return poljeStudenata;
}

// Pozivamo funkciju za ID "studenti"
let studenti = dodajStudente('studenti');

// U polje filtriraniStudenti spremamo sve studente koji imaju ispravnu ocjenu
const filtriraniStudenti = studenti.filter(student => student.opisnaOcjena != "nema ocjenu");
filtriraniStudenti.forEach(student => student.oStudentu());

function prosjekStudenata(poljeStudenata) {
  let sumaOcjena = poljeStudenata.reduce((total, student) =>
total+Number.parseInt(student.ocjena), 0);
  let prosjek = (sumaOcjena/poljeStudenata.length).toFixed(2);
  return prosjek;
}

console.log(`Prosjek ocjena studenata: ${prosjekStudenata(filtriraniStudenti)}`);
```

1.4 Dodavanje i brisanje DOM elemenata

Dodavanje i brisanje elemenata omogućuje dinamičke izmjene stranice temeljem korisničkih akcija ili događaja.

Dosad smo naučili kako da dohvaćamo i mijenjamo elemente, međutim dodavanje novih elemenata je dosta nezgodno koristeći svojstvo `innerHTML`. Iz tog razloga postoje i metode za dodavanja, umetanje i brisanje elemenata:

Metoda	Objašnjenje	Sintaksa
`createElement()`	Stvara novi HTML element prema definiranom `tag`-u	`document.createElement(tagName)`
`append()`	Dodaje element(e) (`newElement`) kao posljednje dijete.	`element.append(child1, child2, ...)`
`prepend()`	Dodaje element(e) (`newElement`) kao prvo dijete.	`element.prepend(child1, child2, ...)`
`before()`	Dodaje element(e) (`newElement`) ispred odabranog elementa.	`element.before(newElement)`
`after()`	Dodaje element(e) (`newElement`) iza odabranog elementa.	`element.after(newElement)`
`insertAdjacentElement()`	Dodaje novi element u odabrani element, na zadanu poziciju (`position`).	`element.insertAdjacentElement(position, newElement)`
`insertAdjacentHTML()`	Dodaje HTML tekst u odabrani element, na zadanu poziciju (`position`).	`element.insertAdjacentHTML(position, html)`
`remove()`	Uklanja element iz DOM-a.	`element.remove()`
`replaceWith()`	Zamjenjuje odabrani element novim elementom (`newElement`).	`element.replaceWith(newElement)`

U kôdu za `insertAdjacentElement()` i `insertAdjacentHTML()` koristimo atribut `position` koji označava gdje se sadržaj dodaje, mora biti postavljen na jednu od sljedećih vrijednosti:

- `beforebegin`: prije elementa
- `afterbegin`: unutar elementa, prije njegovog prvog djeteta
- `beforeend`: unutar elementa, nakon njegovog posljednjeg djeteta
- `afterend`: nakon elementa

Primjer 5 - Stvaranje, dodavanje, brisanje i izmjena DOM elemenata

```
<style>
    #mojID {
        background-color: lightgray;
    }
    .hello-world {
        background-color: darkGray;
    }
</style>
<div id="mojID">
    <div class="hello-world">
        Hello, World!
    </div>
</div>
```

Dodavanje novih elemenata koristeći metode: `append()`, `prepend()`, `before()`, `after()`

```
const mojElement = document.getElementById('mojID');

// Stvaramo nove elemente
const divAppend = document.createElement('div')
const divPreppend = document.createElement('div')
const divAfter = document.createElement('div')
const divBefore = document.createElement('div')

// Postavljamo sadržaj novih elemenata
```

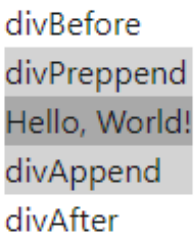
```
divAppend.innerHTML = 'divAppend'
divPreppend.innerHTML = 'divPreppend'
divAfter.innerHTML = 'divAfter'
divBefore.innerHTML = 'divBefore'

// Raspoređujemo nove elemente
mojElement.append(divAppend);
mojElement.prepend(divPreppend);
mojElement.after(divAfter);
mojElement.before(divBefore);
```

Ili s metodom: `insertAdjacentElement()`

```
const mojElement = document.getElementById('mojID');

// Stvaramo nove elemente
const divAppend = document.createElement('div')
const divPreppend = document.createElement('div')
const divAfter = document.createElement('div')
const divBefore = document.createElement('div')

// Postavljamo sadržaj novih elemenata
divAppend.innerHTML = 'divAppend'
divPreppend.innerHTML = 'divPreppend'
divAfter.innerHTML = 'divAfter'
divBefore.innerHTML = 'divBefore'

// Raspoređujemo nove elemente
mojElement.insertAdjacentElement("beforebegin", divBefore);
mojElement.insertAdjacentElement("afterbegin", divPreppend);
mojElement.insertAdjacentElement("beforeend", divAppend);
mojElement.insertAdjacentElement("afterend", divAfter);
```

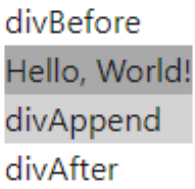
Ili pak s metodom: `insertAdjacentHTML()`

```
const mojElement = document.getElementById('mojID');

// Stvaramo nove elemente u obliku HTML stringa
const divAppend = "<div> divAppend </div>"
const divPreppend = "<div> divPreppend </div>"
const divAfter = "<div> divAfter </div>"
const divBefore = "<div> divBefore </div>"

// Raspoređujemo nove elemente
mojElement.insertAdjacentHTML("beforebegin", divBefore);
mojElement.insertAdjacentHTML("afterbegin", divPreppend);
mojElement.insertAdjacentHTML("beforeend", divAppend);
mojElement.insertAdjacentHTML("afterend", divAfter);
```



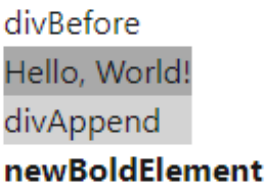
Brisanje radimo jednostavno koristeći metodu `remove()`. Na primjer, brisanje prvog `child` elementa, `div`-a gdje je `id` = `"mojDiv"`

```
const elementZaBrisanje = mojElement.firstElementChild;
elementZaBrisanje.remove()
```



Izmjena `div` elementa nakon `div`-a gdje je `id` = `"mojDiv"`:

```
const elementZaMjenjanje = mojElement.nextElementSibling;

const newBoldElement = document.createElement('b')
newBoldElement.innerHTML = "newBoldElement"

elementZaMjenjanje.replaceWith(newBoldElement)
```



Vježba 4

EduCoder šifra: `html_from_object`

U web programiranju često ćemo morati grafički prikazati podatke iz dinamičkih struktura i različitih izvora podataka. Ako se mijenjaju samo pojedinačne vrijednosti u strukturi, dovoljno je samo izmjeniti postojeće definirane HTML elemente. Međutim, ako se struktura mijenja, ili se povečava/smanjuje količina podataka, tada je potrebno dinamički i dodavati/brisati HTML elemente.

U praksi često za nas ove probleme rješavaju razvojni okviri za JavaScript, ili neka biblioteka, ali je dobro znati kako stvari funkcioniraju "ispod haube".

Zadan je sljedeći HTML/CSS kôd:

```
<style>
```

```
    #kupac {
        margin: 16px;
        padding: 16px;
        border-radius: 8px;
        background: #dedede;
        border: 1px solid darkgray;
        color: black;
        width: 400px;
        position: absolute;
        left: 50%;
        transform: translatex(-50%);
    }
    h1 {
        font-size: 32px;
        font-weight: bold;
        text-align: center;
    }
    hr {
        border: 1px solid black;
        margin: 12px 0px;
    }
    table {
        width: 100%;
    }
    th {
        text-align: left;
        border-bottom: 1px solid darkgray;
        padding-bottom: 4px;
    }
</style>
<div id="kupac">
</div>
```

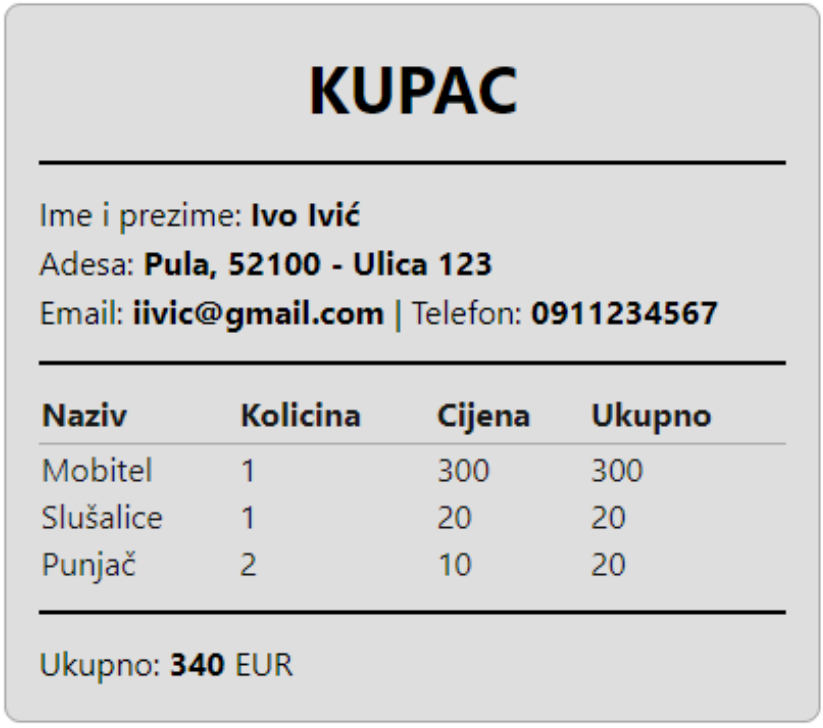
Zadan je objekt `kupac`:

```
let kupac = {
  ime: "Ivo",
  prezime: "Ivić",
  adresa: {
    ulica: "Ulica 123",
    grad: "Pula",
    postanskiBroj: "52100",
  },
  kontakt: {
    telefon: "0911234567",
    email: "iivic@gmail.com",
  },
  narudzbe: [
    {
      stavke: [
        {
          naziv: "Mobitel",
          kolicina: 1,
```

```
          cijena: 300,
        },
        {
          naziv: "Slušalice",
          kolicina: 1,
          cijena: 20,
        },
        {
          naziv: "Punjač",
          kolicina: 2,
          cijena: 10,
        },
      ],
      ukupnaCijena: function () {
        return this.stavke.reduce((ukupno, stavka) =>
ukupno+stavka.kolicina*stavka.cijena,0);
      },
      valuta: "eur",
    },
  ],
};
```

Objekt treba prikazati u obliku HTML-a koristeći metode za dodavanje elemenata.

✅Rezultat:



KUPAC

Ime i prezime: **Ivo Ivić**
Adesa: **Pula, 52100 - Ulica 123**
Email: **iivic@gmail.com** | Telefon: **0911234567**

Naziv	Kolicina	Cijena	Ukupno
Mobitel	1	300	300
Slušalice	1	20	20
Punjač	2	10	20

Ukupno: **340** EUR

> Rješenje:

```
// Dohvaćamo element s ID-om "kupac"
const divKupac = document.getElementById("kupac");
// Dodajemo naslov "KUPAC" u div element
divKupac.insertAdjacentHTML("beforeend", "<h1>KUPAC</h1>")
// Dodajemo horizontalnu liniju
divKupac.append(document.createElement("hr"));
```

Kada smo dodali naslov i liniju, slijedi ispisivanje informacije o kupcu. Stvarat ćemo nove `div` elemente te im dodavati sadržaj koristeći `innerHTML` svojstvo.

```
const divImePrezime = document.createElement("div");
divImePrezime.innerHTML = `Ime i prezime: <b> ${kupac.ime} ${kupac.prezime} </b`
divKupac.append(divImePrezime);

const divAdresa = document.createElement("div");
divAdresa.innerHTML = `Adesa: <b> ${kupac.adresa.grad}, ${kupac.adresa.postanskiBroj} -
${kupac.adresa.ulica}</b>`
divKupac.append(divAdresa);

const divKontakt = document.createElement("div");
divKontakt.innerHTML = `Email: <b> ${kupac.kontakt.email} </b> | Telefon: <b>
${kupac.kontakt.telefon} </b>`
divKupac.append(divKontakt);

// Dodajemo horizontalnu liniju
divKupac.append(document.createElement("hr"));
```

Nakon što smo dodali informacije o kupcu, slijedi dodavanje tablice s narudžbama. Stvorit ćemo prvo tablicu, a naslove stupaca dodati koristeći `insertAdjacentHTML()` metodu.

```
const tableNarudzbe = document.createElement("table");
const tableHeaders = document.createElement("tr");

tableHeaders.insertAdjacentHTML("beforeend", "<th>Naziv</th>")
tableHeaders.insertAdjacentHTML("beforeend", "<th>Kolicina</th>")
tableHeaders.insertAdjacentHTML("beforeend", "<th>Cijena</th>")
tableHeaders.insertAdjacentHTML("beforeend", "<th>Ukupno</th>")

tableNarudzbe.append(tableHeaders);
```

Nakon što smo dodali naslove stupaca, slijedi dodavanje redova tablice s podacima o narudžbama. Iterirat ćemo kroz sve stavke narudžbe i dodati ih u tablicu.

