



Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Predavanje 09 - HTML Document Object Model.

5. prosinca 2025.

Sastavio: Zvonimir Bujanović



Document Object Model (DOM)

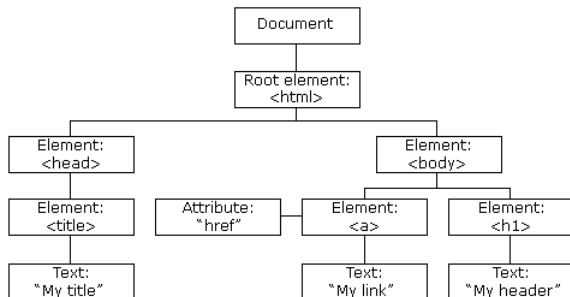
- API za HTML i XML dokumente. Neovisan o prog. jeziku, postoje implementacije u JavaScriptu, Pythonu, ...
- Reprezentira dokument u obliku stabla.
- Svaki čvor stabla je objekt koji ima svojstva, metode i eventualne funkcije kojima reagira na događaje.
- Omogućava programsko dodavanje, brisanje i modifikaciju pojedinih elemenata dokumenta – kako njegove strukture, tako i sadržaja i stila.

Ovdje dajemo samo letimičan pregled.

- Referenca $\rightsquigarrow$ [Mozilla Developer Network – DOM](#).
- Ranije se često koristila i biblioteka [jQuery](#).
- Alternativni (napredniji) pristup: [React](#), [Vue](#), [Svelte](#).

U nastavku, najčešće korištene metode su označene [crveno](#).

Reprezentacija HTML dokumenta pomoću DOM



```
1 <html>
2   <head>
3     <title>My title</title>
4   </head>
5   <body>
6     <a href="https://example.com">My link</a>
7     <h1>My header</h1>
8   </body>
9 </html>
```

JavaScript i DOM – Primjer dinamičkog stvaranja elementa

```
1 <script>
2   window.onload = function() {
3     let h1 = document.createElement( 'h1' );
4     let h1_text = document.createTextNode( 'Naslov!' );
5     h1.append( h1_text );
6     document.body.append( h1 );
7   }
8 </script>
```

Napomena:

- Kako se učitava HTML dokument, tako se izvršava kod skripti.
- Često moramo pričekati dok se cijeli HTML dokument ne učitava prije manipuliranja njegovim elementima (ako je JavaScript kod iznad elemenata kojim manipulira, neće se ispravno izvršiti).
- U gornjem primjeru, kod funkcije se poziva nakon što se dogodi *dogadjaj* `window.onload`, a to je upravo po učitavanju HTML dokumenta.

Node

- Bazna klasa kojom se reprezentiraju čvorovi u stablu DOM-a.
- Važnija svojstva:
 - `n.firstChild` – prvo dijete čvora `n` ili `null`.
 - `n.lastChild` – zadnje dijete čvora `n` ili `null`.
 - `n.nextSibling` – idući brat čvora `n` ili `null`.
 - `n.parentNode` – roditelj čvora `n` ili `null`.
 - `n.nodeName` – tip čvora `n` (na primjer "DIV").
 - `n.children` – Kolekcija (polje) djece čvora `n`. Ne uključuje komentare i text-čvorove (nego samo `Element`-e).
 - `n.children.length` – broj djece čvora `n`.
- Važnije metode (iz izvedene klase `Element`):
 - `n.append(c)` – dodaje čvor ili string `c` kao zadnje dijete čvora `n`.
 - `n.remove()` – briše čvor `n`.
 - `n.prepend(c)` – dodaje čvor ili string `c` kao prvo dijete čvora `n`.
 - `n.before(c)` – dodaje čvor ili string `nc` prije `n` (kao brata).
 - `n.after(c)` – dodaje čvor ili string `nc` poslije `n` (kao brata).
 - `n.replaceWith(c)` – zamjenjuje čvor `n` čvorom `c` u stablu.

