

# Klijentske Web aplikacije

## Uvodno predavanje

Nikola Savanović

[nsavanovic@singidunum.ac.rs](mailto:nsavanovic@singidunum.ac.rs)

Dr. Mlađan Jovanović

[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

## ◆ Klijentske Web aplikacije

- Principi rada i gradivni elementi-tehnologije

## ◆ Ciljevi predmeta

## ◆ Oblasti

## ◆ Smernice

## ◆ Ispit

## ◆ Literatura

# **Pre nego krenemo sa kursem ...**

Pitanje broj 1 koje studeni postavljaju:  
**Koju tehnologiju da izaberem ?**

# Kako držati korak sa tehnologijama ?

- ◆ Postoji veliki broj tehnologija ...
- ◆ Kako da znam koju da koristim ?
- ◆ Gde da učim o novim tehnologijama ?
- ◆ Da li će to što naučim uskoro biti zastarelo ?

**Odgovor: postavljate pogrešna pitanja !**

- ◆ Biti u koraku sa tehnologijama nije najvažnija stvar
- ◆ Tehnologija se ne menja suštinski vrlo često ili vrlo brzo

# Osnove se ne menjaju

- ◆ Tehnologije se ne menjaju tako brzo:
- ◆ Dobar deo Facebook sistema je još uvek napisan u PHP
- ◆ Većina Google sistema je napisana u C++/Java
- ◆ Nećete (i ne bi trebalo) ponovo pisati kod u drugoj tehnologiji svake ili svake 2. godine
- ◆ Nova tehnologija ne znači nužno i napredak
  - Greške, stabilnost, skalabilnost, pouzdanost, ...

# Pravo pitanje je:

- ◆ **Kako da izaberem tehnologiju koja će mi na najlakši i najefikasniji način pomoći da obavim zadatak koji imam ?**
- ◆ Ili:
  - ◆ Imate dovoljno znanja i iskustva da sami procenite da li je alat ili tehnologija korisna
  - ◆ Ili:
    - ◆ Nemate dovoljno znanja i iskustva da sami procenite
  - ◆ Konsultujete **validne** izvore
    - Pouzdane osobe sa iskustvom
    - Članke, knjige i slično (sa dobrom reputacijom)

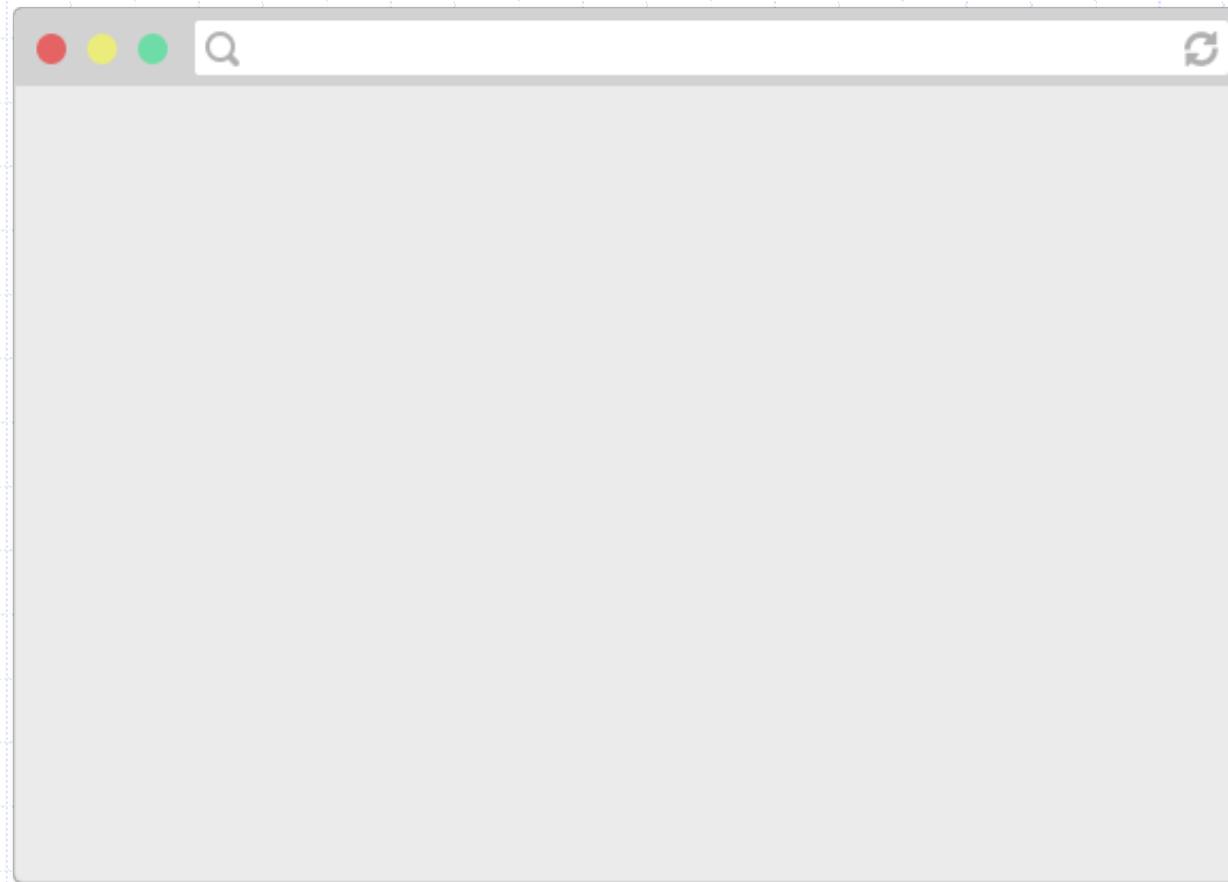
# Generalni savet

- ◆ Radite na tome da postanete dobar **inženjer**
- ◆ **Apstraktno** razmišljanje na nivou praktičnog **problema** koji rešavate
  - Problem-solving skills
- ◆ Naučite da rešite problem korišćenjem **različitih** tehnologija - frontend, backend, web, mobile, ...
  - Koje učite na studijama
- ◆ U praksi:
- ◆ Radite N godina kao full-time inženjer softvera sa drugim (dobrim) inženjerima
  - Čak i posle 1. godine rada vaše inženjerske veštine će zнатно uznapredovati
- ◆ Na ovaj način razvijate ne samo vaše znanje i veštine, već i sposobnost za procenu tehnologije

# Gde je ovaj kurs u **svemu tome** ?

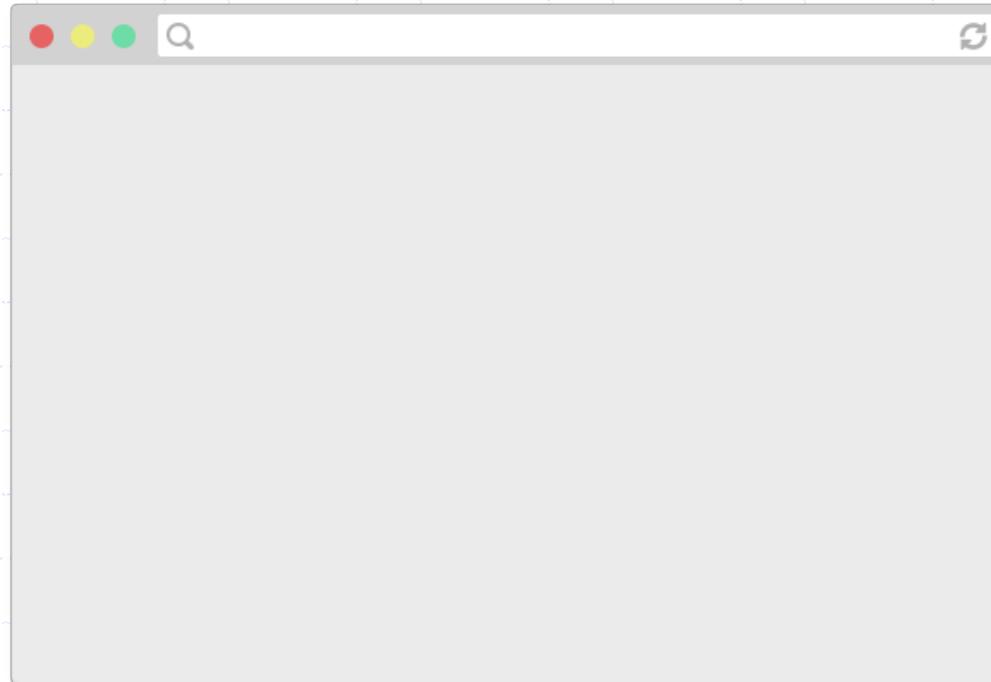
- ◆ Mi vas upoznajemo sa osnovnim **principima** i **konceptima** dobrih Web tehnologija za kreiranje klijentskih aplikacija
- ◆ Kako ?
- ◆ Kroz predavanja: **razumevanje** principa i koncepata
- ◆ Kroz vežbe: **primena** principa i koncepata
- ◆ Podjednako važni:
  - ◆ Bez razumevanja nema pravilne primene
  - ◆ **Bez primene i samostalnog rada nema učenja !**
  - ◆ Dugoročan cilj: lako ovladavanje novim tehnologijama zbog toga što će biti zasnovane na naučenim

# Kako rade Web stranice ?



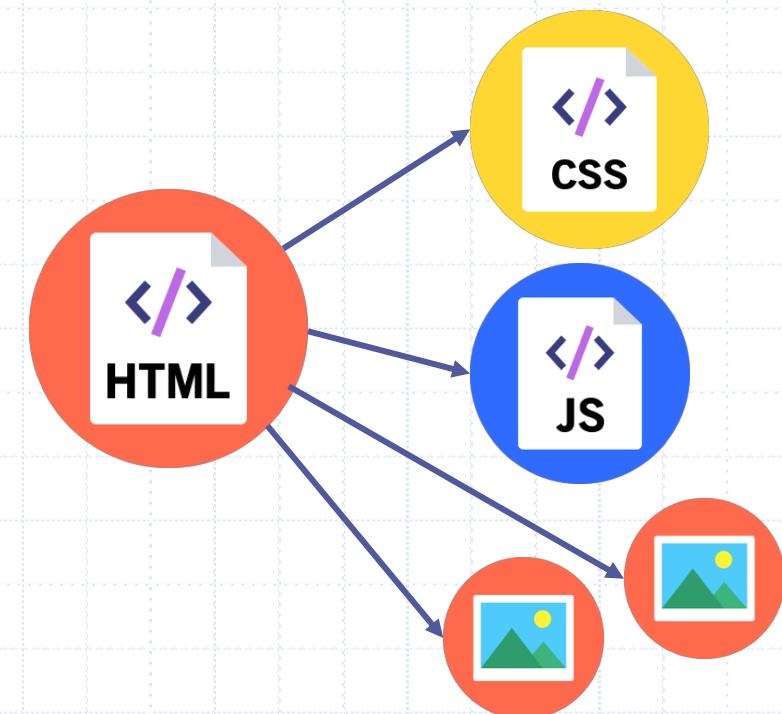
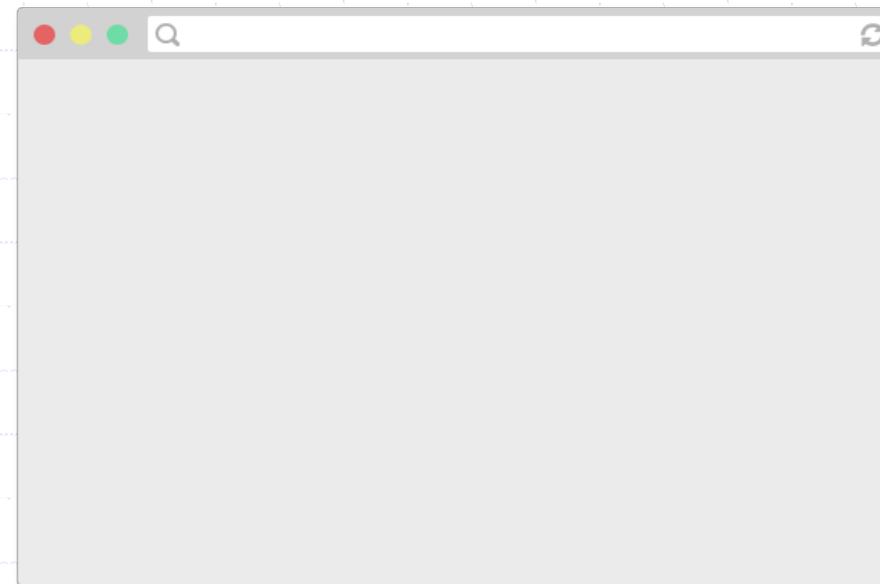
Pretraživači su aplikacije koje prikazuju Web stranice  
Na primer, Chrome, Safari, IE, Firefox, ....

# Kako rade Web stranice ?



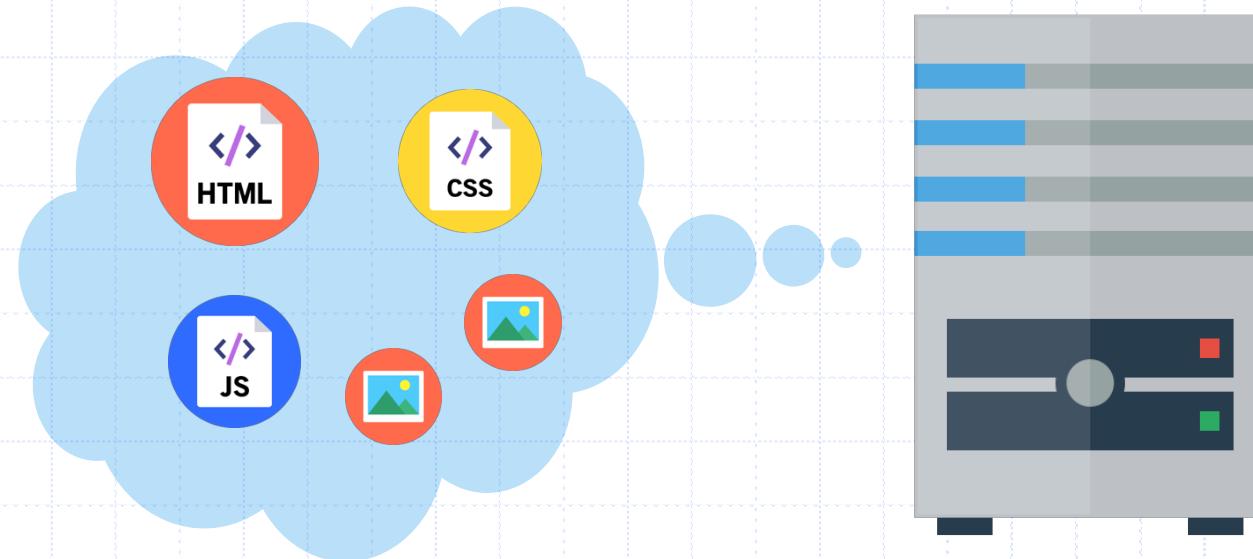
Web stranice su napisane u deklarativnom (markup) jeziku koji se zove HTML. Drugim rečima, pretraživač prikazuje Web stranicu tako što čita i interpretira HTML elemente

# Kako rade Web stranice ?



HTML datoteka može imati veze (linkove) ka drugim resursima, kao što su slike, video, kao i JavaScript i CSS datotekama koje pretraživač takođe učitava i interpretira

# Kako rade Web stranice ?

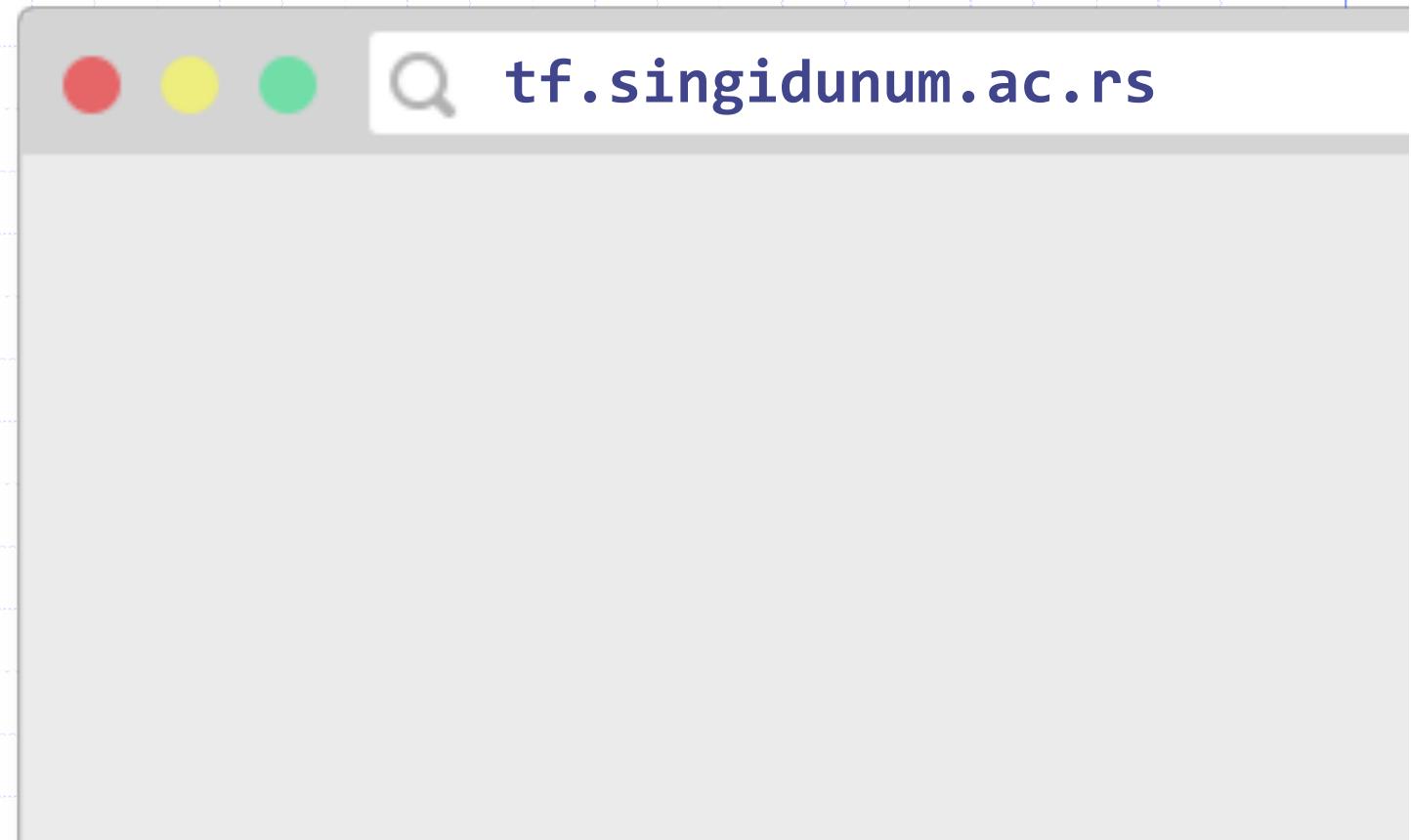


Web server je program koji se izvršava na računaru koji isporučuje Web stranice kao odgovore na zahteve klijenata.

Stranice mogu biti unapred pripremljene (statičke) ili generisane dinamički, po zahtevu

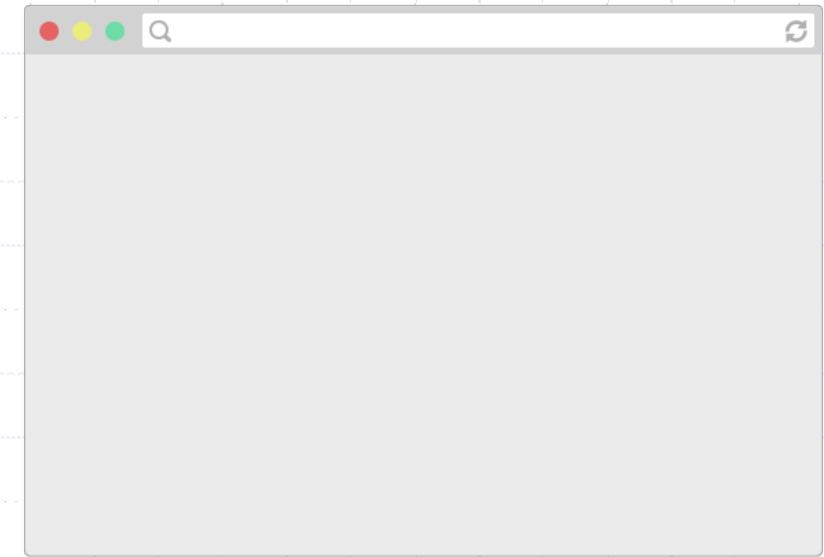
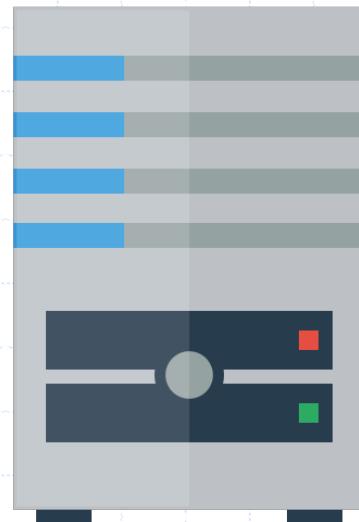
# Kako rade Web stranice ?

1. Unosite adresu Web stranice kao URL, koji predstavlja adresu HTML datoteke na Internetu



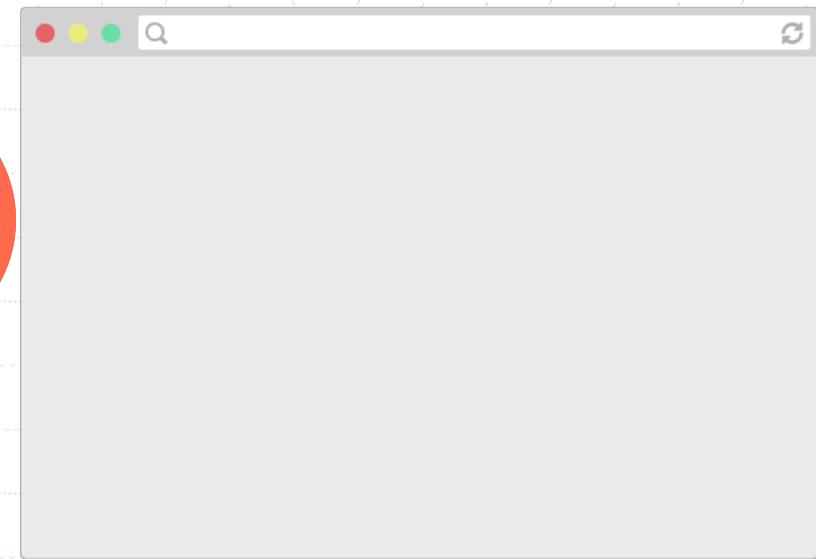
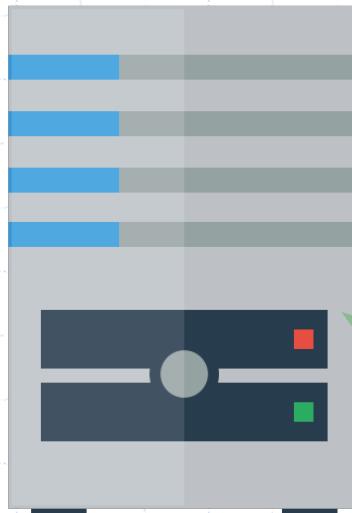
# Kako rade Web stranice ?

2. Pretraživač traži od web servera koji čuva dokument da isti pošalje



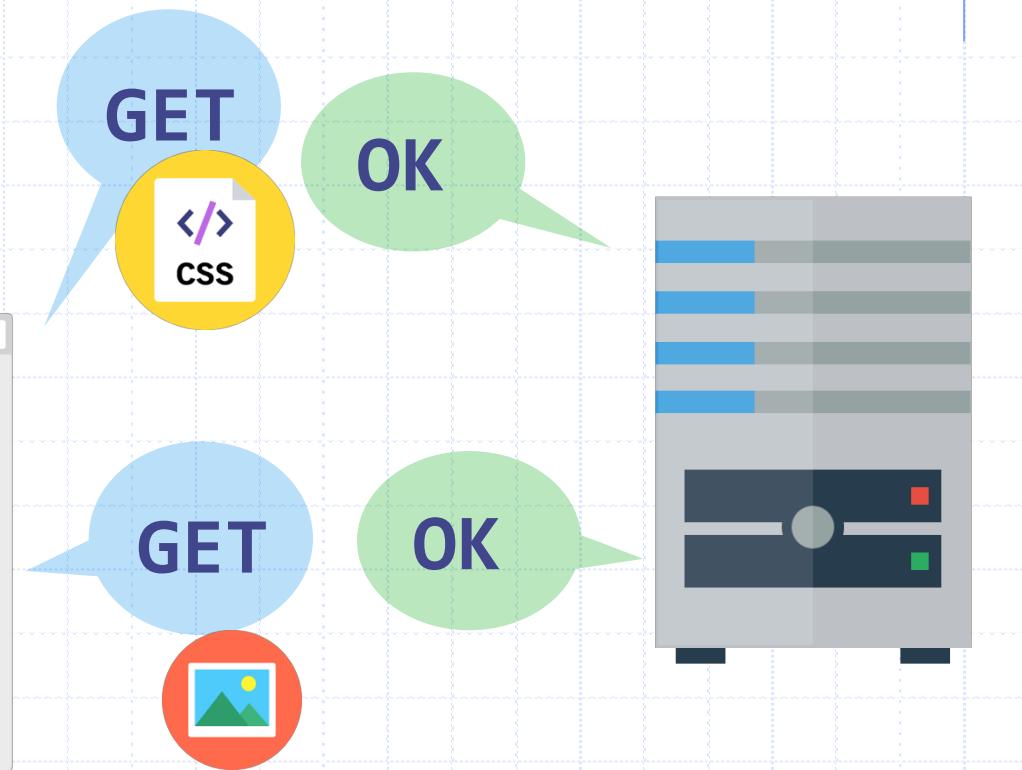
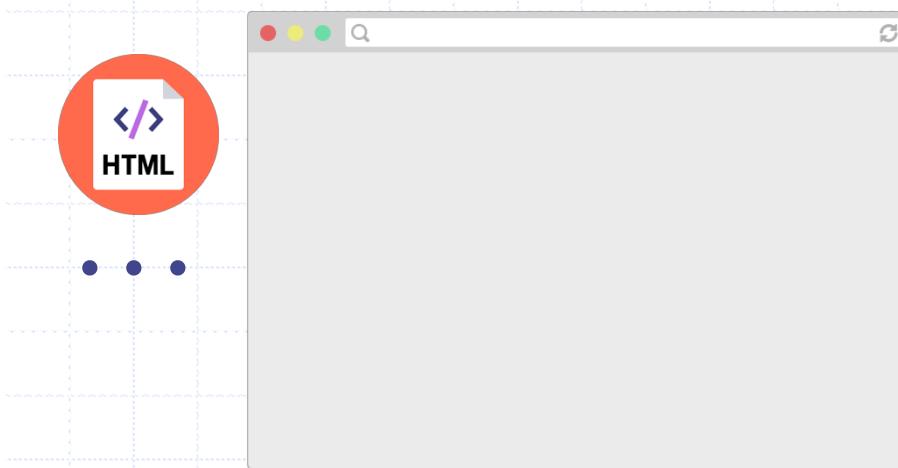
# Kako rade Web stranice ?

3. Web server odgovara pretraživaču  
i šalje traženu HTML datoteku



# Kako rade Web stranice ?

4. Pretraživač čita i interpretira HTML datoteku, nalazi ugrađene resurse i iste takođe traži od servera



# Kako rade Web stranice ?

5. Web stranica je učitana kada su dobavljeni svi resursi i prikazana



Izvor: Holmes, S., (2019). Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2nd Ed.

# Kurs o klijentskim Web aplikacijama

- ◆ Tehnologije za kreiranje savremenih Web aplikacija sa naglasom na klijentsku stranu
- ◆ Stek tehnologija
  - Pretraživač
  - Web server
  - Skladište podataka
- ◆ Cilj: razvoj funkcionalne i upotrebljive Web aplikacije
- ◆ Kako se uče tehnologije koje će biti obrađene na kursu ?
  - ◆ Učenje pokušajima (learn by doing)
  - ◆ Samostalan rad van časova vežbi i predavanja !

