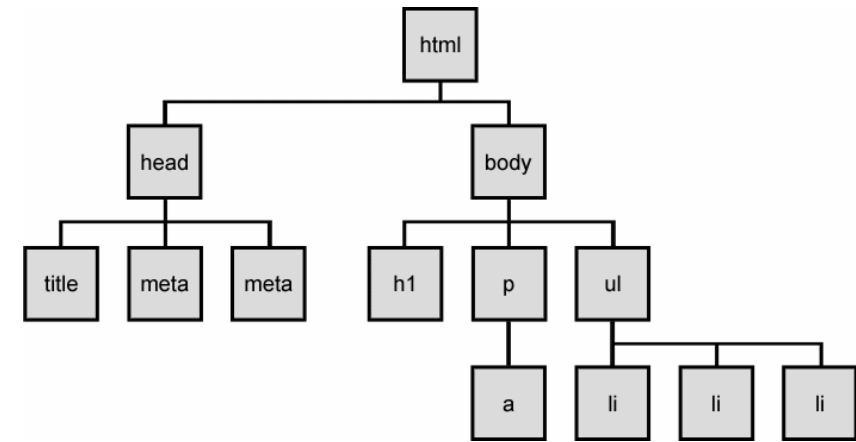


Što je HTML?

- HyperText Markup Language:
 - prezentacijski jezik (jezik označavanja) za kreiranje web stranica
 - aktualna verzija: HTML5
 - služi za definiciju pojedinih komponenti od kojih se sastoji web stranica →
 - struktura dokumenta (layout)
 - sadržaj dokumenta i njegove karakteristike



HTML elementi



- web stranica se gradi pomoću HTML elemenata
- tip elementa određuje njegova oznaka (tag)
 - označava sadržaj dokumenta i daje informaciju što sadržaj predstavlja
 - ključne riječi unutar uglatih zagrada → `<title>`, `<div>`, `<h1>`, `<p>`, ...
 - obično dolaze u paru → `<title>` `</title>`
 - neki elementi nemaju sadržaj ili ne mogu sadržavati pod elemente →
 - definiraju se samo s jednom oznakom → npr. `
` ili `
`

HTML atributi

- HTML elementi mogu sadržavati attribute
 - pružaju dodatne informacije o HTML elementu
 - uvijek se definiraju u početnoj oznaci
 - definicija:
 - naziv="vrijednost"
 - naziv
 - lista atributa: https://www.w3schools.com/tags/ref_attributes.asp

Anatomija HTML elemenata



Struktura i sadržaj HTML dokumenta

```
<!DOCTYPE html>  
<html lang="hr">  
<head>  
    <title>Naziv dokumenta</title>  
</head>  
<body>  
    Sadržaj dokumenta  
</body>  
</html>
```

Meta oznake

```
<meta charset="UTF-8" />
```

```
<meta name="author" content="Pero Perić" />
```

```
<meta name="keywords" content="ključne riječi" />
```

```
<meta name="description" content="opis stranice" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Elementi za označavanje teksta

Naslovi

<h1>Naslov 1. razine</h1>

<h2>naslov 2. razine</h2>

<h3>Naslov 3. razine</h3>

<h4>Naslov 4. razine</h4>

<h5>Naslov 5. razine</h5>

<h6>Naslov 6. razine</h6>

Elementi za označavanje teksta

Paragraf

`<p>Lorem ipsum sit amet</p>`

Unaprijed oblikovan tekst

`<pre>Lorem ipsum sit amet</pre>`

Novi red

`<p>Lorem ipsum
 sit amet</p>`

Vodoravna linija

`<hr />`

Elementi za označavanje teksta

Oblikovanje teksta

`Lorem ipsum dolor`

`Lorem ipsum dolor`

`<i>Lorem ipsum dolor</i>`

`Lorem ipsum dolor`

`<mark>Lorem ipsum dolor</mark>`

`<small>Lorem ipsum dolor</small>`

`Lorem ipsum dolor`

`<ins>Lorem ipsum dolor</ins>`

`_{Lorem ipsum dolor}`

`^{Lorem ipsum dolor}`

Elementi za označavanje teksta

Simbolička lista

```
<ul>  
  <li>Lorem ipsum</li>  
  <li>Lorem ipsum</li>  
</ul>
```

Brojčana lista

```
<ol>  
  <li>Lorem ipsum</li>  
  <li>Lorem ipsum</li>  
</ol>
```

Opisna lista

```
<dl>  
  <dt>Lorem ipsum</dt>  
  <dd>Lorem ipsum</dd>  
  <dt>Lorem ipsum</dt>  
  <dd>Lorem ipsum</dd>  
</dl>
```

Multimedijski sadržaj

- Kod dohvaćanja vanjske datoteke upisujemo putanju (URL) do nje:
 - apsolutna putanja
 - sadrži URL adresu datoteke smještene u vanjskoj web domeni
 - ne upravljamo datotekom koja nije smještena u našoj web mapi
 - da li kršimo autorska prava?
 - primjer:
 - relativna putanja
 - putanja od HTML dokumenta iz kojeg želimo dohvatiti datoteku do same datoteke
 - datoteka je smještena u našoj web mapi te mi njome upravljamo
 - kraći zapis putanje
 - primjer:

```

```

```

```



Multimedijski sadržaj

- Grafike

```

```

```
<picture>
```

```
  <source media="" srcset="" width="" height="" />
```

```
  <source media="" srcset="" width="" height="" />
```

```
  <img src="" alt="" width="" height="" />
```

```
</picture>
```

Multimedijski sadržaj

- Video sadržaj

```
<video controls width="560" height="320" poster="slike/promo.jpg">  
  <source src="video/promo-video.mp4" type="video/mp4" />  
  <source src="video/promo-video.ogv" type="video/ogg" />  
  <source src="video/promo-video.webm" type="video/webm" />  
</video>
```

```
<video src="video/promo-video.mp4" controls width="560" height="320" poster="slike/promo.jpg"></video>
```

Multimedijski sadržaj

- Audio sadržaj

```
<audio controls>
```

```
  <source src="audio/promo-audio.mp3" type="audio/mpeg" />
```

```
  <source src="audio/promo-audio.oga" type="audio/ogg" />
```

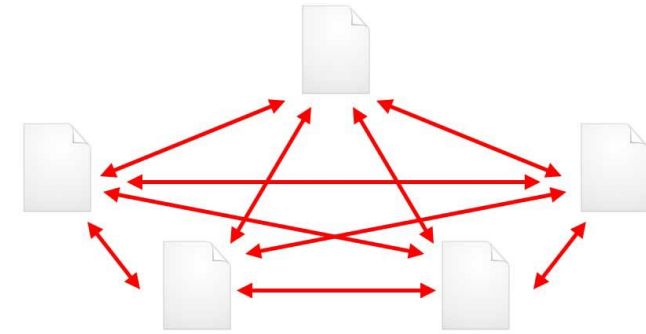
```
  <source src="audio/promo-audio.wav" type="audio/wav" />
```

```
</audio>
```

```
<audio src="audio/promo-audio.mp3" controls></audio>
```

Poveznice

- vrste poveznica:
 - unutarnje poveznice
 - `O nama`
 - vanjske poveznice
 - `Algebra`
 - poveznice prema elektroničkoj pošti ili za uspostavljanje tel. poziva
 - `Kontakt`
 - ` +123456789 `
 - poveznice unutar samog HTML dokumenta – sidra
 - `Skok na Poglavlje 3`
 - ...
 - `<h2 id="p3">Poglavlje 3</h2>`



Poveznice

- primjeri dodatnih atributa:

```
<a href="images/image.jpg" download></a>
```

```
<a href="https://www.algebra.hr" target="_blank"  
    hreflang="hr">Algebra</a>
```

```
<a href="ispis.asp?output=print"  
    media="print and (resolution:300dpi)">Ispiši</a>
```

```
<a rel="nofollow" href="http://www.example.com/">Hello World</a>
```

```
<a rel="noreferrer noopener" href="http://www.example.com/">  
    Hello World</a>
```

...

Rad s tablicama

- definiranje tablice
 - `<table></table>`, `<tr></tr>`, `<th></th>`, `<td></td>`, `<caption></caption>`
 - `<thead></thead>`, `<tbody></tbody>`, `<tfoot></tfoot>`
- atributi:

Naziv atributa	Opis	HTML elementi
colspan	Spajanje ćelija na razini reda (vodoravno).	<code><th></th></code> , <code><td></td></code>
rowspan	Spajanje ćelija na razini kolone ili stupca (okomito).	<code><th></th></code> , <code><td></td></code>
border	Dodavanje okvira tablici (bolje rješenje CSS).	<code><table></table></code> , ...

Rad s tablicama

Anatomija tablica

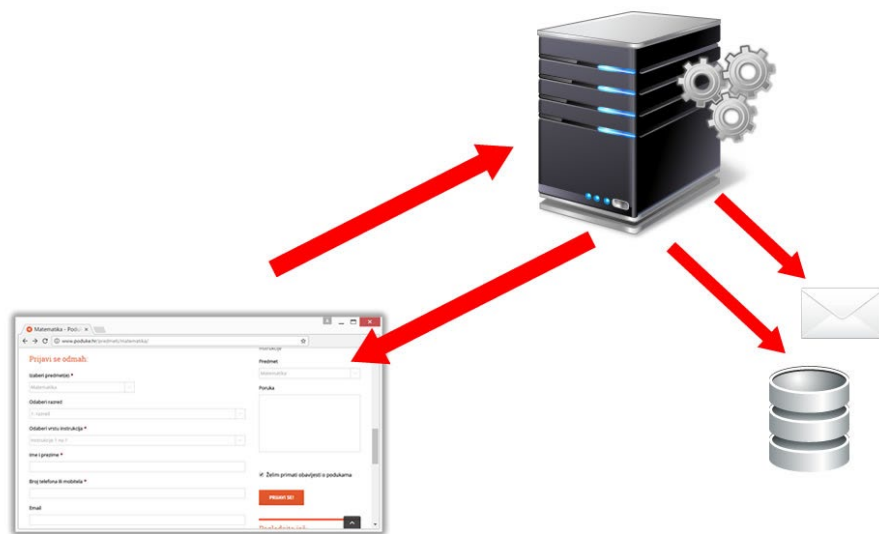
```
<table border="1">
  <caption>Naslov</caption>
  <thead>
    <tr>
      <th>1</th>
      <th>2</th>
      <th>3</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>4</td>
      <td>5</td>
      <td>6</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>7</td>
      <td>8</td>
      <td>9</td>
    </tr>
  </tfoot>
</table>
```

Obrasci

```
<form action="" method="post" enctype="text/plain" autocomplete="on">
```

...

```
</form>
```

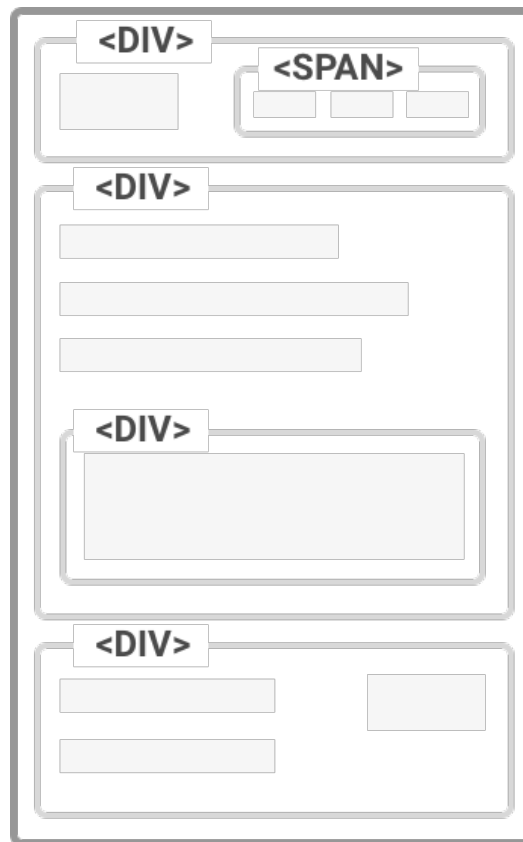


Primjeri elemenata obrasca

```
<label for=""></label>
<input type="text">
<textarea name="" id="" cols="30"
rows="10"></textarea>
<select name="" id="">
    <option></option>
    <option></option>
</select>
<button></button>
<datalist></datalist>
```

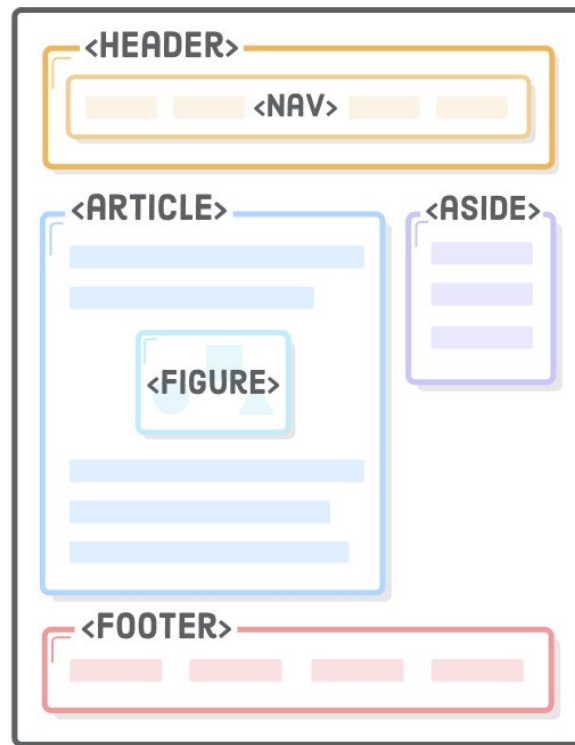
Strukturiranje sadržaja