```
for (let stavka of kupac.narudzbe[0].stavke) {
  const tableRow = document.createElement("tr");
  tableRow.insertAdjacentHTML("beforeend", `<td>${stavka.naziv}</td>`)
  tableRow.insertAdjacentHTML("beforeend", `<td>${stavka.kolicina}</td>`)
  tableRow.insertAdjacentHTML("beforeend", `<td>${stavka.cijena}</td>`)
  tableRow.insertAdjacentHTML("beforeend", `<td>${stavka.cijena*stavka.kolicina}</td>`)
  tableNarudzbe.append(tableRow);
}

divKupac.append(tableNarudzbe);
// Još jedna horizontalna linija
divKupac.append(document.createElement("hr"));
```

Na kraju, dodajemo boldano ukupnu cijenu narudžbe.

```
const divUkupno = document.createElement("div");
divUkupno.innerHTML = `Ukupno: <b> ${kupac.narudzbe[0].ukupnaCijena()} </b>
${kupac.narudzbe[0].valuta.toUpperCase()}`
divKupac.append(divUkupno);

console.log(divKupac.outerHTML);
```

1.4 DOM events

DOM događaji (**eng. DOM events**) omogućuju JavaScriptu da reagira na korisničke akcije kao što su klikovi mišem, različite kretnje mišem, unos na tipkovnici i sl. Događaj se na `element` dodaje metodom `addEventListener(event, callbackFn)`.

- `callbackFn`: callback koji prima argument `event` a koji se odnosi na pozvani događaj. Da bi se moglo pristupati elementu nad kojim se pozvao `event`, koristi se svojstvo `target`.

Sintaksa ove callback funkcije u metodi `addEventListener` je sljedeća:

```
const element = document.querySelector('#elementID');

function callbackFn(event) {

    // Prevencija defaultne akcije (npr. spriječavanje slanja forme unutar <form> elementa)
    event.preventDefault();

    // Dohvaćanje svojstava: target, type, itd (najčešće se dohvaća target)
    const target = event.target;
    const eventType = event.type;

    // Neka akcija koja se izvršava kada se događaj aktivira
    console.log(`Event type: ${eventType}`);
    console.log(`Event target: ${target}`);

    // Primjer: change the background color of the element
    target.style.backgroundColor = 'yellow';
}
```

Uzmimo primjer gdje želimo promijeniti boju pozadine elementa kada se klikne na njega:

```
<button id="btn">Klikni me</button>
```

Prvo dohvaćamo element na koji želimo dodati događaj:

```
const btn = document.getElementById("btn");
```

Zatim dodajemo događaj klikanja na taj element:

```
// 1. način
btn.addEventListener("click", function (event) {
    console.log(event.target.outerHTML)
});
// Output: "<button id="btn">Klikni me</button>"
```

`callbackFn` se može pisati i kao arrow funkcija:

```
// 2. način
btn.addEventListener("click", (event) => {
    console.log(event.target.outerHTML)
});

// 3. način (kratki zapis jer je samo jedan argument)
btn.addEventListener("click", event => {
    console.log(event.target.outerHTML)
});

// 4. način (kratki zapis jer je samo jedan argument i jedna naredba)
btn.addEventListener("click", event => console.log(event.target.outerHTML));

// 5. način (callback funkcija je definirana izvan metode addEventListener)
const ispis = (event) => console.log(event.target.outerHTML);
btn.addEventListener("click", ispis);
```

`event` argument sadrži informacije o događaju koji se dogodio.
Napament ih nema smisla učiti (osim najčešće korištenih poput `click`, `input`, `focus`...) jer se mogu lako pronaći na internetu, ovisno o potrebi.

Ima ih mnogo, neki od najčešće korištenih na webu su:

Metoda	Objašnjenje	Sintaksa	Primjer
click	Poziva se kada se klikne mišem na element.	`element.addEventListener('click', function() {})`	`button.addEventListener('click', function() { console.log('Kliknuto!'); })`
dblclick	Poziva se kada se dvaput klikne na element mišem.	`element.addEventListener('dblclick', function() {})`	`image.addEventListener('dblclick', function() { console.log('Dvaput kliknuto!'); })`
focus	Poziva se kada element dobije fokus.	`element.addEventListener('focus', function() {})`	`input.addEventListener('focus', function() { console.log('Fokusiranje!'); })`
focusin	Poziva se kada element ili njegovi potomci dobiju fokus.	`element.addEventListener('focusin', function() {})`	`div.addEventListener('focusin', function() { console.log('Fokusiranje!'); })`
focusout	Poziva se kada element ili njegovi potomci izgube fokus.	`element.addEventListener('focusout', function() {})`	`input.addEventListener('focusout', function() { console.log('Izgubio fokus!'); })`
blur	Poziva se kada element izgubi fokus.	`element.addEventListener('blur', function() {})`	`input.addEventListener('blur', function() { console.log('Izgubio fokus!'); })`
mousedown	Poziva se kada se pritisne miš na element.	`element.addEventListener('mousedown', function() {})`	`div.addEventListener('mousedown', function() { console.log('Miš pritisnut!'); })`
mouseenter	Poziva se kada miš uđe u element.	`element.addEventListener('mouseenter', function() {})`	`div.addEventListener('mouseenter', function() { console.log('Miš unutar elementa!'); })`
mouseleave	Poziva se kada miš napusti element.	`element.addEventListener('mouseleave', function() {})`	`div.addEventListener('mouseleave', function() { console.log('Miš izvan elementa!'); })`
mousemove	Poziva se kada se miš pomiče preko elementa.	`element.addEventListener('mousemove', function() {})`	`div.addEventListener('mousemove', function() { console.log('Miš se pomiče!'); })`
mouseout	Poziva se kada miš napusti element ili njegovog potomka.	`element.addEventListener('mouseout', function() {})`	`div.addEventListener('mouseout', function() { console.log('Miš napustio element!'); })`
mouseover	Poziva se kada miš uđe u element ili njegovog potomka.	`element.addEventListener('mouseover', function() {})`	`div.addEventListener('mouseover', function() { console.log('Miš ušao u element!'); })`
mouseup	Poziva se kada se miš otpusti iznad elementa.	`element.addEventListener('mouseup', function() {})`	`div.addEventListener('mouseup', function() { console.log('Miš otpušten!'); })`
input	Poziva se kada se promijeni vrijednost input elementa, a korisnik i dalje ostaje u polju.	`element.addEventListener('input', function() {})`	`input.addEventListener('input', function() { console.log('Vrijednost promijenjena!'); })`
change	Poziva se kada se promijeni vrijednost elementa, a korisnik se miče od polja.	`element.addEventListener('change', function() {})`	`inputElement.addEventListener('change', function() { console.log('Vrijednost promijenjena!'); })`

> Kroz sljedeće primjere i vježbe ćemo pokazati kako se koriste DOM događaji u JavaScriptu.

U svim primjerima koristit ćemo sljedeći CSS kôd:

```
<style>
    div {
        padding: 4px 16px;
    }
    input, button {
        background: transparent;
        border: 1px solid gray;
```

```
        border-radius: 4px;
        padding: 2px 8px;
        margin: 4px 2px;
        &:hover {
            background: #a9a9a950;
        }
        &:focus {
            background: #4cb05050;
        }
        &:active {
            background: #ffeb3c50;
        }
    }
</style>
```

Primjer 6 - `click` event

Dodajemo `input` polje i dva `button` elementa. Input polje predstavlja brojač, a dva button elementa povećavaju i smanjuju brojač za jedan.
`click` događaj smo rekli da se poziva kada se klikne na element jedanput.

> Nemojte zaboraviti kopirati CSS kôd iznad.

```
<div>
    <button id="increaseBtn">+</button>
    <input type="number" disabled name="broj" value="0"/>
    <button id="decreaseBtn">-</button>
</div>
```

Prvi korak je naravno dohvat elemenata:

```
const increaseBtn = document.getElementById("increaseBtn");
const decreaseBtn = document.getElementById("decreaseBtn");
const broj = document.getElementsByName("broj")[0];
```

Zatim dodajemo naredbe (inkrement/dekrement) na `click` event za oba buttona:

```
increaseBtn.addEventListener("click", () => broj.value++) // povećava brojač za 1
decreaseBtn.addEventListener("click", () => broj.value--) // smanjuje brojač za 1
```

Vježba 5

EduCoder šifra: `methods_to_methods`

Želimo napraviti aplikaciju koja će omogućiti korisniku da unese podatke o korisniku (ime, prezime, email) i da ih dodaje u listu korisnika. Korisnik može dodavati korisnike na početak ili kraj liste, te ih može brisati s početka ili kraja liste.

Koristit ćemo dobro poznate metode `Array` objekta. Za dodavanje korisnika koristit ćemo metode `push()` i `unshift()`, a za brisanje korisnika metode `pop()` i `shift()`.

Zadan je sljedeći HTML kôd:

```
<div id="forma">
    Ime: <input type="text" name="Ime" placeholder="Ime..." />
    Prezime: <input type="text" name="Prezime" placeholder="Prezime..." />
    Email: <input type="text" name="Email" placeholder="Email..." />
</div>
<div>
    <!--Svaku metodu predstavit ćemo zasebnim gumbom-->
    <button id="push">push</button>
    <button id="pop">pop</button>
    <button id="unshift">unshift</button>
    <button id="shift">shift</button>
</div>
<div style="font-size: 24px;"><b>Korisnici:</b></div>
<div id="lista">
</div>
```

Zadatak je napisati implementacije funkcija: `dohvatiVrijednosti()`, `dodajNoviElement(pozicija)`, `ukloniElement(pozicija)` i dodati eventListenere za svaki button.

```
const inputs = document.getElementsByTagName('input');
const lista = document.getElementById('lista');
const forma = document.getElementById('forma');

const btn_push = document.getElementById('push');
const btn_pop = document.getElementById('pop');
const btn_unshift = document.getElementById('unshift');
const btn_shift = document.getElementById('shift');

let emailPolje = []

function dohvatiVrijednosti() {
    // Pseudokod:

    // Funkcija dohvaća vrijednosti iz "inputs" i vraća novi formatirani div element
    // Koristeći "for of" petlju ili "forEach" metodu iterira se kroz "inputs" (name, value)
za svaki input
    // Ako je input prazan, defaultna vrijednost je "blank"
    return div;
    /* Primjer div-a:
        <div>
            <b>Ime</b>: Ivan
            <b>Prezime</b>: Ivić
            <b>Email</b>: iivic@gmail.com
        </div>
    */
}
function dodajNoviElement(pozicija) {
    // Pseudokod:

    // Funkcija dodaje novi element ovisno o poziciji ("push", "unshift") switch(pozicija)
    // Vrijednost elementa dohvaća pomoću funkcije dohvatiVrijednosti()
    // Prije dodavanja provjerava je li email već dodan, ako je:
    // dodaje upozorenje "<div id="upozorenje" style="color: red;">Email već postoji!</div>"
```

```
    // inače dodaje novi element u polje i html te miče upozorenje
}
function ukloniElement(pozicija) {
    // Pseudokod:

    // Funkcija briše element ovisno o poziciji ("pop", "shift") switch(pozicija)
    // Prije brisanja provjerava postoji li element
    // Ako postoji briše element iz polja
    // Miče upozorenje bez obzira postoji li element ili ne
}

//dodati eventListener-e za svaki button
```

✅Rezultat:

> Rješenje:

```
// Dohvaćamo sve elemente
const inputs = document.getElementsByTagName('input');
const lista = document.getElementById('lista');
const forma = document.getElementById('forma');

const btn_push = document.getElementById('push');
const btn_pop = document.getElementById('pop');
const btn_unshift = document.getElementById('unshift');
const btn_shift = document.getElementById('shift');

let emailPolje = []
// Funkcija koja dohvaća vrijednosti iz input polja i vraća novi formatirani div element
function dohvatiVrijednosti() {
  let div = document.createElement("div");
  for (const input of inputs) {
    div.innerHTML += `<b>${input.name}</b>: `;
    if (input.value == "") {
      div.innerHTML += "blank ";
    } else {
      div.innerHTML += input.value + " ";
    }
```

```
  }
  return div;
}

// Funkcija koja dodaje novi element ovisno o poziciji ("push", "unshift")
function dodajNoviElement(pozicija) {
  const email = document.getElementsByName('Email')[0].value;
  if (emailPolje.includes(email)) {
    if (document.getElementById("upozorenje") == undefined)
      forma.insertAdjacentHTML("afterend", `<div id="upozorenje" style="color: red;">Email
već postoji!</div>`);
  }
  else {
    // Brišemo upozorenje
    let element = document.getElementById("upozorenje");
    if (element) element.remove();

    let vrijednost = dohvatiVrijednosti();
    switch(pozicija) {
      case "push":
        lista.append(vrijednost);
        emailPolje.push(email);
        break;
      case "unshift":
        lista.prepend(vrijednost);
        emailPolje.unshift(email);
        break;
    }
  }
}

// Funkcija koja briše element ovisno o poziciji ("pop", "shift")
function ukloniElement(pozicija) {
  switch(pozicija) {
    case "shift":
      if (lista.firstElementChild != undefined)  {
        lista.firstElementChild.remove()
        emailPolje.shift();
      }
      break;
    case "pop":
      if (lista.lastElementChild != undefined)  {
        lista.lastElementChild.remove()
        emailPolje.pop();
      }
      break;
  }
// Brišemo upozorenje
let element = document.getElementById("upozorenje");
if (element) element.remove();

}

// Dodajemo eventListenere za svaki button
```

```
btn_push.addEventListener("click", () => dodajNoviElement("push"))
btn_pop.addEventListener("click", () => ukloniElement("pop"))
btn_unshift.addEventListener("click", () => dodajNoviElement("unshift"))
btn_shift.addEventListener("click", () => ukloniElement("shift"))
```

Primjer 7 - `focus` events

`focus` familija događaja se poziva kada element dobiva/gubi fokus u nekom kontekstu. U primjeru ćemo koristiti `focus`, `focusin`, `focusout` i `blur` događaje.

Dodat ćemo dva input polja za ime i prezime, te jedno za broj godina. Kada se fokusira na polje, ispisuje se koji je element fokusiran.

> Nemojte zaboraviti kopirati CSS kôd iznad.