Document

- Izvedena klasa iz `Node`, reprezentira korijen DOM stabla.
- Važnija svojstva:
 - `document.documentElement` – referenca na element `<html>` (node).
 - `document.body` – referenca na element `<body>` (node)
 - `document.title` – naslov dokumenta (string)
 - `document.URL` – kompletna adresa dokumenta (string)
 - `document.referrer` – adresa stranice s koje je link doveo na ovu
 - `document.lastModified` – datum zadnje izmjene dokumenta

Document

- Važnije metode:
 - `document.querySelector(q)` – vraća prvi element koji odgovara CSS selektoru zapisanom u stringu `q`
 - `document.querySelectorAll(q)` – vraća kolekciju elementa koji odgovaraju CSS selektoru zapisanom u stringu `q`
 - `e=document.createElement(t)` – kreira novi element tipa `t`, ne dodaje ga još u stablo
 - `e.textContent = 'Hello'` postavlja tekstualni sadržaj.
- Starije metode:
 - `document.getElementById(id)` – vraća referencu na element sa zadanim id-om
 - `document.getElementsByTagName(t)` – vraća kolekciju (ima `length`, `[]`) referenci na sve elemente zadanog tipa (`t="img", "div", "p"...`)
 - `document.getElementsByClassName(c)` – vraća kolekciju (ima `length`, `[]`) referenci na sve elemente zadane klase `c`
 - `t=document.createTextNode(s)` – kreira čvor tipa `TextNode` u kojem se nalazi tekst stringa `s`. `TextNode` su listovi u DOM stablu.

Element / HTMLElement

- Izvedena klasa iz `Node`, reprezentira pojedini element dokumenta.
- Važnija svojstva:
 - `e.id` – daje `id` atribut elementa `e` (kao string).
 - `e.classList` – manipulacija CSS klasama elementa `e`.
 - `e.classList.add('active');`
 - `e.classList.remove('hidden');`
 - `e.classList.toggle('open');`
 - `e.classList.contains('active');`
 - `e.innerHTML` – string koji sadrži HTML kod unutar elementa `e`.
 - `e.textContent` – dohvaća/postavlja "goli" tekst unutar elementa `e`, bez HTML tagova. Izbjegava XSS napad!
 - `e.clientHeight`, `e.clientWidth` – daje visinu (širinu) elementa u pixelima, uključuje padding ali ne border i margin.
- Važnije metode:
 - `e.getAttribute(a)` – vraća vrijednost (string) atributa `a`.
 - `e.removeAttribute(a)` – uklanja atribut `a` elementa `e`.
 - `e.setAttribute(a, s)` – postavlja string `s` kao vrijednost atributa `a` elementa `e`.

Često je potrebno uz element pohraniti neki dodatni podatak.
↪ dinamički atributi.

```
1 <button id="btn" data-podatak="42">Klikni me!</button>
```

Do tog podatka u JavaScriptu dolazimo ovako:

```
1 let btn = document.querySelector( '#btn' );
2 let podatak = btn.dataset.podatak;
3
4 // Možemo dinamički spremiti i neki novi podatak.
5 // Vrijednost je uvijek tipa string.
6 btn.dataset.drugi = '128';
```

Element – pristup CSS-u

Pomoću JavaScripta moguće je čitati i mijenjati stil elementa.

- `e.style.color` – boja teksta elementa `e`
- `e.style.fontFamily` – font-family svojstvo elementa `e`
- Općenito, CSS svojstvo oblika `ime-svojstva` prelazi u `e.style.imeSvojstva`.
- Čitanje svojstva na ovaj način detektira samo ona CSS svojstva eksplicitno postavljena u html dokumentu, a ne u vanjskoj CSS datoteci! Na primjer, ovako:

```
1 <p style="color: red;" id="p1">Crveno</p>
```

- Do svojstva kakvo je zaista prikazano u browseru dolazimo ovako:

```
1 let p = document.querySelector( '#p1' );  
2 let style = window.getComputedStyle( p );  
3 let c = style.getPropertyValue('color'); // 'rgb(255,0,0)'
```

- Bolje: manipulirati klasama sa `e.classList` umjesto direktno stilom.

Pojedini HTML elementi imaju izvedene klase koje olakšavaju manipuliranje specifičnim svojstvima. Na primjer:

- `HTMLTableElement` ima svojstva: `rows`, `tHead` i metode: `insertRow()`, `deleteRow()`; vidi [detalje](#).
- `HTMLTableRowElement` ima svojstva: `cells` i metode: `insertCell()`, `deleteCell()`.
- `HTMLImageElement` ima svojstva: `width`, `height`, `src`; vidi [detalje](#).

Zadatak 1

Neka je zadan HTML kod kao u donjem primjeru (može biti po volji mnogo listi s po volji mnogo životinja).

Treba napisati skriptu koja će u konzolu ispisati tekst "Hello, *životinja*" za svaku od životinja, te ispisati ukupan broj životinja.