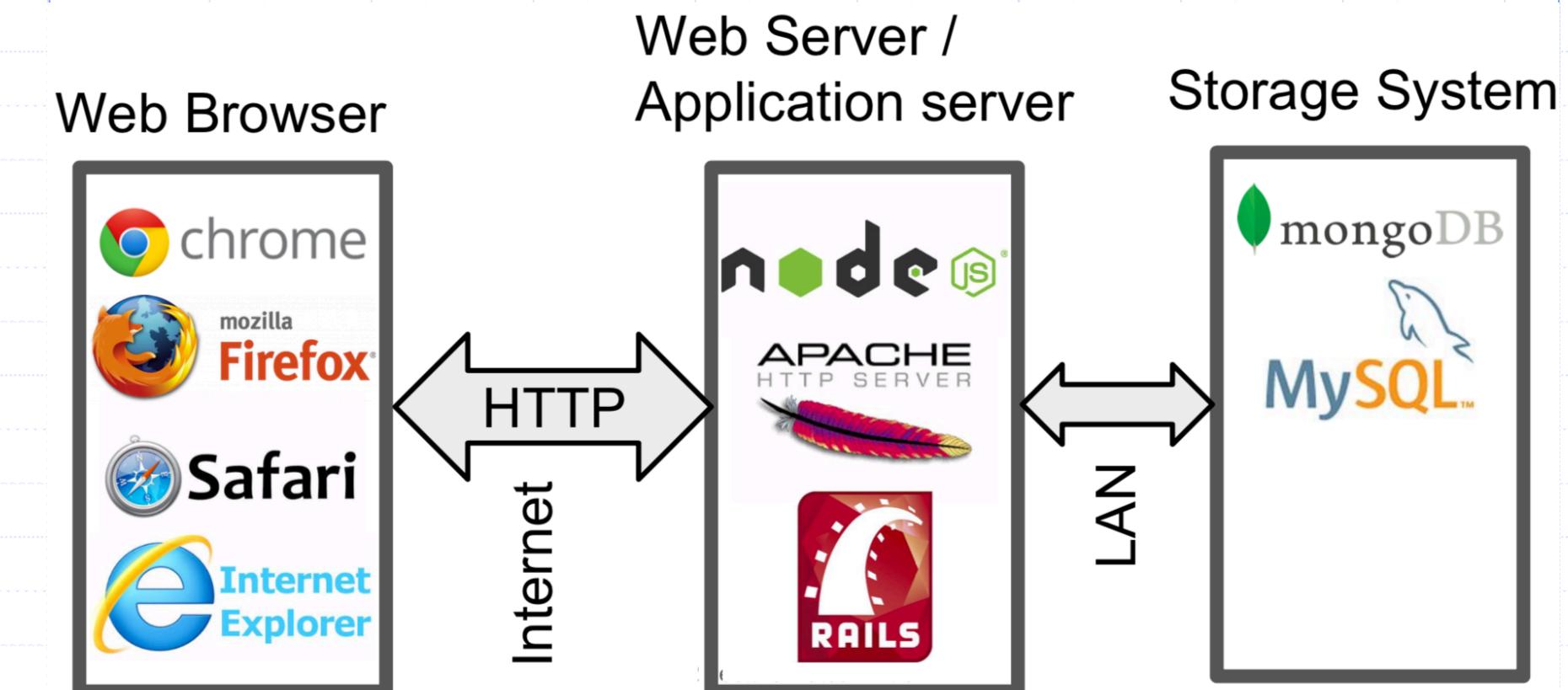
# Tehnologije i koncepti

- ◆ HTML/CSS/JavaScript – Sadržaj, izgled, ponašanje, ponovna upotrebljivost
- ◆ Document Object Model (DOM) – Struktura dokumenta, tj. Web stranice
- ◆ Angular.js – MVC (Model-View-Controller) obrazac, Single page aplikacije
- ◆ Node.js – Serverske funkcionalnosti
- ◆ HTTP/AJAX – Komunikacija
- ◆ Cookies/Sessions – Stanje
- ◆ DB (Database) – Skladište podataka

# Struktura kursa

- ◆ **Lekcije** su fokusirane na **objašnjenje** elemenata tehnologije
- ◆ **Vežbe** su fokusirane na **korišćenje** elemenata tehnologije
  
- ◆ Kolokvijum 1 – praktičan zadatak iz oblasti obrađenih do prve kolokvijumske nedelje
- ◆ Kolokvijum 2 i Završni ispit – semestralni projekat na zadatu temu

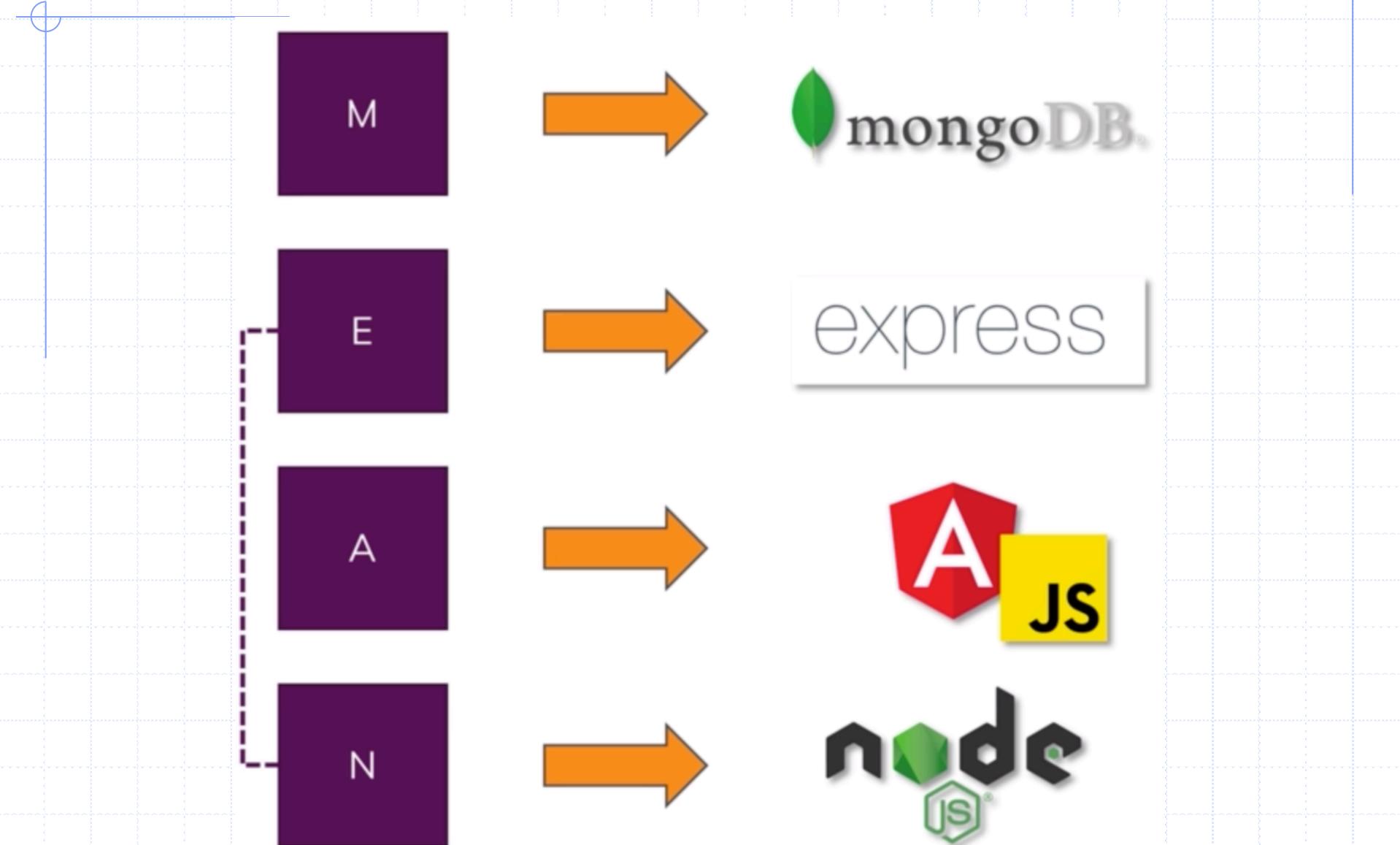
# Stek tehnologija Web aplikacije



# MEAN Stek



# MEAN Stek



Izvor: Holmes, S., (2019). Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2nd Ed.

# MEAN



A Client-Side (Browser) Framework which allows you to build Single-Page-Applications (SPA)

Render UI with Dynamic Data

Handle User Input

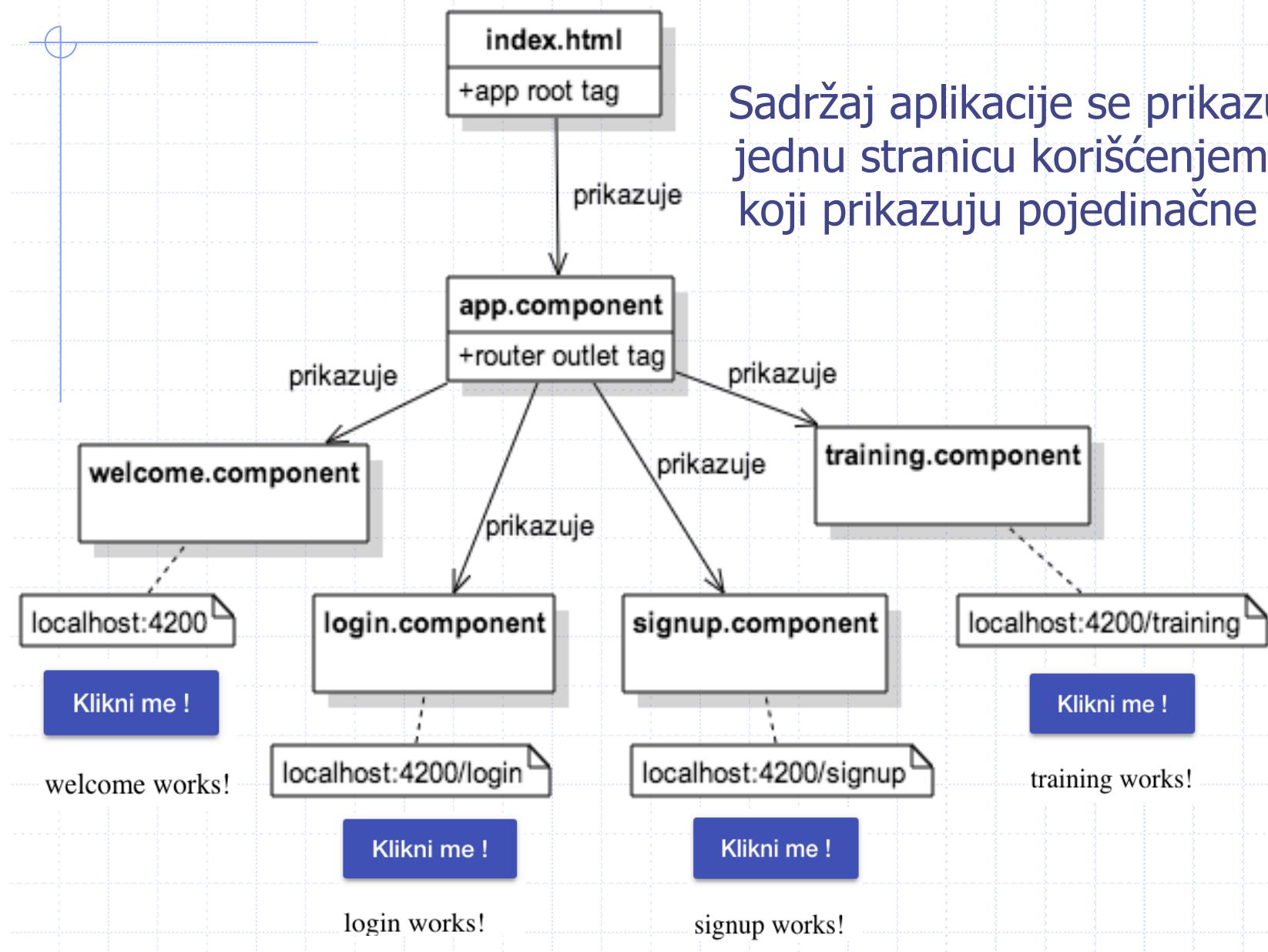
Communicate with Backend Services

Provides a “Mobile App”-like User Experience

# Angular

- ◆ Izvršava se na strani klijenta, u pretraživaču
- ◆ Okvir za razvoj Web stranica zasnovan na TypeScript jeziku
- ◆ Prikaz korisničkog interfejsa sa dinamičkim podacima
  - Podaci koji se menjaju u realnom vremenu, tj. u toku korišćenja aplikacije
- ◆ Prijem i obrada ulaznih akcija i podataka korisnika
- ◆ Prosleđivanje na dalju obradu i prijem rezultata za prikaz od pozadinske logike (backend)
- ◆ *Mobile-like* doživljaj korisnika – sadržaj se menja dinamički, bez ponovnog učitavanja čitave stranice, već samo delova koji su zahtevani ili promenjeni

# SPA – Single Page Application



Sadržaj aplikacije se prikazuje kroz jednu stranicu korišćenjem tagova koji prikazuju pojedinačne prikaze

# MEAN



A Server-side Library: JavaScript on the Server-side

Listen to Requests and  
Send Responses

Execute Server-side  
Logic

Interact with  
Databases and Files

An Alternative to PHP, Ruby on Rails, Java etc. Is rarely used Standalone!

# Node

- ◆ Serverska JavaScript biblioteka
  - Serverska: obavlja funkcije servera
  - Napisana u JavaScript-u
- ◆ Rad sa fajl sistemom, tj. pristup lokalnim datotekama
- ◆ Rad sa bazom podataka
  - Teorijski moguće i korišćenjem Angular-a, ali nije bezbedno
  - Parametri za pristup se mogu videti u developer prozoru pretraživača
- ◆ Obrada HTTP zahteva
  - Prijem zahteva i slanje odgovora(resultata)
- ◆ Autentikacija i autorizacija

# MEAN

# express

A Node Framework which simplifies writing Server-side Code and Logic.

Based on Node, offers  
same Functionalities

Middleware-based:  
Funnel Requests  
through Functions

Includes Routing,  
View-rendering &  
More

# Express

- ◆ Node nadogradnja
- ◆ Korisne, tzv. utility funkcije koje olakšavaju korišćenje Node servisa
- ◆ Kako ?
- ◆ Obrada zahteva kroz sekvencu koraka ili lanac u kojem se u svakom koraku obavlja specifična, atomična funkcija
  - Prijem zahteva
  - Čitanje podataka
  - Specifična obrada
  - Slanje odgovora
- ◆ Analogija: Java *wrapper* funkcije za obradu tokova
- ◆ Zbog čega ?
- ◆ Struktuiran kod koji je lak za razumevanje i održavanje
- ◆ Mehanizam rutiranja
  - Zahtevi koji potiču iz različitih domena

# MEAN



Mogu biti korišćene i  
druge BP (npr. MySQL)



A NoSQL Database which stores “Documents” in “Collections” (instead of “Records” in “Tables” as in SQL).

Store Application Data  
(Users, Products, ...)

Enforces no Data  
Schema or Relations

Easily connected to  
Node/ Express (NOT to  
Angular!)

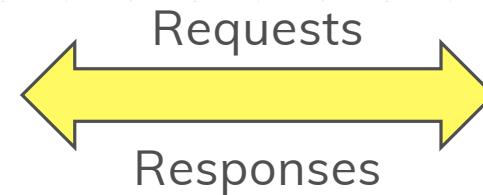
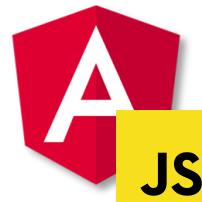
A powerful Database which can easily be integrated into a Node/ Express Environment.

# MongoDB

- ◆ Ne-relaciona baza podataka
- ◆ Čuva kolekcije dokumenata
- ◆ Relaciona baza: tabele, zapis, relacije
- ◆ Ne postoji unapred definisana šema, tj. model podataka koji će se skladištiti
- ◆ Jednostavna integracija sa Node/Express
- ◆ Pokazuje bolje performanse od relacione baze:
- ◆ Veliki broj upisa i pristupa u bazu podataka (logovi, notifikacije)
  - read per second, write per second
- ◆ Relacione baze pokazuju bolje performanse kod veće količine složenije struktuiranih poslovnih podataka sa relacijama i izveštavanja

# MEAN stek – funkcionalni pogled

Client (Browser)



Server



Presentation/ UI

Single-Page- Application

Data

JSON Format

Business Logic

Persistent Data Storage

Authentication Logic

Single Page is not necessarily served by Node Backend!

# Ispit

- ◆ Redovno pohađanje nastave i aktivnost na času (max 10 poena)
- ◆ Kolokvijum 1 (max 30 poena)
  - Zadatak iz oblasti obrađenih do prve kolokvijumske nedelje
- ◆ Kolokvijum 2 (max 30 poena)
  - Projekat prvi deo
- ◆ Završni ispit (max 30 poena)
  - Projekat drugi deo

# Vrednovanje

- ◆ Završna ocena se formira na osnovu skale:
- ◆ 10: 100 => poeni => 91
- ◆ 9: 90 => poeni => 81
- ◆ 8: 80 => poeni => 71
- ◆ 7: 70=> poeni => 61
- ◆ 6: 60 => poen => 51
- ◆ nije položila/o: poeni <= 50

# Saveti za slušanje kursa



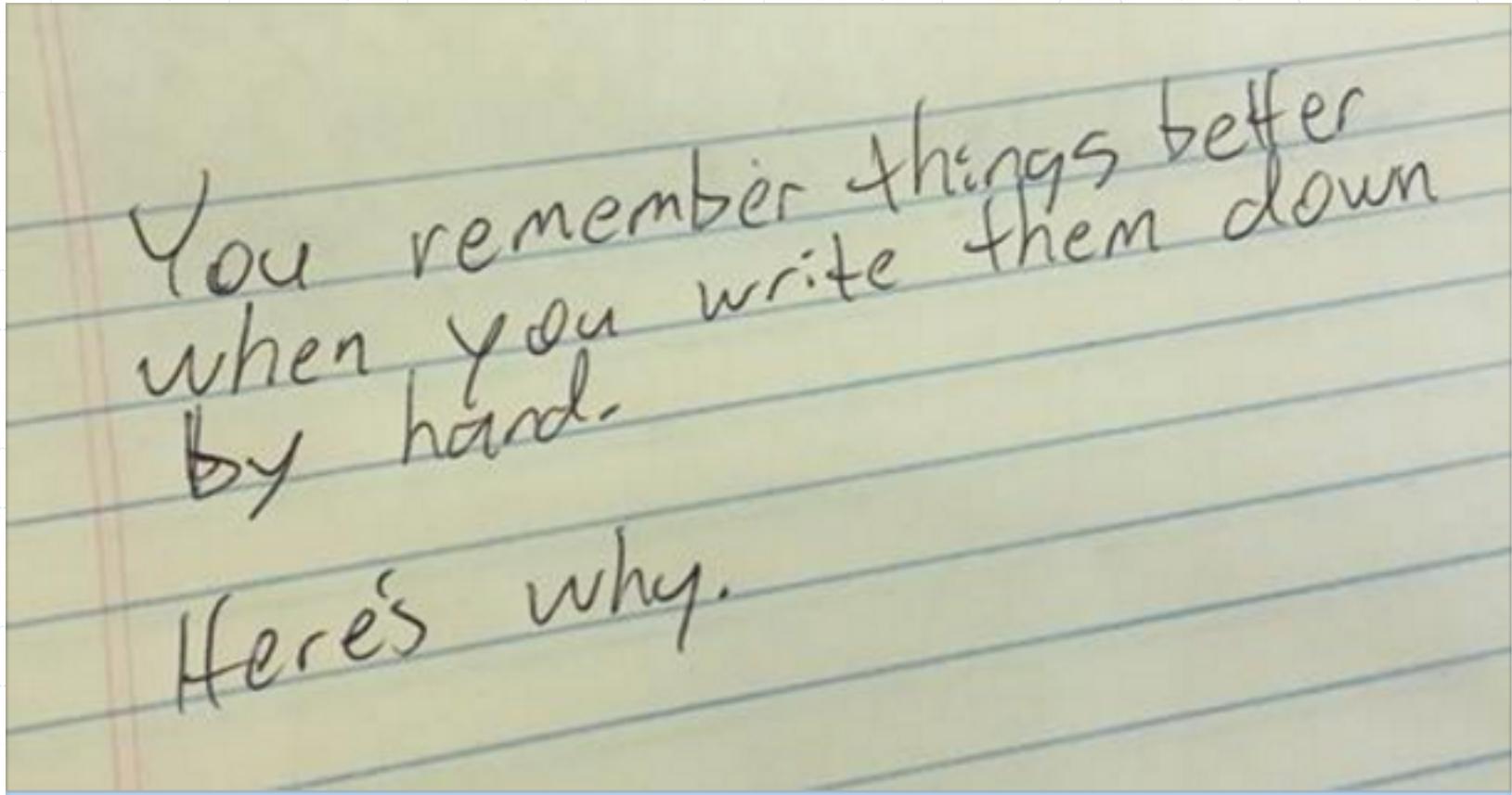
- ◆ Da bi Vašim kolegama i Vama obezbedili uslove za **održavanje pažnje i koncentracije** u toku nastave
1. Izbegavajte korišćenje mobilnih telefona u toku nastave (osim ako nije u vezi sa nastavom)
  2. Dođite na vreme na čas (ukoliko kasnите nemojte uznemiravati ostale kolege i na pauzi ih pitajte šta je rađeno dok ste bili odsutni)
  3. Svedite priču i šaputanje u pozadini na minimum
  4. Izbegavajte zabavne aktivnosti na vašim ili univerzitetskim računarima (gleđanje videoa u toku časa)

# Saveti za slušanje kursa



- ◆ Da bi poboljšali učenje, tj. **usvajanje** materijala u toku izvođenja nastave
1. Izbegavajte više zadatak u isto vreme (pisanje mejlova i slušanje nastave nisu kompatibilni)
  2. Ostavite Vaš telefon u torbi (osim ako se koristi za potrebe nastave)
  3. Hvatajte beleške u toku nastave i prelistajte u toku večeri
  4. Ukoliko nešto ne razumete, prekinite me kako bih pojasnio
  5. Ukoliko imate generalnije ili složenije pitanje, sačekajte pauzu da bi pitali
  6. **Učestvujte i budite proaktivni**

# Ručno vodenje beleški



Mueller (2014) The pen is mightier than the keyboard: Advantages of longhand over laptop note taking

# Zbog čega su beleške korisne ?

- ◆ **hipoteza enkodovanja:** kada osoba hvata beleške, procesiranje koje se dešava će poboljšati učenje i zadržavanje materijala
- ◆ **hipoteza eksternog skladišta:** ljudi uče gledajući ponovo u svoje zapise, ili čak zapise drugih ljudi

# Dodatna literatura

- ◆ Holmes, S., (2016). Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 1st Ed.
- ◆ HTML, CSS i DOM документација:  
<https://developer.mozilla.org/en-US>
- ◆ JavaScript документација:  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
- ◆ Angular документација: <https://angular.io/docs>
- ◆ Angular Material документација:  
<https://material.angular.io>
- ◆ Node.js документација: <https://nodejs.org/en/docs/>

# Klijentske Web aplikacije

## Uvodno predavanje

Nikola Savanović

[nsavanovic@singidunum.ac.rs](mailto:nsavanovic@singidunum.ac.rs)

Dr. Mlađan Jovanović

[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# **HTML i CSS**

## **Prvi deo**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

- ◆ HTML osnovni elementi
- ◆ CSS osnovni elementi
- ◆ Primer: [codepen.io](https://codepen.io)

# Šta je HTML ?

- ◆ HTML (HyperText Markup Language)
- ◆ Opisuje **sadržaj** i **strukturu** Web stranice
- ◆ Sastoji se od gradivnih elemenata koji se zovu **tagovi**

```
<h1>KWA: Klijentske Web Aplikacije</h1>
<strong>Obaveštenja</strong><br/>
Gradivo za prvi kolokvijum<br/>
```

**KWA: Klijentske Web Aplikacije**

**Obaveštenja**

Gradivo za prvi kolokvijum

# Osnovna struktura HTML stranice

```
<!DOCTYPE html>
<html>
  <head>
    <title>KWA</title>
  </head>

  <body>
    ... sadržaj stranice...
  </body>
</html>
```

Sačuvano u datoteci *ime\_fajla.html*

# Osnovna struktura HTML stranice

Metapodaci koji se ne pojavljuju u vidljivom prozoru pretraživača

```
<!DOCTYPE html>
<html>
  <head>
    <title>KWA</title>
  </head>

  <body>
    ... sadržaj strane...
  </body>
</html>
```

Sadržaj koji se prikazuje u vidljivom prozoru pretraživača

sadržaj **<title>** se pojavljuje kao naziv tab-a prozora

# HTML elementi

< p >

HTML je < em > sjajan ! ! ! < / em >

< img src = "html5.png" />

< / p >

- ◆ Element obično ima početni i krajnji tag (< p > i < / p >)
  - Sadržaj: sve što se nalazi između početnog i krajnjeg taga
- ◆ Element može biti samozatvarajući (img)
- ◆ Element može imati atribut (src = "html5.png")
- ◆ Element može sadržati druge elemente (p sadrži em i img)

# HTML Tagovi

- ◆ Mogu imati sledeća značenja (semantiku):
- ◆ Značenje tekstualnog sadržaja
  - <h1> označava zaglavje (naslov) prvog nivoa
  - <p> označava paragraf teksta
  - <ul><li> for unordered (bulleted) list
- ◆ Instukcije za formatiranje (<i> za italic)
- ◆ Dodatna informacija za prikaz (<img>)
- ◆ Tagovi mogu sadržati druge tagove (ugnježdavanje)

# Sintaksna pravila HTML elemenata

- ◆ **Dokument:** hijerarhijska kolekcija **elemenata** koja počinje sa <html>
- ◆ **Element:** početni tag, sadržaj, krajnji tag
- ◆ Elementi mogu biti ugnježđeni
- ◆ Svaki element mora imati početak i kraj
  - Mogu se koristiti i skraćeni oblici:
  - <div/> umesto <div></div>
- ◆ Početni tagovi mogu imati **attribute:**



<input type="text" value="94301" name="zip">

<div class="header">

# Najčešće korišćeni HTML tagovi

< p >

Novi paragraf

< br >

Prelazak u novi red u okviru paragrafa

< h1>, < h2>, ...

Zaglavlja (N=6)

< b >

**bold** tekst

< i >

*italic* tekst

< pre >

Čuva originalno formatiranje: uvlačenje, prazan prostor i prelazak u novi red

< img >

Grafika (slike)

< a href="..." >

Link ka drugoj stranici

<!-- comments -->

Komentar

# Najčešće korišćeni HTML tagovi

<table><tr><td> Tabela

<ul> <li> Neuređena (*bullet*) lista

<ol> <li> Uređena (numerisana) lista

<div> Logičko grupisanje elemenata gde se grupa rasprostire u više redova, forsira prelazak u novi red

<span> Logičko grupisanje elemenata gde grupa zauzima jedan, ne forsira prelazak u novi red

<form>, <input>, <textarea>, <select> Elementi forme za unos podataka korisnika

# HTML elementi <head> sekcije

**<title>** Naslov strane koji će se prikazati u title bar-u prozora pretraživača

**<link>** Link ka CSS datoteci za definisanje stilova na stranici

**<script>** Link ka JavaScript funkcijama koje se koriste na stranici

# HTML specijalni karakteri

- ◆ Potreba za prikazivanjem specijalnih znakova u sadržaju

&lt; prikazuje <

&gt; prikazuje >

&amp; prikazuje &

&quot; prikazuje "

&nbsp; space (**nonbreaking space**)

- ◆ Prazan prostor se ne uzima u obzir osim u slučaju posebnih elemenata (textarea, pre)

# HTML elementi primer

Zaglavlja: h1, h2, ... h6

```
<h1>Moby Dick</h1>  
<h2>Or, the Whale</h2>
```

**Moby Dick**  
**Or, the Whale**

Paragraf: p

```
<p>Call me Ishmael.</p>
```

Call me Ishmael.