```
<div></div>  
<span></span>
```

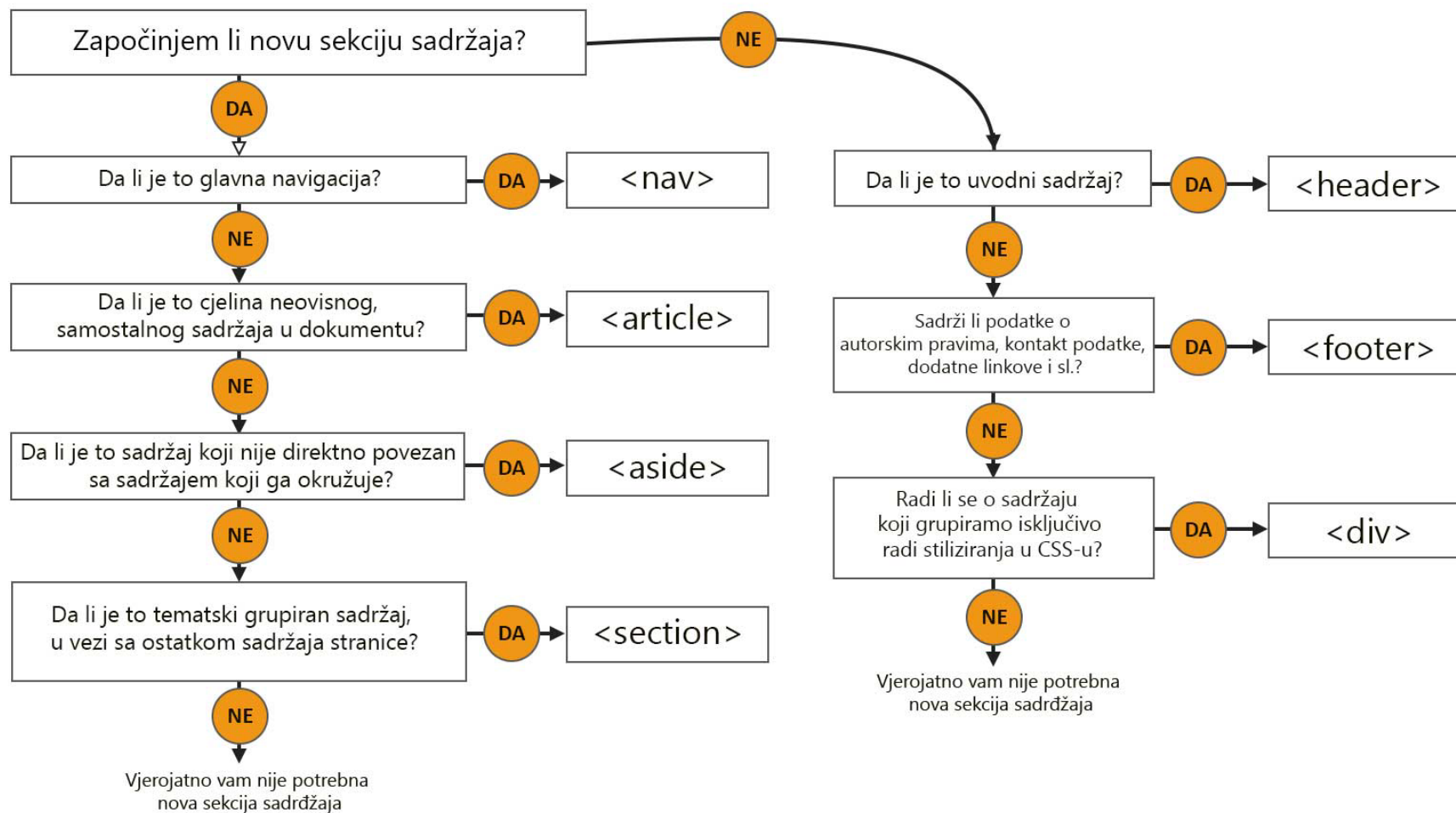


Strukturiranje sadržaja

```
<header></header>  
<footer></footer>  
<nav></nav>  
<main></main>  
<section></section>  
<article></article>  
<aside></aside>  
<figure></figure>  
<address></address>  
<details></details>  
<summary></summary>
```



Strukturiranje sadržaja



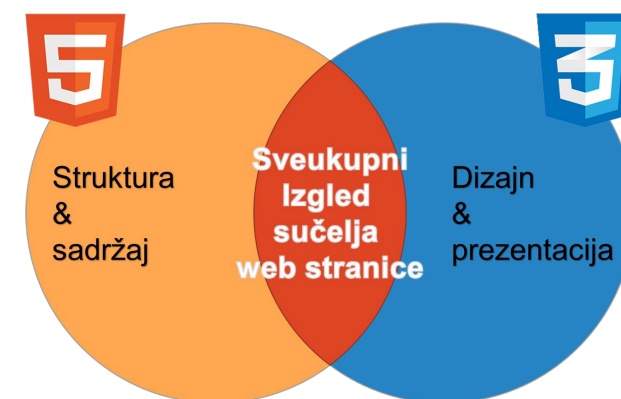
Temelji pristupačnosti internetskih stranica

- Pristupačnost web rješenja:
 - mogućnost da se web rješenje koristi prema potrebama i mogućnostima korisnika
- Postiže se:
 - primjenom semantičkih HTML elemenata
 - pravilnom definicijom HTML atributa (npr. "alt" kao zamjenski tekst za grafike, "role" za davanje značenja elementima, "lang" kao oznaka jezika sadržaja itd.)
 - osiguranjem tekstualnog sadržaja kao alternative za ne-tekstualni sadržaj (npr. tekstualni transkript za audio sadržaj)
 - omogućavanjem korisnicima da prilagode vlastito sučelje (veličinu znakova, boje itd.)
 - više: <https://www.w3.org/WAI/fundamentals/accessibility-principles/>

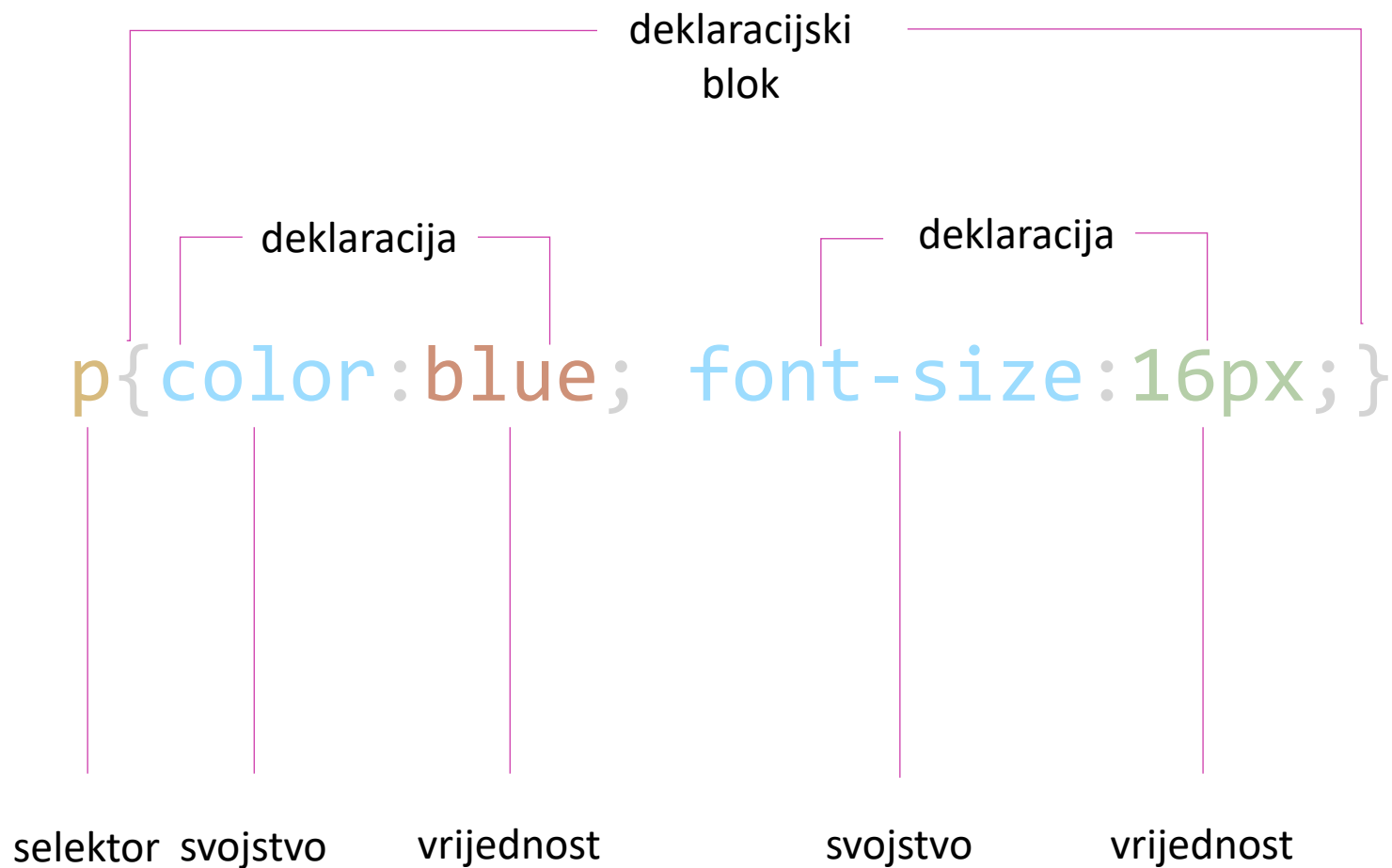
Što je CSS?



- Cascading Style Sheets:
 - računalni jezik koji koristimo za definiciju načina na koji će pojedini HTML elementi web stranice biti prikazani
 - izgled i format HTML elemenata web stranice
 - aktualna verzija: CSS3
 - nadopunjuje HTML
 - objavljen sadržaj dobiva skroman, unaprijed postavljen stil
 - daljnje uređivanje vrši se CSS jezikom
 - omogućuje razdvajanje sadržaja dokumenta (definiranog HTML-om) od prezentacije (stila) dokumenta
 - pojednostavljuje te unaprijeđuje razvoj web stranica



Pravila CSS-a



Kako primijeniti CSS na HTML?

- Linijski CSS

- unutar HTML oznaka → dodavanje atributa “style” HTML elementu
 - ne koriste se selektori
 - ograničava primjenu CSS pravila samo na element u kojemu su napisana
 - daje nepregledan kod
- primjer:

```
<p style="color:red;font-size:16px;">Odlomak teksta</p>
```

Kako primijeniti CSS na HTML

- Unutarnji CSS
 - pisanje u zaglavlju HTML dokumenta, unutar elementa `<style></style>`
 - traži upotrebu selektora
 - ograničava primjenu CSS pravila samo na HTML dokument u kojemu su napisana
 - uređivanje jedne web stranice na specifičan način

```
...  
<head>  
  <meta charset="UTF-8">  
  <title>HTML dokument</title>  
  <style>  
    p {  
      color: red;  
      font-size: 16px;  
    }  
  </style>  
</head>  
<body>  
...
```

Kako primijeniti CSS na HTML

- Vanjski CSS
 - pisanje u zasebnoj vanjskoj datoteci
 - traži upotrebu selektora
 - **optimalna metoda pisanja CSS-a**
 - omogućuje istovremeno uređivanje više HTML datoteka → manje koda
 - razdvajanje sadržaja dokumenta (definiranog HTML-om) od prezentacije dokumenta
 - vanjska CSS datoteka se sprema u memoriji web preglednika → bolje performanse
 - viša ocjena web rješenja kod indeksiranja od strane tražilica

Kako primijeniti CSS na HTML

- Vanjski CSS

...

```
<head>
```

```
  <meta charset="UTF-8">
```

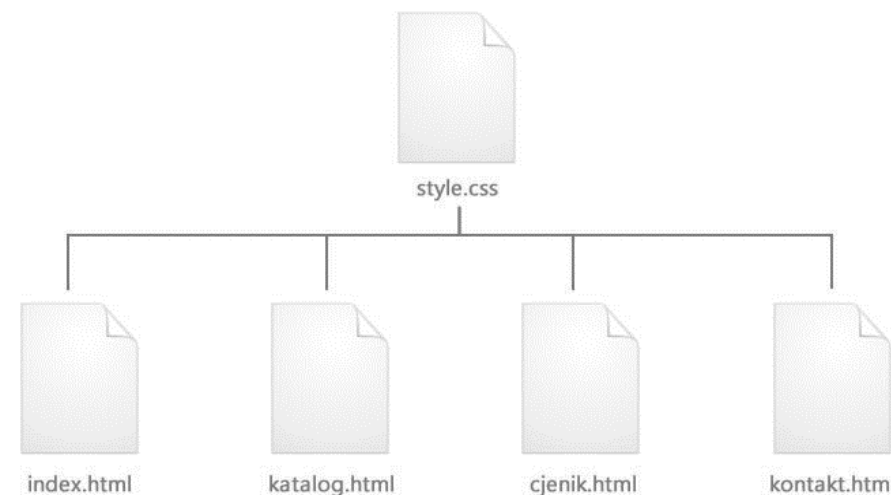
```
  <title>HTML dokument</title>
```

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

```
</head>
```

```
<body>
```

...



Minifikacija koda

- Prije objavljivanja web rješenja možemo minificirati CSS kod
 - s time ćemo smanjiti veličinu datoteke te ubrzati njeno preuzimanje sa poslužitelja

```
body {  
  background-color: azure;  
  font-family: Verdana, Geneva, Tahoma, sans-serif;  
  font-size: 18px;  
  color: #333;  
}
```

```
#zaglavlje img {  
  max-width: 100%;  
  height: auto;  
}
```

```
body{background-color:azure;font-family:Verdana, Geneva, Tahoma, sans-serif;font-size:18px;color:#333}#zaglavlje img{max-width:100%;height:auto}
```

Selektori

- selektori određuju na koji će HTML element/e CSS pravilo biti primijenjeno
 - ne pišemo ih ako koristimo linijski zapis pravila – unutar HTML oznake
- vrste selektora:
 - jednostavni selektori
 - selektori atributa
 - pseudoklase
 - pseudoelementi
 - kontekstualni selektori
 - grupni selektori
- detaljan opis svih tipova selektora: https://www.w3schools.com/cssref/css_selectors.asp

Selektori

Jednostavni selektori

- selektori HTML elmenata

```
<h1>Hello World!</h1>
```

```
<p>Lorem ipsum dolor sit.</p>
```

```
<ul>
```

```
  <li>Lorem ipsum</li>
```

```
  <li>Lorem ipsum</li>
```

```
  <li>lorem ipsum</li>
```

```
</ul>
```

```
h1 {  
  color: red;  
  font-size: 32px;
```

```
}
```

```
p {  
  color: blue;  
  font-size: 16px;
```

```
}
```

```
ul {  
  color: green;  
}
```


Selektori

Jednostavni selektori

- selektori atributa class

```
<ul>  
  <li class="stavka">Lorem ipsum</li>  
  <li class="stavka">Lorem ipsum</li>  
  <li class="stavka">lorem ipsum</li>  
</ul>
```

```
.stavka {  
  color: red;  
}
```

Selektori

Jednostavni selektori

- selektori atributa id