```
<div id="inputi">
    <b>Ime:</b> <input id="ime" placeholder="Ime ..."/>
    <b>Prezime:</b> <input id="prezime" placeholder="Prezime ..."/>
</div>

<div>
    <b>Godine:</b> <input id="brojGodina" type="number" placeholder="Godina ..."/>
</div>

<div>
    <b>Element event:</b> <span id="event"> </span>
</div>
```

```
const inputi = document.getElementById('inputi');
const inputBrojGodina = document.getElementById('brojGodina');

const span = document.getElementById('event');
// Focus event se poziva kada element dobije fokus
inputBrojGodina.addEventListener('focus', event => span.textContent = "focus: " +
event.target.outerHTML);
// Focusin event se poziva kada element ili njegovi potomci dobiju fokus
inputi.addEventListener('focusin', event => span.textContent = "focusin: " +
event.target.outerHTML);
// Focusout event se poziva kada element ili njegovi potomci izgube fokus
inputi.addEventListener('focusout', event => span.textContent = "focusout: " +
event.target.outerHTML);
// Blur event se poziva kada element izgubi fokus
inputBrojGodina.addEventListener('blur', event => span.textContent = "blur: " +
event.target.outerHTML);
```

> `focus` i `blur` se pozivaju samo za trenutni element, ignoriraju potomke, dok se `focusin` i `focusout` pozivaju za element i svakog potomka zasebno.

Vježba 6

EduCoder šifra: `please_focus`

Imate zadana dva input polja za unos lozinke i ponovnu lozinku.

Kada je **input fokusiran**, ovisno o inputu treba pisati odgovarajući hint:

za **Lozinku**:

- Minimalna duljina lozinke mora biti 8
- Lozinka mora imati barem jedno veliko slovo
- Lozinka mora imati barem jedan broj

za **Ponovi lozinku**:

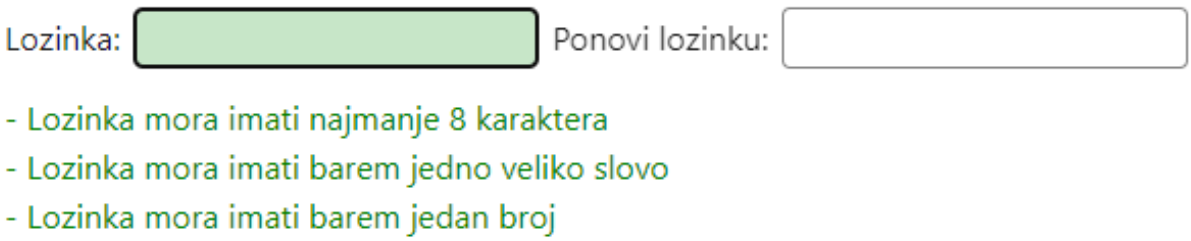
- Lozinke moraju biti iste
  Kada se **izađe iz fokusa**, hint treba biti prazan.

Zadan je sljedeći kôd:

```
<div id="inputs">
    Lozinka: <input name="password" type="password"/>
    Ponovi lozinku: <input name="repeatPassword" type="password"/>
</div>
<div id="hint" style="color: green;">
</div>
```

```
const inputs = document.getElementById('inputs');
const hint = document.getElementById('hint');
/*
Vaš kôd ovdje...
*/
```

✅Rezultat:



> Rješenje:

```
const inputs = document.getElementById('inputs');
const hint = document.getElementById('hint');
// Koristimo focusin i focusout jer se focus i blur pozivaju samo za trenutni element,
ignoriraju potomke
// Focusin se propagira kroz sve potomke unutar DOM stabla.
inputs.addEventListener("focusin", (event) => {
  switch(event.target.name) {
    case "password":
      hint.innerHTML = `
      - Minimalna duljina lozinke mora biti 8 <br>
      - Lozinka mora imati barem jedno veliko slovo <br>
      - Lozinka mora imati barem jedan broj
      `;
```

```
      break;
    case "repeatPassword":
      hint.innerHTML = `
      - Lozinke moraju biti iste
      `;
      break;
  }
})
inputs.addEventListener("focusout", () => {
    hint.innerHTML = ``;
})
```

Primjer 8 - `mouse` events

`mouse` familija događaja se poziva kada se događa neka akcija mišem. U primjeru ćemo koristiti `mouseover`, `mouseout`, `mouseenter`, `mouseleave`, `mousedown`, `mouseup` i `click` događaje.

Zadan je sljedeći kôd:

```
<div id="buttons" style="background: lightblue;">
    <button id="btn_1">1</button>
    <button id="btn_2">2</button>
    <button id="btn_3">3</button>
    <button id="btn_4">4</button>
</div>
<!-- Ispisuje koji je događaj aktiviran -->
<b>Mouse Event:</b> <span id="event"> </span>
```

```
// div koji sadrži sve gumbe
const buttons = document.getElementById('buttons');
// 4 gumba, svaki ima svoj id
const btn_1 = document.getElementById('btn_1');
const btn_2 = document.getElementById('btn_2');
const btn_3 = document.getElementById('btn_3');
const btn_4 = document.getElementById('btn_4');

// dodajemo ih u polje
const btnList = [btn_1, btn_2, btn_3, btn_4]

// dohvaćamo span element za ispis događaja
const span = document.getElementById('event');

// dodavanje event listenera za različite mouse događaje
buttons.addEventListener('mouseover', event => span.textContent = "mouseover: " +
event.target.id);
buttons.addEventListener('mouseout', event => span.textContent = "mouseout: " +
event.target.id);

buttons.addEventListener('mouseenter', event => console.log("mouseenter: " +
event.target.id));
buttons.addEventListener('mouseleave', event => console.log("mouseleave: " +
event.target.id));
```

```
// dodavanje event listenera za mousedown, mouseup i click događaje te ispis događaja
for (const btn of btnList) {
  btn.addEventListener('mousedown', event => {
    console.log("[1] mousedown: " + event.target.id);
    span.textContent = "mousedown: " + event.target.id
  });
  btn.addEventListener('mouseup', event =>  {
    console.log("[2] mouseup: " + event.target.id);
    span.textContent = "mouseup: " + event.target.id
  });
  btn.addEventListener('click', event => console.log("[3] click: " + event.target.id));
}
```

> `mouseenter` i `mouseleave` se pozivaju samo za trenutni element, ignoriraju potomke, dok se `mouseover` i `mouseout` pozivaju za element i svakog potomka zasebno.

> `click` se uvijek poziva nakon `mousedown` i `mouseup` i točno tim redoslijedom.

Vježba 7

EduCoder šifra: `gallery`

Želimo izložiti galeriju slika na našu web stranicu. Svaka slika ima svoj naziv, umjetnika i godinu nastanka. Kada se mišem pređe preko slike, treba se prikazati opis slike (naziv, umjetnik, godina). Kada se mišem "izađe" sa slike, opis se mora maknuti.

Imamo već zadan CSS i HTML kôd, a potrebno je dodati event listenere za `mouseover` i `mouseleave` događaje te ispisati podatke o slici u opisu.

```
<style>
.gallery {
  display: flex;
  flex-wrap: wrap;
}

.artwork {
  transition: all 0.2s ease-in-out;
  position: relative;
  height: 200px;
  width: auto;
  border-radius: 4px;
  margin: 10px;
  &:hover {
    scale: 105%;
  }
}

.opis { padding: 16px; }
h1 { font-size: 24px; font-weight: bold; }
h2 { font-size: 20px; }
h3 { font-size: 14px; }
</style>

<div class="gallery">
```

```
    <img class="artwork" id="artwork1"
src="https://upload.wikimedia.org/wikipedia/commons/thumb/e/ec/Mona_Lisa%2C_by_Leonardo_da_Vi
nci%2C_from_C2RMF_retouched.jpg/1200px-
Mona_Lisa%2C_by_Leonardo_da_Vinci%2C_from_C2RMF_retouched.jpg">
    <img class="artwork" id="artwork2"
src="https://upload.wikimedia.org/wikipedia/en/1/14/Picasso_The_Weeping_Woman_Tate_identifier
_T05010_10.jpg">
    <img class="artwork" id="artwork3"
src="https://upload.wikimedia.org/wikipedia/commons/thumb/6/66/VanGogh-
starry_night_ballance1.jpg/290px-VanGogh-starry_night_ballance1.jpg">
</div>
<!--Opis slike koji je ažurirati kroz JavaScript DOM manipulacijom-->
<div class="opis">
    <h1></h1>
    <h2></h2>
    <h3></h3>
</div>
```

Naši podaci o slikama spremljeni su u polju `galerija`.

```
let galerija = [
  {
    id: "artwork1",
    naziv: "Mona Lisa",
    umjetnik: "Leonardo da Vinci",
    godina: 1503
  },
  {
    id: "artwork2",
    naziv: "The Weeping Woman",
    umjetnik: "Pablo Picasso",
    godina: 1937
  },
  {
    id: "artwork3",
    naziv: "The Starry Night",
    umjetnik: " Vincent van Gogh",
    godina: 1889
  }
]
```

Potrebno je dodati dva event listenera:
Pseudokod:

- kada se mišem pređe preko slike
  - treba iz polja `galerija` iščitati točne podatke i prikazati ih u opisu
- kada se mišem izađe iz slike
  - isprazniti opis

✅Rezultat:



The Starry Night

Vincent van Gogh

1889.

> Rješenje:

```
let galerija = [
  {
    id: "artwork1",
    naziv: "Mona Lisa",
    umjetnik: "Leonardo da Vinci",
    godina: 1503
  },
  {
    id: "artwork2",
    naziv: "The Weeping Woman",
    umjetnik: "Pablo Picasso",
    godina: 1937
  },
  {
    id: "artwork3",
    naziv: "The Starry Night",
    umjetnik: " Vincent van Gogh",
    godina: 1889
  }
]
// Dohvaćamo galeriju i opis
const gallery = document.getElementsByClassName("gallery")[0];
const opis = document.getElementsByClassName("opis")[0];

// Dodajemo event listenere za mouseover i mouseleave
gallery.addEventListener("mouseover", event => {
    // Pronalazimo točnu sliku
    let id = event.target.id;
    if (id == "") return;
    let artwork = galerija.find(g => g.id == id);

    // Ažuriramo opis
    opis.children[0].innerHTML =  artwork.naziv;
```

```
    opis.children[1].innerHTML =  artwork.umjetnik;
    opis.children[2].innerHTML =  artwork.godina+".";
})
gallery.addEventListener("mouseleave", event => {
    // Brišemo opis
    opis.children[0].innerHTML = "";
    opis.children[1].innerHTML = "";
    opis.children[2].innerHTML = "";
})
```

Primjer 9 - `input` event

Kroz primjer ćemo pokazati kako koristiti `input` događaj. `input` događaj se poziva kada se promijeni vrijednost `input` elementa u kojeg korisnik unosi tekst.

Definirat ćemo dva input polja za unos lozinke i ponovnu lozinku. Kada korisnik unese lozinku, ispod polja će se prikazati poruka je li ponovljena lozinka jednaka prvoj.