Prelazak u novi red: br

```
since feeling is first<br/>  
who pays any attention<br/>  
to the syntax of things
```

since feeling is first  
who pays any attention  
to the syntax of things

# HTML elementi primer

Grafika: **img**

```

```



Link: **a**

```
<a href="google.com">click here!</a> click here!
```

Isticanje (bold): **strong**

```
<strong>Be BOLD</strong>
```

**Be BOLD**

Isticanje (em vs. italic): **em**

He's my **em>brother</em>** and all

He's my *brother* and all



# **CSS**

# **osnove**

# Motivacija za CSS

- ◆ Koji tip i veličinu fonta kreira `<h1>Uvod</h1>` ?
- ◆ Odgovor: predefinisani iz pretraživača (HTML govori **šta**; pretraživač određuje **kako**)
- ◆ Rani HTML je definisao attribute  
`<table border="2" bordercolor="black">`

**Datoteke sa stilovima** (style sheets) su rešile problem:

Definisanje stila umesto podrazumevanih vrednosti pretraživača ili kodiranja atributa za svaki element

# Princip: razdvajanje stila od sadržaja

- ◆ Sadržaj (**šta** se prikazuje) je u HTML datotekama
- ◆ Instrukcije za formatiranje (**kako** se prikazuje) su u odvojenim datotekama sa stilovima (.css datoteke)
- ◆ Koristi se class atribut elementa za povezivanje sadržaja i formatiranja

```
<div class="test">
```

- ◆ Rezultat: stil se definiše jednom, zatim se koristi na različitim mestima
- ◆ Prednosti:
  - ◆ Fleksibilnost
  - ◆ Preglednost
  - ◆ Ponovna upotrebljivost

# CSS sintaksa

- ◆ CSS: Cascading Style Sheets
- ◆ Opisuje **pojavljivanje** i **raspored** elemenata Web stranice
- ◆ Sastavljena od pravila (CSS rules) gde svako pravilo definiše stil specifičnog elementa

```
selector {  
    property: value;  
}
```

# CSS sintaksa

CSS datoteka se sastoji od skupa pravila:

```
selector {  
    property: value;  
}
```

***selector***: HTML element za koji se definiše stil

***property***: Naziv elementa stila (boja, veličina, itd.)

***value***: Vrednost elementa stila

Sačuvano u datoteci *ime\_datoteke.css*

# CSS datoteka sadrži jedno ili više pravila (rules)

Selector

body {

Declaration  
Block

font-family: Tahoma, Arial, sans-serif;  
color: black;  
background: white;  
margin: 8px;

}

Property

Value

# Vrste CSS selektora, tj. ciljnih elemenata



## CSS Selector

	CSS	HTML
Tag name	<pre>h1 {     color: red; }</pre>	<code>&lt;h1&gt;Today's Specials&lt;/h1&gt;</code>
Class attribute	<pre>.large {     font-size: 16pt; }</pre>	<code>&lt;p class="large"&gt;...</code>
Tag and Class	<code>p.large {...}</code>	<code>&lt;p class="large"&gt;...</code>
Element id	<pre>#p20 {     font-weight: bold; }</pre>	<code>&lt;p id="p20"&gt;...</code>

# Povezivanje CSS sa HTML dokumentom

```
<!DOCTYPE html>
<html>
  <head>
    <title>Klijentske Web aplikacije</title>
    <link rel="stylesheet" href="ime_fajla.css" />
  </head>

  <body>
    ... sadržaj stranice...
  </body>
</html>
```

# CSS elementi (properties)

- ◆ Postoji više od 500 CSS elemenata

Font face	<code>font-family: Helvetica;</code>
Font color	<code>color: gray;</code>
Background color	<code>background-color: red;</code>
Border	<code>border: 3px solid green;</code>
Text alignment	<code>text-align: center;</code>

- ◆ Mozilla Developer Network (MDN) je najbolja referenca za HTML i CSS elemente

# Definisanje CSS boja

## 140 predefinisanih imena ([spisak](#))

color: black;

rgb() i rgba()

color: rgb(34, 12, 64);

color: rgba(0, 0, 0, 0.5);

### Heksadecimalne vrednosti

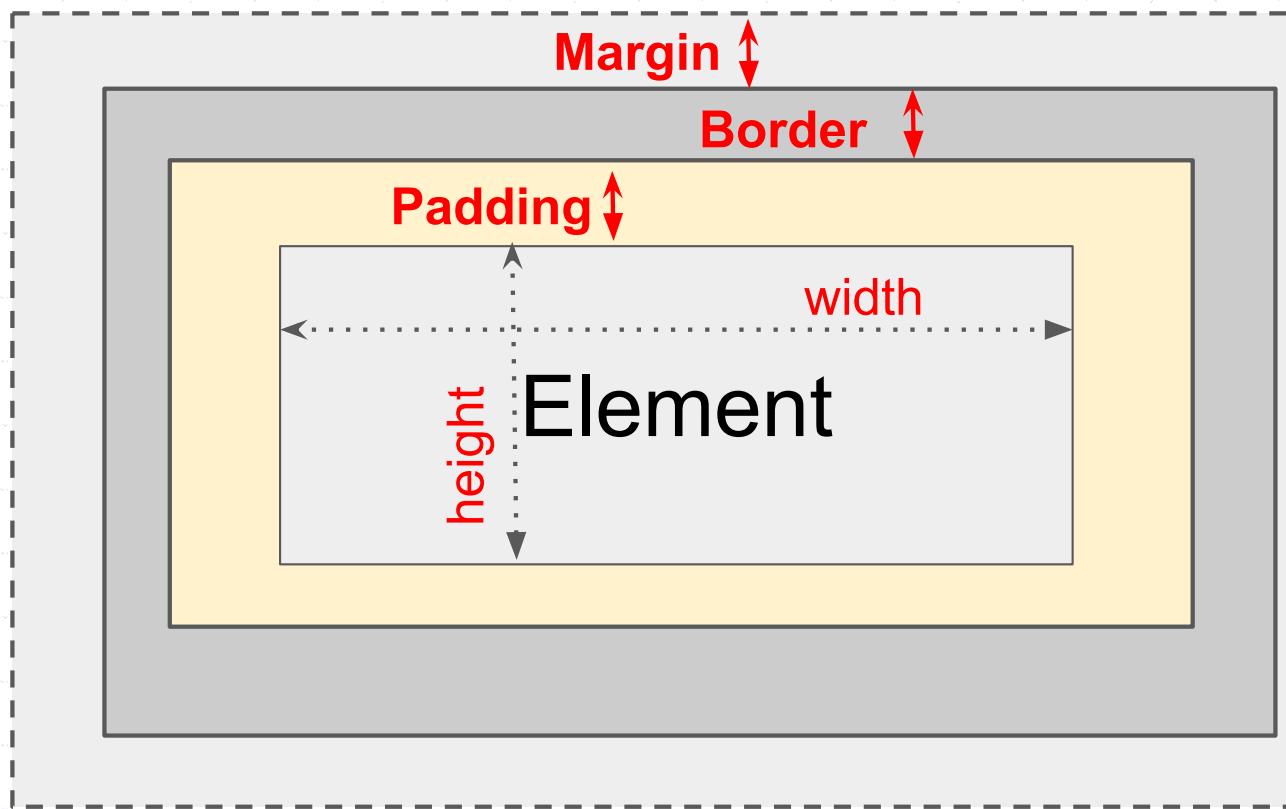
color: #00ff00;

color: #0f0;

color: #00ff0080;

- "a" označava alfa vrednost i definiše transparentnost boje
- Koristiti deskriptivne, umesto numeričkih vrednosti:
  - predefinisan naziv
  - rgb/rgba
  - heksadecimalno

# CSS Box Model



Ukupna širina (width) elementa = width + left padding + right padding +  
left border + right border +  
left margin + right margin

# CSS jedinice mere

## Apsolutne

2px pikseli

1mm milimetri

2cm centimetri

0.2in inči

2pt printer point (1/72 inča)

## Relativne

2em 2 puta veličine fonta tekućeg elementa

3rem 3 puta veličine fonta root elementa

# **Vežba: Web stranica kursa**

## **Klijentske Web aplikacije: KWA**

### **Obaveštenja**

Silabus je postavljen

Materijali će biti postavljeni

[Vidi silabus](#)

# HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>KWA</title>
</head>
<body>
  <h1>Klijentske Web aplikacije: KWA</h1>
  <p>
    <strong>Obaveštenja</strong><br/>
    Silabus je postavljen<br/>
    Materijali će biti postavljeni<br/>
  </p>
  <a href="http://predmet.singidunum.ac.rs/mod/page/view.php?id=11883">
    Vidi silabus
  </a>
</body>
</html>
```

## Klijentske Web aplikacije: KWA

### Obaveštenja

Silabus je postavljen  
Materijali će biti postavljeni

[Vidi silabus](#)

# CSS

```
body {  
    font-family: Helvetica;  
    text-align: center;  
}  
  
h1 {  
    text-align: center;  
}  
  
a {  
    text-align: center;  
    display: block; ← ?  
}  
  
p {  
    border: 3px solid hotpink;  
    display: inline-block; ← ?  
    background-color: lavenderblush;  
    text-align: left;  
}
```

## Klijentske Web aplikacije: KWA

### Obaveštenja

Silabus je postavljen  
Materijali će biti postavljeni

[Vidi silabus](#)

**Font:** Helvetica

**Granica:** hotpink 3px

**Boja pozadine:** lavenderblush

Kutija i link su **centrirani**

Sadržaj kutije je **poravnat levo**

# Linkovi

CSS Elementi:

<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

CSS Boje:

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/Values\\_and\\_units#Colors](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Values_and_units#Colors)

Predefinisani nazivi boja:

<http://htmlcolorcodes.com/color-names/>

HTML primer:

<https://codepen.io/mljovanovic/pen/VggdvW?editors=1100>

# Sažetak

## ◆ HTML

- **Sadržaj** koji se prikazuje
- Struktura
- Elementi
- Sintaksa

## ◆ CSS

- **Formatiranje** sadržaja koji se prikazuje
- Sintaksa
- Elementi

## ◆ Primer: HTML + CSS

◆ U pripremi prezentacije korišćene su ilustracije sa  
<https://developer.mozilla.org>

# **HTML i CSS**

## **Prvi deo**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# **HTML i CSS**

## **Drugi deo**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

- ◆ HTML:
- ◆ Block elementi
- ◆ Inline elementi
- ◆ CSS:
- ◆ Selektori elemenata
- ◆ CSS Box model
- ◆ Flexbox
  
- ◆ Primer: [codepen.io](https://codepen.io)

# Podsetnik: HTML elementi

< p >

HTML is < em > sjajan ! ! ! </ em >

< img src = "html5.png" />

< / p >

- ◆ Element obično ima početni i krajnji tag (< p > i < / p >)
  - Sadržaj: sve što se nalazi između početnog i krajnjeg taga
- ◆ Element može biti samozatvarajući (img)
- ◆ Element može imati atribut (src = "html5.png")
- ◆ Element može sadržati druge elemente (p sadrži em i img)

# Tipovi HTML elemenata

- ◆ Svaki HTML element spada u jednu od tri kategorije:
- ◆ **block:** veći blokovi sadržaja koji imaju visinu (height) i širinu (width)  
`<p>, <h1>, <blockquote>, <ol>, <ul>, <table>`
- ◆ **inline:** manje količine sadržaja, nemaju visinu ili širinu  
`<a>, <em>, <strong>, <br>`
  - **inline block:** inline sadržaj sa visinom i širinom `<img>`
- ◆ **metadata:** metainformacije o stranici, nekada nisu vidljive `<title>, <meta>`

# Block elementi

- ◆ Primeri:

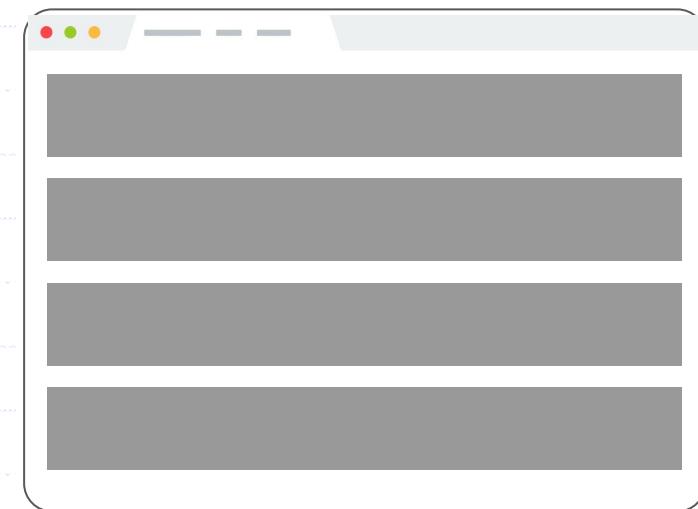
- `<p>, <h1>, <blockquote>, <ol>, <ul>, <table>`

- ◆ Zauzimaju čitavu širinu stranica

- ◆ Raspored: odozgo na dole

- ◆ Mogu imati visinu i širinu

- ◆ Mogu sadržati block ili  
inline elemente



# Block element: primer

```
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS: Block example</title>
  </head>
  <body>
    <h1>About block</h1>
    <p>
      Takes a lot of <em>space</em>
    </p>
  </body>
</html>
```

```
h1 {
  border: 5px solid red;
}
```

`<h1>` je block element  
i podrazumevano  
zauzima čitavu širinu  
stranice

## About block

Takes a lot of *space*

# Širina block elementa se može menjati

```
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS: Block example</title>
  </head>
  <body>
    <h1>About block</h1>
    <p>
      Takes a lot of <em>space</em>
    </p>
  </body>
</html>
```

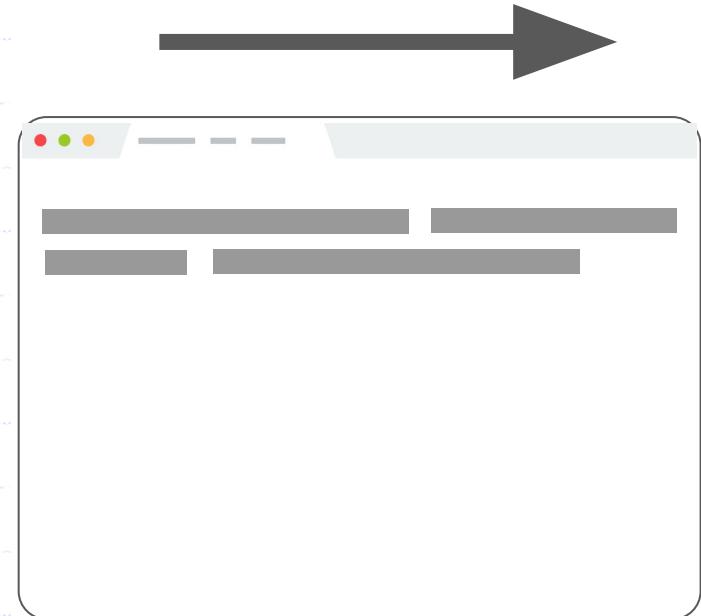
```
h1 {
  border: 5px solid red;
  width: 50%;
}
```

**About block**

Takes a lot of *space*

# Inline elementi

- ◆ Primeri:  
`<a>, <em>, <strong>, <br>`
- ◆ Uzimaju samo onoliko širine koliko im je potrebno
- ◆ Raspored: sa leva na desno
- ◆ **Ne mogu** imati visinu i širinu
- ◆ **Ne mogu** imati block element kao dete
- ◆ **Ne mogu** biti eksplisitno pozicionirani (CSS atributi kao što su float i position ne važe za ovaj tip)
  - Umesto toga, pozicionira se block element koji sadrži dati inline element



# Inline element: primer

```
<strong>Web client applications  
resources:</strong>  
<a  
href="http://predmet.singidunum.ac.  
rs/course/view.php?id=819">WCA</a>  
<a  
href="https://developer.mozilla.org  
/en-US/">MDN</a> <a  
href="http://google.com">Google</a>
```

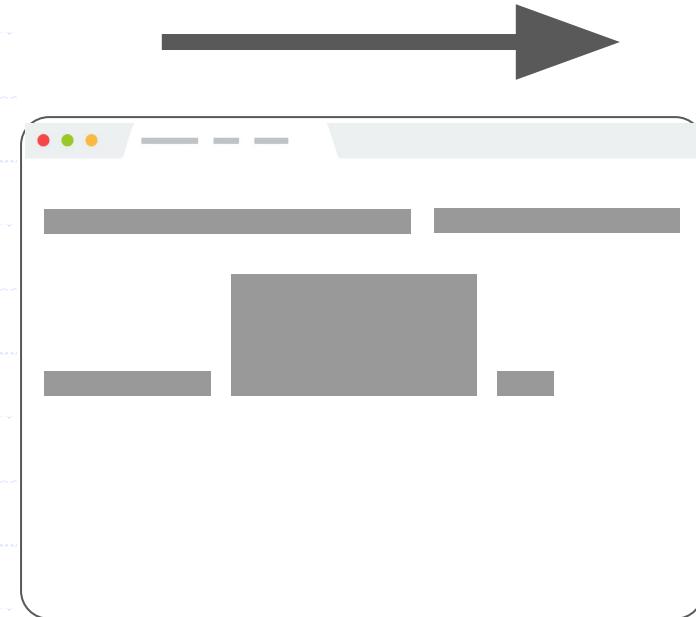
```
strong {  
    border: 5px solid red;  
    width: 1000px;  
}
```

Inline elementi ignoraju atribut širine (ne može biti promenjena !)

Web client applications resources: [WCA](#) [MDN](#) [Google](#)

# Inline-block element

- ◆ Primer:  
`<img>`,  
svaki element sa stilom  
display: inline-block
- ◆ Uzimaju samo onoliko širine  
koliko im je potrebno
- ◆ Raspored: sa leva na desno
- ◆ Mogu imati visinu i širinu
- ◆ Mogu imati block element kao dete
- ◆ Mogu biti eksplicitno pozicionirani  
(CSS atributi kao što su float i position važe)



# Inline-block: primer

- ◆ Ima visinu i širinu (50px); prikaz sa leva na desno

```
  
  
  
  
  
  

```

```
img {  
    width: 50px;  
}
```



# Kako eksplisitno definišemo način prikaza HTML elementa ?

- ◆ Možete promeniti podrazumevan način prikazivanja element pomoću CSS display atributa

```
p { display: inline; }
```

```
a { display: block; }
```

- ◆ Moguće vrednosti display atributa:

block

inline

inline-block

i druge:

<https://developer.mozilla.org/en-US/docs/Web/CSS/display>

# Sažetak

- ◆ **block:** raspored **odozgo na dole; ima visinu i širinu**  
`<p>, <h1>, <blockquote>, <ol>, <ul>, <table>`
- ◆ **inline:** raspored **sa leva na desno; nema visinu i širinu**  
`<a>, <em>, <strong>, <br>`
  - **inline block:** raspored **sa leva na desno; ima visinu i širinu** jednake dimenzijama sadržaja  
`<img>`

# Savet iz prakse

Ako se vaš HTML element ne prikazuje kao što želite, proverite da li pokušavate da примените block attribute na inline elemente

Videti Inline element: primer

# Vratimo se na primer iz prošle lekcije 1/2

```
p {  
    border: 3px solid hotpink;  
    /* display: inline-block; */  
    background-color: lavenderblush;  
    text-align: left;  
}
```

## Klijentske Web aplikacije: KWA

### Obaveštenja

Silabus je postavljen

Materijali će biti postavljeni

[Vidi silabus](#)

**Razlog:** p je block element, širina == širina strane

# Vratimo se na primer iz prošle lekcije 2/2

```
p {  
    border: 3px solid hotpink;  
    display: inline-block;  
    background-color: lavenderblush;  
    text-align: left;  
}
```

## Klijentske Web aplikacije: KWA

Obaveštenja  
Silabus je postavljen  
Materijali će biti postavljeni

[Vidi silabus](#)

**Razlog:** p je block element, širina == širina strane

**Rešenje:** sa display: inline-block zauzima samo neophodnu širinu

# <div> i <span>

- ◆ Dva taga za grupisanje elemenata:
- ◆ <div>: generički **block** element
- ◆ <span>: generički **inline** element

# Primer: span

```
<p>  
  <strong>Obaveštenja</strong><br/>  
  Silabus je postavljen!  
  <span>Od ponedeljka</span><br/>  
  Materijali će biti postavljeni<br/>  
</p>
```

```
p {  
  border: 3px solid hotpink;  
  display: inline-block;  
  background-color: lavenderblush;  
  text-align: left;  
}  
  
span, div {  
  background-color: yellow;  
}
```

Obaveštenja  
Silabus je postavljen! Od ponedeljka  
Materijali će biti postavljeni

# Primer: div

```
<p>  
<strong>Obaveštenja</strong><br/>  
Silabus je postavljen!  
<div>Od ponedeljka</div><br/>  
Materijali će biti postavljeni<br/>  
</p>
```

```
p {  
    border: 3px solid hotpink;  
    display: inline-block;  
    background-color: lavenderblush;  
    text-align: left;  
}  
  
span, div {  
    background-color: yellow;  
}
```

Obaveštenja  
Silabus je postavljen!

Od ponedeljka

Materijali će biti postavljeni

# Podsetnik: CSS elementi

CSS datoteka se sastoji od skupa pravila:

```
selector {  
    property: value;  
}
```

***selector***: HTML element za koji se definiše stil

***property***: Naziv elementa stila (boja, veličina, itd.)

***value***: Vrednost elementa stila

Sačuvano u datoteci *ime\_datoteke.css*

# **CSS Selektori**

**Selektori klasa**

**Selektori tipa elemenata**

**Selektori instanci elemenata**

# Selektori klasa i elemenata

- ◆ Osnovne i kombinovane vrste CSS selektora:

Vrsta	Sintaksa	Ciljni HTML element
Element	p	Svi <p> elementi
Instanca elementa	#abc	element sa id="abc"
Klasa stila	.abc	elementi sa class="abc"
Nezavisni elementi	h1, h2, div	Svi <h1>, <h2> i <div> elementi
Sadržavajući elementi	div strong	<strong> elementi unutar <div> elementa

# class i id HTML elementi

◆ **class** i **id** su specijalni HTML atributi koji se mogu koristiti u okviru svakog elementa

- class: za jedan ili više elemenata; identificuje kolekciju elemenata
- id: identificuje jedan element po stranici

◆ Može se dodeljivati veći broj klasa (odvojenih praznim prostorom):

```
<span class="hw new">HW1</span>
```

◆ Često se koristi sa span i div tagovima za stilizovanje grupe obuhvaćene datim elementom

```
<span class="highlight">
```

# Upotreba sadržavajućih elemenata ?

Umesto:

```
<h1 class="hw">Homework 0</h1>
<p class="hw">Due Fri</p>
<p class="hw">Late cutoff Sun</p>
```

Preporučuje se:

```
<div class="hw">
  <h1>Homework 0</h1>
  <p>Due Fri</p>
  <p>Late cutoff Sun</p>
</div>
```

Umesto primene stila na odvojene susedne elemente, oni se mogu grupisati pomoću `<div>` elementa na koji se primenjuje stil

# Preklapanje stilova

- ◆ Kada se stilovi preklapaju, najspecifičniji stil se primenjuje

```
div strong { color: red; }  
strong { color: blue; }
```

```
<div>  
    <strong>Koje sam boje ?</strong>  
</div>
```

# Preklapanje stilova

- ◆ Kada se stilovi preklapaju, najspecifičniji stil se primenjuje

```
div strong { color: red; }  
strong { color: blue; }
```

```
<div>  
    <strong>Koje sam boje ?</strong>  
</div>
```

# Preklapanje stilova

- ◆ Pravila specifičnosti ([detalji](#)):
- ◆ Selektori instanci (id) su specifičniji od selektora klase (class)
- ◆ Selektori klase (class) su specifičniji od selektora tipova elemenata (npr. p)
- ◆ Pravila stila koja se primenjuju direktno na elemente su specifičnija od nasleđenih pravila stila

# Preklapanje stilova

- ◆ Ako elementi imaju istu specifičnost, poslednje navedeno pravilo se primenjuje

```
strong { color: red; }  
strong { color: blue; }
```

```
<div>  
    <strong>Koje sam boje ?</strong>  
</div>
```

Napomena: Proces utvrđivanja koje pravilo stila se primenjuje sa naziva cascade. Odavde potiče "C" u nazivu jezika (**Cascading Style Sheets**).

# Nasleđivanje stilova

- ◆ CSS stilovi se nasleđuju od roditelja ka deci

Umesto navođenja svakog elementa ponaosob:

```
a, h1, p, strong {  
    font-family: Helvetica;  
}
```

Možemo stilizovati roditeljski element tako da će elementi deca naslediti stilove:

Stilovi se mogu redefinisati u elementima deci:

```
body {  
    font-family: Helvetica;  
}  
  
h1, h2 {  
    font-family: Consolas;  
}
```

# Nasleđivanje stilova

- ◆ Dok se većina CSS atributa nasleđuje od roditelja ka detetu, **nisu svi CSS atributi nasledni**
  - ◆ Ne postoje precizno definisana pravila o tome koji se atributi nasledni, a koji ne
  - ◆ Detalji se mogu naći u MDN opisu atributa:

# font-family:

## Inherited yes

# display:

no

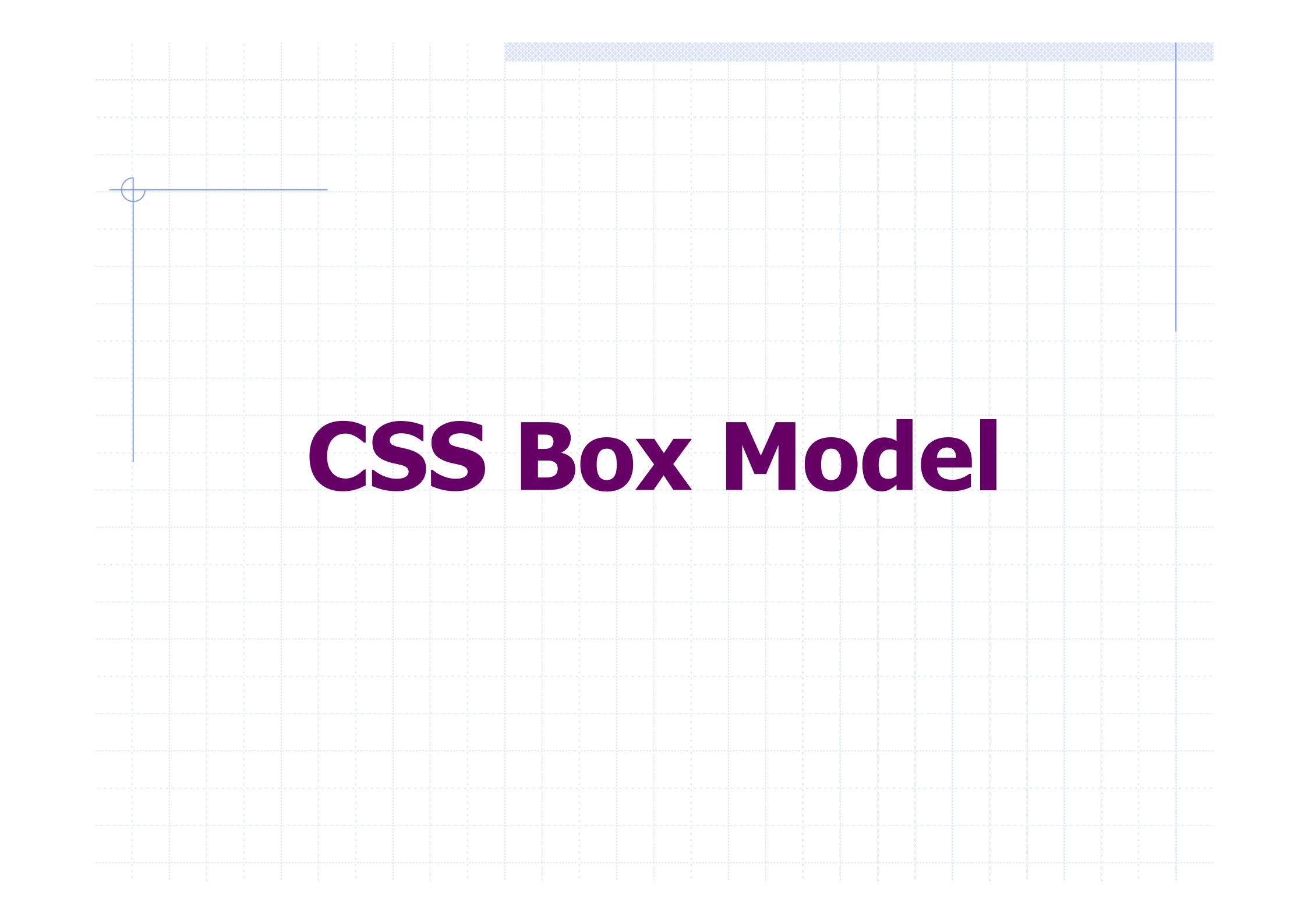
- ◆ Opšte pravilo: atributi koji se odnose na tekst su nasledni, dok atributi koji se odnose na raspored (layout) nisu nasledni
  - ◆ display atribut nije nasledan !

# CSS pseudo-klase

- ◆ Ključne reči koje se mogu dodati selektorima i koje dodatno opisuju stanje ili specifičnu osobinu selektora

Sintaksa	Objašnjenje
a	Svi a tagovi u svim stanjima
a:visited	Posećeni link
a:link	Neposećen link
a:hover	Stil kada je pokazivač pozicioniran između linka
a:active	Stil u trenutku aktivacije linka (downclick)

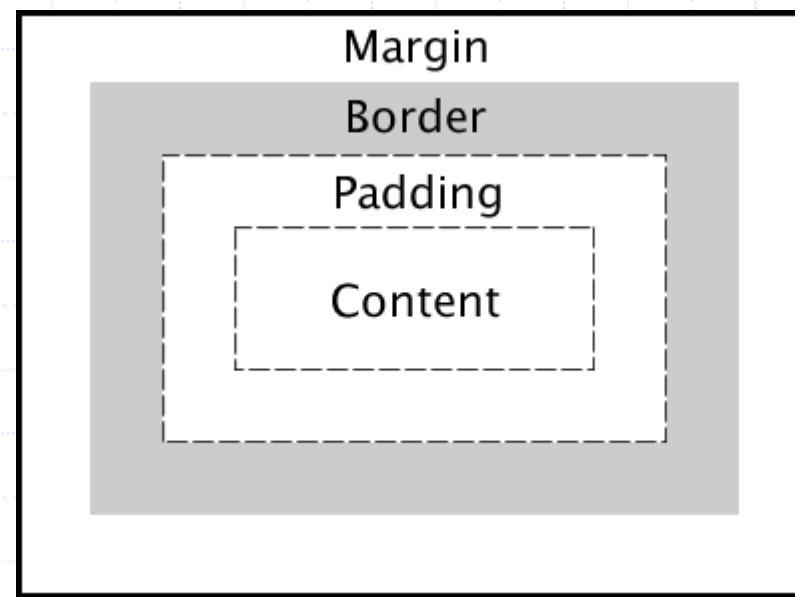
- ◆ Postoje i druge pseudo klase ([link](#))



# CSS Box Model

# CSS Box Model

- ◆ Svaki element je sastavljen od četiri sloja:
- ◆ Sadržaj elementa (**content**)
- ◆ Granica oko sadržaja elementa (**border**)
- ◆ Prostor između sadržaja i granice elementa (**padding**)
- ◆ Prostor margine oko granice (**margin**)



# CSS Box Model: border