```
<p id="odlomak1">Lorem ipsum</p>
```

```
<p id="odlomak2">Lorem ipsum</p>
```

```
<p id="odlomak3">Lorem ipsum</p>
```

```
#odlomak1 {  
    color: red;  
}
```

```
#odlomak2 {  
    color: green;  
}
```

```
#odlomak3 {  
    color: blue;  
}
```

Selektori

Selektori atributa

[attr]

[attr=val]

[attr~=val]

[attr^=val]

[attr|=val]

[attr\$=val]

[attr*=val]

Selektori

Pseudoklase → stil se dodjeljuje prema stanju ili poziciji elementa

:first-child

:last-child

:nth-child()

:first-of-type

:last-of-type

:nth-last-child()

:link

:visited

:hover

:focus

:active

- omogućuju dodjeljivanje drugačijeg stila poveznicama, ovisno o njihovom stanju

Selektori

Pseudoelementi:

- omogućuju postavljanje sadržaja iza ili ispred sadržaja elementa
- omogućuju uređivanje dijela sadržaja elementa

::after

::before

::first-letter

::first-line

::selection

::marker

Selektori

Kontekstualni selektori (*Combinators*)

- omogućuju dohvaćanje željenog elementa s obzirom na kontekst u kojem se nalazi
 - stil se dodjeljuje elementima s obzirom na njihovu poziciju unutar HTML koda
- mogu uključiti sve tipove selektora
- primjeri – stil se dodjeljuje:

`div p {...}`

/ elementu p unutar elementa div */*

`div > p {...}`

/ elementu p kojemu je roditelj element div */*

`#sadrzaj > p a {...}`

/ elementu a unutar elementa p kojemu je roditelj element s id-om sadrzaj*/*

`div + p {...}`

/ svakom elementu p koji je smješten odmah iza elementa div unutar istog roditelja */*

`div ~ p {...}`

/ svakom elementu p koji je smješten iza elementa div unutar istog roditelja */*

Selektori

Grupiranje selektora

```
p, a, table{...}
```

```
p, .element1, #odlomak1{...}
```

```
[title], a[target="_blank"]{...}
```

```
div a, header a{...}
```

```
...
```

CSS variable

- Spremnik za vrijednost koja se može više puta iskoristiti unutar dokumenta
 - naziv joj započinje sa znakom "--"
 - pristupa joj se preko funkcije var()

```
:root {  
  --blue: #1e90ff;  
  --white: #ffffff;  
}  
  
body {  
  background-color: var(--blue);  
}
```


Uređivanje fonta znakova

- za uređivanje fonta znakova koristimo sljedeća svojstva:

Naziv svojstva	Opis
font-family	Određivanje obitelji fonta znakova – određuje stilska svojstva znakova.
@font-face	Moguće je koristiti fontove koji nisu instalirani na računalima korisnika.
font-size	Veličina znakova.
font-style	Prikaz nakošenih ili kurzivnih znakova.
font-weight	Debljina znakova.
font-variant	Veliki znakovi manjeg formata.
line-height	Visina linije znakova.

Kraći zapis → font: font-style font-variant font-weight font-size/line-height font-family;

Uređivanje teksta

- za uređivanje teksta koristimo sljedeća svojstva:

Naziv svojstva	Opis
color	Boja znakova.
text-align	Poravnanje sadržaja unutar elementa (znakovi i pod elementi reda).
text-decoration	Dekoriranje znakova.
text-transform	Prikaz slovnih znakova u velikom ili malom formatu.
text-shadow	Primjena sjene nad znakovima – moguće postaviti više sjena.
text-indent	Uvlačenje prve linije znakova.
word-spacing	Razmak između riječi.
letter-spacing	Razmak između znakova.
vertical-align	Vertikalno poravnanje pod elementa reda.
word-wrap	Prijelom riječi koje ne stanu u jedan red.

Uređivanje pozadine

- za uređivanje pozadine elemenata, koristimo sljedeća svojstva:

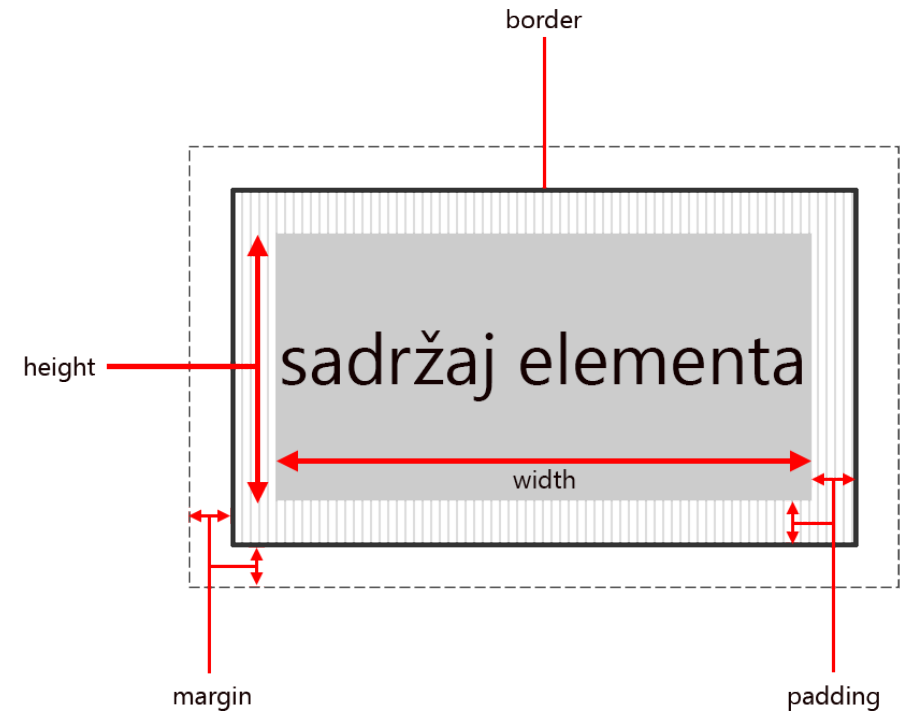
Naziv svojstva	Opis
background-color	Pozadinska boja.
background-image	Pozadinska slika (jedna ili više) te gradijent (linearni, radijalni i konusni).
background-repeat	Način ponavljanja pozadinske slike.
background-attachment	Da li je pozadinska slika fiksirana ili se pomiče s ostatkom sadržaja.
background-position	Pozicija pozadinske slike.
background-size	Veličina pozadinske slike.

Kraći zapis:

background: bg-image1 bg-repeat1 position1/bg-size1 bg-attachment1, bg-color bg-image2 bg-repeat2 position2/bg-size2 bg-attachment2;

Box model

- prikazuje strukturu HTML elementa
- HTML element može sadržavati:
 - sadržaj
 - tekst, slika ili drugi oblik sadržaja
 - ispunu
 - područje oko sadržaja, preuzima pozadinu elementa kojemu pripada
 - okvir
 - uokviruje ispunu i sadržaj
 - marginu
 - prazno područje oko okvira
 - transparentna – na nju ne utječe pozadina elementa kojemu pripada



Podešavanje dimenzija elementa

- za postavljanje dimenzija elementa koristimo sljedeća svojstva:

Naziv svojstva	Opis
width	Širina područja sadržaja elementa.
height	Visina područja sadržaja elementa.
max-width / max-height	Maksimalna širina / visina područja sadržaja elementa.
min-width / min-height	Minimalna širina / visina područja sadržaja elementa.
box-sizing	Promjena zadanog načina računanja dimenzija elementa.

Podešavanje ispune, okvira i margine

- za postavljanje ispune, okvira i margine koristimo sljedeća svojstva:

Naziv svojstva	Opis
padding	Dimenzije ispune.
padding-[top, right, bottom, left]	Dimenzije pojedinih strana ispune.
border-width	Debljina okvira.
border-style	Stil okvira.
border-color	Boja okvira.
border-[top-, right-, bottom-, left-]-svojstvo	Podešavanje pojedinih strana okvira.
margin	Dimenzije margine.
margin-[top, right, bottom, left]	Dimenzije pojedinih strana margine.

Kako su elementi prikazani

- dvije osnovne skupine HTML elemenata:
 - blok elementi
 - zauzima cijelu dostupnu širinu roditelja – može se utjecati na širinu i visinu
 - iza i ispred elementa slijedi novi red
 - moguće je utjecati na sve strane ispune i margine

Kako su elementi prikazani

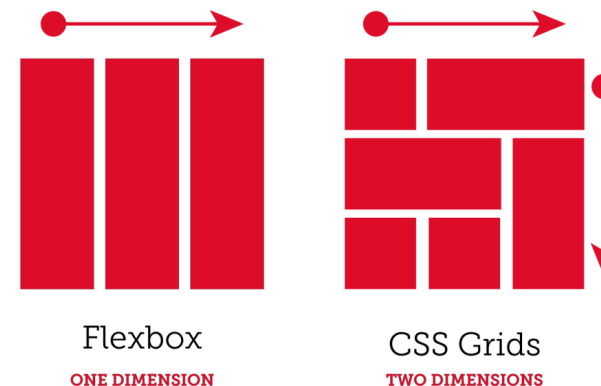
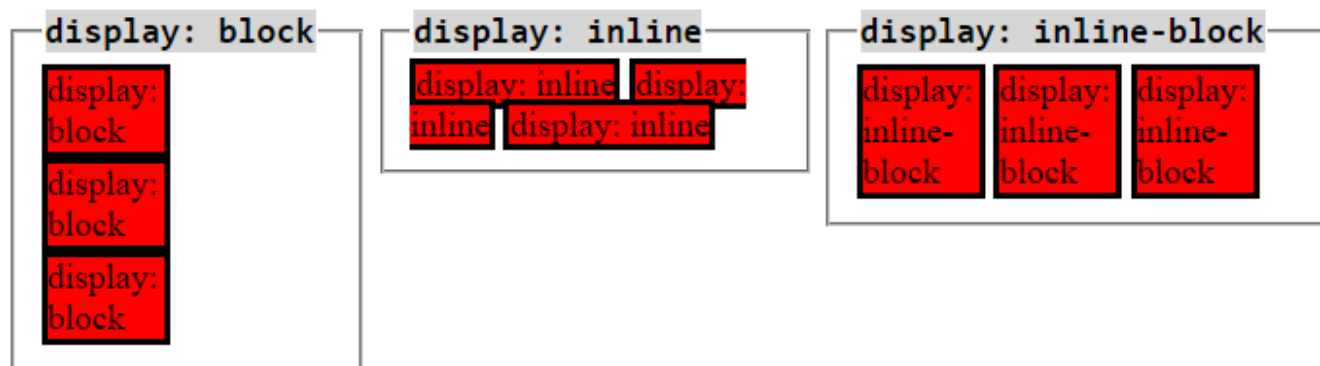
- dvije osnovne skupine HTML elemenata:
 - elementi reda
 - zauzima samo potrebnu širinu za prikaz sadržaja – ne može se utjecati na širinu i visinu
 - postoje iznimke → img, video ...
 - moguće je postaviti druge elemente reda u isti red
 - ne mogu sadržavati pod elemente iz grupe blok
 - moguće je utjecati samo na lijevu i desnu stranu ispunje i margine
 - gornja i donja ispuna se mogu postaviti (pozadina i okvir okolo), ali neće utjecati na pozicije susjednih elemenata
 - postoje iznimke → img, video ...

Lorem ipsum dolor sit amet [consect etuer](#) adipi scing elit
sed diam nonummy nibh euismod tinunt ut laoreet dolore magna
aliquam erat volut. Ut wisi enim ad minim veniam, quis nostrud exerci
tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo
consequat.

Kako su elementi prikazani

- promjena načina kako su HTML elementi prikazani:

Naziv svojstva	Opis	Moguće vrijednosti
display	Način prikazivanja elementa (promjena grupe kojoj pripada).	• inline / block / inline-block / flex / grid / none / ...



Uređivanje elemenata – dodatna svojstva

- dodatna svojstva:

Naziv svojstva	Opis
border-radius border-top-left-radius border-top-right-radius border-bottom-right-radius border-bottom-left-radius	Postavljanje zaobljenih kutova elementu.
box-shadow	Postavljanje sjene elementu – moguće postaviti više sjena.
overflow	Ponašanje elementa ako sadržaj prekoračuje njegove dimenzije.
clip-path	Isječak vidljivog dijela elementa.
opacity	Definiranje transparentnosti elementa.
resize	Omogućuje promjenu dimenzije elementa od strane korisnika.

Metode pozicioniranje elemenata

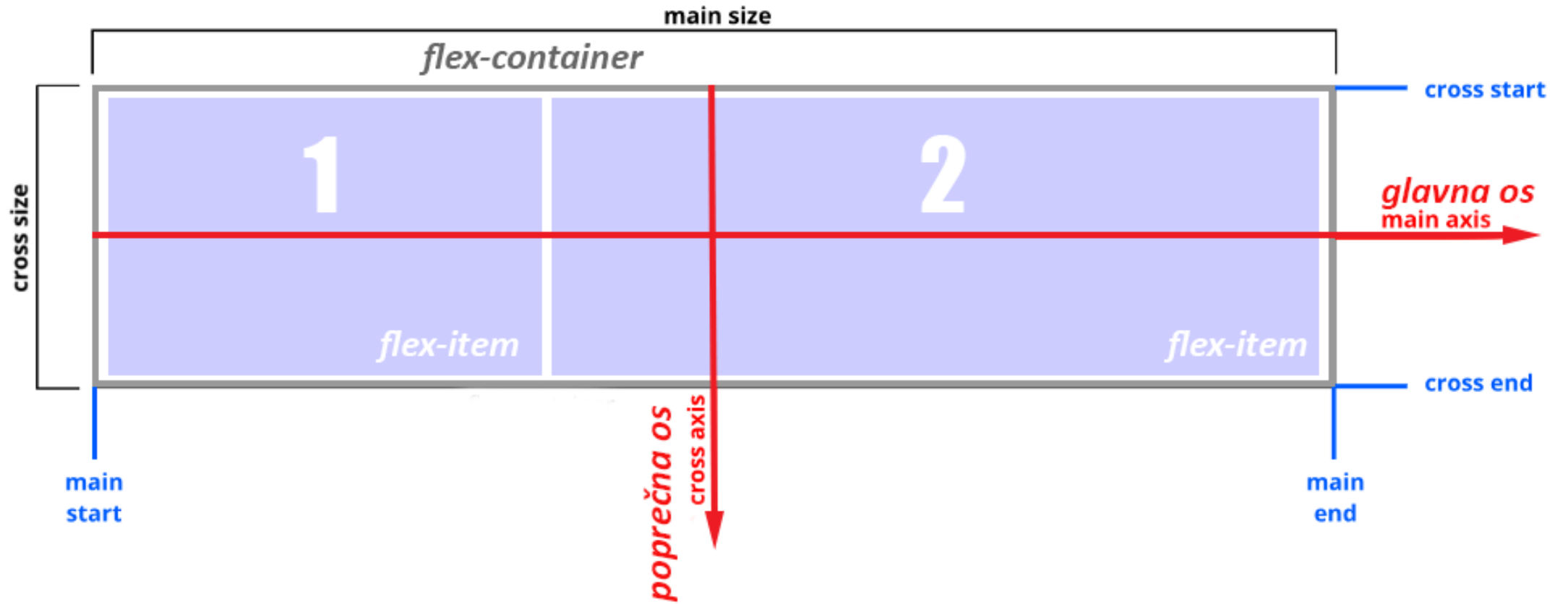
- metode pozicioniranja:
 - static – statično pozicioniranje (unaprijed zadano, pomaci nemaju efekta)
 - relative – relativno pozicioniranje
 - absolute – apsolutno pozicioniranje
 - fixed – fiksno pozicioniranje
 - sticky – ljepljivo pozicioniranje (relative + fixed)
- metodu pozicioniranja postavljamo svojstvom position:
 - primjer:

```
header {position: fixed;}
```

Metode pozicioniranje elemenata

- nakon postavljanja metode, elementi se pozicioniraju sljedećim svojstvima
 - horizontalni pomak (px, %, ...)
 - left
 - right
 - vertikalni pomak (px, %, ...)
 - top
 - bottom
 - preklapanje – jedan iza drugog (-10, -1, 0, 1, 10, ...)
 - z-index

Izrada layout-a: Flexible Box Layout



Izrada layout-a: Flexible Box Layout

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
  <div class="flex-item">6</div>
  <div class="flex-item">7</div>
  <div class="flex-item">8</div>
</div>
```

```
.flex-container {
  display: flex;
  flex-direction: column;
  flex-wrap: wrap;
  justify-content: space-between;
  align-content: space-between;
  height: 600px;
}
```

```
.flex-item {
  flex-basis: 32%;
  width: 24%;
}
.flex-item:nth-child(4n + 1) {
  flex-basis: 100%;
}
```



CSS transformacije

- omogućuju promjenu oblika, pozicije i veličine HTML elementa
- definicija:
 - transform: metoda();
 - transform: metoda1() metoda2() ...;
- postoje
 - 2D transformacije
 - 3D transformacije

CSS transformacije

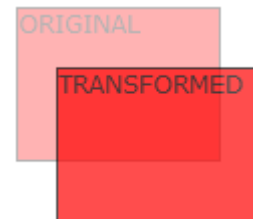
- Primjeri metoda transformacija:

Naziv metode	Opis
<code>translate(x,y)</code>	Element se pomiče po x i y osi (px, %, ...).
<code>rotate(deg)</code>	Element se rotira (deg) – 2D rotacija.
<code>scale(x,y)</code>	Element se skalira u zadanom omjeru.
<code>skew(deg)</code>	Element se iskrivljava po x i y osi (deg).
<code>rotatex(deg)</code>	Element se rotira oko x osi (deg) – 3D rotacija.
<code>rotatey(deg)</code>	Element se rotira oko y osi (deg) – 3D rotacija.
...	

- napomena: deg → stupnjevi

CSS transformacije

```
div {  
  transform: translate(50px, 100px);  
}
```



CSS tranzicije

- omogućuje prijelaz elementa iz jednog stila u drugi, koji se odvija postepeno u zadanom vremenskom trajanju
- prijelaz se aktivira:
 - pomoću pseudo klase
 - :hover → promjena stila kada pokazivač miša prijeđe preko elementa
 - :focus → promjena stila kada element dobije fokus
 - pomoću skripte (JavaScript)

CSS tranzicije

- definicija:

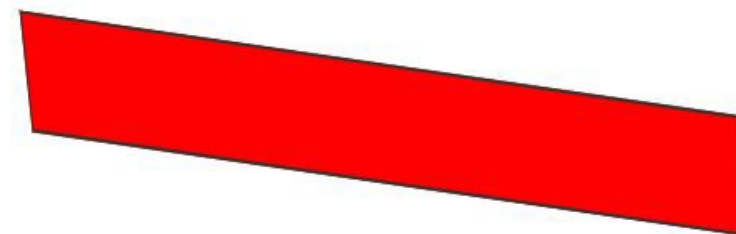
Naziv svojstva	Opis
transition-property	Svojstva nad kojima se primjenjuje tranzicija.
transition-duration	Vrijeme koliko dugo tranzicija traje (ms ili s).
transition-timing-function	Određuje krivulju brzine tranzicije.
transition-delay	Odgoda početka tranzicije (ms ili s).

Kraći zapis:

transition: property duration timing-function delay;

CSS tranzicije

```
div {  
  transition: background 1s ease-in-out, transform 0.5s linear;  
}  
  
div:hover {  
  background: red;  
  transform: scale(1.2, 2) skew(10deg, 5deg);  
}
```



CSS animacije

- više povezanih tranzicija
- omogućuju da se stil HTML elementa postepeno mijenja iz jednog u drugi
- CSS animacije se definiraju u dva osnovna koraka:
 - 1. korak
 - pravilo @keyframes
 - definira samu animaciju, njezine ključne kadrove i svojstva koja se animiraju
 - 2. korak
 - svojstvo animation
 - dodjeljuje animaciju određenom elementu i definira kako je animiran

CSS animacije

- svojstvo animation:

Naziv svojstva	Opis
animation-name	Naziv animacije.
animation-duration	Vrijeme trajanja ciklusa animacije (ms ili s).
animation-timing-function	Određuje krivulju brzine animacije.
animation-delay	Odgoda početka animacije (ms ili s).
animation-iteration-count	Broj koliko puta će se animacija izvršiti (infinite: neograničeno).
animation-direction	Smjer izvršavanja animacije.
animation-fill-mode	Određuje stil elementa u vremenu kada je animacija zaustavljena.
animation-play-state	Određuje da li se animacija izvršava.

Kraći zapis → `animation: name duration timing-function delay iteration-count direction fill-mode play-state;`

CSS animacije

```
@keyframes animacija1 {  
  0% {  
    top: 0px;  
    left: 0px;  
    background: red;  
  }  
  50% {  
    top: 100px;  
    left: 100px;  
    background: yellow;  
  }  
  100% {  
    top: 0px;  
    left: 0px;  
    background: red;  
  }  
}
```

```
div {  
  animation: animacija1 5s linear 2s infinite alternate;  
}
```

Responzivni web dizajn

- Responzivni dizajn predstavlja standard suvremenog web razvoja
- Web rješenje je responzivno:
 - kada se prilagođava karakteristikama uređaja na kojemu se pregledava
 - najčešće veličini ekrana uređaja
 - kada omogućava dobro korisničko iskustvo, bez obzira na uređaj koji se koristi za njegovo pregledavanje
 - kada se samo automatski prilagodi karakteristikama uređaja
 - izrađuje se samo jedna verzija web rješenja, što omogućava njegovo jednostavno održavanje i optimizaciju radu tražilica (SEO)


Responzivni web dizajn

- Viewport:
 - korisniku vidljivo područje web stranice – ovisi o karakteristikama uređaja na kojemu se stranica pregledava
 - jednako veličini prostora prozora web preglednika unutar kojeg se pregledava web stranica
 - HTML5 je omogućio kontrolu nad vidljivim područjem upotrebom <meta> elementa:
 - daje uputu web pregledniku kako da kontrolira dimenzije web stranice i njeno skaliranje

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```



Širina sučelja je jednaka
širini zaslona uređaja



Početna razina zumiranja stranice kod njenog
učitavanja u web preglednik

Media Queries

- Media Query:
 - pomoću @media upita određeni CSS blok se uzima u obzir samo kod ispunjenja zadanog uvjeta
 - omogućuje primjenu drugačijih stilova stranice na uređajima različitih karakteristika
 - uvjet za primjenu definiranog stila može biti:
 - tip medija
 - širina i visina vidljivog područja stranice (viewport)
 - širina i visina zaslona uređaja
 - orijentacija (portret ili pejzaž)
 - rezolucija
 - ...

Media Queries

- primjena određenog CSS bloka samo ako je zadovoljeni zadani uvjet:

```
@media not|only mediatype and (media feature) and|, (media feature) {  
    .my-code { ... }  
}
```

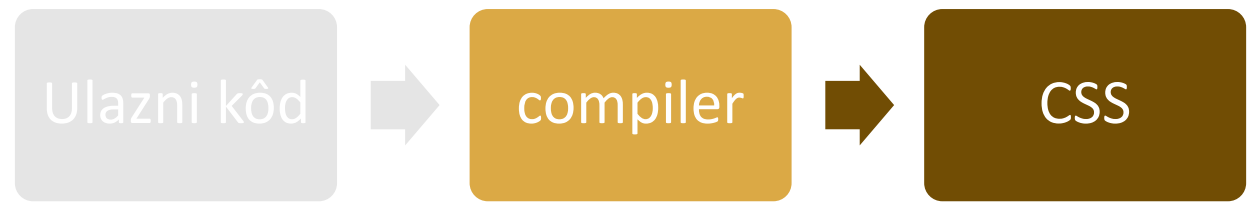
```
@media (media feature) {  
    .my-code { ... }  
}
```

- primjena cijele vanjske CSS datoteke samo ako je zadovoljeni zadani uvjet:

```
<link rel="stylesheet" media="mediatype and (media feature)" href="style.css" />
```

Predprocesori

- Predprocesori:
 - skriptni jezici koji proširuju zadane mogućnosti CSS-a
 - omogućuju upotrebu logike, varijabli, ugnježđivanja, nasljeđivanja, funkcija te matematičkih operacija
 - automatiziraju ponavljajuće zadatke, smanjuju broj pogrešaka, stvaraju dijelove koda za višekratnu upotrebu te osiguravaju kompatibilnost sa starijim verzijama web preglednika
 - najpopularniji: Sass, Less i Stylus



Variable

```
$red: #833;
```

```
body {  
  color: $red;  
}
```

Nesting

```
.markdown-body {  
  p {  
    color: blue;  
  }  
  
  &:hover {  
    color: red;  
  }  
}
```

Extend

```
.button {  
  ...  
}  
  
.push-button {  
  @extend .button;  
}
```

Composing

```
@import './other_sass_file';
```

Mixins

```
@mixin heading-font {  
  font-family: sans-serif;  
  font-weight: bold;  
}  
  
h1 {  
  @include heading-font;  
}
```

```
@mixin font-size($n) {  
  font-size: $n * 1.2em;  
}  
  
body {  
  @include font-size(2);  
}
```

```
@mixin pad($n: 10px) {  
  padding: $n;  
}  
  
body {  
  @include pad(15px);  
}
```

For loops

```
@for $i from 1 through 4 {  
  .item-#{$i} { left: 20px * $i; }  
}  
  
@mixin pad($n: 10px) {  
  padding: $n;  
}  
  
body {  
  @include pad(15px);  
}
```

Each loops

```
$menu-items: home about services contact;  
  
@each $item in $menu-items {  
  .photo-#{$item} {  
    background: url('images/#{$item}.jpg');  
  }  
}
```


Conditionals

```
@if $position == 'left' {  
  position: absolute;  
  left: 0;  
}  
@else {  
  position: static;  
}
```

Maps

```
$map: (key1: value1, key2: value2, key3: value3);
```

```
map-get($map, key1)
```

Što je JavaScript?



- JavaScript:
 - skriptni jezik → programski jezik koji omogućuje upravljanje aplikacijom u koju je ubačen
 - automatizira izvršavanje zadataka
 - može se koristiti i na strani klijenta i na strani poslužitelja. Mi učimo njegovu primjenu na strani klijenta:
 - osigurava interaktivnost i dinamičnost HTML stranice
 - njime se programira ponašanje web stranice
 - za napredniji rad dostupan je veliki broj razvojnih okvira i biblioteka (jQuery, React, ...)

Klasifikacija jezika

JavaScript je skriptni programski jezik visoke razine, dobar i za **objektno-orijentirano** i za **funkcijsko programiranje**.

Skup funkcija koje pruža ovisi o okolini u kojoj se koristi.

JavaScript jezik se razvija prema **ECMAScript specifikaciji**:

- opisuje sintaksu i ponašanje programskog jezika
- definira standard za implementaciju JavaScript jezika

Ugradnja JavaScripta u HTML

1. Linijski (inline), između oznaka `<script>` i `</script>`
2. **Iz vanjske datoteke definirane src atributom `<script>` elementa**
3. U atributu HTML elementa kojemu se dodjeljuje događaj (onclick, onmouseover itd.)
4. U URI-u koji koristi poseban javascript: protokol (blokirano od strane modernih web preglednika)

```
<script>
```

```
...
```

```
</script>
```

```
<script src="script.js"></script>
```

Sinkrone, asinkrone i odgođene skripte

Različiti tipovi učitavanja:

1. **Sinkrono** - uključenje JS skripte prije kraja `<body>` elementa - kad je 99% html-a već učitano
2. **Asinkrono** - stavljanjem **async** atributa na `<script>` element: loadanje skripte ne blokira dokument
3. **Odgođeno** - stavljanjem **defer** atributa na `<script>` element: ni loadanje ni izvođenje JS-a neće blokirati dokument

Vrijednosti

Literali su nepromjenjive vrijednosti, poput brojeva i teksta.

Varijable su promjenjive, i koriste se za spremanje podataka.

- koriste ključnu riječ **var** za deklaraciju
- koriste znak jednakosti **=** za dodjeljivanje vrijednosti
- puni izraz za varijablu x

var x = 1;

Deklaracija

Djelokrug (doseg) varijabli:

- globalni djelokrug (globalne varijable)
 - definiraju se izvan funkcija
 - mogu se koristiti u čitavom programu, zatvaranjem stranice brišu se
- djelokrug funkcije (lokalne varijable)
 - definiraju se unutar funkcije te se mogu koristiti samo unutar nje
 - izlaskom iz funkcije brišu se
- djelokrug bloka
 - definiraju se unutar bloka {} pomoću ključne riječi **let**
 - dostupne su samo unutar bloka u kojemu su definirane
 - moraju se deklarirati prije upotrebe te se ne mogu ponovo deklarirati

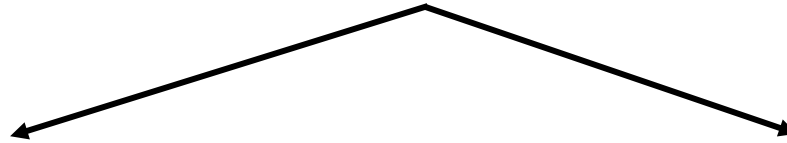
Imenovanje

Identifikatori

Identifikator je jednostavno ime za imenovanje varijabli i funkcija i za pružanje oznaka za određene petlje u JavaScript kodu. Ima par pravila kod imenovanja:

1. JavaScript razlikuje velika i mala slova
2. Naziv mora početi sa **slovom**, **podvlakom** (**_**) ili **znakom dolara** (**\$**)
3. Sljedeći znakovi mogu biti **slova**, **znamenke**, **podvlake** ili **\$**
4. Ključne riječi se ne smiju koristiti kao identifikatori

Tipovi podataka u JavaScriptu



Primitivni

- brojevi
- znakovni nizovi
- logičke vrijednosti
- undefined
- null

Objektni

- objekti
- nizovi
- funkcije

String

String je niz znakova (0 ili više) okružen navodnicima.