Velike životinje treba na web-stranici ispisati fontom od 24pt.

```
1 <html>
2   <head>...</head>
3   <body>
4     <ul>
5       <li>Pas</li>
6       <li>Mačka</li>
7       <li class="velika">Žirafa</li>
8     </ul>
9     <ul>
10      <li class="velika">Slon</li>
11      <li>Mrav</li>
12    </ul>
13  </body>
14 </html>
```

jQuery

- Biblioteka funkcija koja čini manipulaciju HTML dokumentima bitno jednostavnijim.
- Izuzetno popularna do pojave React/Vue/Svelte i sličnih biblioteka.
- Podržava i događaje (events), animacije, Ajax.
- Postoje dva izdanja biblioteke:
 - `jquery.js` – nekomprimirana, podržava debugiranje;
 - `jquery.min.js` – komprimirana, za produkcijsku upotrebu.
- Dokumentacija:
 - [TutorialsPoint JQuery](#)
 - [w3schools JQuery](#)
 - [Quick API Reference](#)
- Moderne aplikacije ipak preferiraju *vanilla* JavaScript, no smijete koristiti i jQuery. Primjeri i zadaci će biti riješeni na oba načina.

```
1 <head>
2   <title>...</title>
3   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.js">
4   </script>
5 </head>
```

- Sva manipulacija HTML DOM-om obavlja se preko funkcije `$`.
- Alternativno, umjesto `$` može se koristiti `jQuery`.
- `$(sel)` vraća objekt tipa `jQuery` koji sadrži kolekciju svih HTML elemenata koji odgovaraju CSS selektoru `sel`.

```
1 // kolekcija svih ul elemenata iz HTML-a
2 let uls = $( 'ul' );
3
4 // kolekcija svih elemenata klase veliki
5 let veliki = $( '.veliki' );
6
7 // kolekcija koja sadrži jedini element s id-om naslov
8 let naslov = $( '#naslov' );
9
10 // kolekcija svih paragrafa unutar sekcija klase clanak
11 // i svih naslova tipa h1
12 let para = $( 'section.clanak p, h1' );
```

Neka svojstva i metode klase jQuery (neka je `j` objekt tipa jQuery):

- `j.length` – vraća broj elemenata u kolekciji;
- `j.eq(i)` – vraća jQuery objekt za $(i+1)$ -vi element u kolekciji (ako $i < 0$, onda broji od kraja);
- `j.css(prop)` – vraća vrijednost CSS svojstva `prop` za prvi element kolekcije;
- `j.css(prop, val)` – svim elementima kolekcije postavlja CSS svojstvo `prop` na vrijednost `val`;
- `j.html()` – vraća HTML sadržaj koji se nalazi unutar prvog elementa kolekcije. Usporedi sa `j.text()`;
- `j.html(x)` – svim elementima kolekcije postavlja HTML sadržaj na `x`;

```
1 // Primjer ulančavanja poziva funkcija u jQuery:  
2 // Treći h1 naslov će biti plave boje i imat će sadržaj "Yes!"  
3 $('h1').eq(2).css('color', 'blue').html('Yes!');
```

Neka svojstva i metode klase jQuery (neka je `j` objekt tipa jQuery):

- `j.attr(a)` – vraća vrijednost atributa `a` za prvi element kolekcije;
- `j.attr(a, val)` – svim elementima kolekcije postavlja atribut `a` na vrijednost `val`;
- Postoji i `j.prop(a)` – razlika u detaljima.
- `j.val()` – vraća vrijednost (za elemente tipa `input`, `select`, `textarea`) za prvi element kolekcije;
- `j.val(x)` – svim elementima kolekcije postavlja vrijednost na `val`;

```
1 // Elementu (npr. input) s id-om ime postavlja vrijednost Mirko
2 $('#ime').val('Mirko');
3
4 // Svim select elementima postavlja vrijednosti jabuke i kruške
5 $('select').val( ['jabuke', 'kruške'] );
```

Postoji još **puno funkcija**, npr:

- `$(html)` – stvara novi jQuery objekt sastavljen od zadanog html koda. Novi objekt nije uklopljen u stablo DOM-a.
- `j.append(x)`, `j.prepend(x)` – na kraj/početak HTML koda svih elemenata iz kolekcije se dodaje `x` (HTML ili jQuery objekt)
- `j.addClass(c)`, `j.removeClass(c)` – dodaje/uklanja CSS klasu `c` svim elementima kolekcije
- `j.children()` – kolekcija koja sadrži svu djecu svih elemenata iz `j`
- `j.next()` – kolekcija koja sadrži idućeg brata svih elemenata iz `j`
- `j.parent()` – kolekcija koja sadrži roditelje svih elemenata iz `j`
- `j.parents()` – kolekcija koja sadrži pretke svih elemenata iz `j`

```
1 let x = $( '<h1>Naslov!</h1>' ); // stvori novi element
2 $('body').prepend( x ); // stavi naslov na početak body-a
```

Riješite **Zadatak 1** koristeći jQuery.

Dodatno, procesirajte samo one ul-ove kojima je postavljena klasa `zivotinja`.

Stvorite novi element tipa ul na kraju web-stranice koji sadrži popis svih životinja u jednoj listi.

Dodajte uskličnik iza imena svake životinje u originalnim listama.

Zadatak 2

Na web-stranici se nalazi nekoliko paragrafa.

Sve paragrafe koji sadrže riječ "crveno", ispišite crvenom bojom.

Paragrafima koji sadrže riječ "žuto", postavite boju pozadine na žutu.

Ako želimo pričekati da se HTML dokument učita prije izvršavanja koda:

```
1 document.addEventListener( 'DOMContentLoaded', function () {  
2     // kod koji treba izvršiti kad se učita kod web-stranice  
3 } );  
4  
5 // jQuery: $(document).ready( function() { ... } );
```

Ako želimo pričekati da se učitaju i sve slike i drugi povezani objekti:

```
1 window.addEventListener( 'load', function () {  
2     // kod koji treba izvršiti kad se učita kod web-stranice  
3     // i sve slike i drugi povezani objekti na njoj  
4 } );  
5  
6 // jQuery: $(window).on( 'load', function() { ... } );
```

Browser Object Model

Document Object Model (DOM) $\rightsquigarrow$ manipulacija HTML dokumentima.
Browser Object Model (BOM) $\rightsquigarrow$ manipulacija browserom neovisno o prikazanom sadržaju.

BOM se sastoji od nekoliko objekata:

- **window** – glavni BOM objekt, (ne samo) za manipulaciju prozorom browsera.
- **window.location** – informacije o učitanoj dokumentu u browser. Uočiti: `window.location === document.location`, tj. `location` je svojstvo i `window` i `document` objekta.
- **window.navigator** – svojstva browsera, npr. verzija, operativni sustav, popis pluginova, jezik korisnikovog OS-a, ...
- **window.screen** – podaci o dimenzijama ekrana, npr. njegova širina i visina, broj boja, ...
- **window.history** – povijest browsanja unutar aktualnog prozora (tj. taba)

Browser Object Model: window

window = "globalni objekt", svaka funkcija ili globalna varijabla deklarirana sa **var** (ali ne i sa **let**, **const**) je njegov član

```
1 function hello() { console.log( 'Hello' ); }
2 var ime = 'Pero';
3
4 console.log( ime ); // "Pero"
5 console.log( window.ime ); // "Pero"
6 window.hello(); // ispiše "Hello" u konzoli
```

Otvaranje novih *modalnih* prozora:

- **alert**(poruka) – pop-up prozor u kojem piše poruka, klik na OK
- **confirm**(poruka) – pop-up prozor u kojem piše poruka, klik na OK ili Cancel, vraća true/false
- **prompt**(poruka) – pop-up prozor u kojem piše poruka, očekuje unos teksta i onda klik na OK ili Cancel, vraća utipkani string ili null
- Moderne aplikacije: vlastite implementacije modalnih prozora pomoću HTML/CSS ili npr. **Bootstrap**.