```
<p>  
<strong>Obaveštenja</strong><br/>  
Silabus je postavljen<br/>  
Materijali će biti postavljeni<br/>  
</p>
```

```
p {  
    border: 3px solid hotpink;  
    display: inline-block;  
    background-color: lavenderblush;  
    text-align: left;  
}
```

**Obaveštenja**  
Silabus je postavljen  
Materijali će biti postavljeni

Skraćena sintaksa:  
border: *debljina stil boja*

# CSS Box Model: border

- ◆ Svaka strana granice se može specificirati individualno:
  - ◆ border-top
  - ◆ border-bottom
  - ◆ border-left
  - ◆ border-right
- ◆ Svako svojstvo se može navesti odvojeno:
  - ◆ border-style: dotted
  - ◆ border-width: 3px
  - ◆ border-color: purple

# CSS Box Model: border radius

- ◆ Može se specificirati i border-radius radi dobijanja zaobljenih ivica granica
- ◆ U tom slučaju nije potrebno koristiti border atribut

```
<p>
<strong>Obaveštenja</strong><br/>
Silabus je postavljen<br/>
Materijali će biti postavljeni<br/>
</p>
```

```
p {
background-color: purple;
border-radius: 10px;
color: white;
display: inline-block;
}
```

**Obaveštenja**  
Silabus je postavljen  
Materijali će biti postavljeni

# CSS Box Model: border

- ◆ Sama granica se nalazi odmah uz sadržaj



- ◆ Na koji način dodajemo prostor između granice i sadržaja elementa ?

# CSS Box Model: padding

```
p {  
    border: 2px solid black;  
    padding: 20px;  
    display: inline-block;  
}
```

**Obaveštenja**  
Silabus je postavljen  
Materijali će biti postavljeni

**padding** je prostor između granice i sadržaja elementa  
Može se navesti padding-top, padding-bottom, padding-left,  
padding-right

Skraćene sintakse:

padding: 2px 4px 3px 1px; <- vrh | desno | dno | levo  
padding: 10px 2px; <- vrh+dno | levo+desno

# CSS Box Model: padding

Ako dodamo  
granice susednim  
div elementima oni  
izgledaju spojeno

```
<div>Lekcije</div>
<div>Domaci zadaci</div>
```

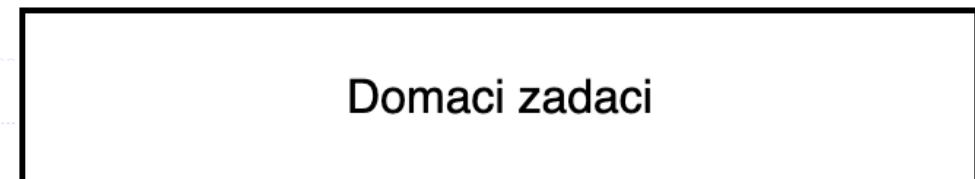
```
div {
    border: 2px solid black;
    padding: 20px;
    display: block;
}
```

Kako možemo  
dodati prostor  
između višestrukih  
elemenata ?



# CSS Box Model: margin

```
div {  
    border: 2px solid black;  
    padding: 20px;  
    margin: 20px;  
    display: block;  
}
```



**margin** je prostor između granice elementa i drugih, susednih elemenata

Može se navesti margin-top, margin-bottom, margin-left, margin-right

Skraćena sintaksa:

margin: 2px 4px 3px 1px; <- vrh | desno | dno | levo

margin: 10px 2px; <- vrh+dno | levo+desno

# CSS Box Model: margin

- ◆ Spajanje margina (margin collapsing)
- ◆ Gornja i donja margina blok elemenata se nekada spajaju u jednu marginu čija je veličina jednaka najvećoj individualnoj margini
- ◆ Tri osnovna slučaja:
  - ◆ Margine susednih elemenata
  - ◆ Roditelj i prvo/poslednje dete
  - ◆ Prazni blok elementi

20px donja margina +  
20px gornja margina =  
margina 40px ?

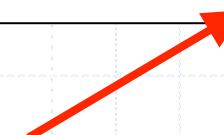


# Podrazumevane vrednosti margina

```
<div>
<p>Žuta boja pozadine!</p>
<p>Koja okružuje paragraf</p>
</div>
```

```
div {
    display: inline-block;
    background-color: yellow;
}
```

Žuta boja pozadine!  
Koja okružuje paragraf



Margine oko div elementa

# body element ima podrazumevane margine

- ◆ Vrednost se mora eksplisitno redefinisati

```
<div>
<p>Žuta boja pozadine!</p>
<p>Koja okružuje paragraf</p>
</div>
```

```
body {
    margin: 0;
}

div {
    display: inline-block;
    background-color: yellow;
}
```

Žuta boja pozadine!

Koja okružuje paragraf

# CSS position atribut

Sintaksa	Značenje
<code>position: static;</code>	podrazumevana pozicija elementa u dokumentu na tok raspoređivanja (top, right, bottom, left atributi se ne uzimaju u obzir)
<code>position: relative;</code>	pozicija relativna u odnosu na top, right, bottom i left attribute
<code>position: fixed;</code>	fiksna lokacija na ekranu na osnovu top, right, bottom i left attribute
<code>position: absolute;</code>	pozicija relativna u odnosi na element-roditelj na osnovu top, right, bottom i left atributa

- ◆ Koordinatni početak elementa (0,0) je uvek gornji levi ugao

# CSS display atribut

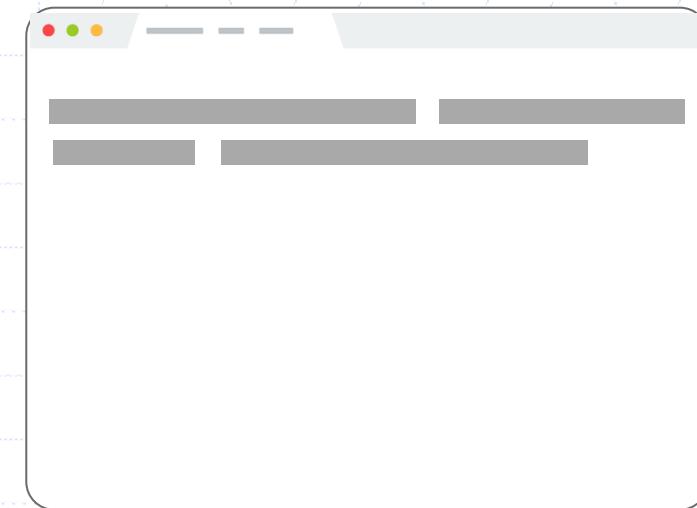
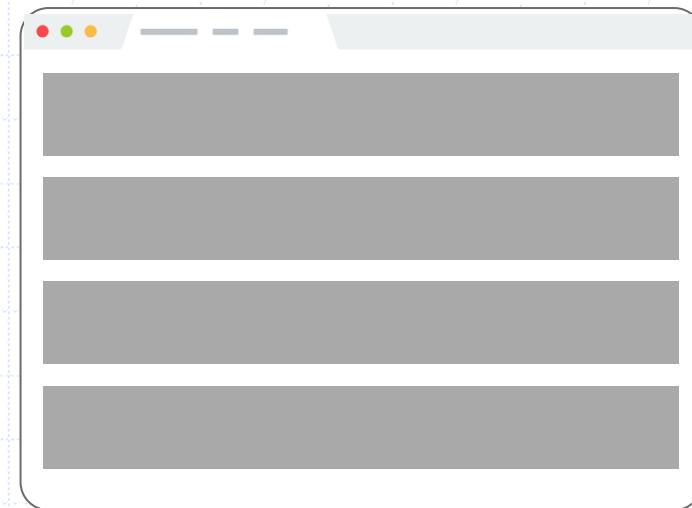
- ◆ Definiše način prikazivanja elementa
- ◆ Najčešće korišćeni CSS element
  - Jedan od najčešćih uzroka problema

Vrednost	Značenje
display: none;	element nije prikazan i ne zauzima prostor
display: inline;	element se tretira kao inline tip elementa
display: block;	element se tretira kao block tip elementa
display: inline-block;	element se tretira kao inline-block tip elementa
visibility: hidden;	element nije prikazan, ali je prostor alociran
display: flex;	FlexBox kontejner ?



# CSS Flexbox

# CSS layout do sada

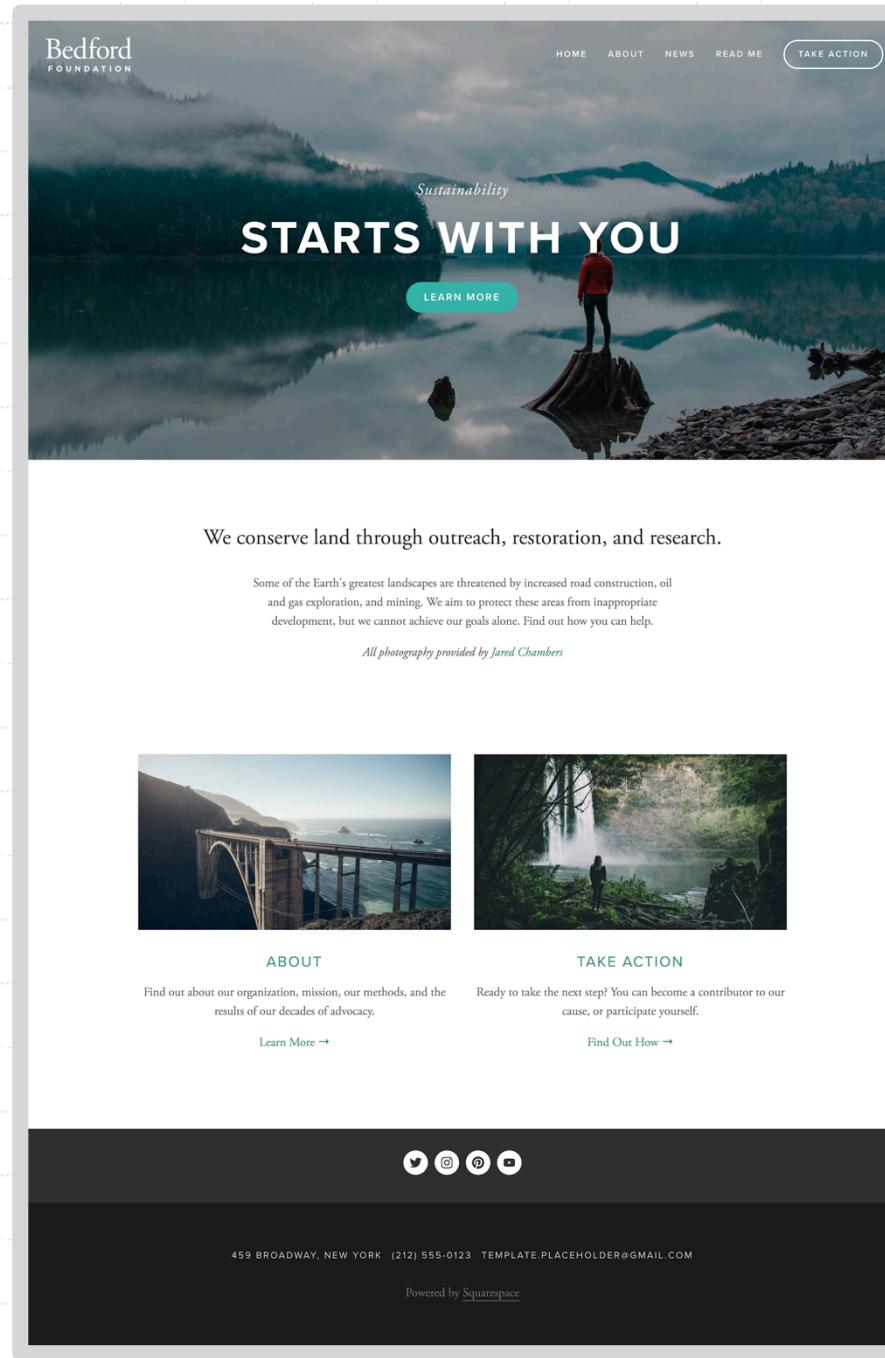


## Block layout:

Raspoređivanje većih sekcija strane

## Inline layout:

Raspoređivanje teksta i manjeg sadržaja unutar sekciije



The Bedford Foundation website features a large banner at the top with a photograph of a person standing on a log in a lake, with mountains in the background. The text "Sustainability" is written in a small font above the main title "STARTS WITH YOU". Below the title is a "LEARN MORE" button. The navigation bar at the top includes links for HOME, ABOUT, NEWS, READ ME, and TAKE ACTION.

We conserve land through outreach, restoration, and research.

Some of the Earth's greatest landscapes are threatened by increased road construction, oil and gas exploration, and mining. We aim to protect these areas from inappropriate development, but we cannot achieve our goals alone. Find out how you can help.

All photography provided by Jared Chambers

**ABOUT**

Find out about our organization, mission, our methods, and the results of our decades of advocacy.

[Learn More →](#)

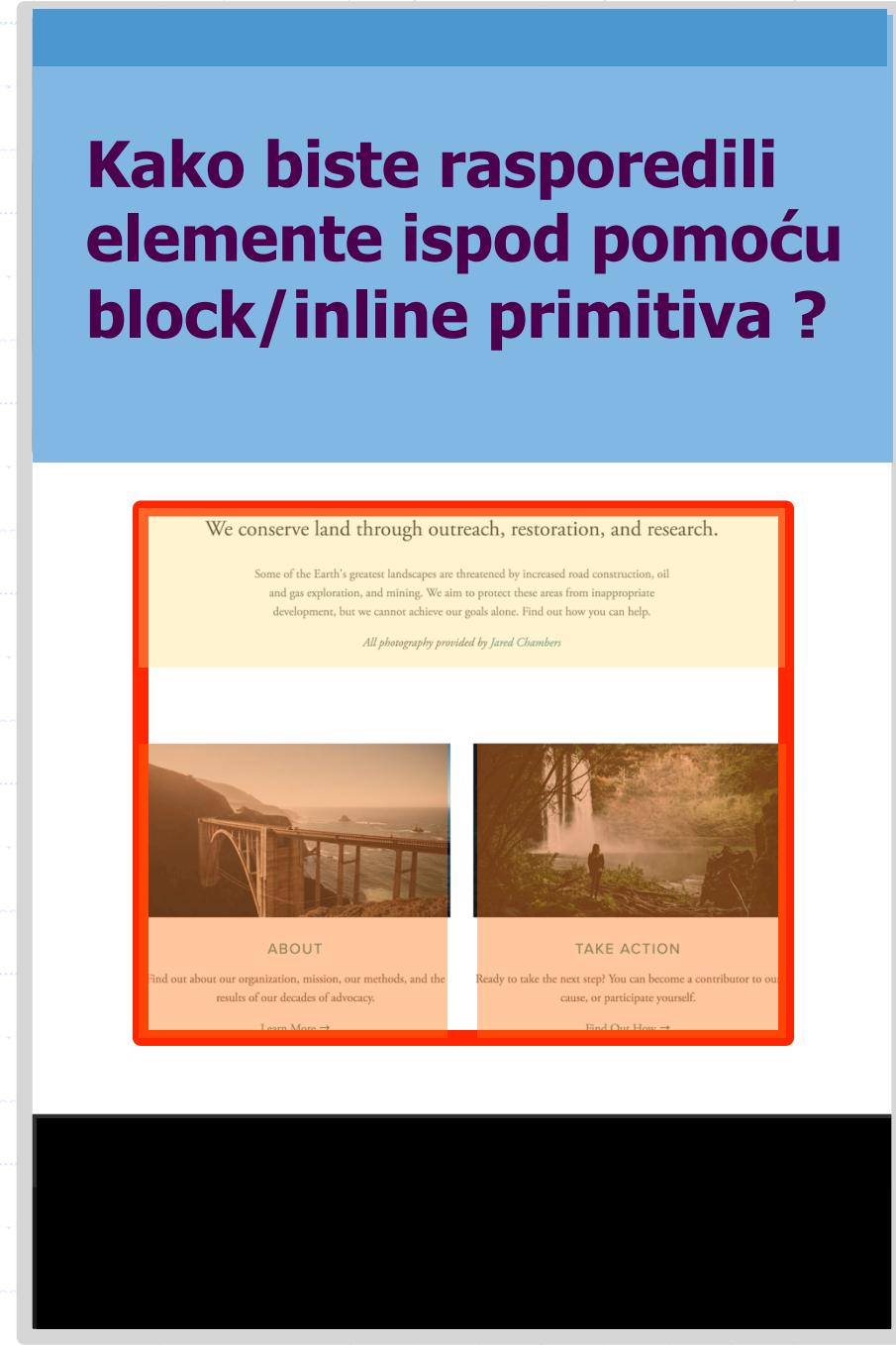
**TAKE ACTION**

Ready to take the next step? You can become a contributor to our cause, or participate yourself.

[Find Out How →](#)

459 BROADWAY, NEW YORK (212) 555-0123 TEMPLATE.PLACEHOLDER@GMAIL.COM

Powered by Squarespace



Kako biste rasporedili elemente ispod pomoću block/inline primitiva ?

We conserve land through outreach, restoration, and research.

Some of the Earth's greatest landscapes are threatened by increased road construction, oil and gas exploration, and mining. We aim to protect these areas from inappropriate development, but we cannot achieve our goals alone. Find out how you can help.

All photography provided by Jared Chambers

**ABOUT**

Find out about our organization, mission, our methods, and the results of our decades of advocacy.

[Learn More →](#)

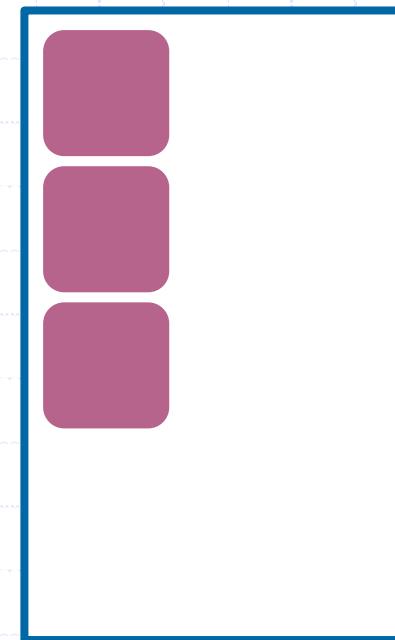
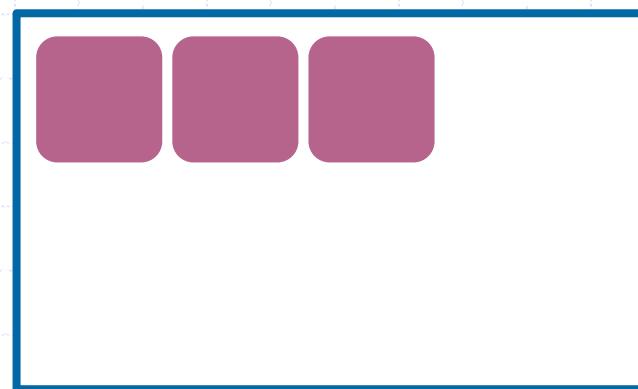
**TAKE ACTION**

Ready to take the next step? You can become a contributor to our cause, or participate yourself.

[Find Out How →](#)

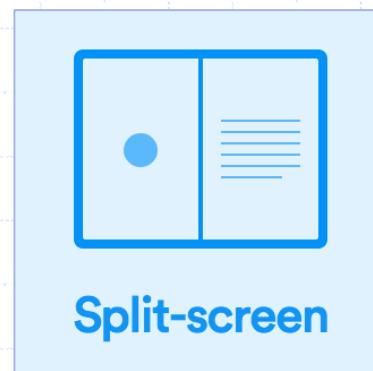
# CSS Flexbox layout

- ◆ **Flex** = fleksibilno
- ◆ **Box** = kontejner, tj. roditelj
- ◆ Definiše skup jednostavnijih pravila za raspoređivanje elemenata po redovima i/ili kolonama

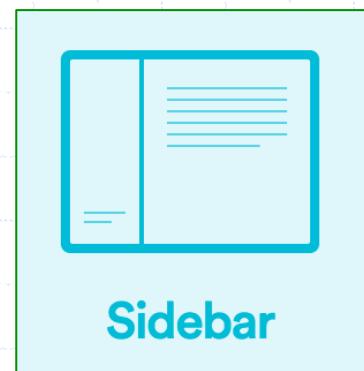


# CSS Flexbox

- ◆ Primeri rasporeda (layout-a) koji se mogu jednostavnije dizajnirati
  - Inače znatno komplikovanje



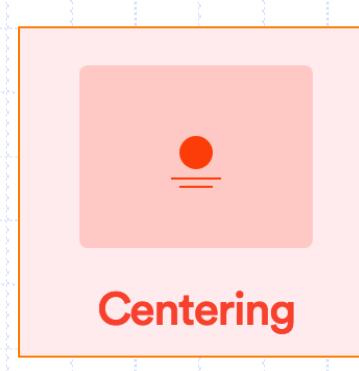
Split-screen



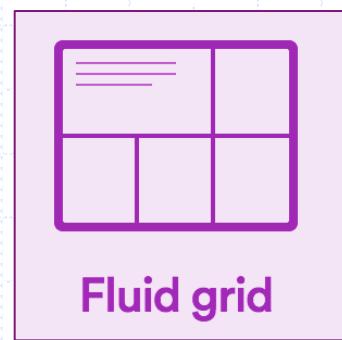
Sidebar



Sticky footer



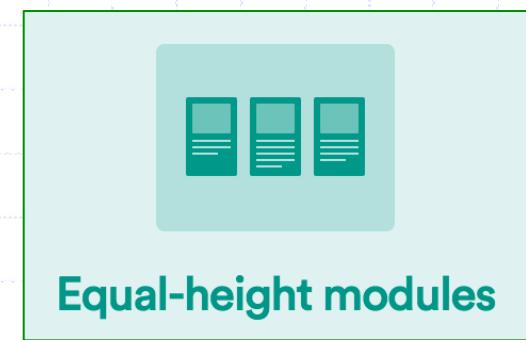
Centering



Fluid grid

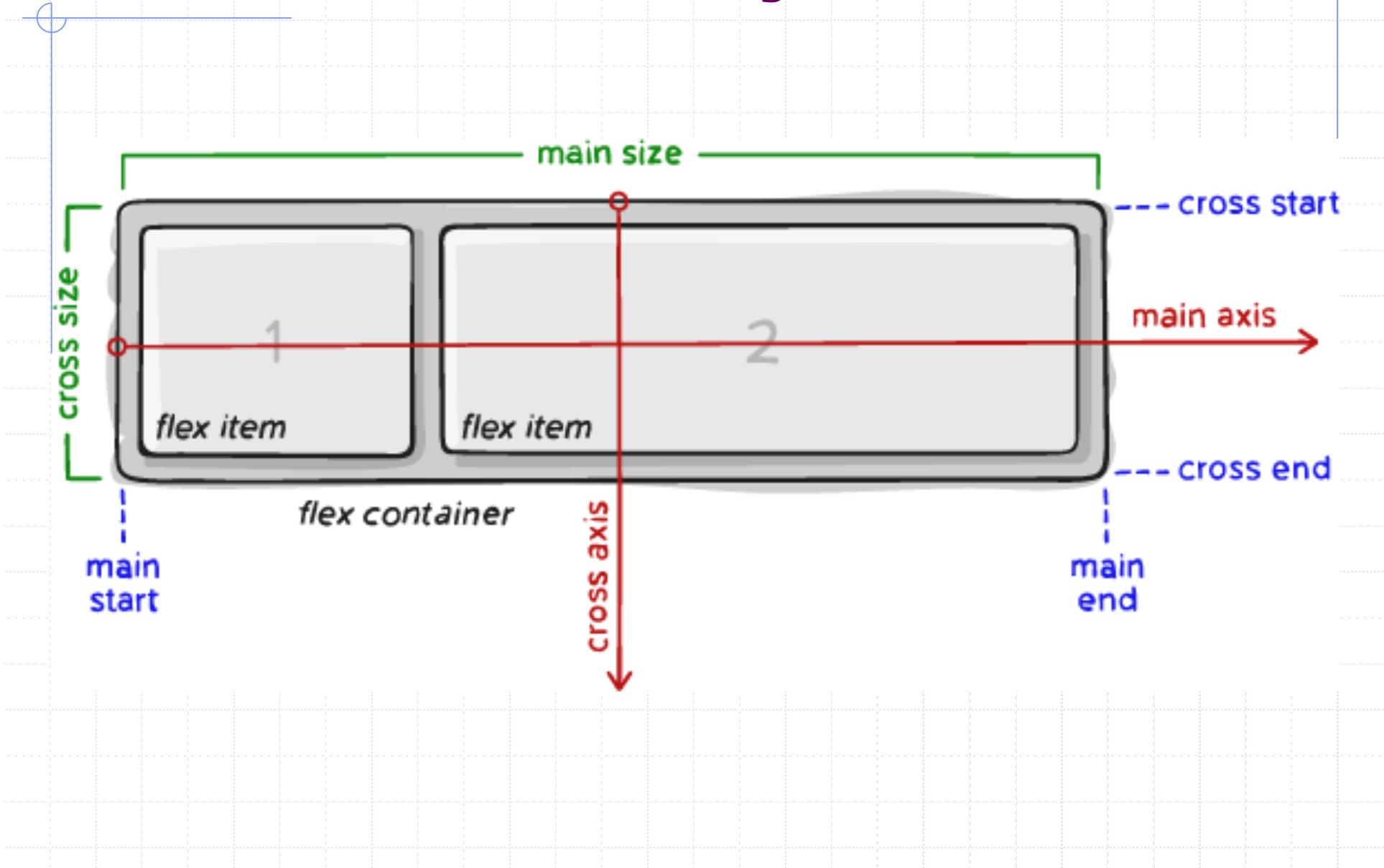


Collection grid



Equal-height modules

# Šema Flexbox kontejnera



# Osnove Flexbox-a

Flex rasporedi se sastoje od  
**Flex kontejnera**, koji sadrži jedan ili više  
**Flex elemenata (items)**

Možete primenjivati CSS atribut na **flex kontejner**  
da uredite raspored elemenata

`id=flex-container`

`class=`  
`flex-`  
`item`

# Osnove Flexbox-a

- ◆ Da biste element proglašili flex kontejnerom, promenite display atribut:
  - ◆ **block** kontejnera: **display: flex;**
  - ◆ **inline** kontejnera: **display: inline-flex;**
  
- ◆ Pravila:
- ◆ **flex-direction** definiše pravac glavne ose (*main-axis*)
  - vrednosti: row ili column
- ◆ **justify-content** raspoređuje decu po glavnoj osi
- ◆ **align-items** raspoređuje decu po upravnoj osi (*cross-axis*)

# Osnove Flexbox-a: justify-content

Možete pozicionirati element po **horizontali** u kontejneru pomoću **justify-content**:

```
#flex-container {  
    display: flex;  
    justify-content: flex-start;  
}
```



# Osnove Flexbox-a: justify-content

Možete pozicionirati element po **horizontali** u kontejneru pomoću **justify-content**:

```
#flex-container {  
    display: flex;  
    justify-content: flex-end;  
}
```



# Osnove Flexbox-a: justify-content

Možete pozicionirati element po **horizontali** u kontejneru pomoću **justify-content**:

```
#flex-container {  
    display: flex;  
    justify-content: center;  
}
```



# Osnove Flexbox-a: align-items

Možete pozicionirati element po **vertikali** u kontejneru pomoću **align-items**:

```
#flex-container {  
    display: flex;  
    align-items: flex-start;  
}
```



# Osnove Flexbox-a: align-items

Možete pozicionirati element po **vertikali** u kontejneru pomoću **align-items**:

```
#flex-container {  
    display: flex;  
    align-items: flex-end;  
}
```



# Osnove Flexbox-a: align-items

Možete pozicionirati element po **vertikali** u kontejneru pomoću **align-items**:

```
#flex-container {  
    display: flex;  
    align-items: center;  
}
```



# Flexbox: razmeštaj većeg broja elemenata

Možete pozicionirati element po **horizontali** i **vertikali** u kontejneru pomoću:

```
#flex-container {  
    display: flex;  
justify-content: flex-start;  
align-items: center;  
}
```



# Flexbox: razmeštaj većeg broja elemenata

Možete pozicionirati element po **horizontali** i **vertikali** u kontejneru pomoću:

```
#flex-container {  
    display: flex;  
    justify-content: flex-end;  
    align-items: center;  
}
```



# Flexbox: razmeštaj većeg broja elemenata

Možete pozicionirati element po **horizontali** i **vertikali** u kontejneru pomoću:

```
#flex-container {  
    display: flex;  
justify-content: center;  
align-items: center;  
}
```



# Flexbox: razmeštaj većeg broja elemenata

Možete pozicionirati element po **horizontali** i **vertikali** u kontejneru pomoću:

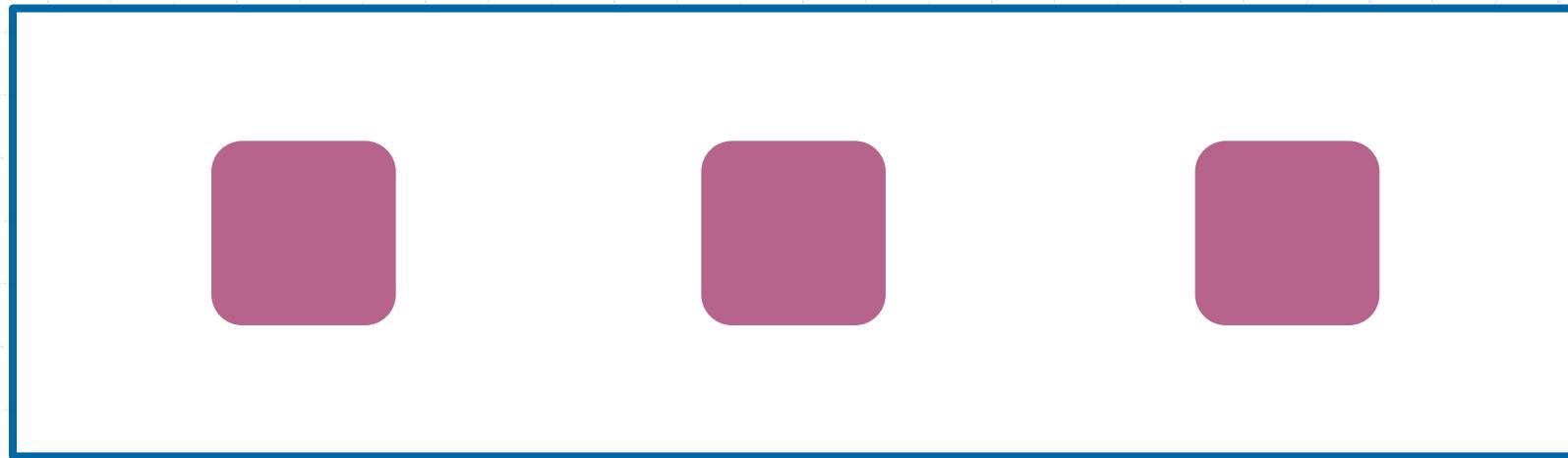
```
#flex-container {  
    display: flex;  
justify-content: space-between;  
align-items: center;  
}
```



# Flexbox: razmeštaj većeg broja elemenata

Možete pozicionirati element po **horizontali** i **vertikali** u kontejneru pomoću:

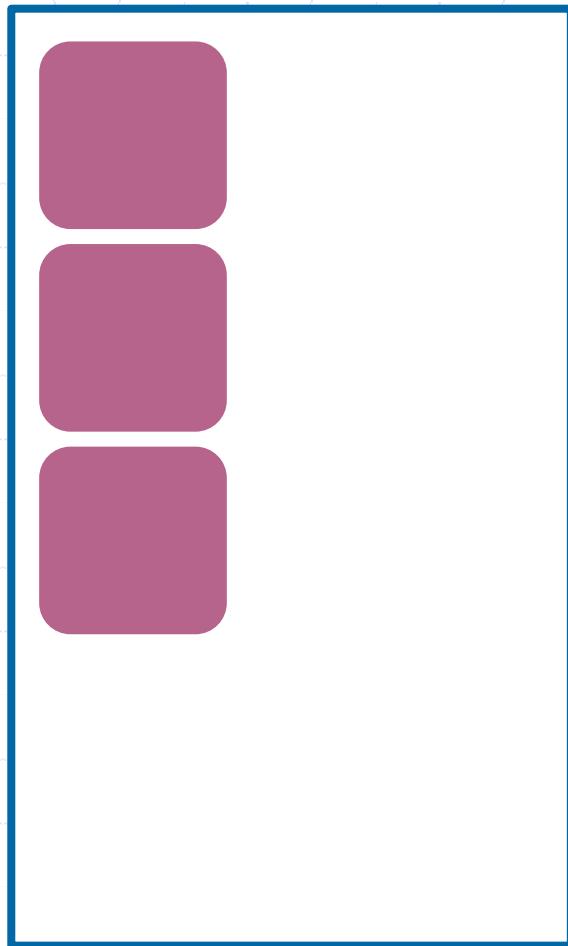
```
#flex-container {  
    display: flex;  
justify-content: space-around;  
align-items: center;  
}
```



# flex-direction

- ◆ Elementi se mogu rasporedjivati i po kolonama

```
#flex-container {  
    display: flex;  
flex-direction: column;  
}
```

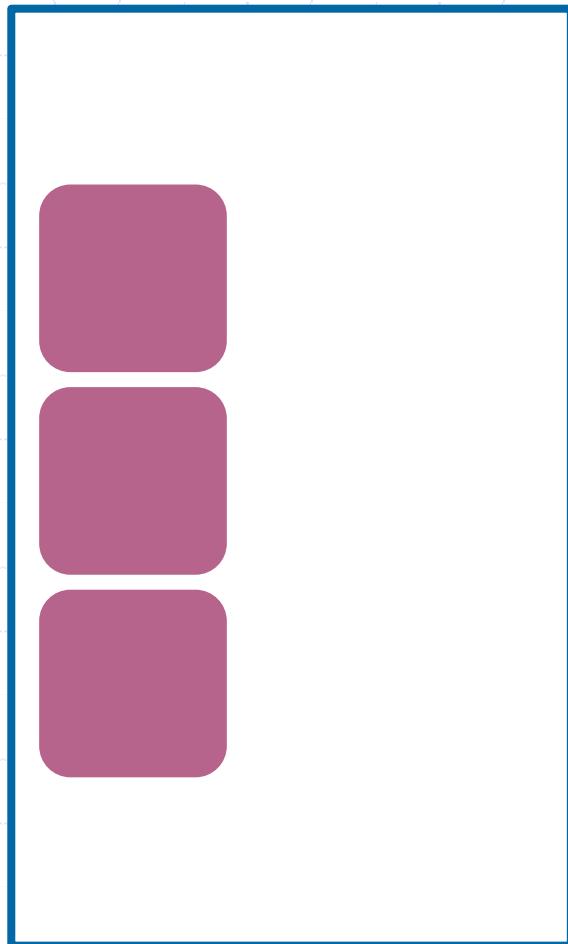


# flex-direction

- ◆ Elementi se mogu raspoređivati i po kolonama

```
#flex-container {  
    display: flex;  
    flex-direction: column;  
    justify-content: center;  
}
```

Sada **justify-content**  
raspoređuje elemente vertikalno

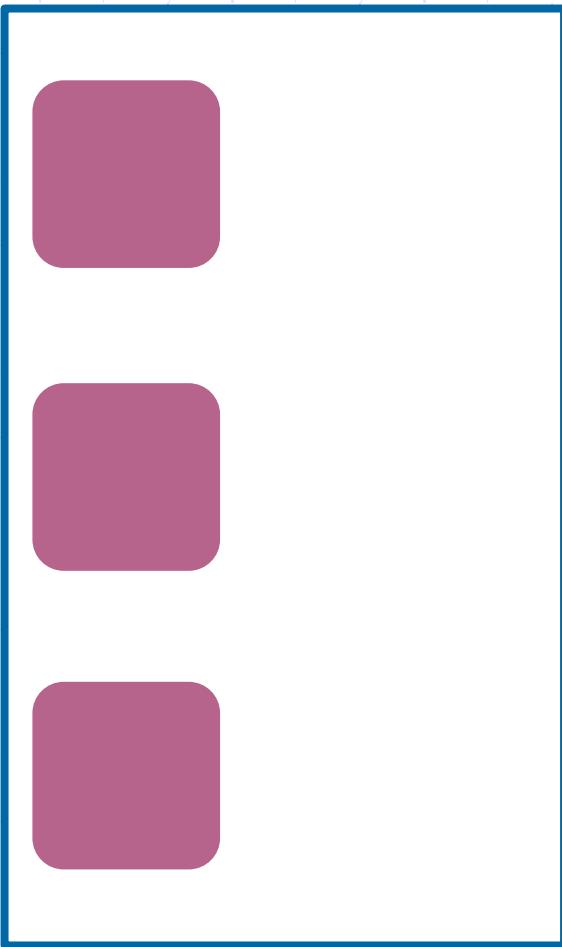


# flex-direction

- ◆ Elementi se mogu raspoređivati i po kolonama

```
#flex-container {  
    display: flex;  
    flex-direction: column;  
    justify-content: space-around;  
}
```

Sada **justify-content**  
raspoređuje elemente vertikalno

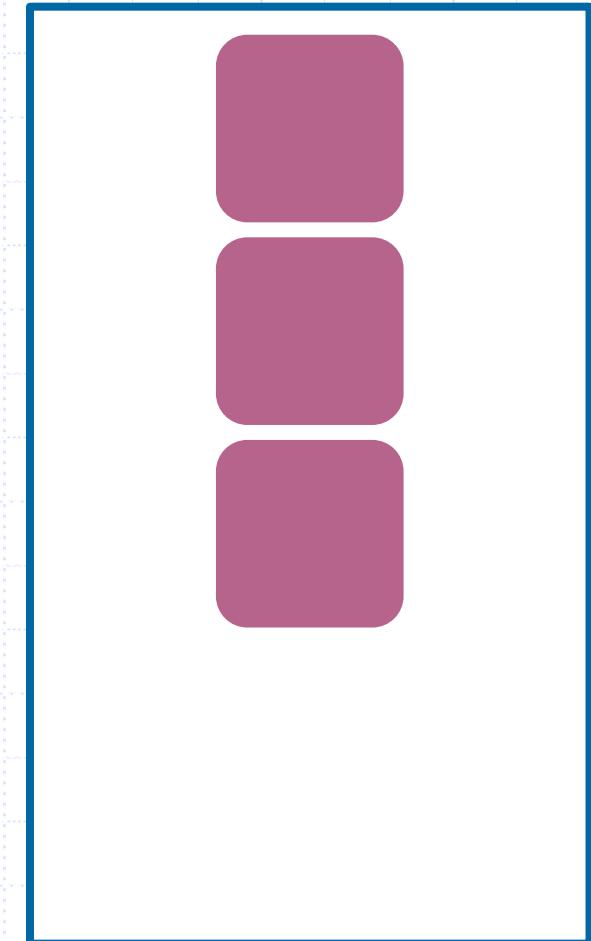


# flex-direction

- ◆ Elementi se mogu raspoređivati i po kolonama

```
#flex-container {  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
}
```

Sada **align-items** kontroliše poziciju kolone horizontalno

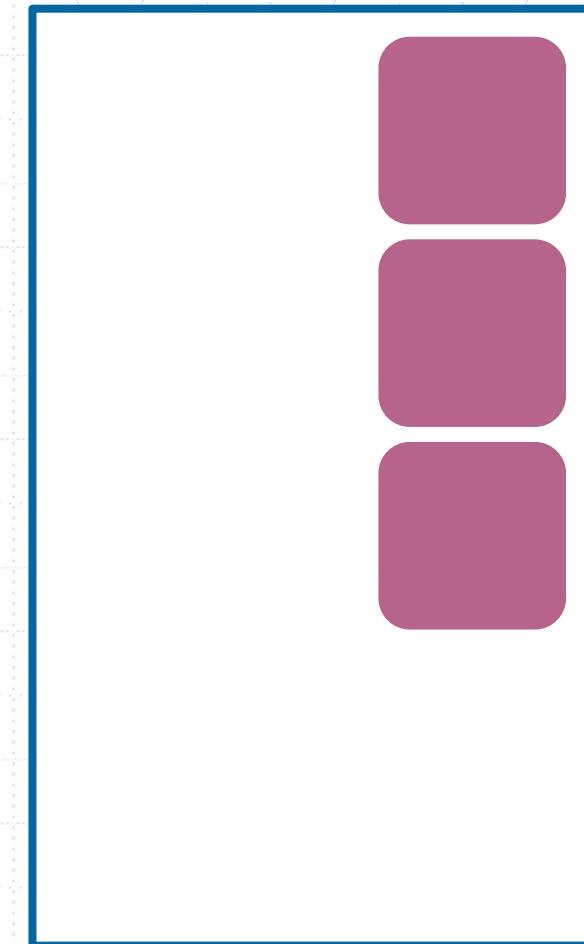


# flex-direction

- ◆ Elementi se mogu raspoređivati i po kolonama

```
#flex-container {  
    display: flex;  
    flex-direction: column;  
    align-items: flex-end;  
}
```

Sada **align-items** kontroliše poziciju kolone horizontalno



# Linkovi ka primerima

Block primer:

<https://codepen.io/mljovanovic/pen/zbOMyO>

Inline primer:

<https://codepen.io/mljovanovic/pen/vPBvbz>

Inline-block primer:

<https://codepen.io/mljovanovic/pen/vPBPQx>

Div vs. span primer:

<https://codepen.io/mljovanovic/pen/vPYwoq>

CSS Box model primer:

<https://codepen.io/mljovanovic/pen/WmQmwN>

CSS Box model default vrednosti:

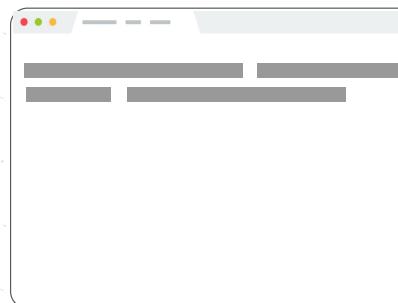
<https://codepen.io/mljovanovic/pen/bZELYj>

# Sažetak 1/2

## ◆ Block HTML elementi



## ◆ Inline HTML elementi



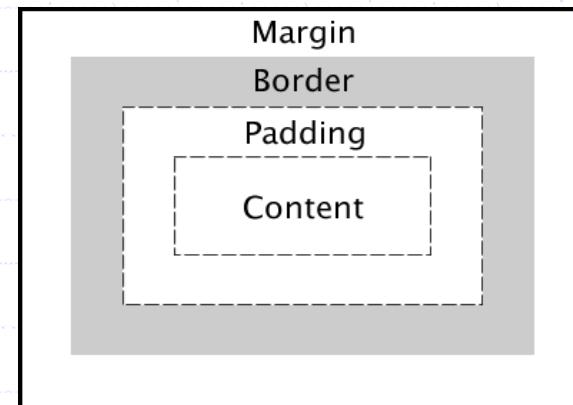
## ◆ CSS display (block ↔ inline)

# Sažetak 2/2

## ◆ CSS selektori

- Selektori instanci elemenata (#p)
  - Selektori klasa (.class)
  - Selektori tipa elemenata (p)
  - Nezavisni i sadržavajući elementi
- } opadajuća specifičnost ↓

## ◆ CSS Box model



## ◆ Flexbox

- Fleksibilno pozicioniranje po horizontali i/ili vertikali

# **HTML i CSS**

## **Drugi deo**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# JavaScript osnove

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

- ◆ JavaScript:
- ◆ Princip rada
- ◆ Proceduralni aspekti
  - Operatori
  - Funkcije
  - Varijable
  - Tipovi podataka (templejt stringovi)
  - Nizovi
- ◆ Objektno-orientisani aspekti
  - Metode
  - Funkcije kao objekti
  - Prototipi
  - Klase i nasleđivanje
- ◆ Aspekti funkcionalnog programiranja
  - Anonimne funkcije
  - Arrow funkcije
- ◆ Primeri: [codepen.io](http://codepen.io)

# JavaScript istorijat

- ◆ Nastao 1995 (Brendan Eich)
- ◆ Ko-osnivač Mozilla-e
- ◆ Nema naročitih veza sa Java jezikom
  - Razlozi isključivo marketinški
- ◆ Prva verzija jezika napisana za 10 dana
- ◆ Glavne odluke o jeziku su donete zbog politike kompanije, ne tehničkih (stručnih) razloga
- ◆ Vremenom je jezik unapređivan
  - Tradicionalni JavaScript – globalni opsezi; nema klasa/modula; mogu biti veće količine koda uređenog sekvencialno
  - Moderni (unapređeni) JavaScript – klase, moduli; unapređene mogućnosti jezika, i izgled i organizacija koda;
  - ECMAScript (TypeScript)

# JavaScript programski jezik

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

# JavaScript u Web stranici

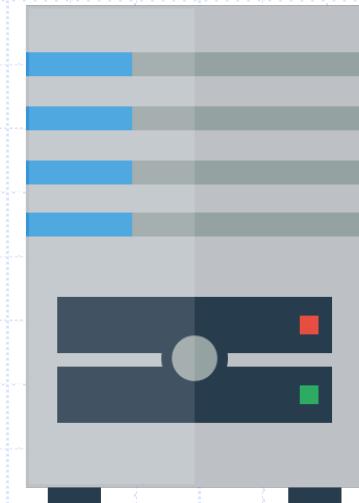
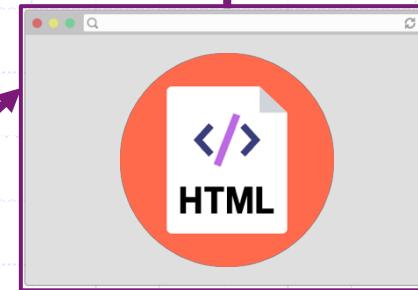
```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>CS 193X</title>  
    <link rel="stylesheet" href="style.css" />  
    <script src="filename.js"></script>  
  </head>  
  <body>  
    ... contents of the page...  
  </body>  
</html>
```

# console.log()

- ◆ Ispisivanje log poruka u JavaScript-u

# Kako radi JavaScript ?

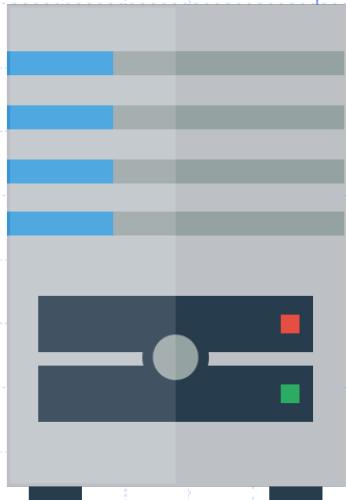
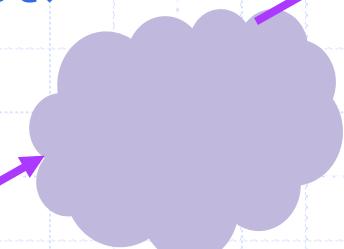
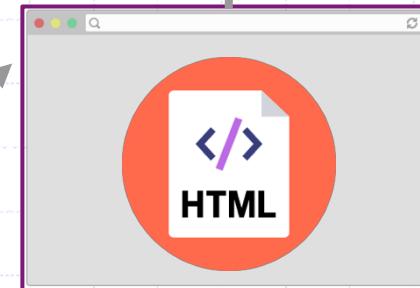
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



Pretraživač parsira HTML datoteku, i iz script taga čita nazine JavaScript datoteka.

# Kako radi JavaScript ?

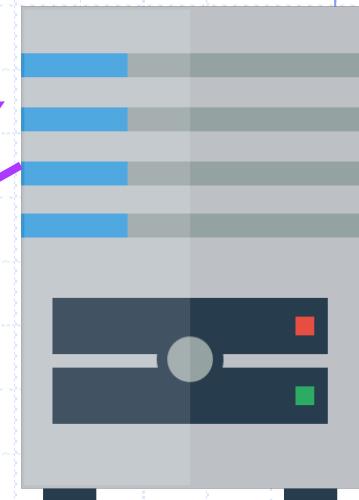
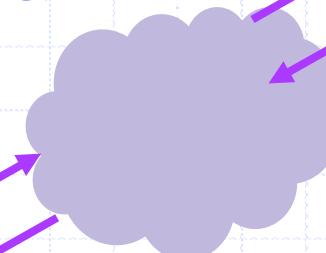
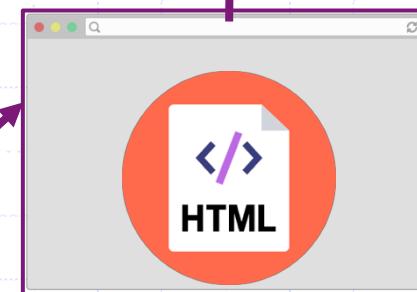
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  → <script src="script.js"></script>
</head>
```



Pretraživač šalje zahtev serveru za datoteku script.js, slično  
kao za CSS datoteku ili sliku ...

# Kako radi JavaScript ?

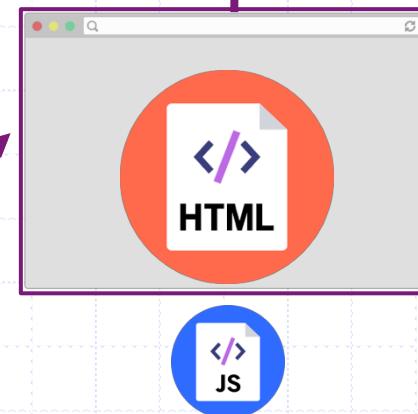
```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



I server šalje JavaScript datoteku, kao što bi poslao CSS datoteku ili sliku ...

# Kako radi JavaScript ?

```
<head>
  <title>CS 193X</title>
  <link rel="stylesheet" href="style.css" />
  <script src="script.js"></script>
</head>
```



```
console.log('Hello, world!');
```

U ovom trenutku, JavaScript datoteka se izvršava na klijentskoj strani, tj. u pretraživaču računara korisnika.

# Kako radi JavaScript ?

- ◆ Ne postoji **main** metoda:
- ◆ Instrukcije se izvršavaju sekvencijalno od vrha ka dnu
- ◆ Ne postoji prevođenje (kompilacija) koda od strane programera
- ◆ JavaScript datoteke se prevode i izvršavaju kako se učitavaju od strane pretraživača
- ◆ Just-in-time (JIT) prevođenje
- ◆ V8-Ignition je najpopularniji JavaScript engine (Chrome, NodeJS):
  - ◆ Princip je da se kod prevodi i izvršava što je brže moguće:
  - ◆ Izvorni JavaScript kod se prevodi u byte kod
  - ◆ Byte kod se izvršava od strane interpretera
  - ◆ SuperMonkey (Mozilla)



# **Elementi JavaScript jezika**

# Operatori

```
i = 3;  
i = i * 10 + 3 + (i / 10);  
while (i >= 0) {  
    sum += i*i;  
    i--;
```

Standardne operacije:

/ % + - ! >= <= > < && || ?:

Naredbe skoka:

continue/break/return

# Petlje i naredbe grananja

**for** petlja:

```
for (let i = 0; i < 5; i++) { ... }
```

**while** petlja:

```
while (notFinished) { ... }
```

**komentari:**

```
// comment ili /* comment */
```

uslovne (**if**) naredbe:

```
if (...) {  
  ...  
} else {  
  ...  
}
```

# Funkcije

Jedan od načina za definisanje JavaScript funkcije je kao ispod:

```
function name() {  
    statement;  
    statement;  
    ...  
}
```

# Funkcije

```
function hello() {  
    console.log('Hello!');  
    console.log('Welcome to JavaScript');  
}  
  
hello();  
hello();
```

Console

"Hello!"

"Welcome to JavaScript"

"Hello!"

"Welcome to JavaScript"

# Da li će biti izvršeno ?

```
hello();  
hello();
```

```
function hello() {  
    console.log('Hello!');  
    console.log('Welcome to JavaScript');  
}
```

Hoće, za ovaj način definisanja funkcija !

Definicija funkcije je *hoisted* (može se pozivati pre definisanja)

Console  
"Hello!"

"Welcome to JavaScript"

"Hello!"

"Welcome to JavaScript"

# JavaScript funkcije

- ◆ Imaju promenjiv broj argumenata:
- ◆ Varijabilan niz argumenata (`arguments[0]` je prvi argument)
- ◆ Nenavedeni argumeti imaju vrednost `undefined`
- ◆ Sve funkcije imaju povratnu vrednost (podrazumevana je `undefined`)
- ◆ Funkcije su objekti prvog reda (*first-class objects*):
- ◆ Mogu biti vrednost varijable ili elementa niza, prosleđeni kao argument funkcije, povratna vrednost funkcije

# Funkcija kao objekat prvog reda

```
var aFuncVar = function (x) {  
    console.log('Called with', x);  
    return x+1;  
};  
function myFunc(routine) {  
    console.log('Called with', routine.toString());  
    var retVal = routine(10);  
    console.log('retVal', retVal);  
    return retVal;  
}  
myFunc(aFuncVar);
```

Rezultat ?

```
Called with function (x) {  
    console.log('Called with', x);  
    return x + 1;  
}  
Called with 10  
retVal 11
```

# Varijable var, let, const

Varijable se mogu definisati na tri načina:

```
// Opseg vidljivosti je globalan
```

```
var x = 15;
```

```
// Opseg vidljivosti je blok
```

```
let fruit = 'banana';
```

```
// Konstanta
```

```
const isHungry = true;
```

Tip varijable se ne navodi pre definisanja

Dinamičko tipiziranje – tip se definiše dinamički, prilikom dodelje vrednosti

Varijable dobijaju tip poslednje dodeljene vrednosti

# Parametri funkcije

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
}
```

Parametri funkcije se ne deklarišu sa var, let, ili const

# var – opseg vidljivosti

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
    console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Šta se dešava ako želimo da ispišemo vrednost i van petlje ?

# var – opseg vidljivosti

```
function printMessage(message, times) {  
    for (var i = 0; i < times; i++) {  
        console.log(message);  
    }  
    console.log('Value of i is ' + i);  
}  
printMessage('hello', 3);
```

Console

"hello"

"hello"

"hello"

"Value of i is 3"

Vrednost *i* je vidljiva van petlje zbog toga što varijable deklarisane sa var imaju opseg vidljivosti na nivou funkcije u kojoj su deklarisani

# var – opseg vidljivosti

- ◆ Varijable deklarisane sa **var** imaju opseg vidljivosti na nivou **funkcije** u kojoj se definišu, ne bloka naredbi
- ◆ Možete koristiti varijablu i nakon završetka bloka koda (tj. posle petlje ili if naredbe u kojoj su deklarisane)

```
var x = 10;  
if (x > 0) {  
    var y = 10;  
}  
console.log('Value of y is ' + y);
```

"Value of y is 10"

# let – opseg vidljivosti

```
function printMessage(message, times)
{
    for (let i = 0; i < times; i++) {
        console.log(message);
    }
    console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

Šta se dešava ako želimo da ispišemo vrednost i van petlje ?

# let – opseg vidljivosti

```
function printMessage(message, times)
{
    for (let i = 0; i < times; i++) {
        console.log(message);
    }
    console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

3 hello

✖ ► Uncaught ReferenceError: i is not defined  
at printMessage (pen.js:29)  
at VM479 pen.js:23

let ima opseg  
vidljivosti  
**bloka** !

# const – opseg vidljivosti

```
let x = 10;  
if (x > 0) {  
    const y = 10;  
}  
console.log(y);
```

Value of i is 3

Value of y is 10

✖ Uncaught ReferenceError: y is not defined

[console\\_runner-1df7d...81e5fba982f6af.js:1](#)

[console\\_runner-1df7d...81e5fba982f6af.js:1](#)

[pen.js:25](#)

const ima opseg vidljivosti **bloka** !

