Napredni JavaScript (napredni koncepti)

Dragi polaznici, ova prezentacija nije primarno namijenjena za učenje već služi kao **pomoćni materijal**. Kao takva, ne može zamijeniti predavanja, literaturu kao niti vašu (pro)aktivnost na nastavi i konzultiranje s predavačem u slučaju potencijalnih pitanja i/ili nejasnoća.

Napredni JavaScript

Ključne riječi

closures, scope, “use strict”, OOP, patterns, ES6, API

Cjeline

Uvod u napredne koncepte

Funkcije - napredni koncepti

Vježba 1

JavaScript OOP - 1.dio

JavaScript OOP - 2.dio

Vježba 2

Obrasci programskog koda

ES6 standard - 1.dio

ES6 standard - 2.dio

Rad s API-jima

Vježba 3

Animacije u JavaScriptu

Osnove programiranja

Uvod u napredne koncepte

Eksplicitna konverzija podataka

Par je metoda koje možemo koristiti kad želimo konvertirati vrijednost:

1. Upotrijebiti **Number, String, Boolean i Object** funkcije
2. **toString** metoda za konverziju bilo koje vrijednosti u string
3. **parseInt** i **parseFloat** parsiraju vrijednost i izvlače broj

Hoisting (izvlačenje)

Hoisting je premještanje svih deklaracija na vrh trenutnog opsega (na vrh trenutne skripte ili trenutne funkcije).

Varijable je uvijek dobro deklarirati na početku funkcije!

Objekt

Osnovni oblik podataka JavaScripta-a, zbirka svojstava od kojih svako ima ime (ključ) i vrijednost.

```
{ name: 'Golden retriever' }

function bark() {}

['John', 'Jane', 'Joe']
```

I objekti i funkcije i nizovi su objektni tip podatka i mogu imati metode i svojstva!

Opseg (scope)

Opseg određuje dostupnost varijabli, objekata i funkcija iz različitih dijelova koda.

JavaScript koristi **leksički opseg** implementiran **funkcijama.**

Postoje 2 tipa dosega opsega u Javascriptu: **lokalni** i **globalni**.

```
var x = 2;

function() {
    var x = 1;
}

console.log(x);
```

Globalne i lokalne varijable

```
var x = 2;

function() {
    x = 1;
}

console.log(x);
```

Koju će vrijednost ispisati console.log?

U slučaju da je varijabla deklarirana unutar funkcije, ona je **lokalna,** *osim ako se izostavi ključna riječ var*.

U slučaju da je varijabla deklarirana van funkcije, ona je **globalna**.

Namespace

Problem s globalnim varijablama je taj što ih dijele svi kodovi u vašoj JavaScript aplikaciji ili web stranici.

Oni žive u istom **globalnom prostoru imena** i uvijek postoji mogućnost sudara imenovanja - kada dva odvojena dijela aplikacije definiraju globalne varijable s istim nazivom, ali s različitim svrhama.

this

Ključna riječ **this** odnosi se na objekt konteksta izvođenja.

1. Van objekta ili u funkciji se odnosi na *globalni objekt*
2. U metodi objekta se to odnosi na sam objekt

   … o drugim kontekstima pričat ćemo na Naprednom Javascript dijelu

```
var person = {
     name: 'John',
     speak: function(word) {
          return this.name;
     }
};
```

Zadaci

Napredni JavaScript

Funkcije - napredni koncepti

Načini korištenja funkcije

U JavaScriptu funkcije su objekti prve klase, što znači se se tretiraju kao bilo koja druga varijabla i da mogu imati vlastita svojstva i metode.

Funkcije se mogu:

- dodijeliti varijabli
- proslijediti drugoj funkciji kao argument
- vratiti iz druge funkcije (koristeći return)

Funkcija koja vraća drugu funkciju zove se HOF (Higher-Order Function)

Načini korištenja funkcije

Kao varijabla

```
function vratiNesto(x) {
          return x;
}

var y = vratiNesto();
```

```
function vratiNesto(x) {
          return x;
}

var y = vratiNesto;

y(“Hello world”);
```

Načini korištenja funkcije

Kao argument druge funkcije

```
function vratiNesto(x) {
          var y = x(1);
          return y;
}

function vratiMene(z){
          return z/2;
};

var y = vratiNesto(vratiMene);
```

Načini korištenja funkcije

Kao vraćena vrijednost

```
function vratiNesto(x) {
        return function() {
                return x;
    };
}

var y = vratiNesto(true);
```

```
function vratiNesto(x) {
        return function() {
                return x;
    };
}

var y = vratiNesto(true)();
```

Anonimna funkcija

```
(function vratiNesto() {
        var x = 1;
        return x;
})
```

Anonimna funkcija je funkcija koja nema ime. Može biti izravno dodijeljena varijabli, ili obuhvaćena zagradama (npr. da ne zagađuje globani scope).

```
(function vratiNesto() {
        var x = 1;
        return x;
})()
```

Immediately Invoked Function Expression (IIFE)
tj.
Samopozivajuća anonimna funkcija