Rad s vremenskim intervalima (uči: JavaScript je *single-threaded*!):

- `id=setTimeout(f, t)` – izvrši kod zadan funkcijom `f` nakon čekanja od `t` milisekundi
- `id=setInterval(f, t)` – izvršavaj kod zadan funkcijom `f` svakih `t` milisekundi
 - Nekad je bolje koristiti `setTimeout`, i onda ponovno u funkciji `f` pozvati `setTimeout`.
 - Razlog je taj što samo izvršavanje funkcije `f` može trajati dulje od perioda `t`, pa `setInterval` ima nepredvidivo ponašanje.
- `clearTimeout(id)/clearInterval(id)` – otkaži štopericu `id`

Primjer 2

Sljedeći kod prikazuje štopericu unutar paragrafa s id-om p. Vrijeme u štoperici se osvježi svake sekunde.

```
1 <html>
2   <head></head>
3   <body>
4     <p id="p"></p>
5
6     <script>
7       function updateClock() {
8         let d = new Date();
9         let p = document.querySelector('#p');
10        p.textContent = d.toLocaleTimeString();
11      }
12
13      setInterval( updateClock, 1000 );
14    </script>
15  </body>
16 </html>
```

Zadatak 3

Postavite neku sliku na web-stranicu. Zatim napravite sljedeće:

- Neka se slika postupno povećava do svoje dvostruke veličine.
- Nakon toga, slika se smanjuje do svoje originalne veličine.
- Ovaj postupak se ponavlja.

Napomene:

- `x=getComputedStyle(img)` vraća širinu s mjernom jedinicom, npr. "200px".
jQuery: `x=img.css('width')`
- `x=img.clientWidth` vraća širinu bez mjerne jedinice, npr. "200".
jQuery: `x=img.width()`
- Postavljanje nove vrijednosti: `img.style.width=x` (treba mjerna jedinica).
jQuery: `img.width(x)` (bez mjerne jedinice)
- U ovom zadatku treba prvo pričekati da se učitaju sve slike, a ne samo HTML document.

Događaji (Events)

- Događaj (“event”) – omogućava izvršavanje JavaScript koda kada se pojavi određena akcija unutar browsera ili učitanoj dokumenta. Na primjer:
 - klik mišem na neki HTML element,
 - prelazak mišem preko nekog HTML elementa,
 - scroll sadržaja web-stranice,
 - utipkavanje teksta u neki `input` tag,
 - promjena veličine prozora browsera i slično.
- Pojedine funkcije se mogu “pretplatiti” na događaje:
 - Čim se neki događaj pojavi, automatski se pozivaju sve funkcije pretplaćene na njega.
 - Ovaj model se tradicionalno naziva *Observer pattern*.
 - Također: event-listener model – funkcija “isčekuje” događaj sve dok se ne dogodi.

Događaji unutar HTML-a

Svaki događaj na kojem pojedini HTML element reagira može imati pridruženu akciju preko odgovarajućeg HTML atributa.

```
1 <!-- Uoči jednostruke navodnike oko parametra od alert!  
2      this = element koji je generirao događaj (input). -->  
3 <input type="button" value="Klikni me"  
4       onclick="alert('Na gumbu pise ' + this.value);">  
5  
6 <script>  
7     function handler() { alert( 'Klik na button!' ); }  
8 </script>  
9 <input type="button" value="Klikni" onclick="handler();">
```

Ovaj pristup se izbjegava jer ima nekoliko mana:

- Dogodi se greška ako se događaj dogodi prije no što se učita kod funkcije koja na njega reagira.
- Ne poštuje se ideja razdvajanja sadržaja dokumenta (HTML) i dinamičkog aspekta dokumenta (JavaScript).

Prva alternativa: DOM Level 0 Event Handler

Funkcija koja reagira na događaj se dodaje kao član odgovarajućeg čvora u DOM stablu.

```
1 <input type="button" id="btn1" value="Klikni me!">
2
3 <script>
4     let btn = document.querySelector( '#btn1' );
5     btn.onclick = function() {
6         alert( 'Klik na gumb na kojem piše ' + this.value );
7     };
8 </script>
```

Funkcija koja reagira na događaj ("event handler") se uklanja ovako:

```
1 btn.onclick = null;
```

I dalje možemo imati max. jednu funkciju koja reagira na događaj.

Druga alternativa: DOM Level 2 Event Handler (Primjer 3)

Funkcija koja reagira na događaj se “pretplati” na odgovarajući događaj.

```
1 <input type="button" id="btn1" value="Klikni me!">
2
3 <script>
4   let btn = document.querySelector( '#btn1' );
5   btn.addEventListener( 'click', function() {
6       alert( 'Klik na gumb ' + this.value );
7   } );
8
9   let f = function() { alert( 'Još jedan prozor!' ); };
10  btn.addEventListener( 'click', f );
11 </script>
```

Funkcija koja reagira na događaj (“event handler”) se uklanja ovako:

```
1 btn.removeEventListener( 'click', f );
```

Ovo je ispravan pristup. Možemo imati više handler-a za isti događaj.