# **var, let, const**

- ◆ Deklracija varijable u JS:

- ◆ Opseg vidljivosti funkcije:

```
var x = 15;
```

- ◆ Opseg vidljivosti bloka:

```
let fruit = 'banana';
```

- ◆ Opseg vidljivost bloka (konstanta):

```
const isHungry = true;
```

# Šta je blok ?

- ◆ Skup naredbi ograničen vitičastim zagradama ({} )

Java, C#, C++:

```
if (...) {  
    int x = 5;  
    ...  
}
```

// ovde se ne može pristupiti x

# Blok i opseg vidljivosti

- Isto važi i u JavaScript-u, pod uslovom da su varijable deklarisane sa **const** i **let**

```
if (...) {  
    let x = 5;  
    ...  
}  
// ovde se x ne vidi
```

- Ali ako koristimo **var**, varijabla se vidi u čitavoj funkciji nezavisno od blokova

```
if (...) {  
    var x = 5;  
    ...  
}  
// ovde se x vidi
```

# JavaScript tipovi podataka

- ◆ JavaScript **varijable nemaju** tipove
- ◆ JavaScript **vrednosti imaju** tipove
- ◆ Osnovni tipovi podataka (primitive):
  - ◆ Boolean – true ili false
  - ◆ Number – double (ne postoji integer)
  - ◆ String – 'primer stringa 1' "primer stringa 2"
  - ◆ Symbol – jedinstven simbol kao objekat
  - ◆ Null – nulta vrednost
  - ◆ Undefined – varijabli nije dodeljena vrednost
  - ◆ Postoje i Object (tj. objektni) tipovi, kao što su Array, Date, String (omotač-wrapper primitivnog tipa)

# JavaScript brojevi

- ◆ Svi brojevi su tipa realnog broja
- ◆ Operatori su identični kao u Java jeziku
- ◆ Tri simboličke konstante kao vrednosti:
  - ◆ -Infinity (Number.MIN\_VALUE)
  - ◆ +Infinity (Number.MAX\_VALUE)
  - ◆ NaN – *not-a-number*
- ◆ Postoji Math klasa: Math.floor, Math.ceil

# Stringovi

```
let mojString = 'primer';
mojString += 'stringa';
mojString = mojString.toUpperCase();
console.log("Sadržaj stringa: " + mojString);
```

- ◆ Mogu se koristiti jednostruki ili dvostruki navodnici
- ◆ Nepromenjivi (immutable)
  - Jednom kreiran, ne može se menjati kao takav
  - Novi string se kreira kao rezultat operacije nad originalom
- ◆ Ne postoji znakovni tip (char)
- ◆ Konkatenacija (+)
- ◆ Veličina stringa – length atribut
- ◆ Korisne metode: indexOf(), charAt(), match(), search(), replace(), toUpperCase(), toLowerCase(), slice(), substr()

# Templejt stringovi (*template strings*)

- ◆ Stringovi koji omogućavaju dinamičko ugrađivanje i izračunavanje izraza pomoću *placeholder-a*
- ◆ String nije ograničen znakovima navoda već back-tick simbolom (`) !

```
string text ${expression} string text  
tag `string text ${expression} string text`
```

- ◆ U prvom slučaju, računa se vrednost izraza, konvertuje u string, i spaja sa ostatkom stringa (pre i posle izraza)
- ◆ U drugom slučaju, izračunavanje, konverzija i spajanje se vrše unutar funkcije (tzv. tag funkcije)

# Templejt stringovi - primer

```
var a = 5;  
var b = 10;  
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
```

"Fifteen is 15 and not 20."

```
var name = 'Student';  
var age = 22;  
function greet(arr, nameArg, ageArg) {  
    console.log(arr[0] + nameArg + arr[1] + ageArg + arr[2]);  
}  
greet`${name} ima ${age} godine?`;
```

"Student ima 22 godine?"

# Logički tip (**boolean**)

- ◆ Dve vrednosti: true i false
- ◆ Operatori: && (logičko i); || (logičko ili); !(negacija)
- ◆ Ne-logičke vrednosti mogu biti korišćene u uslovnim naredbama
- ◆ Konvertuju se u odgovarajuću vrednost:
  - ◆ null, undefined, 0, NaN, "", "" se pretvaraju u false
  - ◆ Sve ostalo se pretvara u true (sadržaji svih tipova koji nisu prazni)

```
if (username) { // username is defined }
```

# object tip

- ◆ JavaScript objekat je kolekcija parova **atribut-vrednost** (*property-value*)

```
var obj1 = {};  
var obj2 = {name: "Alice", age: 23, state:"California"};
```

- ◆ Naziv atributa može biti proizvoljan string
- ◆ Atributi se referenciraju kao:
- ◆ struktura - **obj2.name**
- ◆ hash tabela sa kjučevima - **obj2["name"]**
- ◆ Fleksibilno dodavanje i brisanje atributa:

```
var obj1 = {};  
obj1.name = "Alice"; // ili obj1[name] = "Alice"
```

```
var obj1 = {name: "Alice"};  
delete obj1.name;
```

# Jednakost – starija pravila (*loose equality*)

JavaScript `==` i `!=` su zapravo dve operacije: implicitna konverzija tipa pre poređenja (tabela: poređenje jednakosti)

		Operand B					
		Undefined	Null	Number	String	Boolean	Object
Operand A	Undefined	true	true	false	false	false	false
	Null	true	true	false	false	false	false
	Number	false	false	A === B ToNumber(B)	A === ToNumber(B)	A === ToNumber(B)	A == ToPrimitive(B)
	String	false	false	ToNumber(A) == B	A === B	ToNumber(A) == ToNumber(B)	A == ToPrimitive(B)
	Boolean	false	false	ToNumber(A) == B	ToNumber(A) == ToNumber(B)	A === B	ToNumber(A) == ToPrimitive(B)
	Object	false	false	ToPrimitive(A) == B	ToPrimitive(A) == B	ToPrimitive(A) == ToNumber(B)	A === B

# Jednakost – starija pravila (*loose equality*)

JavaScript `==` i `!=` su zapravo dve operacije: implicitna konverzija tipa pre poređenja)

```
' ' == '0' // false
' ' == 0 // true
0 == '0' // true
NaN == NaN // false
[' '] == '' // true
false == undefined // false
false == null // false
null == undefined // true
```

# Jednakost – novija pravila, ECMA Script (*strict equality*)

Nema implicitne konverzije

Pravila:

Različiti tipovi vrednosti ->  $\neq$

Isti tipovi vrednosti -> poređenje vrednosti

```
'' === '0' // false
'' === 0 // false
0 === '0' // false
[''] === '' // false
false === undefined // false
false === null // false
null === undefined // false
```

**Koristiti  $==$  i  $!=$  umesto  $==$  i  $!=$**

# *loose (==) i strict (===) equality*

x	y	==	===
undefined	undefined	true	true
null	null	true	true
true	true	true	true
false	false	true	true
'foo'	'foo'	true	true
0	0	true	true
+0	-0	true	true
+0	0	true	true
-0	0	true	true
0	false	true	false
""	false	true	false
""	0	true	false
'0'	0	true	false

# Null i undefined

- ◆ Razlika ?
- ◆ Null predstavlja nultu vrednost kao takvu (Java null)
- ◆ Undefined znači da varijabli nije dodeljena vrednost (varijabla je samo deklarisana)

```
x = null;  
let y;  
console.log(x);  
console.log(y);
```

```
null  
undefined
```

# Nizovi (array)

- ◆ **Objektni** tipovi koji se kreiraju i koriste kao **liste**
- ◆ Objektni: memorijska struktura podrazumeva referencu na element niza i sam element niza, tj, njegovi podaci
- ◆ **Lista:** struktura tipa liste

[element0, element1, ..., elementN]

new Array(element0, element1[, ...[, elementN]])

new Array(arrayLength)

Parametri:

**elementN:** element niza kada se niz kreira ručno kao lista

**arrayLength:** ceo broj koji se prosleđuje kada se niz kreira kao objekat tipa Array. Kreira se niz sa praznim elementi (ne undefined)

Dužina niza: atribut length (ne metoda)

Indeks počinje od vrednosti 0

Dužina i tipovi elemenata niza su promenjivi

# Operacije sa nizovima

- ◆ Kreiranje niza

```
var fruits = ['Apple', 'Banana'];
```

```
console.log(fruits.length);
```

- ◆ Pristup elementima niza

```
var first = fruits[0];
```

```
var last = fruits[fruits.length - 1];
```

# Operacije sa nizovima

## ◆ Iteracija niza (for petlja)

```
for(let fruit of fruits) {  
    console.log(fruit);  
}
```

"Apple"

"Banana"

## ◆ Iteracija niza (funkcija)

```
fruits.forEach(function(item, index, array) {  
    console.log(item, index);  
});
```

"Apple" 0

"Banana" 1

# Operacije sa nizovima

- ◆ Dodavanje/uklanjanje elementa sa početka niza

```
var first = fruits.shift();
var newLength = fruits.unshift('Strawberry');
```

- ◆ Dodavanje/uklanjanje elementa sa kraja niza

```
var newLength = fruits.push('Orange');
var last = fruits.pop();
```

- ◆ Indeksiranje elementa

```
var pos = fruits.indexOf('Banana');
// 1
```

# Operacije sa nizovima

- ◆ Uklanjanje podniza od zadate pozicije (indeksa)

```
var vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot'];
console.log(vegetables);

var pos = 1, n = 2;

var removedItems = vegetables.splice(pos, n);

console.log(vegetables);
console.log(removedItems);
```

["Cabbage", "Carrot"]

["Turnip", "Radish"]

# Kopiranje nizova – shallow copy

```
var array1 = [1, 2, 3, 4, 5, 6, 7, 8];  
var array2 = array1;  
var array3 = array1.slice();
```

const array1 =



const array2 =

**Kopiraju se reference na elemente niza, ne i sami element !**

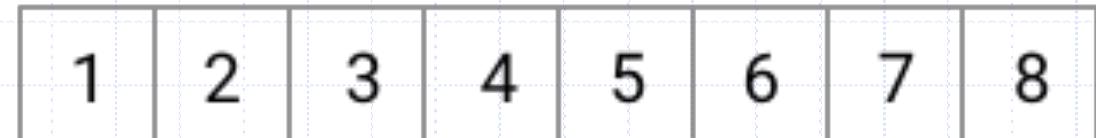
**Napomena:** načini opisani ispod važe samo za jednodimenzione nizove !

# Kopiranje nizova – deep copy

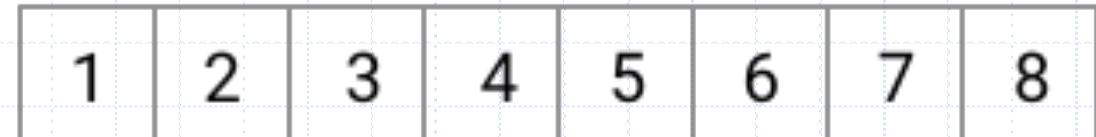
```
var array4 = array1.map(elem => elem);
```

```
var array2 = [];
array1.forEach(elem => {
    array2.push(elem)
});
```

const array1 =



const array2 =



**Kopiraju se reference i elementi niza !**

# Datumi

```
new Date(); //tekuci datum  
new Date(value);  
new Date(dateString);  
new Date(year, monthIndex [, day [, hours [, minutes [,  
seconds [, milliseconds]]]]]);
```

Poseban tip objekta: `typeof date == 'object'`

Broj milisekundi od 1. Januara 1970 (UTC)

Korisne metode (getter/setter):

`Date.now()`; `Date.parse()`  
`getDate`/`Time`/`Hours`  
`setDate`/`Time`/`Hours`

# Datumi - primer

```
var start = Date.now();
doSomethingForALongerTime();
var end = Date.now();
var elapsed = end - start;
```

```
var start = new Date();
doSomethingForALongerTime();
var end = new Date();
var elapsed = end.getTime() - start.getTime();
```

# Obrada izuzetaka – try/catch

- ◆ Obrada, tj. prijavljivanje izuzetaka (grešaka) koje nastanu u toku izvršavanja programa

```
try {  
    nonExistentFunction();  
}  
catch(error) {  
    console.error(error);  
}
```

► ReferenceError: nonExistentFunction is not defined

Sadržaj error poruke može varirati u zavisnosti od pretraživača

# try/catch – bolji primer

```
try {  
    nonExistentFunction();  
}  
catch (err) {  
    console.log("Error call func", err.name, err.message);  
}
```

```
Error call func ReferenceError  
nonExistentFunction is not defined
```

Varijante:

try...catch

try...finally

try...catch...finally

# try/catch – obrada specifičnih grešaka

- ◆ Može se korisititi veći broj clause izjava za specifičan tip grešaka koji može nastati

```
try {  
    myroutine(); // may throw three types of exceptions  
} catch (e if e instanceof TypeError) {  
    // statements to handle TypeError exceptions  
} catch (e if e instanceof RangeError) {  
    // statements to handle RangeError exceptions  
} catch (e if e instanceof EvalError) {  
    // statements to handle EvalError exceptions  
} catch (e) {  
    // statements to handle any unspecified exceptions  
    logMyErrors(e); // pass exception object to error handler  
}
```

# try/catch – česta upotreba

- ◆ Pristup skladištu podataka (server-side JS)

```
openMyFile();
try {
    writeMyFile(theData);
}
finally {
    closeMyFile();
}
```



Naredbe koje se izvršavaju posle izvršenja try naredbi. Izvršavaju se uvek, nezavisno od toga da li nastane izuzetak.

# JavaScript programski jezik

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

# OO JS: metode

- ◆ Atribut (property) objekta može biti i funkcija

```
var o = {count: 0};  
o.increment = function (amount) {  
    if (amount == undefined) {  
        amount = 1;  
    }  
    this.count += amount;  
    return this.count;  
}  
  
console.log(o.increment());  
console.log(o.increment(3));
```

1

4

# this

- ◆ Globalni kontekst:
- ◆ Van funkcije, ukazuje na globalni kontekst izvršavanje, na primer window objekat
- ◆ Kontekst funkcije:
- ◆ Funkcija u okviru koje se poziva

```
var o = {oldProp: 'this is an old property'};  
o.aMethod = function() {  
    this.newProp = "this is a new property";  
    return Object.keys(this);  
}  
console.log(o.aMethod());
```

```
["oldProp", "aMethod", "newProp"]
```

# Funkcije mogu imati attribute

```
function plus1(value) {  
  if (plus1.invocations == undefined) {  
    plus1.invocations = 0;  
  }  
  plus1.invocations++;  
  return value + 1;  
}  
console.log(plus1());  
console.log(plus1(3));
```

NaN

4

# Funkcije se mogu posmatrati kao klase

```
function Rectangle(width, height) {  
    this.width = width;  
    this.height = height;  
    this.area = function() { return this.width*this.height; }  
}  
var r = new Rectangle(26, 14);  
  
console.log(r.constructor.name);  
console.log(r);
```

"Rectangle"

```
Object {  
  area: function () {return this.width * this.height;},  
  height: 14,  
  width: 26  
}
```

# Prototipi (prototypes)

- ◆ Object.prototype: svaki JS objekat je instanca Object
- ◆ Svaki JS objekat ima privatni atribut (*property*) koji se zove **prototype** i sadrži link ka drugom objektu (koji naziva svojim prototipom)
  - Svaka kreirana instanca ima svoj prototip objekat
- ◆ Referencirani objekat ima svoj prototype atribut i tako dalje, formirajući lanac prototipa (*prototype chain*)
- ◆ JS objekti predstavljaju dinamički skup atributa
  - Novi atributi se mogu dodeljivati objektima, tj.instancama u bilo kom trenutku
- ◆ Kada pokušamo da pristupimo atributu objekta, atribut neće biti tražen samo u konkretnom objektu, već i u njegovom prototipu, prototipu prototipa, itd. sve dok se atribut ne pronađe, ili ne dođe do kraja lanca prototipa
- ◆ Uopšteno gledano, atributi objekta se mogu posmatrati kao oni koje on sadrži + atributi lanca prototipa
  - Nasleđivanje zasnovano na prototipima

```
function Rectangle(width, height) {  
    this.width = width;  
    this.height = height;  
}  
Rectangle.prototype.area = function()  
{  
    return this.width*this.height;  
}  
var r = new Rectangle(26, 14);  
var v = r.area();  
console.log(r);  
console.log(v);
```

# Prototipi primer

```
Object {  
    area: function ()  
    {  
        return this.width * this.height;  
    },  
    height: 14,  
    width: 26  
}
```

Funkcije takođe imaju atribut prototype (zbog toga što su i one instance tipa Object, tj. objekti prvog reda)

# Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
console.log(car1.stereo);  
console.log(car2.gps);
```

undefined

undefined

# Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
console.log(car1.wheels);  
console.log(car2.wheels);  
console.log(car1.steeringWheel);  
console.log(car2.steeringWheel);
```

```
4
```

```
4
```

```
1
```

```
1
```

# Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
car1.wheels=3;  
console.log(car1.wheels);  
console.log(car2.wheels);  
console.log(Car.prototype.wheels);
```

Objekti deca ne mogu menjati atribute roditelja u lancu

Kreira se novi atribut deteta sa istim nazivom

3

4

4

# Kako izgleda car3 instanca ?

```
console.log(car1);
```

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  wheels: 3  
}
```

# Prototipi primer

```
function Car(){}
```

```
Car.prototype.wheels = 4;  
Car.prototype.steeringWheel = 1;  
Car.prototype.color = 'metallic';
```

```
var car1 = new Car();  
var car2 = new Car();
```

```
car1.wheels=3;
```

```
Car.prototype.wheels = 5;  
console.log(Car.prototype.wheels);  
console.log(car2.wheels);  
console.log(car1.wheels);
```

Promene u roditelju se automatski reflektuju na decu

Osim za decu u kojoj je promena izvršena eksplicitno

5

5

3

# Prototipi primer

```
car1.stereo = 1;  
car2.gps = 1;  
console.log(Car.prototype);  
console.log(car2);  
console.log(car1);
```

Promene u deci se  
ne vide u roditelju !

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  wheels: 5  
}
```

```
Object {  
  color: "metallic",  
  gps: 1,  
  steeringWheel: 1,  
  wheels: 5  
}
```

```
Object {  
  color: "metallic",  
  steeringWheel: 1,  
  stereo: 1,  
  wheels: 3  
}
```

# Šema

Car.prototype

wheels = 4 (5)  
steering wheel = 1  
color = 'metallic'

car1.\_proto\_

wheels = 3  
steering wheel = 1  
color = 'metallic'  
stereo = 1

car1

car2.\_proto\_

wheels = 5  
steering wheel = 1  
color = 'metallic'  
gps = 1

car2

# Prototipi kao JS mehanizam nasleđivanja

- ◆ Svaka funkcija se može posmatrati kao klasa koja ima jednu unikatnu roditeljsku instancu (prototype) i decu primerke instanci (objekti)
- ◆ Instance su međusobno povezane i čine lanac (prototype chain)
  - Sekvencijalan lanac od unikatne instance ka deci po redosledu njihovog kreiranja
- ◆ Promene prototip instance (`Car.prototype`) će se automatski reflektovati na decu
  - Dodavanje/uklanjanje atributa i funkcija
  - Promene vrednosti atributa
- ◆ Obrnuto ne važi !
  - Novi atributi/funkcije ostaju u instanci u kojoj su kreirani

# Nasleđivanje (ECMAScript v6 2015)

```
class Rectangle extends Shape {  
    constructor(height, width) {  
        super(height, width);  
        this.height = height;  
        this.width = width;  
    }  
    area() {  
        return this.width * this.height;  
    }  
    static countInstances() {  
        ...  
    }  
}  
var r = new Rectangle(10,20);
```

Definicija i  
nasleđivanje

Definisanje  
metode

Statička  
metoda

Nasleđivanje klase se u  
pozadini konvertuje u model  
nasleđivanja prototipa !

# **JavaScript programski jezik**

- ◆ Proceduralni

- ◆ Objektno-orientisani

- ◆ Funkcionalni

# Funkcionalno programiranje

- ◆ Zasnovano na ideji da se program tretira kao matematička funkcija
- ◆ Sve u kodu je ili funkcija ili izraz
- ◆ Ne postoji stanje (variable, polja, objekti, ...)
- ◆ Većina modernih programskih jezika nije isključivo zasnovano na funkcionalnoj paradigmi
- ◆ Međutim, neke ideje su korisne i već nalaze primenu u JS jeziku:
  - ◆ Funkcije kao objekti prvog reda (*first-class functions*)
  - ◆ Anonimne funkcije (*Anonymous functions*)
  - ◆ Arrow funkcije (Angular)

# Funkcije kao objekti prvog reda

- ◆ Funkcije su JS objekti:
- ◆ Mogu se čuvati u, tj. dodeljivati varijablama
- ◆ Mogu se prosleđivati kao parametri
- ◆ Mogu imati atributе
- ◆ Mogu se definisati bez identifikatora ->

# Anonimna funkcija

- ◆ Funkcija koja nema ime i koja se kreira u vreme izvršavanja programa (*runtime*)
- ◆ Mora se definisati pre nego se pozove !
  - Nije hoisted kao standardne JS funkcije

```
//printName("Mlađan");
//ReferenceError: printName is not defined

const printName = function (name){
    console.log(name);
}
printName("Mlađan");
```

"Mlađan"

# Arrow funkcija

- ◆ Klasičan (imperativni) stil

```
for (var i=0; i < anArr.length; i++) {  
    newArr[i] = anArr[i]*i;  
}
```

- ◆ Funkcionalni stil:

```
newArr = anArr.map((val, ind) => val*ind);
```

- ◆ Čitav program se može napisati kao lanac funkcija:  
`anArr.filter(filterFunc).map(mapFunc).reduce(reduceFunc)`

# Linkovi ka primerima i dokumentacijom

JavaScript osnove primer: <https://codepen.io/mljovanovic/pen/BbpjNV>

JavaScript nizovi primer: <https://codepen.io/mljovanovic/pen/XGRroa>

JavaScript objekti i prototipi primer:

<https://codepen.io/mljovanovic/pen/VRWPXR>

JavaScript funkcionalno programiranje primer:

<https://codepen.io/mljovanovic/pen/aMwVjk>

MDN jednakost:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/  
Equality comparisons and sameness](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness)

MDN template stringovi:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/  
Template literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/<br/>Template_literals)

MDN arrow funkcije:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/  
Functions/Arrow functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/<br/>Functions/Arrow_functions)

MDN nasleđivanje:

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/  
Inheritance](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/<br/>Inheritance)

# Sažetak

- ◆ JavaScript:
- ◆ Princip rada
- ◆ Proceduralni aspekti
  - Operatori
  - Funkcije
  - Varijable
  - Tipovi podataka (templejt stringovi)
  - Nizovi
- ◆ Objektno-orientisani aspekti
  - Metode
  - Funkcije kao objekti
  - Prototipi
  - Klase i nasleđivanje
- ◆ Aspekti funkcionalnog programiranja
  - Anonimne funkcije
  - Arrow funkcije
- ◆ Primeri: [codepen.io](http://codepen.io)

# JavaScript osnove

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

U pripremi prezentacije korišćene su ilustracije i primeri sa  
<https://developer.mozilla.org>

# **JavaScript**

# **Nasleđivanje klasa**

# **Arrow funkcije**

# **DOM**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

- ◆ **JS Klase:**
- ◆ Definicija
- ◆ Instanciranje
- ◆ Nasleđivanje
- ◆ **Arrow funkcije:**
- ◆ Princip
- ◆ Sintakse
- ◆ Korišćenje
- ◆ Tajmer
- ◆ **DOM:**
- ◆ Struktura
- ◆ Metode
- ◆ Događaji
- ◆ Korišćenje
- ◆ Primeri: [codepen.io](http://codepen.io)

# ECMA Script

- ◆ JS jezik se kontinuirano usavršava
- ◆ Transponovanje: novi elementi jezika se u pozadini transformišu u postojeće, starije mehanizme
  - Nasleđivanje klasa -> nasleđivanje prototipa
- ◆ Razlika je sintaksna
- ◆ Razlog: kompatibilnost programa sa različitim tipovima i verzijama izvršnih okruženja, tj. pretraživača
- ◆ Moderni okviri za razvoj Web klijentskih aplikacija forsiraju upotrebu novijih elemenata jezika
  - AngularJS nameće upotrebu TypeScript jezika kao proširenja JS-a

# JS definicija klase

```
class Car {  
    constructor(wheel, steeringWheel, color) {  
        this.wheel = wheel;  
        this.steeringWheel = steeringWheel;  
        this.color = color;  
    }  
  
    get wheel() { return this._wheel; }  
    set wheel(newWheel) { this._wheel = newWheel; }  
  
    get steeringWheel() { return this._steeringWheel; }  
    set steeringWheel(newSWheel) { this._steeringWheel = newSWheel; }  
  
    get color() { return this._color; }  
    set color(newColor) { this._color = newColor; }  
}
```

# JS definicija klase - pravila

- ◆ Ključne reči **class** i **constructor**
- ◆ U konstruktoru se navode pararemetri za inicijalizaciju objekata klase
- ◆ **this** se odnosi na **instancu** klase
- ◆ Get/set metode se definišu pomoću ključnih reči **get** i **set**
- ◆ U okviru get/set metoda, atributima objekta se pristupa na specifičan način, *underscore simbol (\_)* kao prefiks:

*\_imeAtributa*

# JS instanciranje klase

```
var carInstance = new Car(4,1,'metallic');  
console.log(carInstance);
```

```
Object {  
  _color: "metallic",  
  _steeringWheel: 1,  
  _wheel: 4  
}
```

# JS klase – korišćenje get/set metoda instanci klase

```
carInstance.color = 'red';
console.log(carInstance.color);
```

"red"

Kod poziva metoda se navodi samo ime bez zagrada () !

# JS nasleđivanje klasa

```
class Car1 extends Car {  
    constructor(wheel, steeringWheel, color, stereo) {  
        super(wheel, steeringWheel, color);  
        this.stereo = stereo;  
    }  
  
    get stereo() { return this._stereo; }  
    set stereo(newStereo) { this._stereo = newStereo; }  
  
    carInfo() {  
        console.log("Wheels:" + this._wheel +  
                    " wheels:" + this._steeringWheel +  
                    " color:" + this._color +  
                    " stereo:" + this._stereo)  
    }  
}
```

# JS nasleđivanje klasa - pravila

- ◆ U konstruktoru izvedene klase se mora pozvati konstruktor roditeljske klase – **super**
  - Sa parametrima roditelja
- ◆ Rad sa novim atrubutima je identičan kao kod roditeljske klase (get/set)
- ◆ Pravila za definisanje metoda su identična kao za proceduralne JS funkcije

# JS instanciranje i poziv metode izvedene klase

```
var car1Instance = new Car1(4,1,'red',1);  
car1Instance.carInfo();
```

"Wheels:4 wheels:1 color:red stereo:1"

# Funkcionalno programiranje

## Arrow funkcije

# Arrow funkcije

- ◆ Koncizniji (sažetiji) način za pisanje JS funkcije  
=> (*flat arrow*)
- ◆ Bez upotrebe nekih ključnih reči (*function*) i simbola
- ◆ Uvedene od ES V6 (2015)
- ◆ Sa obzirom da je većina navikla na pisanje funkcije na standardan način u većini jezika, varijante funkcija ćemo obraditi kroz poređenja sa klasičnim načinom pisanja

# Osnovna sintaksa

(param1, param2, ..., paramN) => { **statements** }

(param1, param2, ..., paramN) => **expression**

// ekvivalentno sa: => { return expression; }

// Spisak parametara može biti prazan

() => { **statements** }

# Osnovna sintaksa se višestrukim parametrima

```
// Osnovna sintaksa sa višestrukim parametrima  
// Klasičan način  
var multiplyOld = function(x, y) {  
    return x * y;  
};  
  
// Arrow funkcija  
let multiplyNew = (x, y) => { return x * y };
```

Isti rezultat se može postići sa manje kodiranja !

# Osnovna sintaksa se jednim parametrom

```
// Osnovna sintaksa sa jednim parametrom
// Klasičan način
var phraseSplitterOld = function phraseSplitter(phrase) {
    return phrase.split(' ');
};

// Arrow funkcija
var phraseSplitterNew = phrase => phrase.split(" ");

console.log(phraseSplitterNew("Osnovna sintaksa sa jednim parametrom"));
```

Parametar se ne mora navoditi u zagradama !