```
<div>
    Lozinka: <input id="password" type="password"/>
    Ponovi lozinku: <input id="repeatPassword" type="password"/>
    Lozinke iste: <b id="same"></b>
</div>
```

I naš JavaScript kôd:

```
// Dohvaćamo input polja
const password = document.getElementById("password");
const repeatPassword = document.getElementById("repeatPassword");
const same =  document.getElementById("same");

// Dodajemo event listenere za input događaj. U dijelu naredbe provjeravamo jednakost lozinki
password.addEventListener("input", event => {
  same.innerHTML = event.target.value == repeatPassword.value;
});
repeatPassword.addEventListener("input", event => {
  same.innerHTML = event.target.value == password.value;
});
```

Vježba 8

EduCoder šifra: `recommend`

Na web stranicama i aplikacijama često se implementira preporuka pretrage. Kada korisnik počne unositi pojam u polje za pretragu, prikazuju se rezultati koji odgovaraju unesenom pojmu.

U ovom primjeru implementirati ćemo preporuku pretrage za unos u polje za pretragu. Kada korisnik počne unositi pojam, prikazat će se rezultati koji odgovaraju unesenom pojmu. Rezultati se prikazuju u padajućem izborniku ispod polja za pretragu. Kada korisnik klikne na rezultat, unos u polje za pretragu postaje taj rezultat, a padajući izbornik se skriva.

Samo filtriranje smo do sada naučili kroz primjere iz prošlih skripti, sada ćemo sve ukomponirati koristeći HTML i CSS te JavaScript za manipulaciju našim `input` poljem odnosno tražilicom.

> Za stilizirani input kopirajte CSS kôd od ranije.

Zadan je sljedeći HTML/CSS kôd:

```
<style>
    #searchInput {
        width: 100%;
        margin-bottom: 20px;
    }
    #searchResults {
        border: 1px solid #ccc;
        border-radius: 4px;
        padding: 10px;
    }
    #searchResults div {
        margin-bottom: 5px;
        cursor: pointer;
        &:hover {
            background-color: #f0f0f0;
        }
    }
</style>

<div>
    <input type="text" id="searchInput" placeholder="Traži...">
    <div id="searchResults"></div>
</div>
```

JavaScript kôd:

```
const inputField = document.getElementById('searchInput');
const resultsContainer = document.getElementById('searchResults');

// Podaci za preporuku
const data = [
    'JavaScript',
    'HTML',
    'CSS',
    'React',
    'Node.js',
    'Express.js',
    'MongoDB',
    'Vue.js',
    'Angular',
    'TypeScript'
];

function showResults(searchTerm) {
    // Vaš kôd ovdje...
}
```

```
//odgovarajući eventListener
// Vaš kôd ovdje...
```

✅Rezultat:

> Rješenje:

```
const inputField = document.getElementById('searchInput');
const resultsContainer = document.getElementById('searchResults');

// Podaci za preporuku
const data = [
    'JavaScript',
    'HTML',
    'CSS',
    'React',
    'Node.js',
    'Express.js',
    'MongoDB',
    'Vue.js',
    'Angular',
    'TypeScript'
];

function showResults(searchTerm) {
    // Filtriramo podatke (filter, normalizacija i Array.includes metoda)
    const filteredData = data.filter(item =>
item.toLowerCase().includes(searchTerm.toLowerCase()));

    resultsContainer.innerHTML = '';

    // Prikazujemo rezultate u padajućem izborniku (za svaki rezultat radimo div element)
    filteredData.forEach(item => {
        const stavka = document.createElement('div');
        stavka.textContent = item;
        stavka.addEventListener("click", event => {
          inputField.value = event.target.textContent;
          resultsContainer.innerHTML = '';
        })
        resultsContainer.appendChild(stavka);
    });
```

```
}

inputField.addEventListener('input', event => {
    const searchTerm = event.target.value;
    showResults(searchTerm);
});
```

Samostalni zadatak za vježbu 8

EduCoder šifra: `kosarica`

Imate zadatak izraditi sučelje za **košaricu web trgovine**. Sučelje aplikacije sastoji se od polja za unos naziva i cijene proizvoda te gumba za dodavanje proizvoda u košaricu. Također, prikazuje se lista proizvoda u košarici s informacijama o nazivu, količini, cijeni po komadu te ukupnoj cijeni za taj proizvod.

Kada korisnik unese naziv i cijenu proizvoda te klikne na gumb `Dodaj artikl`, proizvod se dodaje u košaricu.

- Ako proizvod nema ime, dugme se ne može kliknuti, mora biti zatamnjeno/onemogućeno
- Ako se proizvod s istim imenom već nalazi u košarici, količina se povećava za `1`
- Ako je proizvod novi, dodaje se na listu
- Cijena proizvoda ne može ići ispod `0`

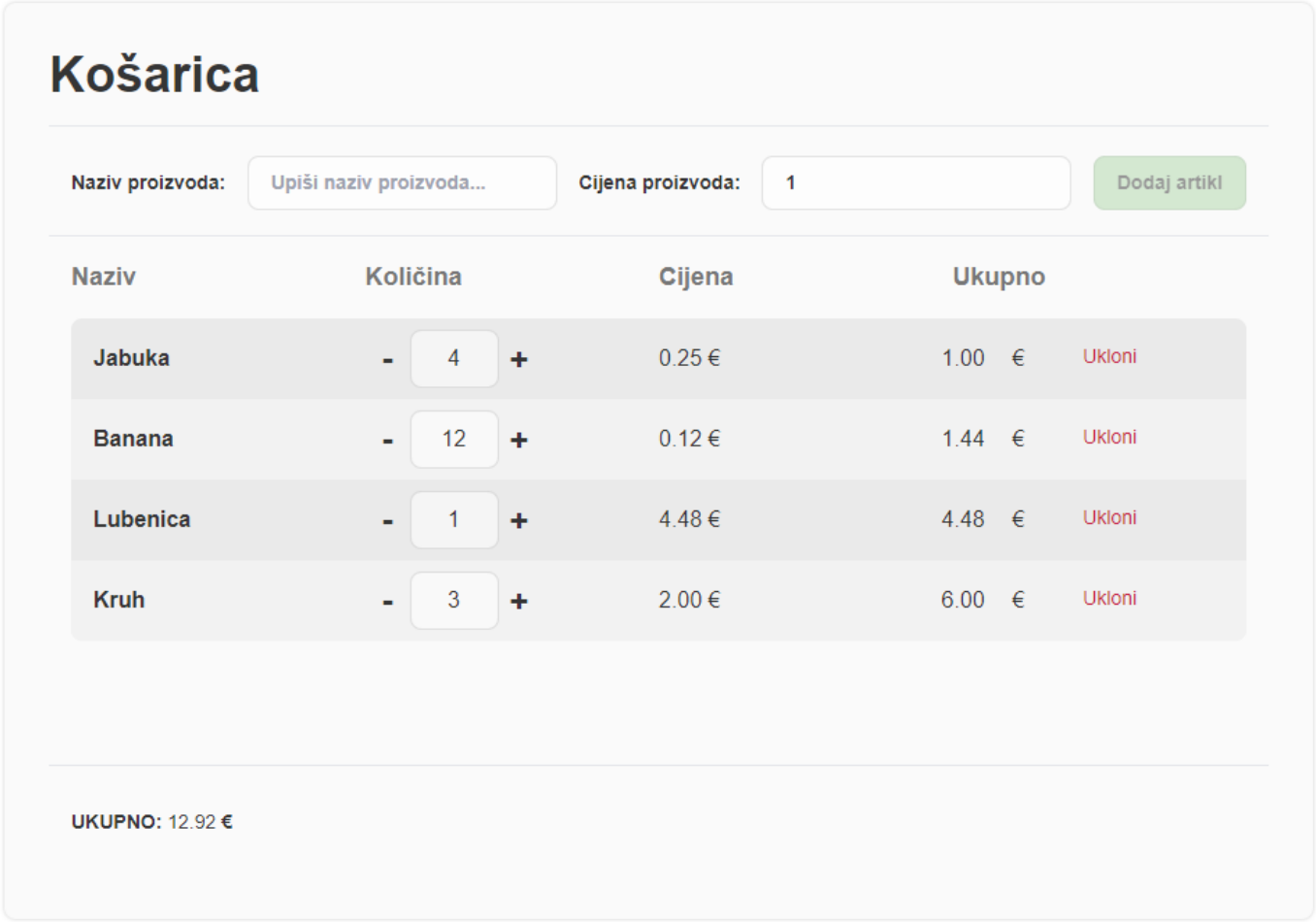
Korisnik može mijenjati količinu proizvoda u košarici koristeći gumbe `+` i `-` pored svakog proizvoda. Također, postoji opcija za uklanjanje proizvoda iz košarice klikom na gumb `Ukloni`.

Nakon svake promjene u košarici, ukupna cijena se automatski ažurira kako bi korisnik imao uvid u trenutni trošak.

Jedan primjer implementacije:

- Izrada konstruktora `Proizvod(naziv, kolicina, cijena)` koji ima atribute `naziv`, `kolicina`, `cijena` i metodu `ukupnaCijena()` koja vraća ukupnu cijenu proizvoda zaokruženu na dvije decimale.
- Izrada objekta `kosarica` koja sadrži:
  - atribut `proizvodi` - lista/polje proizvoda
  - metodu `dodajProizvod(proizvod)` - dodaje proizvod u polje `proizvodi` i HTML element
  - metodu `dodajFunkcionalnosti(naziv)` - dodaje funkcionalnosti (`eventListener`-i) za svaki proizvod u košarici:
    - povećanje količine
    - smanjenje količine
    - brisanje proizvoda
  - metodu `azurirajUkupnuCijenu()` - koristeći `reduce` nad poljem računa ukupnu cijenu svih proizvoda zaokruženu na dvije decimale

> Primjer:



Možete napisati vlastiti HTML i CSS kôd ili koristiti sljedeći:

```
<style>
    body {
        padding: 64px;
        font-family: Sans-Serif;
    }
    .main {
        display: flex;
        width: 100%;
        height: 100%;
        justify-content: center;
    }
    .card {
        overflow: hidden;
        width: 100%;
        padding: 32px;
        display: flex;
        flex-direction: column;
        background-color: rgb(205, 205, 205, 0.1);
        border-radius: 8px;
        color: #353535;
        box-shadow: 0px 0px 3px rgba(0, 0, 0, 0.2);
    }
    input, button {
        transition: all 0.2s ease-in-out;
```

```
        padding: 8px 16px;
        background: rgba(255, 255, 255, 0.5);
        outline: none;
        border: 1px solid rgba(0, 0, 0, 0.1);
        border-radius: 8px;
        &:hover {
            background: rgba(255, 255, 255, 1);
            border: 1px solid rgba(0, 0, 0, 0.25);
        }
        &:focus {
            background: rgba(255, 255, 255, 1);
            border: 1px solid rgba(0, 0, 0, 0.25);
        }
    }
    button {
        background-color: #a5d6a7;
        &:hover {
            background: #83c683;
            cursor: pointer;
        }
    }
    form {
        overflow-y: hidden;
        min-height: 48px;
        overflow-x: auto;
        padding: 0px 16px;
        display: flex;
        margin-top: 16px;
        gap: 8px;
        justify-content: space-between;
        align-items: center;
        font-size: 14px;
        font-weight: bold;
    }
    .content {
        padding: 16px 16px;
        height: 100%;
        display: flex;
        flex-direction: column;
        overflow-y: auto;
    }
    .flex {
        width: 100%;
        display: inline-grid;
        grid-template-columns: repeat(4, minmax(0, 1fr));
    }
    .item-list {
        overflow-y: auto;
        overflow-x: hidden;
        padding: none !important;
        border-radius: 8px;
        margin-top: 16px;
    }
    .item {
```

```
        background-color: rgba(169, 169, 169, 0.2);
        padding: 8px 16px;
        align-items: center
    }
    .item:nth-child(2n) {
        background-color: rgba(169, 169, 169, 0.1);
    }
    .item-kolicina {
        width: 64px;
        text-align: center;
        margin-left: 8px;
        margin-right: 4px;
    }
    .item-kolicina-button {
        padding: 0px 4px;
        font-size: 24;
    }
    .item-kolicina-button:hover {
        padding: 0px 4px;
        color: #1f87e8;
        cursor: pointer;
    }
    .item-ukloni {
        color: #d81b43;
        font-size: 14;
        &:hover {
            text-decoration: underline;
            cursor: pointer;
        }
    }
    #ukupno {
        font-weight: normal;
    }
    .disabled {
        opacity: 0.5;
        cursor: not-allowed !important;
    }
</style>
<div class="main">
    <div class="card">

        <h1 class="text-4xl mb-4 font-bold">Košarica</h1>

        <hr />

        <form>
            <label for="naziv_proizvoda">Naziv proizvoda:</label>
            <input type="text" name="naziv_proizvoda" id="naziv_proizvoda" placeholder="Upiši
naziv proizvoda..." />
            <label for="cijena_proizvoda">Cijena proizvoda:</label>
            <input type="number" value="1" min="0" name="cijena_proizvoda"
id="cijena_proizvoda" placeholder="Upiši cijenu proizvoda..." />
            <button disabled class="whitespace-nowrap disabled" type="button"
name="dodaj_button" id="dodaj_button">Dodaj artikl</button>
```

```
        </form>

        <hr />

        <div class="content">
            <div class="flex" style="font-size: 18; color: #787878;">
                <b> Naziv </b>
                <b> Količina </b>
                <b> Cijena </b>
                <b> Ukupno </b>
            </div>

            <div id="item_list" class="item-list">
                <div class="flex item" id="item_Jabuka"><b>
                        Jabuka
                    </b>
                    <div style="display: flex; align-items: center">
                        <b class="item-kolicina-button item-kolicina-minus"
id="item_kolicina_minus_Jabuka">-</b>
                        <input name="kolicina" id="item_kolicina_Jabuka" class="item-
kolicina" value="4" disabled="">
                        <b class="item-kolicina-button item-kolicina-plus"
id="item_kolicina_plus_Jabuka">+</b>
                    </div>
                    <div>
                        0.25 €
                    </div>
                    <div class="flex">
                        <span id="item_ukupnaCijena_Jabuka">1.00</span> €
                        <div class="item-ukloni" id="item_ukloni_Jabuka">
                        Ukloni
                        <div>
                    </div>
                </div>
            </div>
        </div>