Treća alternativa: jQuery (Primjer 3)

jQuery objekt se pretplati na neki događaj ovako:

```
1 <input type="button" id="btn1" value="Klikni me!">
2
3 <script>
4     let btn = $( '#btn1' );
5     btn.on( 'click', function() {
6         // $(this) napravi jquery objekt od HTML elementa
7         // this na kojem je nastao event
8         alert( 'Klik na gumb ' + $(this).val() );
9     } );
10
11     let f = function() { alert( 'Još jedan prozor!' ); };
12     btn.on( 'click', f );
13 </script>
```

Funkcija koja reagira na događaj ("event handler") se uklanja ovako:

```
1 btn.off( 'click', f );
```

Dohvaćanje vrijednost iz HTML input elemenata

Ako je `e1` čvor koji odgovara:

- `<input type="text">`
- `<input type="password">`
- `<select>`
- `<textarea>`

onda je `e1.value` vrijednost koju je unio korisnik.

jQuery: `e1.val()`

Ako je `e1` čvor koji odgovara:

- `<input type="radio">`
- `<input type="checkbox">`

onda je `e1.checked` jednak `true` ili `false`.

jQuery: `e1.prop('checked')`

Selektor `input[name="obrok"]:checked` dohvaća odabrane čvorove.

Pomoću `e1.value` i `e1.checked` možemo i promijeniti vrijednost.

Zadatak 4

Napišite JavaScript program koji simulira kalkulator.

Jednostavna varijanta:

10	20	+	-	*	/	0.5
----	----	---	---	---	---	-----

Komplicirana varijanta (izazov: što kraći program!)

0			
7	8	9	+
4	5	6	-
1	2	3	*
0	=	C	/

- Događaji korisničkog sučelja (**UIEvent**)
 - **load** – kada se odgovarajući element u potpunosti učita. Najčešće se koristi za **window**: opali se kada se cijela web-stranica, zajedno sa slikama i ostalim elementima učita.
 - **resize** – kada se promijeni veličina prozora browsera ili se on minimizira/maksimizira.
 - **scroll** – kada se pomiče prikazani sadržaj prozora. Trenutni odmak od vrha prozora možemo dobiti pomoću **window.scrollY**.

Napomena:

- Za događaj *event*, odgovarajući HTML atribut ili DOM 0 svojstvo se zove *onevent*: **load** $\rightsquigarrow$ **onload**, **scroll** $\rightsquigarrow$ **onscroll**, ...

- Događaji fokusa (**FocusEvent**)
 - **focus** – kada element dobije fokus (npr. klikom miša na `input` možemo unositi tekst)
 - **blur** – kada element izgubi fokus (jer npr. kliknemo mišem na neki drugi)
- Događaji s mišem (**MouseEvent**)
 - **click**, **dblclick** – klik/dupli klik na lijevu tipku miša
 - **mousedown** – klik bilo koje tipke miša
 - **mouseup** – otpuštanje bilo koje tipke miša
 - **mouseenter**, **mouseleave** – ulazak/izlazak miša u područje koje zauzima neki element
 - **mousemove** – pomicanje miša iznad elementa

Vrste događaja

- Događaji s tipkovnicom (**KeyboardEvent**)
 - **keydown** – pritisnuta je tipka (događaj se ponavlja sve dok je pritisnuta)
 - **keyup** – tipka je otpuštena
- Događaj **input** – korisnik nešto unosi u textbox/textarea ili je kliknuo mišem drugi element tipa input.
- Postoje još razni drugi događaji, npr. **contextmenu** za desni klik miša, **orientationchange** za promjenu orijentacije mobilnog uređaja itd. Vidi **ovdje**.

Funkcije koje reagiraju na događaj mogu primiti jedan parametar. Taj parametar će se prilikom poziva funkcije automatski popuniti informacijama o događaju:

- Događaji s mišem će (na primjer) dati koordinate klika.
- Događaji s tipkovnicom će (na primjer) dati podatke o tome koja tipka je pritisnuta i je li pri tome bio pritisnut shift/alt/control.

Zadatak 5

Napravite web-stranicu na kojoj se nalaze text-box i gumb. U text-box korisnik treba unijeti datum u formatu `dd/mm/yyyy`.

Ako je u text-boxu datum ispravan, gumb treba biti zelen, na njemu treba pisati "Datum je OK!" i na njega se može kliknuti.