Navodnici mogu biti jednostruki ili dvostruki.

Brojanje pozicije znaka u stringu počinje od **nule**!

```
var myString = "";           // prazan string
myString = "Lorem ipsum";
myString = "  Lorem ipsum";  // string sa prazninama na početku
```

Svojstva i metode

length	Duljina stringa
charAt()	Vraća znak na poziciji
trim()	Miće praznine na početku i kraju stringa
indexOf()	Vraća poziciju stringa unutar stringa
search()	Traži da li se izraz nalazi unutar znakovnog niza – vraća poziciju podudaranja
replace()	Vraća novi string sa zamijenjenim izrazom
toUpperCase()	Pretvara stringu u sva velika slova
substring()	Vraća dio znakovnog niza između dvaju indeksa
concat()	Povezuje više nizova u jedan

Tipovi operatora

Po tipu operacije

Logički	&& !
Arifmetički	+ - * / % ** ++ --
Operatori dodjeljivanja	= += -= *= /= %= **=
Relacijski	== === != !== > < >= <= ?
Bitovni (bitwise)	& ~ ^ << >> >>>
Operator konkatencije	+
Ostali	delete, typeof, void, instanceof, in, ?., ??, ??=

Vrste izjava (kontrolne strukture)

Uvjetno grananje:

- izvršava se ili preskače blok naredbi ovisno o istinitosti zadanog uvjeta → kod se grana na više dijelova od kojih će se samo jedan izvršiti.

Petlje (iteracije):

- blok naredbi se izvršava više puta.

Skokovi (poput break, return, throw):

- naređuju interpreteru da skoči na neki drugi dio programa.

Uvjetno grananje

If, else if

```
// Jednostavan if
if (izraz) {
    // Izvrši kod u bloku #1
}
```

```
// If - else if
if (izraz) {
    // Izvrši kod u bloku #1
}
else if (izraz2) {
    // Izvrši kod u bloku #2
}
```

Uvjetno grananje

switch

Switch izjava je bolja alternativa kompliciranim if izjavama.

```
switch(n) {  
    case 1: // Execute code block #1. break;  
    case 2: // Execute code block #2. break;  
    default: // Execute code block #3. break;  
}
```

Smjer provjere



Petlje

while, do ... while

```
var count = 0;
while (count < 10) {
    console.log(count);
    count++;
}
```

```
var count = 0;
do {
    console.log(count);
    count++;
} while (count < 10);
```

for, for ... in


```
for (var count = 0; count < 10; count++) {
    console.log(count);
}
```

```
for (var i in nazivObjekta) {
    /* ... */
}
```


Petlje


break, continue

```
for (i = 0; i < 10; i++) {  
    if (i === 3) { continue; }  
    text += "The number is " + i + "<br>";  
}
```



continue izjava preskače obradu trenutnog člana petlje i nastavlja sa sljedećim

```
for (i = 0; i < 10; i++) {  
    if (i === 3) { break; }  
    text += "The number is " + i + "<br>";  
}
```



break izjava prekida petlju i izlazi iz nje

Niz (array)

Niz je sortirana zbirka vrijednosti.

Svaka pojedina vrijednost u nizu se naziva elementom niza. Može biti bilo kojeg JS tipa podatka, a sam niz je tipa objekt. Niz prepoznamo po uglatim zagradama ([...]).

```
var name1 = 'John';  
var name2 = 'Jane';  
var name3 = 'Mike';
```

```
var names = ['John', 'Jane', 'Mike'];  
  
console.log(names);  
console.log(typeof(names));
```

Elementi niza

Postavljanje i dohvaćanje

Za razliku od klasičnih objekata, elementi u nizu se dohvaćaju brojkom (ne-negativnim intergerom), tj. **indexom** elementa. Indexi elemenata počinju od **nule**.

Nizovi se mogu proširivati i smanjivati po volji.

```
var names = ['John', 'Jane', 'Mike'];  
var name1 = names[0];
```

```
names[3] = 'Holly';
```

```
for (var i = 0; i < names.length; i++) {  
    console.log(names[i]);  
}
```

```
names.forEach(function(value){  
    console.log(value);  
});
```

Svojstva i metode nad nizom

length	Vraća duljinu niza
push()	Dodaje element na kraj niza
pop()	Miće zadnji element niza
splice()	Na zadanoj poziciji dodaje ili briše željeni broj elemenata
filter()	Vraća novi niz popunjen s elementima početnog niza koji zadovoljavaju zadani uvjet
reverse()	Obrne redoslijed elemenata u nizu
sort()	Sortira elemente niza (default: po abecedi)
forEach() (ES5)	Iterira kroz niz, pozivajući funkciju koju odredite za svaki element
map() (ES5)	Prosljeđuje svaki element niza funkciji i vraća niz koji sadrži vrijednosti vraćene tom funkcijom
...	

Funkcija

Funkcija je blok JavaScripta koda koji izvršava neki zadatak.

U drugim jezicima funkcija se naziva procedura ili subrutina.

Funkcije koristimo jer njezin kôd možete **ponovo upotrijebiti**: definirajte ga jednom i koristite ga više puta. Možete koristiti isti kôd **više puta s različitim argumentima** za dobivanje različitih rezultata.

Definicija i pozivanje funkcije

DEFINICIJA

```
function nazivFunkcije() {  
    // nesto napravi  
}
```

POZIVANJE

```
nazivFunkcije();
```

“Return” ključna riječ

Svaka funkcija vraća vrijednost undefined, osim ako upotrijebimo ključnu riječ return. Vraća izračunatu vrijednost iz funkcije.

U slučaju izjave return prije kraja funkcije, ostatak funkcije je ignoriran (izvršavanje funkcije se prekida).

```
function nazivFunkcije() {  
    var value = 1;  
    return value;  
}  
console.log(nazivFunkcije());
```

```
function nazivFunkcije() {  
    var value = 1;  
    return value;  
    value += 1;  
}  
console.log(nazivFunkcije());
```

Parametri i argumenti

Definicija funkcije može (ali ne mora) sadržavati popis identifikatora, poznatih kao parametri, koji djeluju kao lokalne varijable za tijelo funkcije.

PARAMETRI

```
function udaljenostKoordinata(x1, y1, x2, y2)
{
    var dx = x2 - x1;
    var dy = y2 - y1;
    console.log(Math.sqrt(dx*dx + dy*dy));
}
```

ARGUMENTI

```
udaljenostKoordinata(1,1,3,4);
```


JavaScript OOP

- JavaScript je dizajniran da bude objektno-orijentiran jezik.
- Objektno-orijentirano programiranje:
 - metoda programiranja koja se temelji na upotrebi objekata u izradi računalnih programa
 - objekt → programska struktura koja se sastoji od
 - svojstva (podatkovne vrijednosti) koja ga opisuju
 - metoda (radnje), čijim izvršavanjem se njime upravlja
 - primjena objekata program čini razumljivijim, lakšim za održavanje, u pravilu ga skraćuje te omogućuje njegovo brže izvršavanje

Kreiranje objekta

Možemo kreirati objekt pomoću:

1. literala
2. operatora new
3. Object.create() metode
4. funkcije konstruktora

Kreiranje objekta

Možemo kreirati objekt pomoću **literals**:

- kreira se jedan primjerak objekta, zajedno s pripadajućim podacima
- za dohvaćanje svojstva unutar definicije objekta koristimo ključnu riječ **this**

```
var dog = {  
  breed: 'Golden retriever',  
  bark: function() {  
    return this.breed + ' says Wuf Wuf';  
  }  
};
```

```
var firstBreed = dog.breed;  
var whatDogSays = dog.bark();
```

Kreiranje objekata

Možemo kreirati objekt pomoću predloška - **funkcija konstruktora**:

- kreira se predložak, putem kojeg je moguće kreirati neograničen broj objekata istog tipa, ali drugačijih podataka
- objekt → primjerak (instanca) predloška, kreiran u vremenu izvršavanja programa
 - NAPOMENA: kako bi se približio drugim objektno orijentiranim programskim jezicima, JavaScript omogućuje upotrebu klasa umjesto funkcija konstruktora

Objektno orijentirano programiranje

Konstruktor je funkcija dizajnirana za inicijalizaciju novostvorenih objekata.

Riječ **new** ispred bilo koje funkcije pretvara poziv funkcije u poziv konstruktora.

```
function Dog() {  
    this.hasSit = false;  
    this.bark = function() {  
        return 'Wuf Wuf';  
    }  
    this.doSit = function() {  
        this.hasSit = true;  
    }  
}
```

```
var dog = new Dog();  
dog.bark();
```

Objektno orijentirano programiranje

Nasljeđivanje:

- dijeljenje zajedničkih svojstva i metoda između objekata različitih tipova (kreiranih iz različitih funkcija konstruktora)
- u JavaScript jeziku nasljeđivanje se realizira pomoću **prototip objekta**:
 - objekt koji je povezan sa svakom funkcijom i objektom
 - omogućuje:
 - implementaciju nasljeđivanja
 - dodjeljivanje svojstva i metoda svim postojećim objektima određenog tipa (kreiranim iz iste funkcije konstruktora)

Objektno orijentirano programiranje

Upotreba prototip objekta:

- definicija nasljeđivanja

```
function Konstruktor1(){  
  ...  
}
```

```
function Konstruktor2(){  
  Konstruktor1().call();  
  ...  
}
```

```
Konstruktor2.prototype = Object.create(Konstruktor1.prototype);
```

Tipovi objekata po podrijetlu

1. **Nativni** objekt je objekt ili klasa objekata definirana u ECMAScript specifikaciji (Array, Function, Date ...)
2. **Host** objekt je objekt definiran okruženjem domaćina, u našem slučaju web preglednika (npr. HTMLElement objekt)
3. **Korisnički definiran** objekt je svaki objekt kreiran izvršenjem JavaScript koda.

Date objekt

Core JavaScript uključuje konstruktor **Date()** za kreiranje objekata koji predstavljaju datume i vremena. Ovi objekti Date imaju metode koje pružaju API za jednostavno izračunavanje datuma. Objekti datuma nisu osnovna vrsta kao što su brojevi.

```
var d = new Date();  
var d = new Date(2018, 11, 24, 10, 33, 30, 0);  
var d = new Date("October 13, 2014 11:13:00");  
var d = new Date(-1000000000000);
```

Date objekt - metode

getDate()	Vraća redni broj dana u mjesecu
getHours()	Vraća sat iz trenutnog vremena
getFullYear()	Vraća godinu iz trenutnog datuma
setDate()	Postavlja redni broj dana u mjesecu
setHours()	Postavlja vrijednost sata
setFullYear()	Postavlja vrijednost godine
now()	Vraća broj milisekunda od 01.01.1970 00:00:00
toLocaleString()	Vraća vrijednost datuma i vremena u lokalnom formatu korisnika
...	

Math objekt

Pored osnovnih aritmetičkih operatora, JavaScript podržava složenije matematičke operacije kroz skup funkcija i konstanti definiranih kao svojstva objekta Math:

- objekt Math nije potrebno kreirati (instancirati)
 - postoji samo jedan primjerak objekta tog tipa
 - sve metode i svojstva su statička

Math objekt – primjeri svojstva i metoda

PI	Sadrži vrijednost konstante PI
SQRT2	Sadrži vrijednost drugog korijena broja 2
LN10	Sadrži vrijednost prirodnog logaritma broja 10

pow()	Vraća vrijednost x potenciranu s y → Math.pow(x,y)
sqrt()	Vraća drugi korijen broja
random()	Vraća slučajno generiran broj između 0 i 1 (isključujući 1)
floor()	Vraća niži cijeli broj
round()	Zaokružuje vrijednost na najbliži cijeli broj Za zaokruživanje na decimale: broj.toPrecision() ili broj.toFixed()

Regexp

Regular expressions

Opisuje uzorak znakovnog niza

- omogućuje pretragu traženog uzorka unutar znakovnog niza

Redovni izrazi obrasci su koji se koriste za podudaranje kombinacija znakova u tekstu. U JavaScript-u su i redoviti izrazi objekti.

```
var re = new RegExp('[žšđćč]', 'ig');  
var re = /[žšđćč]/ig;
```

Regexp – metode

exec()	Provjerava da li znakovni niz sadrži uzorak znakova – vraća zajedničku vrijednost
test()	Provjerava da li znakovni niz sadrži uzorak znakova – vraća istina ili laž

Za pretragu, moguća upotreba string metoda:

`znakovniNiz.search(uzorak)`

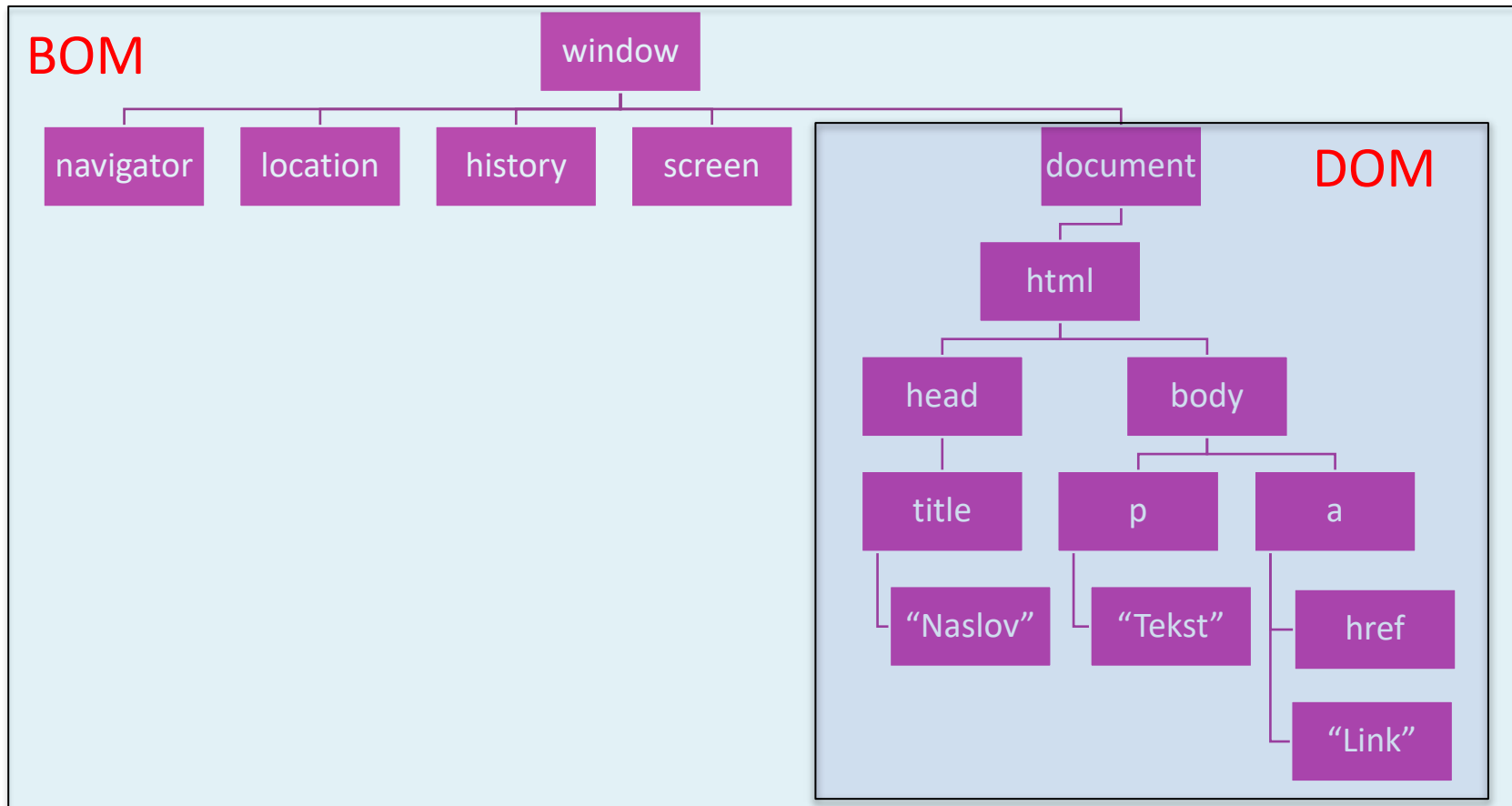
`znakovniNiz.match(uzorak)`

Browser Object Model (BOM)