        <hr />
        <form>
            <div> UKUPNO: <span id="ukupno"></span> € </div>
        </form>

    </div>
</div>
```

2. JSON - JavaScript Object Notation

JSON (**JavaScript Object Notation**) je **string format za razmjenu podataka** koji je jednostavan čovjeku za razumijevanje, ali i računalu za procesiranje. JSON format često se koristi za slanje podataka između web poslužitelja i klijenta. Radi se o tekstualnom formatu koji se sastoji od parova ključ-vrijednost i nizova, vrlo slične sintakse kao i JavaScript objekti.

JSON format je neovisan o jeziku, što znači da se može koristiti u bilo kojem programskom jeziku. Međutim, svojom sintaksom podsjeća na JavaScript objekte i polja. Format je nastao ranih 2000-ih, a danas je de facto **standard za razmjenu podataka na webu**.

JSON podaci se mogu spremiti u datoteku s ekstenzijom `.json` ili kao tekstualni podaci u bazi podataka.

2.1 Struktura JSON-a

- Podaci u JSON zapisu su organizirani kao `ključ:vrijednost` parovi. Ključevi **moraju biti nizovi znakova**, a vrijednosti mogu biti bilo kojeg tipa podataka (string, broj, objekt, polje, boolean, null).
- Vitičaste zagrade `{}` koriste se za definiranje objekata.
- Uglate zagrade `[]` koriste se za definiranje polja/niza.
- Svaki par `ključ:vrijednost` odvojen je zarezom.
- Ključevi su nizovi znakova (stringovi) i **morate ih staviti u dvostruke navodnike**.
- u JSON se **ne mogu pisati komentari**.

Primjer JSON formata s 2 `ključ:vrijednost` para:

```
{
    "kljuc1": "vrijednost1",
    "kljuc2": "vrijednost2"
}
```

Primjer ugniježđenog JSON formata:

```
{
    "kljuc1": "vrijednost1",
    "kljuc2": {
        "kljuc3": "vrijednost3",
        "kljuc4": "vrijednost4"
    }
}
```

> Kao glavnu razliku uočite dvostuke navodnike oko ključeva

Još jedan popularan format za razmjenu podataka je **XML** (Extensible Markup Language). Sintaksa XML-a je nešto složenija od JSON-a te više nalikuje HTML-u.

Primjer istog podataka u XML i JSON formatima:

XML:

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON:

```
{
  "employees": [
    {
      "firstName": "John",
      "lastName": "Doe"
    },
    {
      "firstName": "Anna",
      "lastName": "Smith"
    },
    {
      "firstName": "Peter",
      "lastName": "Jones"
    }
  ]
}
```

2.2 Primjeri ispravnog i neispravnog JSON formata

Primjer JSON formata s podacima o osobi:

✅ Ispravan JSON format:

```
{
    "ime": "Ana",
    "prezime": "Anić",
    "godine": 25,
    "zaposlen": true,
    "adresa": {
        "ulica": "Ulica 1",
        "grad": "Zagreb",
        "poštanski broj": "10000"
    },
    "hobi": ["čitanje", "pisanje", "plivanje"]
}
```

Primjer JSON formata s podacima o knjizi:

✅ Ispravan JSON format:

```
{
    "naslov": "Harry Potter and the Philosopher's Stone",
    "autor": "J.K. Rowling",
    "godina izdanja": 1997,
    "žanrovi": ["fantasy", "drama"],
    "izdavač": {
        "naziv": "Bloomsbury",
        "lokacija": "London"
    }
}
```

Postoji i mogućnost da se JSON podaci nalaze u polju, bez omotavanja vitičastim zagradama:

✅ Ispravan JSON format:

```
[
    {
        "ime": "Ana",
        "prezime": "Anić",
        "godine": 25
    },
    {
        "ime": "Ivan",
        "prezime": "Ivić",
        "godine": 30
    }
]
```

Međutim ne možemo imati više objekata bez omotavanja u polje:

❌ Neispravan JSON format:

```
{
    "ime": "Ana",
    "prezime": "Anić",
    "godine": 25
},
{
    "ime": "Ivan",
    "prezime": "Ivić",
    "godine": 30
}
```

Komentare nije moguće pisati u JSON formatu, jer JSON je strogo definiran format podataka.

❌ Neispravan JSON format:

```
{
    "ime": "Ana",
    "prezime": "Anić",
    "godine": 25,
    "adresa": "Zagreb" // Adresa stanovanja
}
```

Ključeve je potrebno uvijek staviti u dvostruke navodnike:

❌ Neispravan JSON format:

```
{
    ime: "Ana",
    prezime: "Anić",
    godine: 25
}
```

Za razliku od JavaScript objekata, JSON ne podržava funkcije, niti metode.

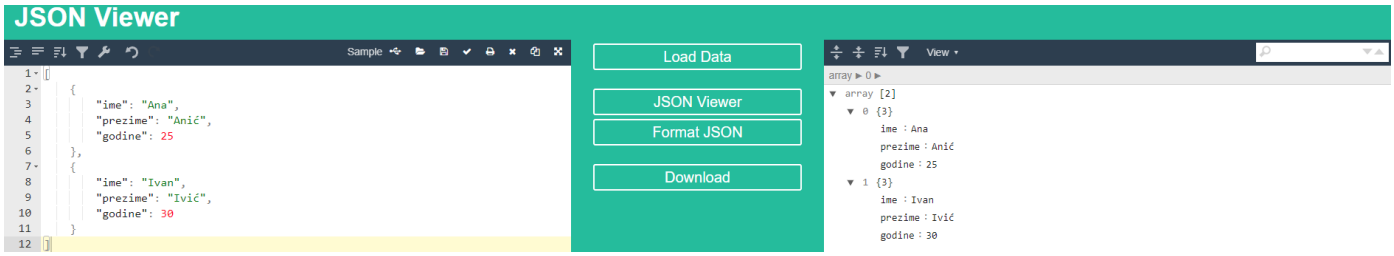
❌ Neispravan JSON format:

```
{
    "ime": "Ana",
    "prezime": "Anić",
    "godine": 25,
    "pozdravi": function() {
        return "Pozdrav!";
    }
}
```

2.3 Online alati za prikaz/validaciju JSON formata

Postoje online alati koji vam mogu pomoći u validaciji i prikazu JSON formata:

- JSONFormatter.org
- Codebeautify.org
- JSONEditorOnline.org
- JSONViwer Chrome ekstenzija



2.4 Rad s JSON formatom u JavaScriptu

JSON je isključivo tekstualni format podataka u koji se ne mogu unositi metode odnosno funkcije.

Moguće je pretvoriti JSON format u JavaScript objekt i obrnuto, koristeći ugrađene metode `JSON.parse()` i `JSON.stringify()`.

2.4.1 `JSON.parse()`

Kada podaci pristignu s web servera, gotovo uvijek su u JSON formatu koji je tekstualni format. Da bismo s njima mogli raditi u JavaScriptu, moramo ih pretvoriti u JavaScript objekt. To radimo pomoću metode `JSON.parse()`.

Recimo da smo dobili JSON podatke o korisniku s web servera:

JSON:

```
{"ime": "Petar", "prezime": "Perković", "email": "pperkovic@unipu.hr", "lozinka":
"98fd88c8fdc81c8efbd3a158007a97a6"}
```

Koristeći metodu `JSON.parse()`, možemo pretvoriti JSON podatke u JavaScript objekt:
Uočite da smo koristili dvostruke navodnike oko ključeva, što je obavezno u JSON formatu. Međutim, cijeli JSON je string tako da ga ovdje moramo omotati u jednostruke navodnike (`''`) ili backticks navodnike.

JavaScript:

```
const korisnik = JSON.parse('{"ime": "Petar", "prezime": "Perković", "email":
"pperkovic@unipu.hr", "lozinka": "98fd88c8fdc81c8efbd3a158007a97a6"}');

console.log(korisnik.ime); // Petar
console.log(korisnik.prezime); // Perković
console.log(korisnik.email); // pperkovic@unipu.hr
console.log(korisnik.lozinka) // 98fd88c8fdc81c8efbd3a158007a97a6
```

Primjetite da jednom kad parsiramo JSON u JavaScript objekt, možemo s njim raditi kao s bilo kojim drugim objektom u JavaScriptu.

Isto vrijedi i za JSON polje:

JSON:

```
{
    "naslov": "Harry Potter and the Philosopher's Stone",
    "autor": "J.K. Rowling",
    "godina izdanja": 1997,
    "žanrovi": ["fantasy", "drama"],
    "izdavač": {
        "naziv": "Bloomsbury",
        "lokacija": "London"
    }
}
```

JavaScript:

```
const knjiga = JSON.parse('{"naslov": "Harry Potter and the Philosopher\'s Stone", "autor":
"J.K. Rowling", "godina izdanja": 1997, "žanrovi": ["fantasy", "drama"], "izdavač": {"naziv":
"Bloomsbury", "lokacija": "London"}}');
```

```
console.log(typeof knjiga); // object

console.log(knjiga.naslov); // Harry Potter and the Philosopher's Stone
console.log(knjiga.žanrovi); // ["fantasy", "drama"]
console.log(knjiga.izdavač.naziv); // Bloomsbury
console.log(knjiga.žanrovi[0]); // fantasy

for (let zanr of knjiga.žanrovi) {
    console.log(zanr);
}
// fantasy
// drama
```

2.4.1 `JSON.stringify()`

Metoda `JSON.stringify()` koristi se za pretvaranje JavaScript objekta u JSON format. Ova metoda je korisna kada želimo poslati podatke na web server, jer web serveri uglavnom očekuju JSON format.

Primjer pretvaranja JavaScript objekta u JSON format:

```
const korisnik = {
    ime: "Petar",
    prezime: "Perković",
    email: "pperkovic@gmail.com",
    lozinka: "super_sigurna_lozinka123"
};

console.log(typeof korisnik); // object

const korisnikJSON = JSON.stringify(korisnik);

console.log(typeof korisnikJSON); // string

console.log(korisnikJSON);

// Ispis:
'{"ime":"Petar","prezime":"Perković","email":"pperkovic@gmail.com","lozinka":"super_sigurna_l
ozinka123"}'
```

Kako funkcije/metode nisu dozvoljene u JSON formatu, metoda `JSON.stringify()` će ih ignorirati prilikom pretvaranja objekta u JSON format, i ključ i vrijednost će biti izostavljeni.

```
const obj = {name: "John", age: function () {return 30;}, city: "New York"};

console.log(JSON.stringify(obj)); // '{"name":"John","city":"New York"}'
```

Ako koristimo `JSON.stringify()` na objektu koji sadrži polje, metoda će automatski pretvoriti polje u JSON format.

```
const obj = {name: "John", age: 30, cars: ["Ford", "BMW", "Fiat"]};

console.log(JSON.stringify(obj)); // '{"name":"John","age":30,"cars":["Ford","BMW","Fiat"]}'
```

Međutim ako ubacite u objekt `Date` objekt, metoda će ga automatski pretvoriti u string.

```
const obj = {name: "John", age: 30, birthdate: new Date()};

console.log(JSON.stringify(obj)); // '{"name":"John","age":30,"birthdate":"2024-05-19T21:12:05.712Z"}'
```

2.5 Lokalno čitanje JSON datoteka

JSON podaci se često koriste za razmjenu podataka između web poslužitelja i klijenta, ali se mogu koristiti i za lokalno čitanje i spremanje podataka.

U JavaScriptu, ovisno o okruženju, možemo koristiti različite metode za čitanje i spremanje JSON datoteka.

2.5.1 Node.js

U **Node.js** okruženju, možemo koristiti ugrađeni modul `fs` (File System) za čitanje i pisanje datoteka.

Funkcija `require` uključuje ugrađeni modul `fs`, i pišemo ju na početku datoteke.

Primjer čitanja JSON datoteke u Node.js okruženju. Pročitajmo datoteku `harry_potter.json` koja sadrži podatke o "Harry Potter" knjigama.

Node.js program možemo pokrenuti u terminalu naredbom `node index.js`, odnosno `node naziv_datoteke.js`.