Ako datum u text-boxu nije ispravan, gumb treba biti crven, na njemu treba pisati "Datum nije OK!" i na njega se ne može kliknuti.

Uputa:

- Textbox treba reagirati na događaj `input`.
- Ako je svojstvo (*property*) `disabled` gumba postavljeno na `true`, onda gumb nije klikabilan; ako je postavljeno na `false`, onda je.

HTML element div (plavi pravokutnik) slijedi pokrete miša.

Ako uz pokrete miša držimo tipku $\mathbf{r}$, pravokutnik je crven.

Ako uz pokrete miša držimo neku drugu tipku, pravokutnik je zelen.

Zadatak 6

Napravite web-stranicu koja ispisuje koordinate na kojima se nalazi miš (događaj `mousemove` na elementu `body`).

Kada korisnik napravi klik, treba ispisati kojom tipkom miša je napravio klik (događaj `mousedown`; `event.button` daje tipku).

Na stranici se nalazi i jedan gumb. Kada korisnik pokuša doći mišem iznad gumba (događaj `mouseover`), gumb treba "preseliti" na neko drugo, slučajno odabrano mjesto.

Upute:

- `html{ height: 100%; }, body{ min-height: 100%; }`
- Za gumb stavite CSS svojstvo `position: absolute`; te mu udaljenost od lijevog ruba postavljajte pomoću CSS svojstva `left`, a od gornjeg pomoću CSS svojstva `top`.
- `Math.floor( Math.random() * 1000 )` – slučajan cijeli broj između 0 i 999.

Događaji za dinamički kreirane elemente

Za element koji kreiramo pomoću JavaScripta (a ne iz HTML-a) kažemo da je kreiran dinamički.

Ako takvom elementu nakon kreiranja dodamo *event handler*, to radi kako je i očekivano.

```
1 document.addEventListener( 'DOMContentLoaded', function()  
2 {  
3     let btn = document.createElement( 'button' );  
4     btn.textContent = 'Klikni me!';  
5     btn.classList.add( 'klasa' );  
6     btn.addEventListener( 'click', function() { alert( 'Klik!' ); } );  
7  
8     document.body.append( btn );  
9     // btn sad reagira na klik.  
10 } );
```

Događaji za dinamički kreirane elemente

Međutim, ako unaprijed definiramo *event handler*, pa onda kreiramo element, on neće reagirati na događaj!

```
1 document.addEventListener( 'DOMContentLoaded', function()
2 {
3     let btns = document.querySelectorAll( 'button.klasa' );
4     for( let btn of btns )
5         btn.addEventListener( 'click',
6             function() { alert( 'Klik!' ); } );
7
8     let btn = document.createElement( 'button' );
9     btn.textContent = 'Klikni me!';
10    btn.classList.add( 'klasa' );
11
12    document.body.append( btn );
13    // btn NE REAGIRA na događaj!!!
14 } );
```

Događaji za dinamički kreirane elemente

Za ovo ipak postoji rješenje: *event handler* treba dodati roditelju elementa kojeg ćemo kasnije stvoriti (ili *body*-ju).

Kažemo da je takav događaj *delegiran*.

```
1 document.addEventListener( 'DOMContentLoaded', function()
2 {
3     let body = document.querySelector( 'body' );
4     body.addEventListener( 'click', function(event)
5     {
6         // event.target je onaj element na kojem se zapravo dogodio događaj.
7         // matches prima CSS selektor i vraća true ako mu event.target pripada.
8         if( event.target.matches( 'button.klasa' ) )
9             alert( 'Klik!' );
10    } );
11
12    let btn = document.createElement( 'button' );
13    btn.textContent = 'Klikni me!';
14    btn.classList.add( 'klasa' );
15
16    document.body.append( btn );
17    // btn sad REAGIRA na događaj!!!
18 } );
```

Događaji za dinamički kreirane elemente: jQuery

Delegirani događaji se slično implementiraju i u jQuery-u.

```
1 $( document ).ready( function()
2 {
3     // Drugi parameter ('button.klasa') je CSS selektor
4     // podelemenata od body.
5     $( 'body' ).on( 'click', 'button.klasa',
6                     function() { alert( 'Klik!' ); } );
7
8     let btn =
9         $( '<button>Klikni me!</button>' )
10         .addClass( 'klasa' );
11
12     $( 'body' ).append( btn );
13     // btn sad REAGIRA na događaj!!!
14 } );
```