- skupina unaprijed definiranih objekata koji stranici (JavaScript-u) omogućuju interakciju s web preglednikom
- omogućava JavaScript-u "razgovor" s web preglednikom
 - objekt preko kojeg to ostvarujemo je globalni objekt window
- ne postoji službeni standard
 - većina web preglednika ipak ima implementirano identično sučelje za komunikaciju s JavaScript jezikom

Browser Object Model (BOM)

- hijerarhijska struktura BOM objekata:



HTML DOM

Platforma i jezično neutralno sučelje koje omogućuje programima i skriptama dinamički pristup i ažuriranje sadržaja, strukture i stila dokumenta.

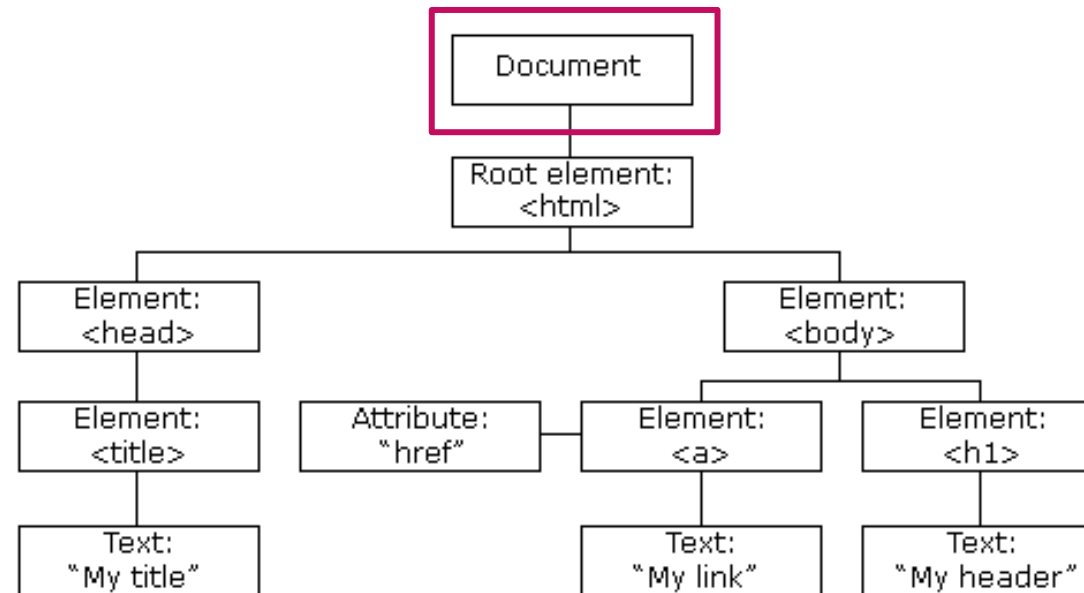
Definira:

- HTML elemente kao objekte
- svojstva svih HTML elemenata
- metode pristupa svim HTML elementima
- događaje za sve HTML elemente

Document objekt

Objekt dokumenta predstavlja našu web stranicu.

Ako želite pristupiti bilo kojem elementu HTML stranice, uvijek započinjemo s pristupom objektu dokumenta.



Selektiranje DOM elementa

Za mijenjanje DOMa ili elementa u DOMu prvo ga moramo 'pronaći' u dokument strukturi i selektirati.

DOM definira brojne **načine odabira elemenata**, koristeći:

- **id elementa**
- **tag elementa (tip)**
- **CSS klasu ili drugi CSS selektor**
- atribut name

```
document.getElementById("header-title");  
document.getElementsByTagName("li");  
document.getElementsByClassName("description");  
document.getElementsByName("gender");  
document.querySelector("#header-title");  
document.querySelectorAll(".description");
```

Kreiranje i brisanje elemenata

Kreiranje:

```
var f = document.querySelector("#footer");  
var d = document.createElement("div");  
f.appendChild(d);
```

Brisanje:

```
var f = document.querySelector("#footer");  
f.remove();  
  
/* ili */  
  
f.parentNode.removeChild(f);
```

Svojstva za navigaciju kroz DOM stablo

Svojstva	Opis
parentNode	Vraća roditelja zadanog člana
children	Vraća kolekciju (HTMLCollection) HTML elemenata kojima je zadani element roditelj
childNodes	Vraća kolekciju (NodeList) čvorova kojima je zadani čvor roditelj
firstChild / firstElementChild	Vraća prvi čvor / HTML element kojemu je zadani čvor / element roditelj
lastChild / lastElementChild	Vraća zadnji čvor / HTML element kojemu je zadani čvor / element roditelj
nextSibling / nextElementSibling	Vraća prvi čvor / HTML element nakon zadanog, na istoj razini DOM stabla
previousSibling / previousElementSibling	Vraća prvi čvor / HTML element prije zadanog, na istoj razini DOM stabla

Dodatne metode

Metoda	Opis
<code>createTextNode()</code>	Kreira tekstualni član DOM stabla
<code>insertBefore()</code>	Roditelju dodaje element prije odabranog podelementa
<code>insertAdjacentHTML()</code>	Ubacuje HTML kod na zadanu poziciju (afterbegin, afterend, beforebegin, beforeend)
<code>replaceChild()</code>	Roditelju zamjenjuje podelement
<code>remove()</code>	Uklanja odabrani element iz DOM stabla

Dohvaćanje ili postavljanje sadržaja

Tekstualni sadržaj:

```
var content = el.textContent;  
var content = el.innerText;
```

```
el.textContent = "Ima 4 rijeke";  
el.innerText = "Ima 4 rijeke";
```

HTML sadržaj:

```
var content = el.innerHTML;  
  
el.innerHTML = "Ima <span>4 rijeke</span>";
```

Dohvaćanje ili postavljanje vrijednosti atributa

Vrijednosti atributa HTML elemenata dostupne su kao **svojstva** **HTMLElement** objekata koji predstavljaju te elemente.

```
var img = document.querySelector("img");  
img.src = "ipsum.png";
```

```
img.getAttribute("width");  
img.setAttribute("height", "40");  
img.removeAttribute("height");
```


Dinamička promjena stilova – inline stilovi

Atribut **style** HTML elementa je njegov inline stil i poništava sve specifikacije stila u prije definiranom stylesheetu – ima **najviši prioritet** u CSS kaskadi kao inline deklaracija stila.

```
document.getElementById(id).style.property = new style;
```

```
element.style.font-family = "Arial"; ❌
```

```
element.style.fontFamily = "Arial"; ✓
```

```
element.style.float = "left"; ❌
```

```
element.style.cssFloat = "left"; ✓
```

Dinamička promjena stilova – mijenjanje klasa

Ponekad želimo umjesto inline stila promijeniti CSS klasu elementa. Za to koristimo **className** i **classList** svojstva elemenata.

```
var element = document.getElementById("id");  
element.className = "containerElement containerElement-left";  
  
console.log(element.classList);  
element.classList.add("containerElement", "containerElement-left");  
element.classList.remove("hide");  
element.classList.toggle("hide");
```

Animiranje CSS-a

Zbog svoje dinamičke i interaktivne prirode, JavaScript se često koristi za animiranje CSS svojstava.

```
var elem = document.getElementById("cube");  
var pos = 0;  
var id = setInterval(frame, 10);  
  
function frame() {  
    pos++;  
    elem.style.left = pos + 'px';  
}
```

Događaj

Kad god se nešto dogodi na našoj web stranici, možemo koristiti JavaScript u reakciji na taj događaj.

Događaji:

- aktivnosti unutar HTML dokumenta koje web preglednik može prepoznati
- mogu pokrenuti izvršavanje određene operacije definirane JavaScript jezikom
- radnja izvršena od strane web preglednika ili korisnika web stranice:
 - završi učitavanje HTML dokumenta
 - promijeni se vrijednost input elementa
 - klik na gumbić unutar web stranice
 - ...

Tipovi događaja

window	load, popstate, resize, ...
document	DOMContentLoaded
body, img, link, style, script	load
form	blur, change, focus, input, invalid, reset, submit, ...
Mouse	click, mouseenter, mouseleave, wheel, ...
Key	keydown, keypress, keyup
HTML5	drag, dragstart, scroll, play, pause, progress, ...
Mobile	touchstart, touchend

Postavljanje događaja

3 su načina postavljanja:

- 1) **inline**
- 2) kao svojstvo na elementu
- 3) addEventListener metoda

```
<input type="button" onblur="blur();" value="Click" />
```

Postavljanje događaja

3 su načina postavljanja:

- 1) inline
- 2) **kao svojstvo na elementu**
- 3) addEventListener metoda

```
<input type="button" value="Click" />
```

```
var button =  
document.querySelector("input");  
button.onblur = function() {  
    ...  
}
```

Postavljanje događaja

3 su načina postavljanja:

- 1) inline
- 2) kao svojstvo na elementu
- 3) **addEventListener metoda**

```
<input type="button" value="Click" />
```

```
var button = document.querySelector("input");  
function handleBlur() { ... }
```

```
button.addEventListener("blur", handleBlur);  
button.removeEventListener("blur",  
handleBlur);
```


ES6 (ECMAScript 6) novosti

1. Nove definicije varijabli (let, const)
2. Template literali
3. “Spread” operator
4. “Arrow” funkcije
5. Klase i moduli
6. Promises

...

Let deklaracija

Prednosti **let** deklaracije (naspram **var**):

1. Označuje da planiramo mijenjati tu varijablu
2. Opseg varijable je ograničen unutar bloka, ne funkcije

```
var x = 10;  
  
{  
    let x = 2;  
}
```

const deklaracija

Ima slično ponašanje kao let deklaracija (block scope), osim što se vrijednost varijable **ne može mijenjati**.

Funkcionalno programiranje izbjegava promjenu stanja i mijenjanje varijabli, tako da je const deklaracija puno bolja od let ili var u 99% slučajeva.

Template literals

Umjesto konkatencije stringova pomoću + operatora, koristimo backtickove.

```
const ime = 'Ivan';  
const pozdrav = `Moje ime je ${ime}`;  
console.log(pozdrav);
```

“Spread” operator (...)

Omogućava nam dekonstrukciju struktura kao što su nizovi ili objekti na njihove elemente.

```
function ispisiImena(prvo, drugo, trece) {  
    console.log(prvo);  
}  
  
const x = ['Ivan', 'Marija', 'David'];  
ispisiImena(...x);
```

“Rest” operator (...)

Omogućava nam definiciju funkcije sa nepoznatim brojem argumenata.

```
function vratiZbroj(...args) { // args -> niz
  let suma = 0;
  for (let arg of args) suma += arg;
  return suma;
}

console.log(vratiZbroj(1)); // 1
console.log(vratiZbroj(1, 2, 3)); // 6
```

“Arrow” funkcije

Omogućuju kraću sintaksu za definiranje funkcija (bez zagrada, ključnih riječi function i return, ili kombinaciji).

```
const bark = () => "Wuf wuf";  
  
myArray.forEach(element => console.log(element));  
  
myArray.forEach(element => {  
    console.log(this);  
    return element;  
}  
);
```

Klase

ES6 je dodao ključnu riječ **class**, kako bi jezik približili klasičnoj sintaksi OOP jezika (Java, C++), ali ništa se u pozadini nije pomijenilo.

Također, imamo par novi izraza koje nam pomažu u radu sa "klasama".

constructor - metoda koja se zove prilikom kreiranja objekta
extends - za nasljeđivanje, kao u klasičnom OOPu

```
class Golden extends Dog {  
    constructor(breed) {  
        this.breed = breed;  
    }  
}
```


Obećanja (Promises)

Obećanje je objekt koji predstavlja eventualni uspješni završetak ili neuspjeh asinkrone operacije

- rezultirati će jednom vrijednosti koja može biti ili riješena ili neriješena (odbačena)

```
new Promise((resolve, reject) => {
```

```
  ...  
  if (...) {  
    return resolve;  
  }  
  return reject;  
})  
.then()  
.catch();
```

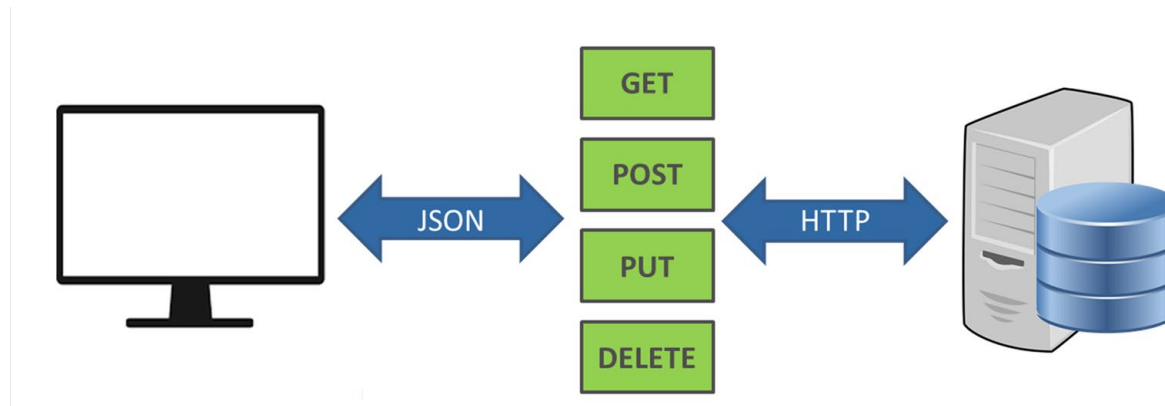
Egzekutor funkcija mora vratiti ili resolve ili reject callback funkciju.

then metoda reagira na stanje fulfilled Promise objekta → može primiti dva argumenta: callback funkcije za slučaj uspjeha ili neuspjeha

API

Application Programming Interface

Sučelje aplikacijskog programa koje se može definirati kao skup metoda komunikacije između različitih softverskih komponenti.