```
{
    "naslov": "Harry Potter and the Philosopher's Stone",
    "autor": "J.K. Rowling",
    "godina izdanja": 1997,
    "žanrovi": ["fantasy", "drama"],
    "izdavač": {
        "naziv": "Bloomsbury",
        "lokacija": "London"
    }
}
```

```
const fs = require('fs');

fs.readFile('harry_potter.json', 'utf8', (err, data) => { // callback funkcija koja se poziva nakon što se datoteka pročita
    if (err) {
        console.log(err);
        return;
    }

    const podaci = JSON.parse(data); // pretvaranje JSON podataka u JavaScript objekt
    console.log(podaci); // ispis podataka
});
```

```
console.log(podaci["harry_potter_books"][0]);
/*
{
  title: "Harry Potter and the Philosopher's Stone",
  author: 'J.K. Rowling',
  publication_date: '1997-06-26',
  publisher: 'Bloomsbury',
  isbn: '978-0747532699',
  summary: 'Harry Potter discovers on his eleventh birthday that he is the orphaned son of two powerful wizards and possesses unique magical powers of his own.'
}
*/
```

2.5.2 Web preglednik

U web pregledniku, možemo koristiti `XMLHttpRequest` ili `fetch` API za čitanje JSON datoteka.

Pokazat ćemo noviji `fetch` API, koji je jednostavniji za korištenje.

Detaljnije o `fetch` API-u u posljednjem poglavlju o asinkronom programiranju.

Primjer čitanja JSON datoteke u web pregledniku koristeći `fetch` API:

```
fetch('harry_potter.json') // Putanja do JSON datoteke lokalno
    .then(response => response.json()) // Više o ovim koracima u sljedećem poglavlju
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.log('Error:', error);
    });
```

Također, isto možemo postići koristćei ES6 `import` sintaksu:

```
import data from './harry_potter.json';

console.log(data);
```

> **Napomena**: Direktno čitanje datoteka iz lokalnog sustava datoteka nije moguće iz sigurnosnih razloga. U stvarnim aplikacijama, JSON podaci se obično čitaju s web poslužitelja. Pokretanje lokalnog http servera često je dobar "workaround" za ovaj problem.

Vježba 9

EduCoder šifra: `JSON`

Ispravite greške u sljedećim JSON podacima:

```
{
{
  "books": [
    {
      "title": "To Kill a Mockingbird",
      "author": "Harper Lee",
      "genre": "Fiction",
      "publishedYear": 1960,
      ISBN: "978-0-06-112008-4",
      availableCopies: 4
    },
    {
      "title": "1984",
      "author": "George Orwell",
      "genre": "Dystopian",
      "publishedYear": 1949,
      ISBN: "978-0-452-28423-4",
      availableCopies: 6
    },
    {
      "title": "The Great Gatsby",
      "author": "F. Scott Fitzgerald",
      "genre": "Fiction",
      "publishedYear": 1925,
      ISBN: "978-0-7432-7356-5",
      availableCopies: undefined
    },
  ]
}
}
```

Nakon što ispravite greške, napišite funkciju `printBooks(JSONbooks)` koja prima JSON podatke o knjigama, pretvara ih u JavaScript objekt i ispisuje sve knjige u konzolu.

```
function printBooks(JSONbooks) {
    // Vaš kôd ovdje...
}
```

3. Asinkrono programiranje

Posljednje poglavlje ove skripte, kao i gradivo ovog kolegija, odnosi se na asinkrono programiranje.

Asinkrono programiranje (***eng. Asynchronous programming***) je način programiranja u kojem se operacije izvršavaju neovisno jedna o drugoj, bez čekanja na završetak prethodne operacije. Ovo je posebno korisno kada se radi s operacijama koje zahtijevaju vrijeme, kao što su čitanje podataka s web poslužitelja, pisanje u bazu podataka i sl.

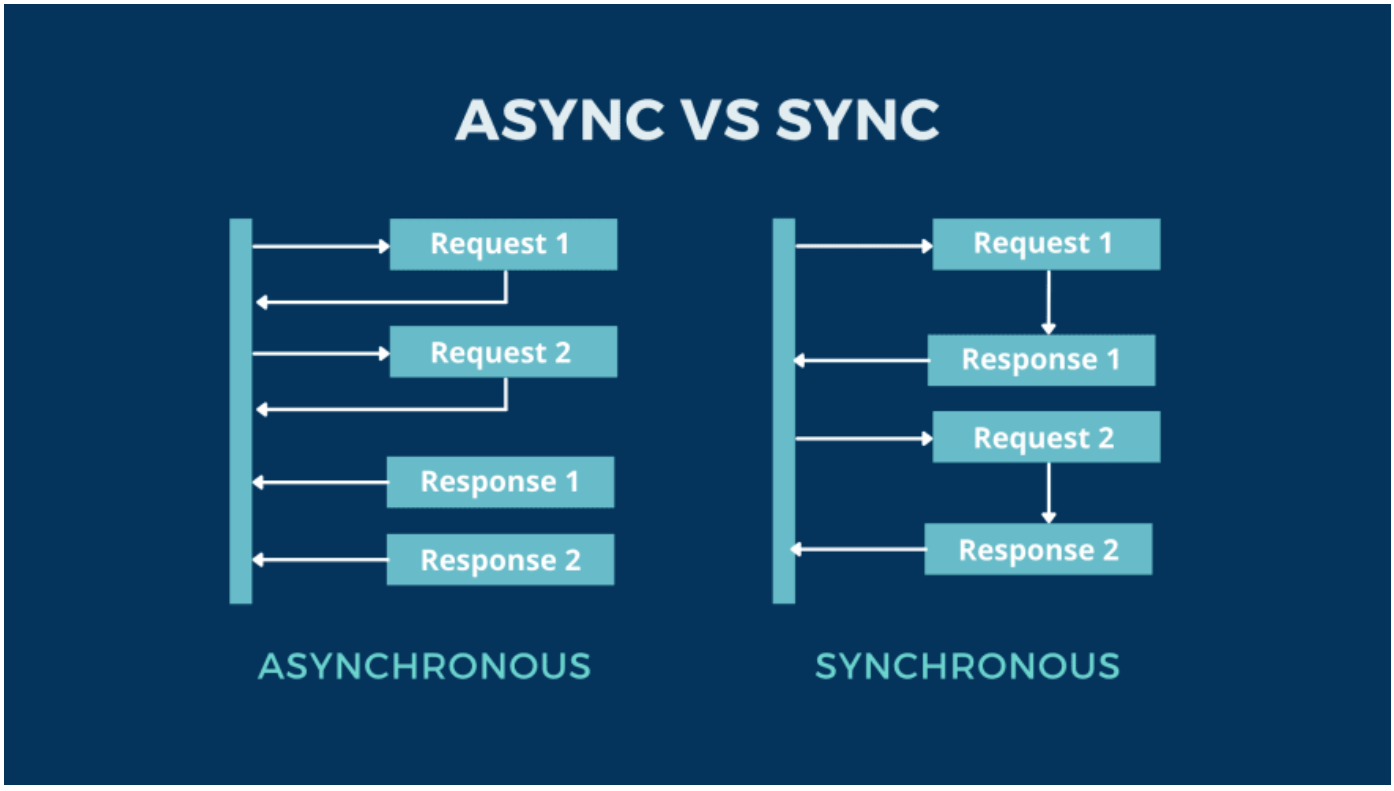
Recimo da želimo dohvatiti podatke s API-a (**eng.** Application Programming Interface) koji se nalazi na udaljenom web poslužitelju. Primjerice, radimo aplikaciju koja prikazuje vremensku prognozu za gradove diljem svijeta. Da bismo dohvatili podatke s API-a, moramo poslati zahtjev na web poslužitelj, pričekati odgovor i zatim prikazati podatke korisniku. Navedena operacija može potrajati nekoliko sekundi, ovisno o brzini interneta i udaljenosti web poslužitelja.

Međutim, protok podataka može biti spor (ili skroz puknuti) zbog različitih faktora, kao što su:

- Spora veza s internetom
- Preopterećenost web poslužitelja
- Dugotrajne operacije na poslužitelju
- Dugotrajne operacije na klijentskoj strani
- Blokirajuće operacije
- ISP (Internet Service Provider) problemi
- Vanjski događaji (npr. kibernetički napadi, prirodne katastrofe, vremenske neprilike)

Kako bi se izbjeglo blokiranje glavne dretve (**eng. main thread**), možemo koristiti asinkrono programiranje bazirano na konkurentnosti. Asinkrono programiranje omogućuje izvršavanje više operacija istovremeno, bez čekanja na završetak prethodne operacije.

Drugim riječima, ako naš korisnik čeka na odgovor s web poslužitelja, kod recimo dohvaćanja podataka o vremenskoj prognozi, ne želimo da mu se cijela aplikacija zamrzne dok čeka. Umjesto toga, želimo da korisnik može nastaviti koristiti aplikaciju dok se podaci dohvaćaju te mu na korektan način dati povratnu informaciju o tome što se događa.



> Izvor: https://dev.to/vinaykishore/how-does-asynchronous-javascript-work-behind-the-scenes-4bjl

Asinkronim programiranjem bavit ćemo se intenzivnije na kolegijima: **Programsko inženjerstvo**, **Web aplikacije** i **Raspodijeljeni sustavi**.

3.1 Razumijevanje asinkronog vs. sinkronog

Najjednostanvije rečeno, u **sinkronom programiranju**, operacije se izvršavaju jedna za drugom, redom. Kada se jedna operacija završi, tek tada se izvršava sljedeća operacija. Sve ispite, zadaće i vježbe do sad iz ovih skripti - pisali smo sinkrono.

Primjer:

Sinkrono programiranje:

```
console.log('Početak');
console.log('Operacija');
console.log('Kraj');
```

Ispisuje:

```
Početak
Operacija
Kraj
```

U **asinkronom programiranju**, kôd se može izvršavati "preko reda". Operacije mogu započinjati i završavati u različito vrijeme, neovisno o glavnom protoku programa. JavaScript je single-threaded jezik, što znači da se sve operacije izvršavaju na jednoj dretvi. Međutim, JS koristi asinkrono programiranje kako bi se izbjeglo blokiranje glavne dretve. Asinkrone potrebe uključuju radnje poput: čitanja podataka s web poslužitelja, pisanja u bazu podataka, čekanja na korisnički unos (I/O operacije).

Idemo simulirati čekanje dohvata podataka s nekog web poslužitelja. Recimo da je web server dosta udaljen i imamo spor internet, pa će dohvat podataka trajati 2 sekunde. Simulirat ćemo navedeno pomoću `setTimeout` funkcije koja prima 2 argumenta:

- callback funkciju koja se izvršava nakon određenog vremena i
- vrijeme čekanja u milisekundama.

Asinkrono programiranje:

```
function fetchData(callback) {
    setTimeout(() => { // simulacija dohvata podataka s web poslužitelja kroz setTimeout
funkciju
        callback('Podaci su dohvaćeni');
    }, 2000);
}

console.log('Start');
fetchData((message) => {
    console.log(message);
});
console.log('End');
```

Kojim redoslijedom će se ispisati poruke?

▶ Spoiler Warning!

```
Start
End
Podaci su dohvaćeni
```

Zašto je ovako?
Sinkroni redoslijed izvršavanja:

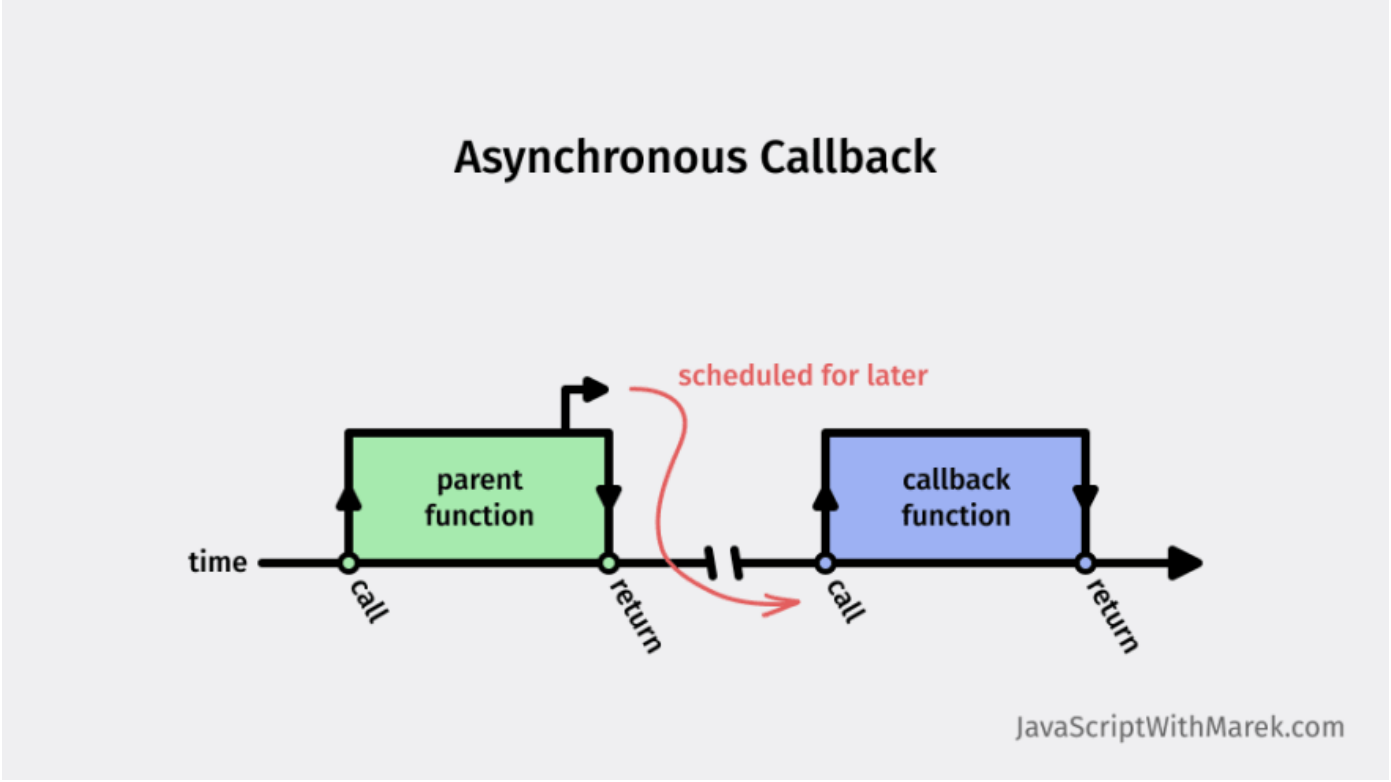
1. Ispisuje se `Start`
2. Poziva se funkcija `fetchData` koja simulira dohvat podataka s web poslužitelja
3. Ispisuje se `End`
4. Nakon 2 sekunde, vraća se odgovor s "web poslužitelja" i ispisuje se `Podaci su dohvaćeni`

3.2 Asinkrone callback funkcije

Upoznali smo se s callback funkcijama u skripti PJS4 u kontekstu pomoćnih funkcija za obradu podataka kod `Array` objekta. Rekli smo da su to funkcije koje prosljeđujemo kao argumente drugim funkcijama (u našem slučaju je bilo metodama `Array` objekta), a koje se pozivaju nakon završetka izvršavanja te funkcije/metode.

U kontekstu manipulacije DOM-om, koristili smo callback funkcije za dodavanje event listenera na HTML elemente.

U kontekstu asinkronog programiranja, callback funkcije koristimo za **rukovanje asinkronim operacijama**.



> Izvor: https://dev.to/marek/are-callbacks-always-asynchronous-bah

U prethodnom primjeru vidjeli smo kako se koristi callback funkcija za rukovanje asinkronim operacijama.

Pojednostavimo primjer:

```
console.log('Start');

setTimeout(() => {
    console.log('Ovo je asinkroni callback');
}, 2000); // 2000 milisekundi = 2 sekundi

console.log('End');
```

Ispisuje:

```
Start
End
Ovo je asinkroni callback (nakon 2 sekunde)
```

Primjer iznad možemo podijeliti na sinkronu i asinkronu egzekuciju:

1. Sinkrona egzekucija:
    - Ispisuje se `Start`
    - Ispisuje se `End`
2. Asinkrona egzekucija:
    - Poziva se `setTimeout` funkcija koja postavlja timer na 2 sekunde i definira asinkronu callback funkciju u `event queue`-u
    - kada timer istekne, callback funkcija se stavlja u `call stack` i izvršava: ispisuje se `Ovo je asinkroni callback`

3.3 Fetch API - dohvaćanje podataka s web poslužitelja

U JavaScriptu, `fetch` API je sučelje koje omogućuje asinkrono dohvaćanje resursa s web poslužitelja preko HTTP protokola. `fetch` API je moderna zamjena za zastarjelu `XMLHttpRequest` metodu.

`fetch` API koristi `Promise` objekte za rukovanje asinkronim operacijama. `Promise` objekt predstavlja eventualni rezultat asinkronog procesa i njegovo konačno stanje (rezoluciju ili odbijanje). Više o `Promise` objektima u sljedećem poglavlju, i nadolazećim kolegijima.

Ovaj API možemo direktno koristiti u web pregledniku ili u Node.js okruženju bez da uključujemo dodatne biblioteke. Bez obzira na to, postoje i biblioteke kao što su `axios`, `jQuery.ajax`, `superagent` koje pojednostavljuju rad s HTTP zahtjevima.

Postoji mnoštvo servisa koji pružaju besplatne API-eve za testiranje i učenje asinkronog programiranja. Na primjer:

- JSONPlaceholder
- PokeAPI
- TheDogAPI
- TheCatAPI

Ogromnu listu razno-raznih API-eva možete pronaći ovdje.
Napomena, neki od API-eva mogu biti zastarjeli, imati ograničenja ili biti nedostupni. **Prije slanja HTTP zahtjeva na API, provjerite dokumentaciju!**

`fetch` API možemo koristiti za dohvaćanje podataka s web poslužitelja, ali i lokalnih JSON datoteka (primjer s "Harry Potter" knjigama).

Sad ćemo pokazati kako koristiti `fetch` API za dohvaćanje podataka s web poslužitelja koristeći `JSONPlaceholder` servis.

```
fetch('https://jsonplaceholder.typicode.com/todos/1') // Obavezni argument je URL (ima i drugih međutim za sad ćemo prekočiti)
      .then(response => response.json()) // arrow funkcija koja pretvara odgovor u JSON format
      .then(json => console.log(json)) // arrow funkcija koja ispisuje JSON podatke

// Ispisuje:
// { userId: 1, id: 1, title: 'delectus aut autem', completed: false }
```

Isto možemo raspisati i bez arrow funkcija:

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
      .then(function(response) { // response kao argument je rezultat fetch metode
          return response.json();
      })
      .then(function(json) { // json kao argument je rezultat prethodne then metode
          console.log(json);
      });
```

U gornjem primjeru, `fetch` funkcija prima URL kao argument i vraća `Promise` objekt. Nakon što se podaci dohvate, koristimo `then` metodu za rukovanje odgovorom. Prva `then` metoda pretvara odgovor u JSON format, a druga `then` metoda ispisuje JSON podatke.

Kôd možemo doslovno čitati kao: "**Dohvati** podatke s URL-a, **onda** pretvori odgovor u JSON format, **onda** ispiši JSON podatke".

`json()` metoda koristi se za parsiranje `Response` odgovora koji je u JSON formatu u JavaScript objekt. Sintaksa je: `response.json()`.

Osim `then()` metode, možemo koristiti i `catch()` metodu za rukovanje greškama. Ako dođe do greške prilikom dohvaćanja podataka, `catch()` metoda će uhvatiti grešku i ispisati je.

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
      .then(response => response.json())
      .then(json => console.log(json))
      .catch(error => console.log('Greška:', error));
```

ili bez arrow funkcija...

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
      .then(function(response) { // response kao argument je rezultat fetch metode
          return response.json();
      })
      .then(function(json) { // json kao argument je rezultat prethodne then metode
          console.log(json);
      })
      .catch(function(error) { // error kao argument je rezultat greške
          console.log('Greška:', error);
      });
```

Detalje o ovom API-ju i njegovim mogućnostima možete pronaći na MDN web dokumentaciji.

Primjetite da smo u gornjem primjeru radili dvostruke `then` metode. Ovo je korisno kada želimo izvršiti više operacija nakon dohvaćanja podataka, međutim može doći do tzv. **callback hell**-a, odnosno dubokog gniježđenja callback funkcija, što može biti teško za održavanje i čitanje kôda, ali i sklono greškama.

U poglavlju o **Async/Await** sintaksi, pokazat ćemo kako možemo izbjeći callback hell i olakšati rad s asinkronim operacijama.

Vježba 10

EduCoder šifra: `cat_fact`

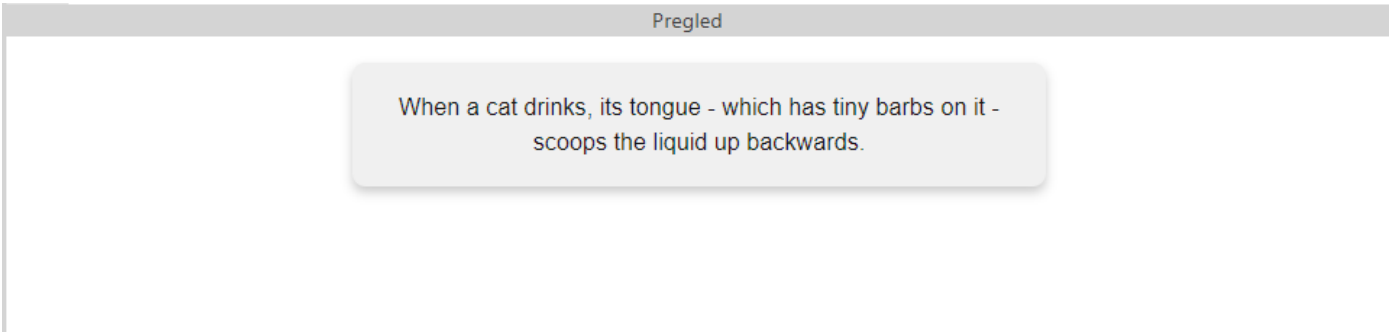
Ako rješavate ovaj zadatak u EduCoderu, ugasite automatsku evaluaciju budući da bi vas servis mogao blokirati zbog prevelikog broja zahtjeva.

Radite svoj web blog i želite korisnicima prikazati slučajno odabrane činjenice o mačkama. Za to koristite **TheCatAPI** servis koji pruža besplatne slučajne činjenice o mačkama. Potrebno je koristeći `fetch` API dohvatiti podatke s **TheCatAPI** servisa i ispisati slučajnu činjenicu o mačkama.

- https://catfact.ninja/fact

Kada ste uspješno dohvatili činjenicu o mačkama, pohranite ju u varijablu `catFact`. Dodajte novi HTML `<div>` element s ID-om `cat-fact` u vaš HTML dokument. Koristeći DOM manipulaciju dodijelite tekstualni sadržaj varijable `catFact` novom `<div>` elementu te dodajte stil po želji.

> Primjer rezultata



3.4 Promise objekt

U JavaScriptu, `Promise` objekt predstavlja **eventualni rezultat asinkronog procesa** i njegovo konačno stanje (**rezoluciju** ili **odbijanje**). `Promise` objekt može biti u jednom od tri stanja:

- **Pending**: Inicijalno stanje, očekuje se rezolucija ili odbijanje

- **Fulfilled**: Operacija je završena uspješno
- **Rejected**: Operacija je završena s greškom

`Promise` objekt ima tri metode: `then()`, `catch()` i `finally()`.

- `then()` metoda se koristi za rukovanje rezolucijom,
- `catch()` metoda se koristi za rukovanje odbijanjem.
- `finally()` metoda se koristi za izvršavanje kôda nakon što se `Promise` završi, bez obzira na rezultat.

Iako mnogima predstavlja muke, za potpuno razumijevanje asinkronog programiranja u JavaScriptu, `Promise` objekt je ključan. Dodatno, potrebno je razumijeti koncepte koje smo prošli u gradivu ovog kolegija, kao što su funkcije, callback funkcije, arrow funkcije, objekti, konstruktori, JSON format i sl.

`Promise` objekt možemo napraviti pozivanjem njegovog konstruktora, a kao argument prosljeđujemo callback funkciju s dva argumenta: `resolve` i `reject`.

`resolve()` se koristi za rezoluciju, `reject` za odbijanje.

Primjer izrade `Promise` objekta:

```
const promise = new Promise((resolve, reject) => {
  const uspjeh = true; // simulacija uspješne operacije, u pravilu je ovo rezultat neke
asinkrone operacije

  if (uspjeh) {
    resolve('Operacija je uspješna!'); // označi operaciju kao uspješnu
  } else {
    reject('Operacija nije uspješna!'); // označi operaciju kao neuspješnu
  }
});
```

U gornjem primjeru, `promise` je `Promise` objekt koji simulira uspješnu operaciju.

- Ako je `uspjeh` varijabla `true`, operacija je uspješna i rezolucija se poziva s porukom "Operacija je uspješna!".
- Ako je `uspjeh` varijabla `false`, operacija nije uspješna i odbijanje se poziva s porukom "Operacija nije uspješna!".

Metode `resolve()` i `reject()` se mogu pozvati samo jednom. Nakon što se `Promise` objekt rezolva ili odbije, ne može se ponovno rezolvati ili odbiti. Ove metode mogu primiti argumente:

```
resolve(vrijednost); // uspješna rezolucija s danom vrijednošću u argumentu

reject(greska); // odbijanje s danom greškom u argumentu (razlogom za odbijanje)
```

Rezultat `Promise` objekta ne možemo direktno ispisati koristeći `console.log()`. Umjesto toga, koristimo `then()` i `catch()` metode za rukovanje rezolucijom i odbijanjem.

```
promise
  .then((rezultat) => { // arrow funkcija koja se poziva nakon rezolucije
    console.log(rezultat); // Operacija je uspješna!
  })
  .catch((greska) => { // arrow funkcija koja se poziva nakon odbijanja
    console.log(greska); // 'Operacija nije uspješna!'
  });
  .finally(() => {
    console.log('Kraj operacije'); // Kraj operacije
  });
```

Prilikom korištenja `fetch` API-a za dohvaćanje podataka s web poslužitelja, `fetch` funkcija vraća `Promise` objekt. Kao takav, možemo koristiti `then()` i `catch()` metode za rukovanje rezolucijom ili odbijanjem direktno!

Primjer 10

EduCoder šifra: `bored`

Napomena, ako radite u EduCoderu, ugasite automatsku evaluaciju budući da bi vas servis mogao blokirati zbog prevelikog broja zahtjeva (limit: 100 zahtjeva svakih 15 minuta).

Idemo složiti malu aplikaciju koja će nam pritiskom na gumb prikazati neku zanimaciju koju bi mogli raditi u slobodno vrijeme, kada nam je dosadno. Podatke ćemo dohvatiti s Bored API i to slučajnu aktivnost: `https://bored-api.appbrewery.com/random`.

Koristit ćemo `fetch` API za dohvaćanje podataka s web poslužitelja i `Promise` objekt za rukovanje rezolucijom i odbijanjem.