```
["Osnovna", "sintaksa", "sa", "jednim", "parametrom"]
```

# Osnovna sintaksa bez parametara

```
//Osnovna sintaksa bez parametara
//Klasičan način
var docLogOld = function docLog() {
    console.log(document);
};

//Arrow funkcija
var docLogNew = () => { console.log(document); };
docLogNew(); // #document... <html> ...
```

Moraju se navesti zagrade () !

# Ugrađene JS funkcije za rad sa nizovima

- ◆ **Map**
- ◆ **Filter**
- ◆ **Reduce**

- ◆ Podrazumevano rade po principu *for* petlje, tj. iteratora
  - Mehanizam iteratora je ugrađen u definicije funkcije
- ◆ Prednost je što se petlja ne mora programirati, tj. navoditi u kodu, već samo željene operacije nad elementima kolekcije

# Arrow funkcija za manipulaciju nizovima

- ◆ Mapiranje nizova
- ◆ Izdvajanje specifičnih vrednosti iz niza elemenata tipa *object*

```
reduced_array = original_array.map(object => object.property)
```

```
const smartPhones = [  
  { name:'iphone', price:649 },  
  { name:'Galaxy S6', price:576 },  
  { name:'Galaxy Note 5', price:489 }  
];
```

# Mapiranje nizova tipa elementa object

```
//Klasičan način  
var pricesOld = smartPhones.map(function(smartPhone) {  
    return smartPhone.price;  
});  
  
console.log(pricesOld);
```

```
//Arrow funkcija  
var pricesNew = smartPhones.map(smartPhone => smartPhone.price);  
console.log(pricesNew);
```

[649, 576, 489]

[649, 576, 489]

# Arrow funkcija za manipulaciju nizovima

- ◆ **Filtriranje nizova**
- ◆ Izdvajanje specifičnih vrednosti iz niza elemenata koji zadovoljavaju specifičan kriterijum (tj. filter)

```
filtered_array = original_array.filter(element => condition)
```

```
const array = [1, 2, 3, 4, 5, 6,  
              7, 8, 9, 10, 11,  
              12, 13, 14, 15];
```

# Filtriranje nizova

```
//Klasičan način  
var filterOld = array.filter(function (v){  
    return v % 3 === 0;  
});  
console.log(filterOld);
```

```
//Arrow funkcija  
var filterNew = array.filter(v => v % 3 === 0);  
console.log(filterNew);
```

[3, 6, 9, 12, 15]

[3, 6, 9, 12, 15]

# Arrow funkcija za manipulaciju nizovima

- ◆ Redukcija nizova
- ◆ Agregacija vrednosti niza (sume, prosečne vrednosti, itd.)

```
value = original_array.reduce((total, current) =>  
    expression with total and current)
```

```
var euros = [29.76, 41.85, 46.5];
```

# Redukcija nizova - suma

```
//Klasičan način
var sumOld = euros.reduce(function(total, amount){
    return total + amount
});
console.log(sumOld);

//Arrow funkcija
var sumNew = euros.reduce((total, amount) => total + amount);
console.log(sumNew);
```

U početnoj iteraciji, *total* ima vrednost prvog elementa niza, *amount* ima vrednost sledećeg elementa. U svakoj narednoj iteraciji drugi parametar (*amount*) se pomera udesno i dodaje kao vrednost datog elementa niza prvom parametru (*total*).

118.11  
118.11

# **setInterval kao klasična funkcija**

- ◆ Standardna JavaScript tajmer funkcija
- ◆ Tajmer: pozivanje funkcije (**function**) u redovnim vremenskim intervalima (**miliseconds**)

```
var timer = setInterval(function, miliseconds)  
clearInterval(timer)
```

- ◆ Pokretanje tajmara: poziv setInterval funkcije
- ◆ Zaustavljanje tajmara: poziv clearInterval funkcije

# setInterval kao arrow funkcija

Naredbe (statements) se izvršavaju u redovnim vremenskim intervalima (milliseconds)

```
var timer = setInterval(() => { statements },  
millseconds);
```

**clearInterval(timer)**

```
function Counter() {  
  var that = this;  
  this.timer = setInterval(() => {  
    console.log(this === that);  
  }, 1000);  
}  
  
var e = new Counter();  
clearInterval(e.timer);
```

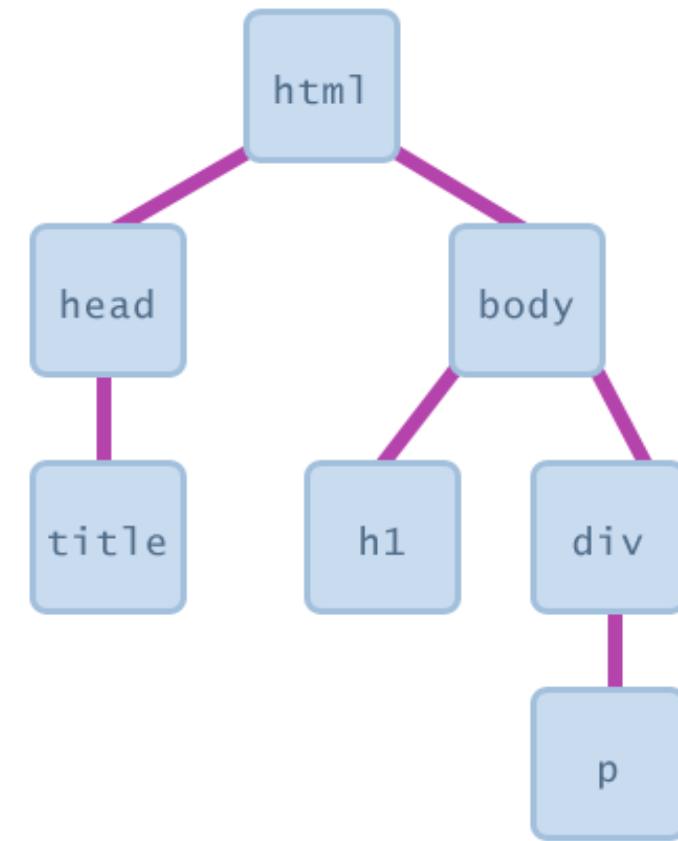


**DOM**

# DOM - Document Object Model

- ◆ Elementi Web stranice u vreme prikazivanja su organizovani kao hijerarhija objekata

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
      <p></p>
    </div>
  </body>
</html>
```



# DOM

- ◆ DOM je dinamičko stablo HTML objekata, tj. čvorova u vreme izvršavanja programa, tj. prikaza HTML dokument
- ◆ JS može **ispitati** čvorove kako bi ustanovio stanje elementa
  - Na primer, da li je korisnik uneo sadržaj u tekstualno polje
- ◆ JS može **ažurirati** atributе ovih objekata
  - Promena stila ili sadržaja elementa, na primer <p>
- ◆ JS kod može **dodavati** ili **uklanjati** čvorove i time menjati izgled stranice

# DOM struktura

- ◆ DOM stablo se sastoji od JS čvorova, instanci tipa **Node**
- ◆ Atributi tipa Node

Atribut	Značenje
textContent	Tekstualni sadržaj čvora i njegove dece
childNodes	Niz čvorova dece
parentNode	Referenca na roditeljski čvor

- ◆ Dva osnovna tipa čvora:
- ◆ **Element** koji predstavlja HTML element (tagovi)
- ◆ **Text** koji predstavlja tekstualni sadržaj tagova, uključujući prazan prostor !
  - Listovi stabla, nemaju elemente decu

```
<head>
</head>
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

**Element** čvor predstavlja HTML element (tag)

**Text** čvor predstavlja tekstualni sadržaj tagova, uključujući prazan prostor !

```
▶ <body>...</body>
▶ NodeList(5) [text, p, text, p, text]
5
```

```
section = document.querySelector('section');
console.log(section.parentNode);
console.log(section.childNodes);
console.log(section.childNodes.length);
```

# Atributi i metode DOM čvora

## ◆ Atributi:

nodeType – tip čvora (Element, Text, Comment, ...)

nodeName – ime čvora

nodeValue – vrednost čvora (null za Element, sadržaj za Text tip čvora); može se koristi i za čitanje i za dodelu vrednosti

## ◆ Kretanje po hijerarhijskoj strukturi:

parentNode() – roditeljski čvor

nextSibling() – naredni čvor u istom nivou hijerarhije

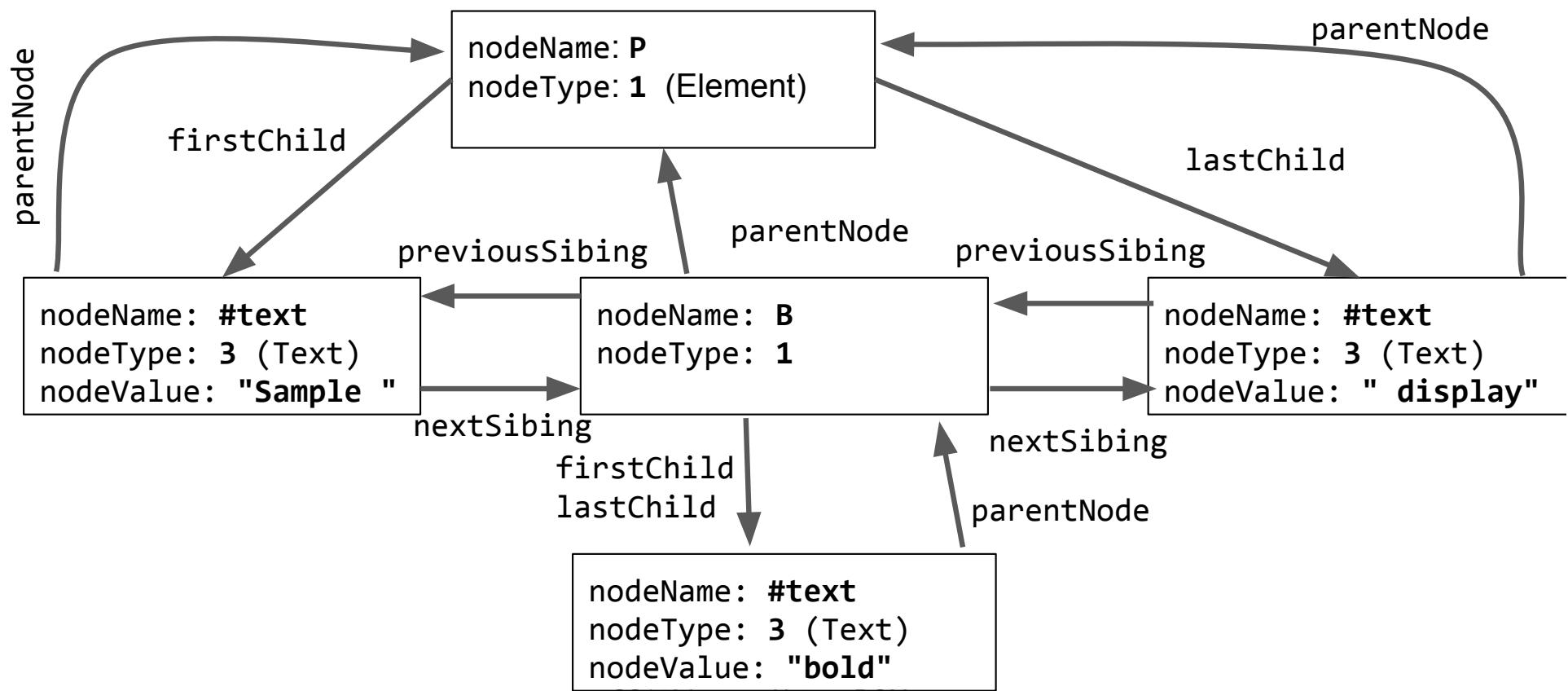
previousSibling() – prethodni čvor u istom nivou hijerarhije

firstChild() – prvi element dete

lastChild() – poslednji element dete

# Primer strukture DOM stabla

<p>Sample <b>bold</b> display</p>



# Kako pristupamo DOM objektu iz JS ?

- ◆ Prvi način:
- ◆ *querySelector* funkcija

**document.querySelector('css selector');**

vraća **prvi** element koji odgovara datom selektoru

**document.querySelectorAll('css selector');**

vraća **sve** elemente koji odgovaraju datom selektoru

# querySelector primer

```
// Vraća sve DOM čvorova koji predstavljaju HTML  
elemente sa id="button"  
let element = document.querySelector('#button');
```

```
// Vraća listu DOM čvorova koji predstavljaju HTML  
elemente sa class="quote"  
let elementList =  
document.querySelectorAll('.quote');
```

# Kako pristupamo DOM objektu iz JS ?

- ◆ Drugi način:
- ◆ *DOM lookup* metode

**document.getElementById('element id');**  
vraća element sa vrednošću id atributa kao parametra

**document.getElementsByClassName('class name');**  
vraća sve elemente sa vrednošću class atributa kao parametra

**document.getElementsByTagName('tag name');**  
vraća sve elemente sa vrednošću naziva taga kao parametra

# DOM lookup primer

HTML: <div id="div42">...</div>

```
let element = document.getElementById('div42');
```

HTML: <div class="div42"> ... </div>

```
let element =  
document.getElementsByClassName('div42');
```

HTML: <div class="div42"> ... </div>

```
let element =  
document.getElementsByTagName('div');
```

# DOM omogućava promenu HTML elemenata u vreme prikazivanja stranice

- ◆ Slučajevi iz prakse:

- ◆ Promena src atributa img taga:

Promena slike na događaj klika dugmeta

```
img.src = "newImage.jpg"
```

- ◆ Prikazivanje/sakrivanje elementa:

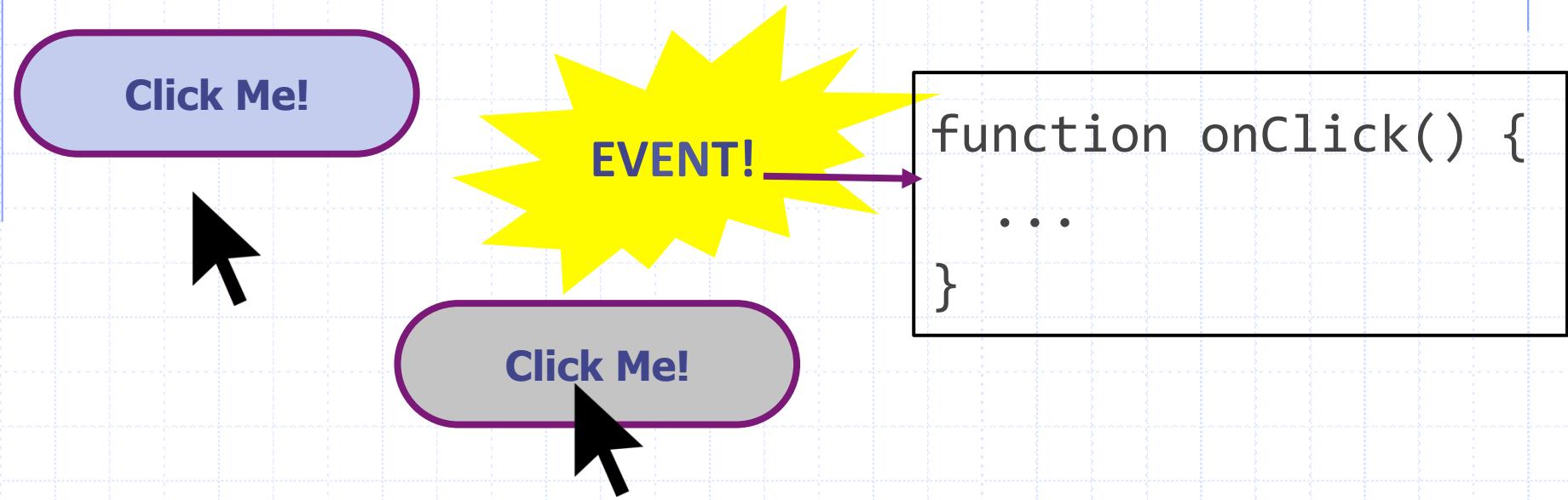
Prikazivanje/sakrivanje generičkih kontejnera (div)

Nevidljivo: element.style.display = "none";

Vidljivo: element.style.display = "";

# DOM događaji

- ◆ Veći deo JS koda se pokreće tako što nastane događaj



Kada korisnik klikne dugme generiše se programski objekat događaja i poziva funkcija koja osluškuje nastanak tog tipa događaja (*event handler*)

# Dinamičko dodavanje osluškivača događaja

- ◆ Svaki DOM objekat ima sledeću metodu:

**addEventListener(event name, function name);**

event name je naziv događaja čiji nastanak  
osluškujemo

uobičajen tipovi: click, focus, blur, ...

function name je ime funkcije koja će biti pozvana  
kada događaj nastane

# Dinamičko uklanjanje osluškivača događaja

- ◆ Da bi uklonili osluškivač događaja, koristimo sledeću metodu:

**removeEventListener**(event name, function name);

event name je naziv događaja čiji nastanak prestajemo da osluškujemo

function name je ime funkcije koja će biti pozvana kada dogđaj nastane

# HTML atributi i atributi DOM objekata

- ◆ Svaki atribut HTML elementa se preslikava u atribut DOM objekta u vreme prikazivanja stranice

## HTML

```

```

## **JavaScript**

```
const element = document.querySelector('img');
element.src = 'newImage.png';
```

**Click for a present:**



**DOM dinamička  
promena  
sadržaja čvora na  
primeru img taga –  
primer 1**

```
function openPresent() {  
    const image = document.querySelector('img');  
    image.src =  
    'https://media.giphy.com/media/27ppQU0xe7KLG/giphy.gif';  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

Funkcija u kojoj pronalazimo img tag i menjamo vrednost src atributa

```
function openPresent() {  
    const image = document.querySelector('img');  
    image.src =  
    'https://media.giphy.com/media/27ppQU0xe7KLG/giphy.gif';  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

Dodela osluškivača događaja img tagu koji na događaj klika dugmeta poziva funkciju iznad

```
<html>
  <head>
    <meta charset="utf-8">
    <title>DOM primer 1</title>
  </head>
  <body>
    <h1>Click for a present:</h1>
    
  </body>
</html>
```

```
body {
  font-family: monospace;
  text-align: center;
}

img {
  height: 200px;
}
```



# Atributi Element tipa čvora

Atribut	Značenje
<b>id</b>	Vrednost id atributa HTML elementa kao string
<b>textContent</b>	Tekstualni sadržaj čvora i njegove dece
<b>innerHTML</b>	Originalan HTML sadržaj između početnog i krajnjeg taga
<b>classList</b>	CSS klase primenjene na dati element

Click for a present:



DOM dinamička  
promena  
sadržaja čvora na  
primeru img taga i h1  
teksta – primer 2

```
function openPresent(event) {  
    const image = document.querySelector('img');  
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  
    const title = document.querySelector('h1');  
    title.textContent = 'Hooray!';  
  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

```
<html>
  <head>
    <meta charset="utf-8">
    <title>DOM primer 1</title>
  </head>
  <body>
    <h1>Click for a present:</h1>
    
  </body>
</html>
```

```
body {
  font-family: monospace;
  text-align: center;
}

img {
  height: 200px;
}
```

**Click for a present:**



**Hooray!**



# Element – izvor događaja

- ◆ Idenične naredbe na dva mesta u primeru

```
function openPresent(event) {  
    const image = document.querySelector('img');  
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  
    const title = document.querySelector('h1');  
    title.textContent = 'Hooray!';  
  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

# JS Event objekat

- ◆ **event.target** – element **izvor** događaja (na primer, slika na koju se klikne). U slučaju da elementu nije pridružen osluškivač, događaj se prenosi kroz DOM čvorove ka roditelju dok se ne nađe na osluškivač registrovan za taj tip događaja
  - ◆ Konkretan element ne mora imati pridružen osluškivač
  - ◆ Mehanizam se naziva delegiranje događaja (*event delegation*)
- ◆ **event.currentTarget** – element kojem je pridružen **osluškivač** za konkretni događaj
  - ◆ U slučaju da je konkretnom HTML elementu pridružen osluškivač događaj, dva atributa se odnose na isti element
  - ◆ U suprotnom mogu vratiti dva različita elementa

# Pristup DOM elementu kao izvoru događaja

```
function openPresent() {  
    //const image = document.querySelector('img');  
    const image = event.currentTarget;  
    image.src =  
        'https://media.giphy.com/media/27ppQU0xe7KLG/giphy.gif';  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

# Dinamičko dodavanje/uklanjanje stilova

- ◆ Mogu se kontrolisati CSS klase primenjene na elemente

**classList.add('class1', 'class2')** – dodavanje specifičnih CSS klasa koje će se primenjivati na element

**classList.remove('class1', 'class2')** – uklanjanje specifičnih klasa koje se se primenjuju na element

**classList.toggle('class1', 'class2')** – ako se specifične klase već primenjuju one se uklanjaju; ako se ne primenjuju dodaju se elementu

# Dinamičko dodavanje/uklanjanje stilova

**JS:**

```
const image = document.querySelector('img');
```

```
// Adds a CSS class called "active"
```

```
image.classList.add('active');
```

```
// Removes a CSS class called "hidden"
```

```
image.classList.remove('hidden');
```

```
//Toggle between active and hidden
```

```
image.classList.toggle('hidden');
```

**CSS:**

```
.hidden {  
    display: none;  
}
```

```
.active {  
    display: inline;  
}
```

```
.hidden {  
    display: none;  
}
```

## DOM dinamička promena sadržaja korišćenjem display – primer 3

```
<head>  
    <meta charset="utf-8">  
    <title>DOM primer 3 display</title>  
</head>  
<body>  
    <div id="gift-outside">  
        <h1>Click for a present:</h1>  
          
    </div>  
    <div id="gift-inside" class="hidden">  
        <h1>Hooray!</h1>  
          
    </div>  
</body>
```

```
function openPresent(event) {  
    const image = event.currentTarget;  
    image.removeEventListener('click', openPresent);  
  
    const giftOutside = document.querySelector('#gift-outside');  
    const giftInside = document.querySelector('#gift-inside');  
    giftOutside.classList.add('hidden');  
    giftInside.classList.remove('hidden');  
}  
  
const image = document.querySelector('#gift-outside img');  
image.addEventListener('click', openPresent);
```

Vidljivost div kontejnera se menja tako što se dodaje i uklanja klasa hidden koja kontroliše display atribut.

# Sažetak – promene izgleda stranice pomoću DOM

## ◆ Promena sadržaja postojećeg elementa

- Primeri 1 i 2: promena sadržaja img i h1 taga
- Jednostavnije i manje promene sadržaja

## ◆ Kontrola prikaza sekcija stranice organizovanih kao blok elementi grupisanja (div, section) pomoću display

- Primer 3: naizmeničan prikaz slika
- Krupnije promene sadržaja stranice

## ◆ Dodavanje/uklanjanje elemenata pomoću naredbi `document.createElement(tag string);` `element.appendChild(element);` `element.remove();`

# Linkovi ka primerima

JavaScript objektno-orientisano nasleđivanje:  
<https://codepen.io/mljovanovic/pen/qvVeVY>

DOM dinamičko dodeljivanje osluškivača i promena  
sadržaja elementa (img):  
<https://codepen.io/mljovanovic/pen/WmzWEL>

DOM dinamičko dodeljivanje osluškivača i promena  
sadržaja elementa (img, h1):  
<https://codepen.io/mljovanovic/pen/PLeqJr>

DOM dinamičko dodeljivanje osluškivača i prikaz/  
sakrivanje elementa pomoću CSS klase (display):  
<https://codepen.io/mljovanovic/pen/wOjWWz>

Arrow funkcije:  
<https://codepen.io/mljovanovic/pen/Qordaq>

Tajmer (setInterval) kao arrow funkcija:  
<https://codepen.io/mljovanovic/pen/PLeOmN>

# Sažetak

## JS Klase:

- Definicija
- Instanciranje
- Nasleđivanje

## Arrow funkcije:

- Princip
- Sintakse
- Korišćenje

## Tajmer

## DOM:

- Struktura
- Metode
- Događaji
- Korišćenje

Primeri: [codepen.io](http://codepen.io)

# JavaScript Nasleđivanje klasa Arrow funkcije DOM

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

U pripremi prezentacije korišćene su ilustracije i primeri sa  
<https://developer.mozilla.org>

# JavaScript Dogadaji **Funkcije (callback funkcije) this**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

# Sadržaj

- ◆ **JS Događaji:**
- ◆ Definicija
- ◆ Vrste
- ◆ Upravljanje događajima: osluškivači i kretanje (propagacija)
- ◆ **this:**
- ◆ Korišćenje u funkcijama
- ◆ **Funkcije kao objekti prvog reda:**
- ◆ Načini definisanja i poziva
- ◆ Callback funkcije
  
- ◆ Primeri: [codepen.io](http://codepen.io)

# JavaScript događaji

- ◆ Većina koda klijentske Web aplikacije se pokreće tako što korisnik inicira akciju na korisničkom interfejsu
- ◆ Fizička akcija korisnika se transformiše u programski objekat **događaja**
- ◆ Aplikacija zna koji element je odgovoran za obradu događaja na osnovu funkcije **osluškivača**
- ◆ Primeri događaja:
  - ◆ Klik miša na elementu interfejsa
  - ◆ Klik na taster tastature
  - ◆ Pomeranje, promena veličine ili zatvaranje prozora
  - ◆ Završetak učitavanje Web stranice od strane pretraživača
  - ◆ Slanje podataka forme
  - ◆ Dodir i pokreti dodira

# JavaScript događaji

- ◆ JavaScript događaji nisu deo jezgra jezika, njihova realizacije je prepuštena JS rutinama koje implementiraju Web pretraživači
- ◆ Tipovi događaja:
- ◆ Događaji miša (klik, pomeraj, ulazak/izlazak pokazivača iz/u region stranice)
- ◆ Događaji tastature (keyup, keydown)
- ◆ Događaji fokusa (focus, blur : dobijanje/gubitak fokusa)
- ◆ Događaji forme (reset, submit)
- ◆ Događaji promene prikaza (resize, scroll)
- ◆ Događaji učitavanja resursa ili pristizanja zahteva (NodeJS)
- ◆ Nisu vezani samo za korisnički interfejs, već uopšte za celokupan stek aplikacije (frontend, backend)
- ◆ Kastomizovani događaji (oni koje sami definišemo)

# Rukovanje, tj, obrada događaja

- ◆ Da bi se radilo sa događajima moraju se definisati sledeći koraci:
  - ◆ **Šta** se generiše: događaja od interesa
  - ◆ **Gde** se generiše: element od interesa (izvor događaja)
  - ◆ **Kako** reagovati: odgovor aplikacije na događaj
  
- ◆ U praksi se često dešava da korisnik ne pokrene akciju direktno nad elementom, ili da različiti elementi reaguju na identičan tip događaja
- ◆ Potrebno je ustanoviti i mehanizam **prosleđivanja** i **dodeljivanja** događaja elementima

# Šta se generiše: objekat tipa Event

- ◆ Logička reprezentacija događaja
- ◆ Korena klasa iz koje su izvedene klase za specifične tipove događaja (MouseEvent, KeyboardEvent)
- ◆ Karakteristični atributi:

Atribut	Značenje
<b>type</b>	Tip događaja kao string simbolička konstanta (click, mousedown, mouseup)
<b>timeStamp</b>	Trenutak generisanja događaja
<b>currentTarget</b>	Element kojem je pridružen osluškivač, tj. koji je registrovan za prijem i obradu događaja
<b>target</b>	Element nad kojim je generisan događaj

# MouseEvent

◆ Različite radnje miša opisane skupom informacija

Atribut	Značenje
<b>button</b>	Dugme miša (0 – levo, 1 – srednje, 2 - desno)
<b>click</b>	Klik dugmeta
<b>mouseDown</b>	Pritisak dugmeta
<b>mouseUp</b>	Otpuštanje dugmeta
<b>mouseMove</b>	Pomeranje miša
<b>mouseEnter, mouseLeave</b>	Pokazivač miša ulazi ili napušta površinu elementa kojem je pridružen osluškivač
<b>pageX, pageY</b>	Pozicija miša relativna u odnosu na gornji, levi ugao dokumenta (x-horizontalno, y-vertikalno)
<b>screenX, screenY</b>	Pozicija miša relativna u odnosu na gornji, levi ugao ekrana (x-horizontalno, y-vertikalno)

```
<div class="box">  
  <p>  
    Pozicija miša u okviru kontejnera.  
  </p>  
  <p>  
    <code>pageX</code>: <span id="x">n/a</span>  
  </p>  
  <p>  
    <code>pageY</code>: <span id="y">n/a</span>  
  </p>  
</div>
```

## Primer – ispisivanje pozicije miša: HTML

Pozicija miša u okviru kontejnera.

pageX: 419

pageY: 76

## Primer – ispisivanje pozicije miša: JS

```
var box = document.querySelector(".box");
var pageX = document.getElementById("x");
var pageY = document.getElementById("y");

function updateDisplay(event) {
    pageX.innerText = event.pageX;
    pageY.innerText = event.pageY;
}

box.addEventListener("mousemove", updateDisplay);
box.addEventListener("mouseenter", updateDisplay);
box.addEventListener("mouseleave", updateDisplay);
```

# KeyboardEvent

◆ Informacije vezane za korišćenje tastera tastature

Atribut	Značenje
<b>key</b>	DOM string kao identifikator tastera tastature ('A', 'a', ...)
<b>keyDown</b>	Pritisak tastera
<b>keyPress</b>	Produžen pritisak tastera (duže od par sekundi)
<b>keyUp</b>	Otpuštanje tastera

# Form event

- ◆ Slanje ili poništavanje podataka forme

Atribut	Značenje
<b>submit</b>	Poziva se na klik <i>submit</i> dugmeta
<b>reset</b>	Poziva se na klik <i>reset</i> dugmeta

## Primer – submit događaj: HTML

```
<form id="test_form">  
  <input type="text" id="txt"  
 name="first_name" value="text"  
 placeholder="Text field..." />  
  <input type="submit"  
 name="form_submit" value="Submit" />  
</form>
```

A screenshot of a web browser displaying a simple HTML form. The form consists of a single text input field and a submit button. The text input field has the placeholder text "Text". The submit button is green with the word "Submit" in white.