AJAX (Asynchronous JavaScript And XML)

- predstavlja standard u web razvoju
- omogućuje asinkronu komunikaciju klijenta i poslužitelja
 - razmjena podataka s web poslužiteljom odvija se u pozadini (bez prekidanja rada skripte u web pregledniku)
 - moguće je **slati podatke** na poslužitelj i **primati ih** s njega, nakon što je stranica već učitana, **bez da se prekida** njen rad u pregledniku klijenta
 - omogućuje asinkrono ažuriranje web stranica
 - moguće je ažurirati dio web stranice, bez ponovnog učitavanja cijele stranice
- izrađuje se više interaktivna web aplikacija



AJAX skripta

- primjer skripte koja šalje zahtjev poslužitelju te rukuje dobivenim odgovorom:

```
function loadData() {  
    var xhttp = new XMLHttpRequest();  
  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("content").innerHTML = this.responseText;  
        }  
    };  
  
    xhttp.open("GET", "data.php", true);  
    xhttp.send();  
}
```

Implementacija asinkronih operacija putem metode fetch

- Kao alternativu objektu tipa XMLHttpRequest za realizaciju asinkronih operacija možemo koristiti metodu **fetch()**
 - rad joj se temelji na objektu tipa Promise (obećanje)
 - daje nam veću fleksibilnost te pojednostavljuje definiciju skripte

Implementacija asinkronih operacija putem metode fetch

- Rad metode fetch() temelji se na objektu tipa **Promise** (obećanje)
 1. dohvaćanje JSON sadržaja iz odgovora
 2. prosljeđivanje dohvaćenih podataka funkciji koja će s njima rukovati
 3. ako Promise rezultira s neuspješnom obradom zahtjeva, opis greške šaljemo funkciji koja će s njime rukovati

```
fetch("URI")
```

```
.then(odgovor => odgovor.json()) // 1.
```

```
.then(podaci => rukujPodacima(podaci)) // 2.
```

```
.catch(greska => rukujGreskom(greska.toString())); // 3.
```

Minifikacija JS-a

Loadanje JavaScript koda (kao i HTML-a i CSS-a) se može smanjiti **minifikacijom skripti**, što smanjuje veličinu datoteke.

```
// return random number between 1 and 6
function dieToss() {
  return Math.floor(Math.random() * 6) + 1;
}
// function returns a promise that succeeds if a 6 is
tossed
function tossASix() {
  return new RSVP.Promise(function(fulfill, reject) {
    var number = Math.floor(Math.random() * 6) + 1;
    if (number === 6) {
      fulfill(number);
    } else {
      reject(number);
    }
  });
}
// display toss result and launch another toss
function logAndTossAgain(toss) {
  console.log("Tossed a " + toss + ", need to try
again.");
  return tossASix();
}

function logSuccess(toss) {}
```



```
function dieToss(){return Math.floor(6*Math.random()+1}function
tossASix(){return new RSVP.Promise(function(a,b){var
c=Math.floor(6*Math.random()+1;6===c?a(c):b(c))}function
logAndTossAgain(a){return console.log("Tossed a "+a+", need to try
again."),tossASix()}function logSuccess(a){}
```

Verzioniranje kôda

- Verzioniranje kôda služi za čuvanje i praćenje promjena u datoteci ili skupu datoteka
- Tijekom vremena, bilježe se promjene koje su se dogodile u datotekama
- Moguće je pozvati se na bilo koju prethodnu verziju
 - kronološki pregled promjena (tko je uveo promjene i kada)

Version Control System (VCS)

Korištenje sustava za kontrolu verzije donosi dvije velike prednosti:

- omogućuje lak oporavak u slučaju da je nešto pošlo po krivu
- uvelike olakšava suradnju više od jedne osobe na istom projektu

Vrste sustava za verzioniranje koda

Lokalni sustav za verzioniranje koda

Centralizirani sustav za verzioniranje kôda

Distribuirani sustav za verzioniranje kôda

Distribuirani sustav za verzioniranje kôda

- Kod distribuiranog sustava za upravljanje verzijama (**Git, Mercurial, Bazaar, Darcs**) klijenti na svom računalu imaju kopiju cijele baze podataka poslužitelja
- Ukoliko dođe do oštećenja podataka kod poslužitelja, podaci klijenata mogu se samo kopirati na novi (popravljeni) poslužitelj
- Jedna od glavnih prednosti DVCS-a je što klijent ne treba stalno biti spojen na poslužitelj

Repozitorij

- Repozitorij (*repository* - *repo*) je mjesto na kojem se pohranjuju sve datoteke jednog projekta
- Repozitoriji mogu biti lokalni ili na nekom poslužitelju (ovisno o sustavu za verzioniranje koji koristimo)
- Na jednom repozitoriju može raditi jedna ili više osoba
- Neki od popularnih repozitorija: GitHub, GitLab, Bitbucket

Git

- Git je **distribuirani sustav za verzioniranje koda** (DVCS)
- Omogućuje praćenje promjena u izvornom kodu tijekom razvoja softvera te koordiniranje rada među programerima
- Git ne nudi mehanizme kontrole pristupa, ali je dizajniran za rad s drugim alatima koji su specijalizirani za kontrolu pristupa (git repozitoriji - GitHub, GitLab, Bitbucket, ..)

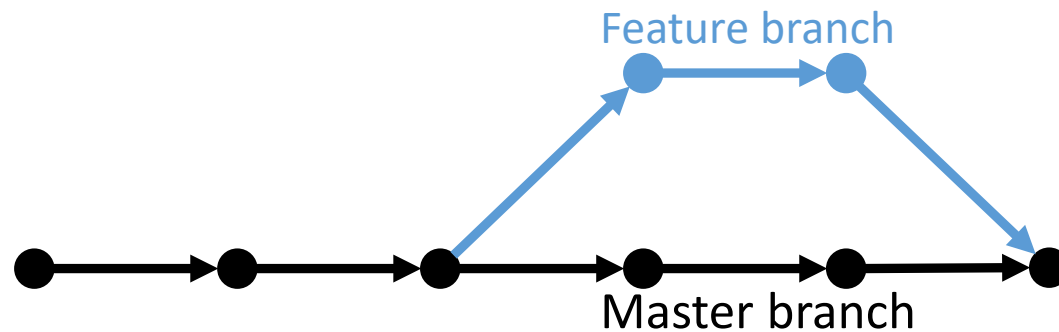
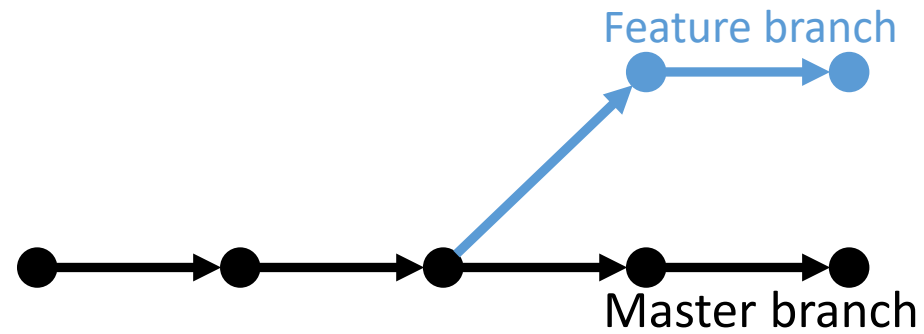
Git naredbe

- `git init` - inicijalizacija git repozitorija
- `git clone` - povlačenje (kloniranje) postojećeg repozitorija na lokalno računalo
- `git add` - dodavanje datoteka na Git repozitorij (ovom naredbom označavamo datoteke čije promjene želimo sačuvati i spremiti na udaljeni repozitorij)
- `git rm` - micanje datoteka s Git repozitorija

Git naredbe

- `git commit -m "message"` - lokalno spremanje trenutne verzije repozitorija (*message* - poruka, opis promjena koje želite spremiti)
- `git checkout` - promjena grane ili vraćanje na prijašnje stanje unutar trenutne grane
- `git push` - zapisivanje svih lokalnih *commitova* na udaljeni repozitorij (npr. GitHub)
- `git pull` - povlačenje promjena s udaljenog na lokalni repozitorij

Grananje



Git naredbe

- `git branch` - kreiranje vlastite *grane* Git repozitorija
- `git checkout <ime-brancha>` - naredba koja služi za promjenu brancha (ime-brancha označava granu na koju se želimo preseliti)
 - uz parametar `-b` omogućuje kreiranje nove grane i postavljanje u nju
- `git merge` - spajanje dvije Git grane
 - git merge se najčešće koristi kako bi se promjene iz nekog brancha zapisale u glavni (master) branch

Biblioteke i razvojni okviri

Biblioteke i razvojni okviri


- programska rješenja čija primjena olakšava razvoj aplikacije
 - osiguravaju unaprijed definirane resurse: podatkovne strukture, tipove objekata, metode, ...
- definiraju vlastite konvencije i standarde pisanja programskog koda
- razlika:
 - biblioteke su jednostavnije, pružaju fleksibilnost u smislu da programeri sami biraju koje će resurse i kako primijeniti
 - razvojni okvir je kompleksniji, daje zadanu arhitekturu prema kojoj se razvija aplikacija → pruža veću pomoć u radu, ali je teže naučiti njegovu primjenu
- primjeri:
 - jQuery
 - Handlebars
 - Bootstrap

Babel

JavaScript compiler

Babel se uglavnom koristi za pretvaranje ECMAScript 2015+ koda u unazad kompatibilnu verziju JavaScripta u trenutnim i starijim preglednicima ili okruženjima.

```
[1, 2, 3].map((n) => n + 1);
```



```
[1, 2, 3].map(function(n) {  
  return n + 1;  
});
```

Node.js

- Node.js je *JavaScript runtime environment* - razvojna okolina koja omogućuje pokretanje JavaScript programa izvan web preglednika.
- Uz Node.js instalaciju dolazi i snažan upravitelj programskim paketima - **npm** (*node package manager*):
 - softverski alat koji automatizira proces instaliranja, nadogradnje, konfiguriranja i brisanja računalnih programa ili biblioteka (paketa).

Naredba	Opis
<code>npm version</code>	trenutna verzija <i>npm-a</i>
<code>npm run</code>	lista raspoloživih skripti
<code>npm run <nazivSkripte></code>	pokretanje odabrane skripte
<code>npm search <nazivPaketa></code>	pretraživanje npm baze paketa
<code>npm init</code>	inicijalizira node paket – kreira package.json datoteku
<code>npm install --save <nazivPaketa></code>	lokalna instalacija paketa i editiranje package.json datoteke (parametar <code>--save</code> se podrazumijeva)
<code>npm install <nazivPaketa>@<verzija></code>	instaliranje točno određene verzije paketa
<code>npm install --save-dev <nazivPaketa></code>	instaliranje paketa koji se koriste samo prilikom razvoja aplikacije i editiranje package.json datoteke
<code>npm install -g <nazivPaketa></code>	globalna instalacija paketa
<code>npm install <lokacija paketa na disku></code>	instaliranje paketa s računala
<code>npm uninstall --save <nazivPaketa></code>	brisanje paketa i brisanje iz package.json datoteke (parametar <code>--save</code> se podrazumijeva)
<code>npm uninstall --no-save <nazivPaketa></code>	brisanje paketa (neće se brisati iz datoteke package.json)
<code>npm uninstall -g <nazivPaketa></code>	brisanje globalnog paketa
<code>npm ls</code>	lista paketa instaliranih u trenutnom direktoriju
<code>npm ls -g</code>	lista globalno instaliranih paketa

Task runner-i

Task runner je alat koji omogućuje automatizaciju ponavljajućih zadataka koji se tipično rade ručno tijekom razvoja projekta.

grunt, gulp, broccoli, brunch, ...

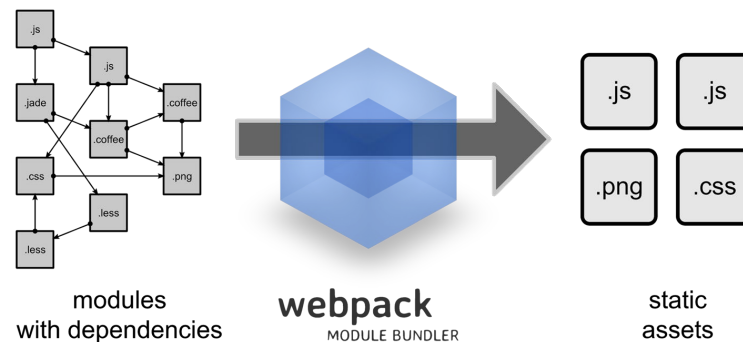
CSS	JavaScript	Ostalo
Kompiliraj SCSS u CSS	Provjeri JavaScript errore	Optimiziraj sve JPG, GIF, or PNG slike
Pokreni Autoprefixer na novom CSS-u	Konkateniraj sve JS fileove	Prati datoteke i pokreni taskove ako je potrebno
Minificiraj CSS	Minificiraj skripte	Pokreni BrowserSync radi testiranja u više preglednika i uređaja odjednom

Bundleri

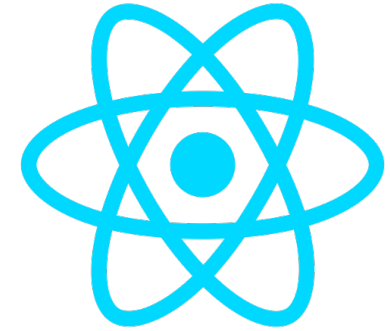
Module bundler je alat koji se koristi prilikom izgradnje aplikacije za produkciju.

Bundling je proces kombiniranja i optimizacije više modula u jedan ili više proizvodno spremnih paketa.

webpack, rollup, browserify, fusebox ...



ReactJS



- vrlo popularna JavaScript biblioteka
- služi za izradu korisničkih sučelja (engl. *user interface* - *UI*)
- pomoću malih izoliranih dijelova koda (komponenti) stvaramo kompleksan, efikasan i fleksibilan UI
- razvio: Facebook Inc. 2013. godine
- React aplikacije su *single page* aplikacije

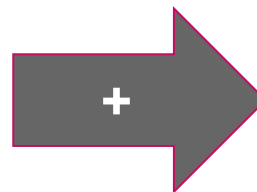
Zašto React?