Primjer JSON objekta kojeg dobivamo s API-a:

```
{
  "activity": "Learn Express.js",
  "availability": 0.25,
  "type": "education",
  "participants": 1,
  "price": 0.1,
  "accessibility": "Few to no challenges",
  "duration": "hours",
  "kidFriendly": true,
  "link": "https://expressjs.com/",
  "key": "3943506"
}
```

Prvo ćemo složiti HTML strukturu:

```
<style>
body {
    font-family: Arial, sans-serif;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
```

```
    background-color: #f0f0f0;
}
h1 {
    color: #333;
}
#activity-container {
    margin-top: 20px;
    padding: 20px;
    background-color: #fff;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}
button {
    padding: 10px 20px;
    font-size: 16px;
    color: #fff;
    background-color: #007bff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}
button:hover {
    background-color: #0056b3;
}
</style>

<body>
    <h1>Što raditi kad nam je dosadno?</h1>
    <!-- Gumb za dohvaćanje aktivnosti (iskoristit ćemo event listener) -->
    <button id="fetch-activity-btn">Prikaži aktivnost</button>
    <!-- Container za prikaz aktivnosti (proširit ćemo ga DOM manipulacijom) -->
    <div id="activity-container"></div>
</body>
```

Prvi korak je dohvatiti gumb i dodati event listener koji će se pozvati pritiskom na gumb. To smo rekli da postižemo koristeći `addEventListener` metodu. Međutim, callback funkciju ćemo definirati izvana, a ona će biti upravo naša funkcija za dohvaćanje aktivnosti s API-a.

```
document.getElementById('fetch-activity-btn').addEventListener('click', fetchActivity); // dodajemo event listener na gumb i pozivamo funkciju fetchActivity
```

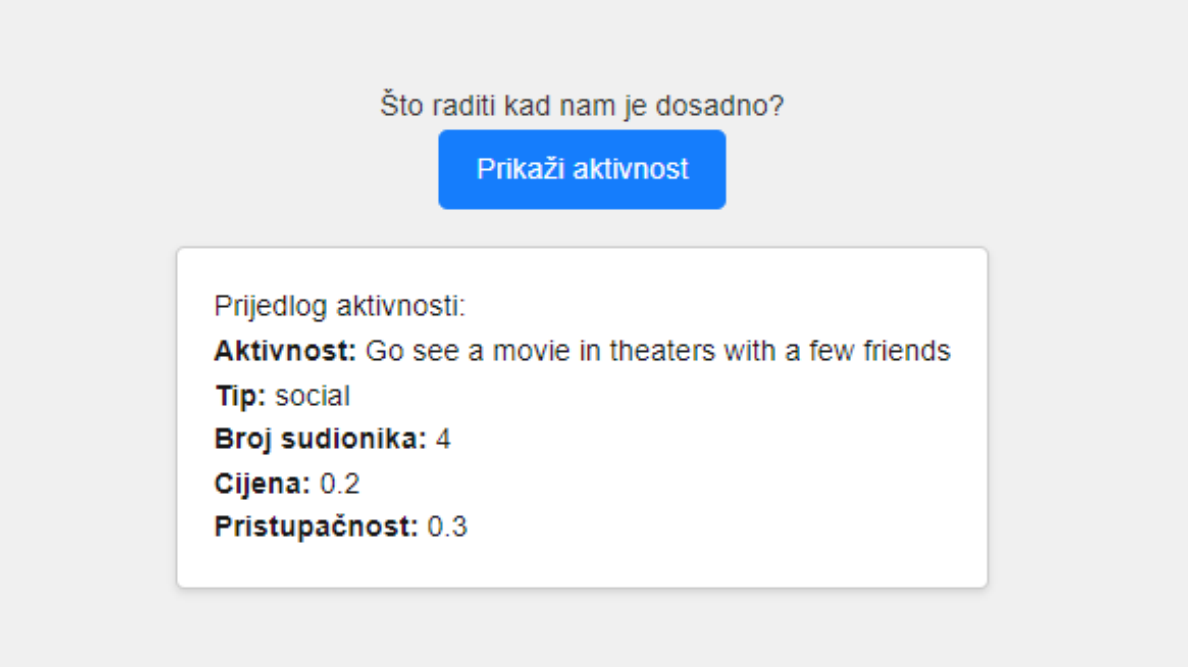
Sada definiramo funkciju `fetchActivity` koja će dohvatiti podatke s API-a i ispisati ih u HTML-u.
Unutar funkcije koristimo `fetch` API za dohvaćanje podataka s web poslužitelja. Nakon što se podaci dohvate, koristimo `then()` metodu za rukovanje rezolucijom i `catch()` metodu za rukovanje odbijanjem.

```
function fetchActivity() {
            fetch('https://bored-api.appbrewery.com/random')
                .then(response => response.json())
                .catch(error => {
                    console.log('Error fetching activity:', error);
                });
        }
```

Sada ćemo dodati `then()` metodu za rukovanje našim podacima koje je odradila prva `then()` metoda. U ovoj metodi ćemo napisati callback funkciju koja će ispisati aktivnost u HTML-u. Rekli smo da koristimo `.json()` metodu za parsiranje JSON podataka u JavaScript objekt onda kada podaci dolaze s web poslužitelja.

```
document.getElementById('fetch-activity-btn').addEventListener('click', fetchActivity);

function fetchActivity() {
    fetch('https://bored-api.appbrewery.com/random')
        .then(response => response.json())
        .then(data => {
            const activityContainer = document.getElementById('activity-container'); //
dohvaćamo container za prikaz aktivnosti i pohranjujemo varijable
            activityContainer.innerHTML = `
                <h2>Prijedlog aktivnosti:</h2>
                <p><strong>Aktivnost:</strong> ${data.activity}</p>
                <p><strong>Tip:</strong> ${data.type}</p>
                <p><strong>Broj sudionika:</strong> ${data.participants}</p>
                <p><strong>Cijena:</strong> ${data.price}</p>
                <p><strong>Pristupačnost:</strong> ${data.availability}</p>
            `;
        })
        .catch(error => {
            console.log('Error fetching activity:', error);
        });
}
```

✅Rezultat:



To bi bilo što se tiče `Promise` objekta. U sljedećem poglavlju ćemo pokazati kako koristiti `async` i `await` sintaksu za olakšavanje rada s asinkronim operacijama.
`Promise` objekt ćemo detaljnije obrađivati na budućim kolegijima.

3.5 Async/Await

`async` i `await` sintaksa je novija sintaksa u JavaScriptu koja olakšava rad s asinkronim operacijama. JavaScript je dodao podršku za `async` i `await` sintaksu u ECMAScript 2017 (ES8).

`async` funkcija vraća `Promise` objekt, dok `await` čeka na rezoluciju `Promise` objekta. `await` se koristi samo unutar `async` funkcija.

> `async` i `await` sintaksa je **syntactic sugar** za rad s `Promise` objektima. Ova sintaksa čini kôd čitljivijim i lakšim za održavanje, posebno kod dubokog gniježđenja callback funkcija (callback hell).

Uzmimo za primjer funkciju koja vraća novi `Promise` objekt koji se uspješno rezolvira.

```
function funkcija() {
    return Promise.resolve('Uspješna rezolucija');
}

funkcija().then(console.log); // Ispisuje: 'Uspješna rezolucija'
```

Isto možemo napisati koristeći `async` i `await` sintaksu. Rekli smo da `async` funkcija vraća `Promise` objekt, a `await` čeka na rezoluciju `Promise` objekta.

```
async function funkcija() { // async funkcija
    return 'Uspješna rezolucija';
}

console.log(await funkcija()); // Ispisuje: 'Uspješna rezolucija'
```

Osnovna `await` sintaksa izgleda ovako:

```
const rezultat = await promise;
```

`await` čeka na rezoluciju `promise` objekta. Kada se `promise` rezolva, `await` vraća rezultat. Ako `promise` odbije, `await` baca grešku.

Idemo primijeniti `async` i `await` sintaksu na primjeru s funkcijom `setTimeout` koja simulira asinkronu operaciju.

Pokazali smo kako se koristi `setTimeout` funkcija za simulaciju asinkronog procesa. U ovom primjeru, koristimo `setTimeout` funkciju unutar `async` funkcije i `await` čekamo na završetak procesa.

```
console.log('Start');

setTimeout(() => {
    console.log('Ovo je asinkroni callback');
}, 2000); // 2000 milisekundi = 2 sekundi

console.log('End');
```

Isto možemo napisati koristeći `async` i `await` sintaksu:

```
async function asinkronaFunkcija() {
    console.log('Start');

    await new Promise(resolve => setTimeout(resolve, 2000)); // čekamo 2 sekunde

    console.log('End');
}
```

U gornjem primjeru, `asinkronaFunkcija` je `async` funkcija koja čeka 2 sekunde prije nego što ispiše `End`. `await` čeka na rezoluciju `Promise` objekta koji se rezolvira nakon 2 sekunde.

Pokazat ćemo kako koristiti `async` i `await` sintaksu za dohvaćanje podataka s web poslužitelja koristeći `fetch` API. Kako više ne koristimo `then()` i `catch()` metode, ali svejedno imamo asinkrone pozive kroz `async` i `await` sintaksu, omotat ćemo kôd u `try` i `catch` blokove. `try` blok sadrži kôd koji može izazvati grešku, dok `catch` blok rukuje greškom.

Sintaksa:

```
try {
    // kôd koji može izazvati grešku
} catch (error) {
    // kôd koji rukuje greškom
}
```

Više o upravljanju iznimkama na budućim kolegijima.

Rješenje Primjera 10 s `async` i `await` sintaksom:

```
async function fetchActivity() {
    try {
```

```
        const response = await fetch('https://www.boredapi.com/api/activity'); // uočite
await i pohranu rezultata u varijablu
        const data = await response.json(); // .json() metoda vraća Promise objekt, koristimo
await za rezoluciju i pohranjujemo podatke u varijablu

        const activityContainer = document.getElementById('activity-container');
        activityContainer.innerHTML = `
            <h2>Prijedlog aktivnosti:</h2>
            <p><strong>Aktivnost:</strong> ${data.activity}</p>
            <p><strong>Tip:</strong> ${data.type}</p>
            <p><strong>Broj sudionika:</strong> ${data.participants}</p>
            <p><strong>Cijena:</strong> ${data.price}</p>
            <p><strong>Pristupačnost:</strong> ${data.accessibility}</p>
        `;
    } catch (error) { // catch block nam se ne mijenja mnogo
        console.log('Error fetching activity:', error);
    }
}
```

KRAJ! WOOHO! 🎉

Samostalni zadatak za vježbu 9

EduCoder šifra: `bitcoin`

Ako rješavate ovaj zadatak u EduCoderu, ugasite automatsku evaluaciju budući da bi vas servis mogao blokirati zbog prevelikog broja zahtjeva.

Student ste na Fakultetu informatike u Puli i polažete kolegij "Programiranje u skriptnim jezicima". Većinu svojeg slobodnog vremena provodite u EduCoder alatu marljivo rješavajući zadatke iz skripti. Pripremate se za ispit i jednostavno ne stignete pratiti vijesti o kriptovalutama iako ste čuli da je Bitcoin u posljednje vrijeme u velikom porastu. Kako EduCoder ima mehanizme za prevenciju varanja, ne možete otvoriti CoinMarketCap ili CoinGecko stranice kako biste provjerili trenutnu cijenu Bitcoina, a silno vas zanima koliko je Bitcoin vrijedan i je li pravo vrijeme za kupiti dip 🤑.

Prilikom rješavanja zadataka, došli ste i do zadnje skripte napokon i naučili koristiti `fetch` API za dohvaćanje podataka s web poslužitelja. Odlučili ste iskoristiti svoje novo znanje i malo nadograditi EduCoder, u kojem sad provodite većinu slobodnog vremena, s jednim widgetom koji će vam u svakom trenutku prikazivati trenutnu cijenu Bitcoina!

Za dohvaćanje trenutne cijene Bitcoina koristite `CoinDesk API: "https://api.coindesk.com/v1/bpi/currentprice.json"` koji vraća JSON objekt s podacima o trenutnoj cijeni Bitcoina u USD, EUR i GBP valutama.

Vaš zadatak je napraviti widget koji će prikazivati trenutnu cijenu Bitcoina u USD, EUR i GBP valutama. Widget treba sadržavati trenutnu cijenu Bitcoina u svim valutama, te datum i vrijeme kada su podaci ažurirani. Dodatno, widget treba sadržavati gumb za ručno ažuriranje podataka.

Koristite `fetch` API za dohvaćanje podataka s web poslužitelja i `async` i `await` sintaksu za olakšavanje rada s asinkronim operacijama.

> Primjer widgeta

Skripta

Programiranje u skriptnim jezicima (PJS)

Nositelj: doc. dr. sc. Nikola Tanković
Asistenti:

- Luka Blašković, univ. bacc. inf.
- Alesandro Žužić, univ. bacc. inf.

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

[1] JavaScript osnove

JavaScript je programski jezik često korišten u web programiranju. Inicijalno je bio namijenjen kako bi učinio web stranice interaktivnijima. Međutim, danas se koristi i za izradu server-side aplikacija, desktop aplikacija, mobilnih aplikacija itd.

Pregled

Trenutna cijena Bitcoina

USD: **69,294.087**

EUR: **63,808.213**

GBP: **54,530.081**

Last updated: May 20, 2024 20:40:33 UTC

Ažuriraj cijenu