## Primer – submit događaj: JS i alert prozor

```
var form = document.querySelector('#test_form');
var inputTxt = document.getElementById("txt");

form.addEventListener('submit', function(event) {
    alert('text: ' + inputTxt.value);
});
```

An embedded page at s.codepen.io says

text: text

OK

# Koji osluškivač će biti izvršen za nastali događaj ?

- ◆ Elementi mogu sadržavati druge elemente i svi mogu imati registrovane osluškivače za isti tip događaja
- ◆ Na primer, korisnik klikće mišem na “xyz”:

```
<body>
  <table>
    <tr>
      <td>xyz</td>
    </tr>
  </table>
</body>
```

- ◆ Ako svi tagovi imaju pridružene osluškivače, koji će biti pozvan ?
- ◆ Nekada unutrašnji elementi (čvorovi) obrađuju događaj
- ◆ Nekada je pogodnije da spoljni (roditeljski) čvorovi obrade događaj

# Kretanje događaja kroz DOM stablo

- ◆ Kretanje od vrha ka dubini hijerarhije (*capture*)

Početak od spoljnog elementa (vrha hijerarhije) ka unutrašnjim elemenatima (deci)

Svaki element na putanji može eksplicitno zaustaviti put događaja

`event.stopPropagation()`

`element.addEventListener(eventType, handler, true)`

- ◆ Kretanje od unutrašnjih elemenata ka vrhu (*bubble*)

Početak od ugnježdenih elemenata ka roditeljima

Svaki element na putanji može eksplicitno zaustaviti put događaja

`event.stopPropagation()`

`element.addEventListener(eventType, handler, false)`

ukoliko se ne navede 3. parametar, `false` je podrazumevano

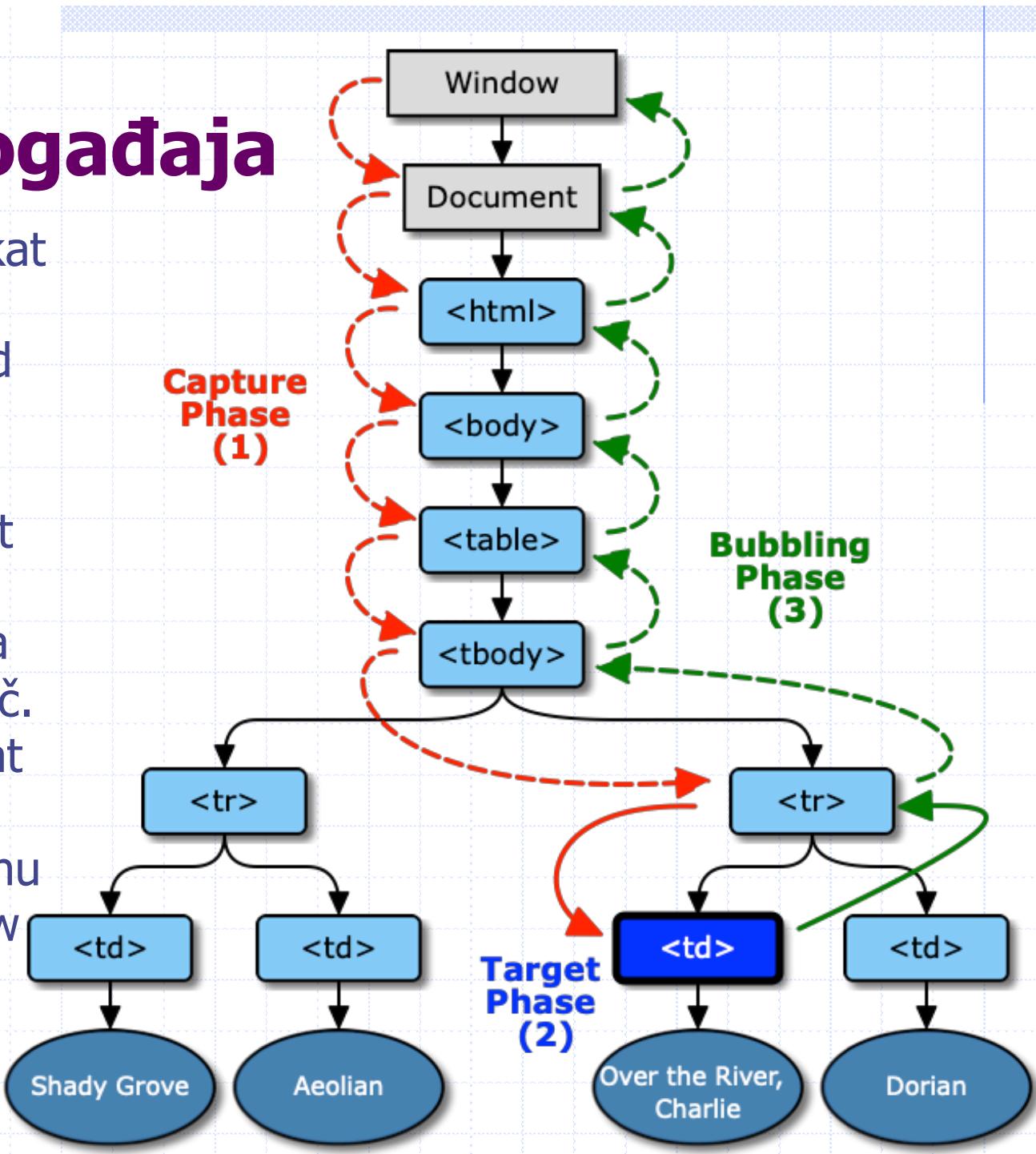
- ◆ U većini slučajeva se koristi *bubble* princip

# Kretanje događaja

**Capture** faza: Objekat događaj putuje od roditelja u dubinu, od window objekta do roditelja izvora

**Target** faza: Objekat događaj stiže na cilj, tj. u element koji ima registrovan osluškivač.

**Bubble** faza: Objekat događaj putuje od roditelja izvora ka vrhu hijerarhije, tj. window objektu



# JS Event objekat

- ◆ **event.target** – element **izvor** događaja (na primer, slika na koju se klikne). U slučaju da elementu nije pridružen osluškivač, događaj se prenosi kroz DOM čvorove dok se ne nađe na osluškivač registrovan za taj tip događaja
  - ◆ Konkretan element ne mora imati pridružen osluškivač
  - ◆ Mehanizam se naziva delegiranje događaja (*event delegation*)
- ◆ **event.currentTarget** – element kojem je pridružen **osluškivač** za konkretni događaj
  - ◆ U slučaju da je konkretnom HTML elementu pridružen osluškivač događaj, dva atributa se odnose na isti element
  - ◆ U suprotnom mogu vratiti dva različita elementa

# Višestruki obrađivači događaja

- ◆ Šta se dešava ako se osluškivač za isti tip događaja pridruži roditelju i detetu ?
- ◆ Da li se pokreću oba ?
- ◆ Koji se pokreće prvi ?

```
<div id="outer">  
  Click me!  
<div id="inner">  
  No, click me!  
</div>
```

```
const outer = document.querySelector('#outer');  
const inner = document.querySelector('#inner');  
outer.addEventListener('click', onOuterClick);  
inner.addEventListener('click', onInnerClick);
```

# Kretanje događaja od deteta ka roditelju (bubble)

- ◆ Oba osluškivača će se izvršiti klikom na element dete
- ◆ Podrazumevano, osluškivač deteta će se prvi izvršavati

div id="outer"

div id="inner"



Click me!

No, click me!

```
<div id="outer">  
    Click me!  
    <div id="inner">  
        No, click me!  
    </div>
```

# Kretanje događaja od deteta ka roditelju (bubble)

- ◆ Oba osluškivača će se izvršiti klikom na element dete
- ◆ Podrazumevano, osluškivač deteta će se prvi izvršavati

div id="outer"

div id="inner"



```
<div id="outer">  
    Click me!  
    <div id="inner">  
        No, click me!  
    </div>
```

Click me!

No, click me!



# stopPropagation()

- ◆ Prosleđivanje događaja kroz lanac roditelja se može eksplicitno zaustaviti sa event.stopPropagation()

```
function onInnerClick(event) {  
    inner.classList.add('selected');  
    console.log('Inner clicked!');  
    event.stopPropagation();  
}
```

# stopPropagation - primer

- ◆ Osluškivač će se izvršiti samo u detetu

```
function onInnerClick(event) {  
    inner.classList.add('selected');  
    console.log('Inner clicked!');  
    event.stopPropagation();  
}
```

div id="outer"

div id="inner"

Click me!

No, click me!

```
<div id="outer">  
    Click me!  
    <div id="inner">  
        No, click me!  
    </div>
```

# stopPropagation - primer

- ◆ Osluškivač će se izvršiti samo u detetu

```
function onInnerClick(event) {  
    inner.classList.add('selected');  
    console.log('Inner clicked!');  
    event.stopPropagation();  
}
```

div id="outer"



div id="inner"



Click me!

No, click me!

```
<div id="outer">  
    Click me!  
    <div id="inner">  
        No, click me!  
    </div>
```

# Kretanje događaja od roditelja ka detetu *(capture)*

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, addEventListener metodi se prosleđuje 3. parametar

**event.addEventListener('click', onClick, true)**

```
const outer = document.querySelector('#outer');
const inner = document.querySelector('#inner');
outer.addEventListener('click', onOuterClick, true);
inner.addEventListener('click', onInnerClick);
```

# Kretanje događaja od roditelja ka detetu (capture)

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, addEventListener metodi se prosleđuje 3. parametar

**event.addEventListener('click', onClick, true)**

`div id="outer"`

`div id="inner"`



```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>
```

Click me!

No, click me!



# Kretanje događaja od roditelja ka detetu (capture)

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, addEventListener metodi se prosleđuje 3. parametar

**event.addEventListener('click', onClick, true)**

`div id="outer"`

`div id="inner"`

"Outer clicked!"

"Inner clicked!"

Click me!

No, click me!



# stopPropagation() (capture)

- ◆ Prosleđivanje događaja kroz lanac roditelja se može eksplicitno zaustaviti sa event.stopPropagation()

```
function onOuterClick(event) {  
    outer.classList.add('selected');  
    console.log('Outer clicked!');  
    event.stopPropagation();  
}
```

# stopPropagation() – capture primer

- ◆ Osluškivač će se izvršiti samo u roditelju

```
function onOuterClick(event) {  
    outer.classList.add('selected');  
    console.log('Outer clicked!');  
    event.stopPropagation();  
}
```

div id="outer"



div id="inner"

Click me!



No, click me!

"Outer clicked!"

# stopPropagation() – capture primer

- ◆ Osluškivač će se izvršiti samo u roditelju

```
function onOuterClick(event) {  
    outer.classList.add('selected');  
    console.log('Outer clicked!');  
    event.stopPropagation();  
}
```

div id="outer"

div id="inner"

Click me!

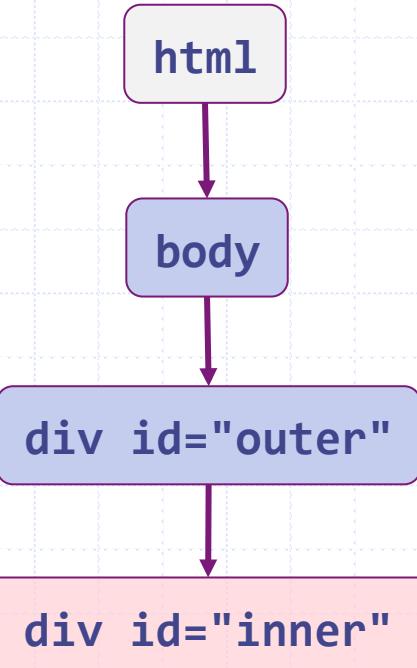
No, click me!



"Outer clicked!"

# Propagacija događaja

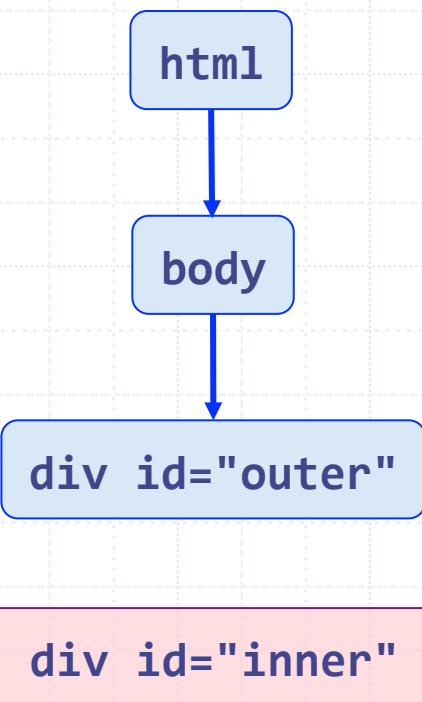
- ◆ U opštem slučaju, pretraživač **uvek** kreira putanju od izvora događaja do vrha hijerhije



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

# 1 - Capture faza

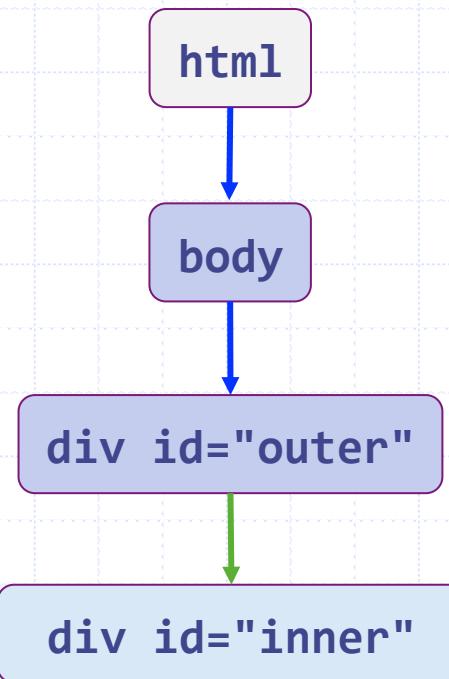
- ◆ Pretraživač kreće od vrha putanje i izvršava sve osluškivače koji imaju **true** vrednost metode do kraja putanje, tj. elementa izvora događaja



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>|
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

## 2 - Target faza

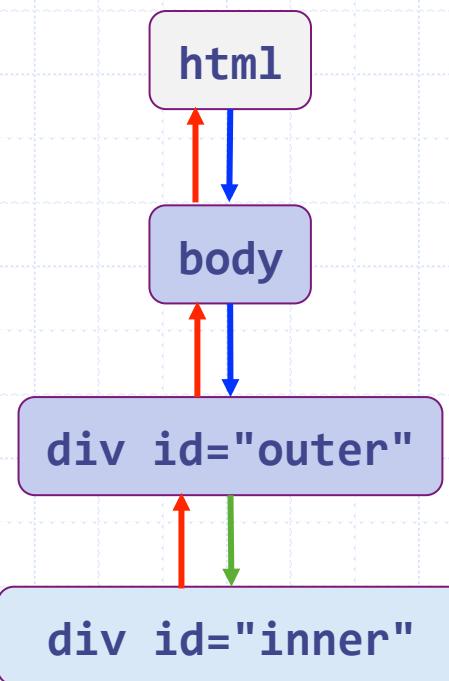
- ◆ Zatim pretraživač poziva osluškivač pridružen samom izvoru događaja (ukoliko postoji)



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>|
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

# 3 - Bubble faza

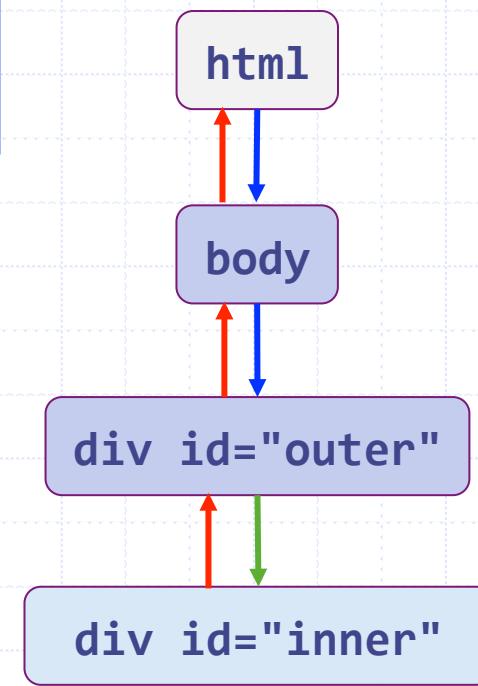
- ◆ Pretraživač se vraća povratnim putem i poziva sve osluškivače koji nisu kreirani kao *capture*, tj. podrazumevano su *bubble*



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>|
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

# stopPropagation()

- ◆ Zaustavlja putovanje događaja u lancu nezavisno od toga u kojoj se fazi nalazi !



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>|
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

# Konkurentnost događaja

- ◆ JS događaji se obrađuju sekvenčno i procesiraju po jedan u datom trenutku
  - Ne postoji višenitna obrada događaja
- ◆ Obrada događaja se odvija kao nezavisan tok procesa
  - Ne meša se sa drugim izvršavanjima
- ◆ Funkcije obrađivača događaja zahtevaju pažljiv dizajn
- ◆ Obrada se mora završiti relativno brzo (<10s)
  - U suprotnom može doći do zastoja
- ◆ Suština kontrole je kroz broj događaja i pravilnu distribuciju obrade: obezbediti dovoljan broj obrađivača koji će za relativno kratko vreme izvršiti potrebne obrade

# JavaScript klase

```
class ClassName {  
    constructor(params) {  
        this.someField =  
            someParam;  
    }  
  
    methodOne() {  
        this.methodTwo();  
        const someValue =  
            this.someField;  
    }  
  
    methodTwo() {  
        ...  
    }  
}
```

Ne koristi se ključna reč  
*function*

Sve metode su *public* (za sada)

Unutar klase, morate uvek  
pozivati druge metode sa **this**

Unutar klase, poljima se mora  
uvek pristupati sa **this**

# this u konstruktoru

```
class Point {  
    constructor(x, y)  
    {  
        this.x = x;  
        this.y = y;  
    }  
}
```

U konstruktoru klase,  
this ukazuje na objekat  
koji se kreira

# this u funkciji

```
class Point {  
    ...  
    toString() {  
        return this.x + ", " + this.y;  
    }  
}
```

this može imati **različita značenja** u zavisnosti od toga kako se funkcija poziva, tj. u kom **kontekstu**

Svaka funkcija (uključujući i one koje se definišu van klase) sadrži this kao atribut

Njegova vrednost zavisi od toga kako se funkcija poziva !

# this u obrađivaču događaja

```
function onClick() {  
    console.log('Clicked!');  
    console.log(this);  
}
```

```
const button = document.querySelector('button');  
button.addEventListener('click', onClick);
```

Kada se koristi u obrađivaču događaja, this pokazuje na  
**element kojem je osluškivač pridružen**

Clicked! [console\\_runner-1df7d...81e5fba982f6af.js:1](#)  
[console\\_runner-1df7d...81e5fba982f6af.js:1](#)  
<button>Click and view the Console!</button>

```
function onClick() {  
  console.log('Clicked!');  
  console.log(this);  
}  
const button =  
document.querySelector('button');  
button.addEventListener('click', onClick);
```

U onClick funkciji, this ukazuje na <button> zbog toga što se funkcija poziva u addEventListener.

# this u funkciji - primer

```
function sayHello() {  
    console.log(this.name + ' says hello');  
}  
  
var person = {  
    name: 'Person',  
    hobbies: ['knitting', 'cooking', 'dancing'],  
    greeting: sayHello  
};  
person.greeting();
```

Kada koristimo this u funkciji koja se ne poziva iz obradivača događaja, **this ukazuje na objekat iz kog se (ili nad kojim) se funkcija poziva**

# this u funkciji

```
function sayHello() {  
    console.log(this.name + ' says hello');  
}  
  
var person = {  
    name: 'Person',  
    hobbies: ['knitting', 'cooking',  
    'dancing'],  
    greeting: sayHello  
};  
  
person.greeting();
```

"Person says hello"

# Klasični programski jezici

- ◆ Kod:
- ◆ Lista instrukcija koje računar izvršava
- ◆ Svaka linija koda je naredba
  
- ◆ Funkcija:
- ◆ Grupa naredbi označena labelom
- ◆ Sve naredbe se izvršavaju prilikom poziva funkcije
  
- ◆ Varijabla:
- ◆ Podaci označeni labelom

U JS funkcije  
predstavljaju i  
klasične funkcije i  
variable !

```
const x = 15;  
let y = true;  
  
const greeting = function() {  
    console.log('hello, world');  
}  
}
```

**Funkcija u JavaScript-u je objekat tipa Function**

memorija JS interpretera:

→ `const x = 15;  
let y = true;`

```
const greeting = function() {  
    console.log('hello, world');  
}
```

**Funkcija u JavaScript-u je objekat tipa Function**

memorija JS interpretera:

```
const x = 15;
```

→ let y = true;

```
const greeting = function() {  
    console.log('hello, world');  
}
```

x

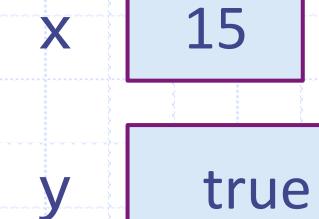
15

Funkcija u JavaScript-u je objekat tipa Function

memorija JS interpretera:

```
const x = 15;  
let y = true;
```

→ const greeting = function() {  
 console.log('hello, world');  
}



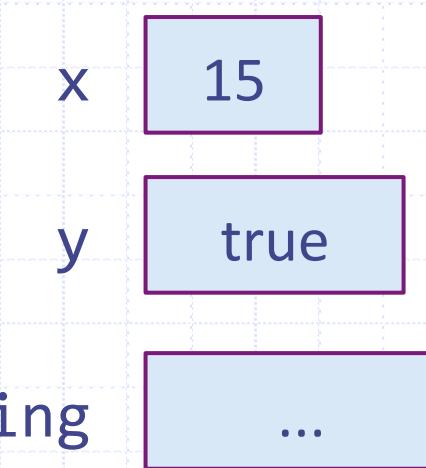
Funkcija u JavaScript-u je objekat tipa Function

```
const x = 15;  
let y = true;
```

```
const greeting = function() { greeting  
    console.log('hello, world');  
}
```



memorija JS interpretera:



**Funkcija u JavaScript-u je objekat tipa Function:**  
Kada se definiše funkcija, kreira se objekat tipa  
Function čiji je sadržaj kod funkcije

# Atributi Function objekta

```
const greeting = function() {  
    console.log('hello, world');  
};  
  
console.log(greeting.name);  
console.log(greeting.toString());  
greeting.call();
```

Function objekat takođe ima `call()` metodu koja poziva izvršni kod funkcije

Sintaksa () u pozadini poziva ovu metodu !

Kada se definiše funkcija, kreira se objekat tipa Function koji sadrži atribute kao što su :

`name`  
`toString()`

"greeting"

"function () {  
 console.log('hello, world');  
}"

"hello, world"

# Funkcije su specifičan tip objekta

- ◆ Jedini tip koji ima pridružen izvršni kod, odnosno objekat koji se može izvršavati
- ◆ Obični objekti se ne mogu izvršavati
- ◆ Ali mogu imati atributi čija je vrednost objekat tipa Function (tj. funkcije) koji se mogu izvršavati

```
function sayHello() {  
    console.log(this.name + ' says hello');  
}  
  
var person = {  
    name: 'Person',  
    hobbies: ['knitting', 'cooking', 'dancing'],  
    greeting: sayHello  
};  
person.greeting();
```

greeting  
atribut je  
objekat tipa  
Function (tj.  
sayHello  
funkcije)

"Person says hello"

# Razlog zbog kog su JS funkcije objekti

- ◆ Mogu se prosleđivati kao parametri drugih funkcija
- ◆ Poznate kao **callback funkcije**
- ◆ Zbog toga što kada definišemo funkciju u kodu u pozadini se kreira objekat tipa Function
- ◆ Programiranje zasnovano na obradi događaja
- ◆ Angular
- ◆ NodeJS

```
function onDragStart(event) {  
    ...  
}  
dragObject.addEventListener('pointerdown', onDragStart);
```

# JS funkcije kao objekti prvog reda

- ◆ Mogu se prosleđivati kao parametri
- ◆ Mogu biti vrednosti varijabli
- ◆ Mogu biti definisane bez imena/identifikatora
  - Anonimne funkcije
  - Arrow funkcije

# Načini definisanja funkcije

```
function myFunction(params) {  
}
```

```
const myFunction = function(params) {  
};
```

```
const myFunction = (params) => {  
};
```

Pozivaju se na identičan način, bez obzira kako se definišu:

```
myFunction();
```

# Načini definisanja funkcije - primer

```
function imeFunkcije(pozvanaFunkcija) {  
    pozvanaFunkcija();  
}
```

```
var varijablaFunkcije = function() {  
    console.log('funkcija kao varijabla');  
};
```

```
var arrowFunkcija = () => {  
    console.log('arrow funkcija');  
};
```

```
imeFunkcije(varijablaFunkcije); "funkcija kao varijabla"  
imeFunkcije(arrowFunkcija);
```

"arrow funkcija"

# Callback funkcija

- ◆ Funkcija koja se prosleđuje kao parametar drugoj funkciji
- ◆ Logično sa obzirom da je funkcija objekat prvog reda u JS (instanca tipa Object)
- ◆ Mehanizam koji često koriste JS okviri za razvoj klijentskih Web aplikacija
  - ◆ Angular
  - ◆ NodeJS
  - ◆ Express
  - ◆ React
  - ◆ ...

# Anonimna funkcija kao callback

```
var fruits = ["apple", "banana", "orange"];
fruits.forEach(function(fruit, index){
    console.log(index + "." + fruit);
});
```

"0.apple"

"1.bannana"

"2.orange"

# Imenovana funkcija kao callback

```
function namedCallback(){  
    alert("namedCallback()");  
}  
  
function testFunction(callback){  
    callback();  
}  
  
testFunction(namedCallback);
```

An embedded page at s.codepen.io says

namedCallback()

OK

# Parametri callback funkcije

```
var author = "Caller";
function namedCallback(param){
    alert("namedCallback() called by " + param);
}
function testFunction(callback){
    callback();
}
testFunction(namedCallback(author));
```

An embedded page at s.codepen.io says

namedCallback() called by Caller

OK

# Callback + parametar kao promenjiva

U konkretnom primeru, callback funkcija je opcionalna kao parametar (može se i ne mora prosleđivati)

```
var message = "hello there!";
function myCallbackFunction(){
    alert("myCallbackFunction is called()");
}

function testCallback(param, callback){
    alert(param);
    if(callback && typeof(callback) === "function"){
        callback();
    }
}
testCallback(message, myCallbackFunction);
```

# Rezultat?

An embedded page at s.codepen.io says  
hello there!

1.

OK

An embedded page at s.codepen.io says  
myCallbackFunction is called()

2.

OK

# Linkovi ka primerima

Tipovi događaja:

<https://developer.mozilla.org/en-US/docs/Web/Events>

Ispisivanje pozicije miša:

<https://codepen.io/mljovanovic/pen/ZPmzNd>

Submit događaj: <https://codepen.io/mljovanovic/pen/aMQWdX>

JS događaji bubble: <https://codepen.io/mljovanovic/pen/NJeKya>

JS događaji bubble stopPropagation:

<https://codepen.io/mljovanovic/pen/JzwjWK>

JS događaji capture: <https://codepen.io/mljovanovic/pen/MxZYEJ>

JS događaji capture stopPropagation:

<https://codepen.io/mljovanovic/pen/OqrNOW>

Načini definisanja funkcije:

<https://codepen.io/mljovanovic/pen/WmLJXb>

Callback funkcije: <https://codepen.io/mljovanovic/pen/pYqZVB>

Function objekat: <https://codepen.io/mljovanovic/pen/RdEqdQ>

this u obrađivaču događaja:

<https://codepen.io/mljovanovic/pen/PLXrye>

this u funkciji: <https://codepen.io/mljovanovic/pen/KEENem>

# Sadržaj

- ◆ **JS Događaji:**
- ◆ Definicija
- ◆ Vrste
- ◆ Upravljanje događajima: osluškivači i kretanje (propagacija)
- ◆ **this:**
- ◆ Korišćenje u funkcijama
- ◆ **Funkcije kao objekti prvog reda:**
- ◆ Načini definisanja i poziva
- ◆ Callback funkcije
  
- ◆ Primeri: [codepen.io](http://codepen.io)

# JavaScript Dogadaji **Funkcije (callback funkcije) this**

Dr. Mlađan Jovanović  
[mjovanovic@singidunum.ac.rs](mailto:mjovanovic@singidunum.ac.rs)

U pripremi prezentacije korišćene su ilustracije i primeri sa  
<https://developer.mozilla.org>