U čistom *JavaScript-u* teško je reflektirati promjene u stanju aplikacije (promjene nastale korisničkom interakcijom)

Jednostavnija izrada i održavanje aplikacije

Bolje performanse – virtualni DOM

Velika i aktivna zajednica



Rezultirajući kôd ima bolje performanse jer se u pozadini koristi zrela i optimizirana biblioteka

Create React App

- Create React App je biblioteka koja omogućuje kreiranje React aplikacije samo jednom naredbom u CLI-ju:

```
npx create-react-app my-app
```

- rezultat ove naredbe je direktorij my-app unutar kojeg je organizirana cijela React aplikacija
- nije potrebno instalirati niti postavljati Webpack i Babel, ti alati su već konfigurirani i aplikacija je spremna za pokretanje

Export/import

Kako bi se programski kôd napisan u jednoj datoteci mogao koristiti u drugoj datoteci, potrebno ga je:

- ***exportati*** iz izvorne datoteke i
- ***importati*** u datoteci u kojoj ga želimo iskoristiti

Export

Dva su načina za napraviti *export*:

- ***default export*** - možemo imati samo jedan *default export* po datoteci, kod *importa* moramo specificirati ime *import* modula
- ***named export*** - možemo imati više *named exporta* po datoteci, ali kod *importa* moramo znati ime svakog modula

JSX

- predstavlja proširenje JavaScript jezika
 - prije izvršavanja prevodi se u standardni JavaScript kod
- koristi se za opisivanje izgleda korisničkog sučelja u React aplikaciji
 - olakšava dodavanje HTML-a unutar React komponenata (JavaScript koda)
- ima sintaksu vrlo sličnu HTML-u te ga, uz znanje HTML-a vrlo lako rabe i početnici

ReactJS komponente

- osnovni koncept React-a - React je u svojoj jezgri biblioteka za stvaranje komponenata
- komponente su neovisni, ponovno iskoristivi dijelovi kôda
- tipična React web-aplikacija može biti prikazana kao stablo komponenti i to tako da postoji jedna korijenska (*root*) komponenta (*“App”*), i potencijalno beskonačna količina ugniježđenih komponenti

ReactJS komponente

- Razlikujemo:
 - komponente definirane funkcijom i komponente definirane klasom
 - složene (pametne, *statefull*) i jednostavne (prezentacijske, *stateless*) komponente

Props

- *props* („propertyji”) su način na koji komponente međusobno komuniciraju
- *propsi* se koriste kako bi se informacije prenijele iz *parent* komponente u *child* komponentu
- protok podataka kroz *propse* je uvijek jednosmjernan - iz *parent* komponente u *child* komponentu
- *child* komponenta ne može i ne smije mijenjati *props* objekt koji je primila od *parenta*

Props - funkcije

```
function ChildComponent(props) {  
    return <h1>Ja sam {props.name}</h1>;  
}  
  
function ParentComponent() {  
    return (  
        <div>  
            <h1>Ja sam ParentComponent</h1>  
            <ChildComponent name="Child" />  
        </div>  
    );  
}
```

Props - klase

```
class ChildComponent extends React.Component{  
  render() {  
    return <h1>Ja sam {this.props.name}</h1>;  
  }  
}
```

```
class ParentComponent extends React.Component{  
  render() {  
    return (<div>  
      <h1>Ja sam ParentComponent</h1>  
      <ChildComponent name="Child" />  
    </div>);  
  }  
}
```

Children props

```
function Component1(props) {  
  return (  
    <div>{props.children}</div>  
  );  
}
```

```
function Component2() {  
  return (  
    <Component1>  
      <h1>Naslov</h1>  
      <p>Paragraf</p>  
    </Component1>  
  );  
}
```

Stanje komponente

- **state** - lokalno stanje komponente
 - komponenta ima kontrolu nad svojim stanjem i može ga mijenjati pomoću metode **setState** – komponenta definirana klasom
 - useState hook omogućuje korištenje stanja u komponentama definiranim funkcijom
- kod promjene stanja komponenta se ponovno iscrtava (renderira) kako bi prikazala promjene u stanju
 - renderiraju se i sve njezine child komponente

```
class MyComponent extends React.Component {
  state = {
    city: "Zagreb",
    state: "Hrvatska"
  };

  changeState = () => {
    const newCity = this.state.city === "Zagreb" ? "Osijek" : "Zagreb";
    this.setState({ city: newCity });
  };

  render() {
    return (
      <div>
        <h1>Grad: {this.state.city}</h1>
        <h1>Država: {this.state.state}</h1>
        <button onClick={this.changeState}>Drugi grad</button>
      </div>
    );
  }
}
```

Hooks

- *hooks* predstavljaju novi dodatak koji se pojavio s React 16.8 verzijom biblioteke
 - omogućuju upotrebu stanja (state) i drugih mogućnosti unutar komponenata definiranih funkcijom (npr. simulacija životnog ciklusa komponente)
- *useState hook* funkcija omogućuje korištenje stanja u komponentama definiranim funkcijom

useState hook

Primjer

```
function Example() {  
  // Deklaracija state varijable (count) i metode za njeno ažuriranje (setCount)  
  const [count, setCount] = useState(0);  
  
  var changeCount = () => {  
    setCount(count + 1);  
  }  
  
  return (  
    <div>  
      <p>You clicked {count} times</p>  
      <button onClick={changeCount}> Change count </button>  
    </div>  
  );  
}
```

Događaji (*events*)

- kao i u HTML-u, postoji definirana lista događaja na koje React komponente mogu reagirati, npr: **onClick**, **onChange**, ...
- aktiviranjem događaja izvršava se funkcija dodijeljena tom događaju - vrlo često ta funkcija mijenja stanje neke komponente ili cijele aplikacije

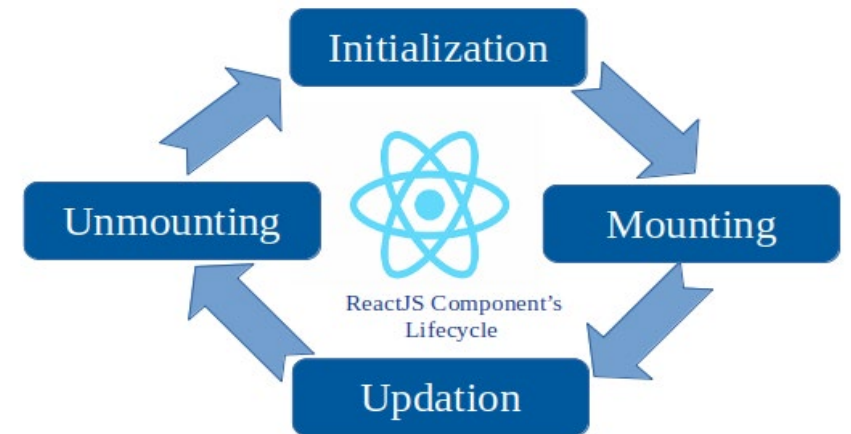
HTML	React
onclick	onClick
onchange	onChange
onmouseover	onMouseOver
onload	onLoad

Primjer korištenja *eventa*:

```
function ReactComponent() {  
  const handleClick = (event) => {  
    console.log("Button click!");  
  };  
  
  return (  
    <button onClick={handleClick}>  
      Klikni me!  
    </button>  
  );  
}
```

Metode životnog ciklusa

- životni ciklus komponente može se definirati kao niz metoda koje se pozivaju u različitim fazama postojanja komponente
- možemo definirati kod koji će se izvršiti u točno određenom trenutku procesa
- te metode nazivamo *lifecycle methods*, a pozivaju se u sljedećim fazama:
 - inicijalizacija komponente (*initialization*)
 - postavljanje na DOM (*mounting*)
 - ažuriranje (*updating*)
 - brisanje iz DOM-a (*unmounting*)



Metode životnog ciklusa

```
class Lifecycle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { data: null };  
  }  
  
  componentDidMount() {  
    this.getList();  
  }  
  
  getList = () => {  
    // method to make api call  
  };  
  
  render() {  
    return (<div>  
      <h3>Hello mounting methods!</h3>  
      <div id="container"></div>  
    </div>);  
  }  
}
```

useEffect hook

- *useEffect hook* koristimo kada želimo uzrokovati neke nuspojave (engl. *side effects*) prije ili poslije iscrtavanja (renderiranja) komponente definirane funkcijom
- *useEffect hook* može zamijeniti glavne *lifecycle* metode: `componentDidMount()`, `componentDidUpdate()` te `componentWillUnmount()`
 - različitom implementacijom *useEffect hook-a* možemo ga pokrenuti u istim životnim fazama (funkcijske) komponente u kojima su se pokretale *lifecycle* metode u komponentama definiranimi klasom

useEffect hook

```
function Component() {  
  
  useEffect(() => {  
    /* ... */  
  }, [stateVariable]); // Pokreni efekt samo ako se promijenila vrijednost stateVariable  
  
  return (<div>  
    <h3>Hello useEffect hook!</h3>  
    </div>);  
}
```

Liste u Reactu

- liste olakšavaju definiciju sadržaja koji vraća određena komponenta te omogućuju veću fleksibilnost u upravljanju njime:

```
function NumberList() {  
  const numbers = [1, 2, 3, 4, 5];  
  return (  
    <ul>  
      {numbers.map(number => <li>{number}</li>)}  
    </ul>  
  );  
}
```

Obrasci

Primjer React obrasca s funkcijama koje reagiraju na događaje promjene imena (onChange) i predavanja obrasca (onSubmit):

- podaci obrasca se pohranjuju u stanje komponente

```
<form onSubmit={this.handleSubmit}>  
  <label>Ime:</label>  
  <input name="name" onChange={this.handleChange} value={this.state.name}/>  
  <input type="submit" value="Potvrdi" />  
</form>
```

CSS u Reactu

- Najčešći načini za stilizaciju sadržaja komponenti u React-u:
 - CSS Stylesheet
 - Inline stilovi
 - CSS Modules

CSS Stylesheet

```
import React from "react";
import "../ShadowCard.css";

export default function Card() {
  return (
    <p className="shadow-card ">
      Dobar dan!
    </p>
  );
}
```

ShadowCard.js

```
.shadow-card {
  border-radius: 8px;
  box-sizing: border-box;
  margin: 24px;
  padding: 24px;
  box-shadow: 0 0 8px 0
darkgreen;
  width: 200px;
  color: darkgreen;
  font-weight: bold;
}
```

ShadowCard.css

Inline stilizacija

ShadowCard.js

```
import React from "react";

const cardStyle = {
  borderRadius: 8,
  boxSizing: "border-box",
  margin: 24,
  padding: 24,
  boxShadow: "0 0 8px 0 darkred",
  width: 200,
  color: "darkred",
  fontWeight: "bold"
};

export default function Card() {
  return <p style={cardStyle}>Dobar dan!</p>;
}
```

CSS Modules

```
import React from "react";
import styles from "../ShadowCard.module.css";

export default function Card() {
  return (
    <p className={styles.shadow-card}>
      Dobar dan!
    </p>
  );
}
```

ShadowCard.js

```
.shadow-card {
  border-radius: 8px;
  box-sizing: border-box;
  margin: 24px;
  padding: 24px;
  box-shadow: 0 0 8px 0 darkgreen;
  width: 200px;
  color: darkgreen;
  font-weight: bold;
}
```

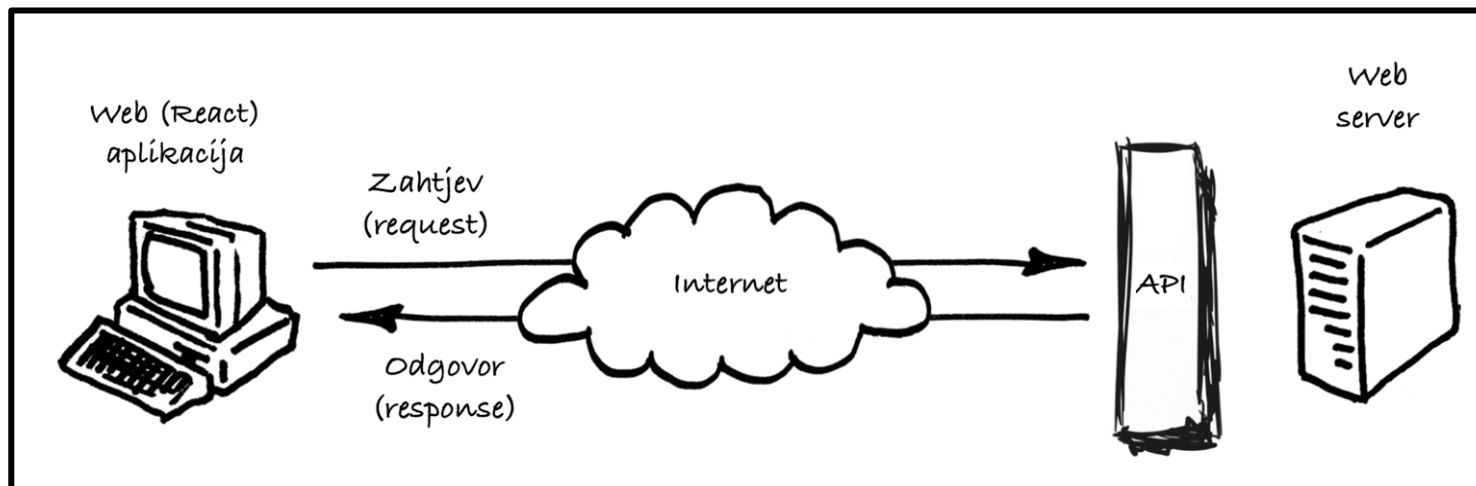
ShadowCard.module.css

Dodatne mogućnosti

- React Context
- *Higher Order Components (HOC)* - komponente višeg reda
- React Router
- GraphQL = **Graph Query Language**, jezik upita baze graf

API

- API = **A**pplication **P**rogramming **I**nterface
- API-ji omogućavaju aplikacijama da komuniciraju jedna s drugom
- nas zanimaju mrežni API-ji koji služe za komunikaciju web- aplikacija (React aplikacija) s poslužiteljem na kojem se nalaze podaci



Implementacija asinkronih operacija putem metode fetch

- Primjer poziva metode fetch()

```
fetch("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY")  
  .then(response => response.json())  
  .then(data => handleData(data))  
  .catch(error => handleError(error.toString()));
```

Skladišta stanja (engl. *stores*)

- u većim React aplikacijama često se pojavljuje potreba da više komponenti dijeli isto stanje aplikacije
- koristimo napredne biblioteke za upravljanje stanjima koje za spremanje podataka koriste skladišta stanja (state store)
- najpopularnije biblioteke:
 - Redux i MobX