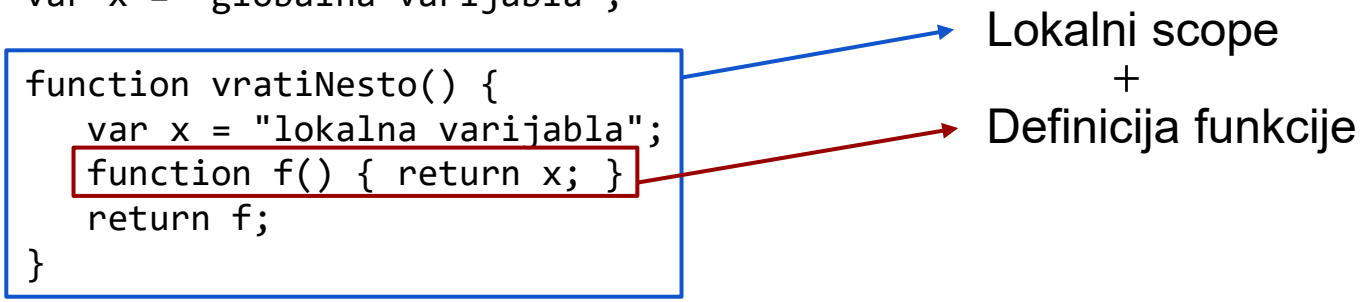
Closure

U računalnoj znanosti, closure je kombinacija objekta funkcije i dosega (scopea) unutar kojeg su varijable funkcije razriješene.

Sve funkcije u JavaScriptu implementiraju **closure** tehniku**.**
Doseg (scope) varijabla funkcije je onaj unutar kojeg je funkcija definirana, a ne unutar kojeg je pozvana!

Closure

```
var x = "globalna varijabla";

function vratiNesto() {
    var x = "lokalna varijabla";
    function f() { return x; }
    return f;
}

var y = vratiNesto()();
```



Funkcionalno programiranje

Funkcionalno programiranje je jedna od programskih paradigmi, uz npr. objektno-orijentirano programiranje. Više o tome u React modulu!

Funkcionalno programiranje je programska paradigma koji računanje smatra kao evaluaciju matematičkih funkcija i izbjegava promjenu stanja i promjenjivih podataka.

Napredni JavaScript

Funkcije - vježba

Napredni JavaScript

OOP JavaScript - 1.dio

Objektno-orijentirano programiranje (OOP)

Objektno-orijentirano programiranje je jedna od programskih paradigmi koja koristi objekte kao modele za računanje.

Programska paradigma je način klasifikacije programskih jezika na temelju njihovih značajki. Jezici se mogu svrstati u više različitih paradigmi.

OOP, tldr: Sve je objekt!

Neki poznati OOP jezici: Java, C++, C#, Python, PHP...

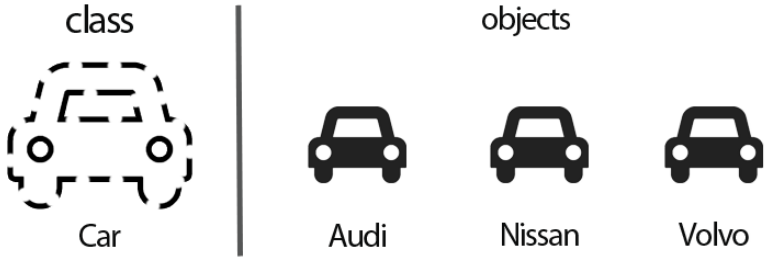
JavaScript OOP

JavaScript je dizajniran da bude objektno-orijentiran jezik.
Većina tipova podataka u JavaScriptu su zapravo objekti (čak i funkcije).

Jezici koji podržavaju objektno orijentirano programiranje (OOP) obično koriste **nasljeđivanje** (metoda i svojstava) u svrhu ponovne upotrebe postojećeg koda i proširivost u obliku klasa ili prototipa.

Primjer: OOP u Javi (tzv. klasično nasljeđivanje)

1. Baziran na klasama
2. Objekti se stvaraju iz klasa
3. Ti objekti se nazivaju instancama klasa



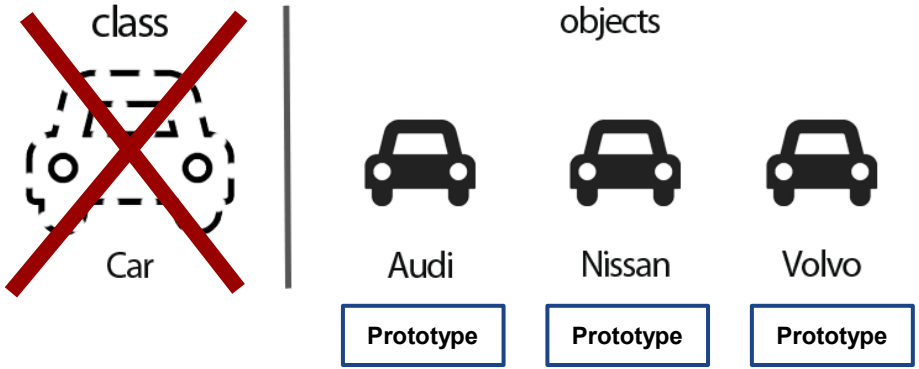
Primjer: OOP u Javi (tzv. klasično nasljeđivanje)

```
class Dog {
          public String bark() {
          return 'Wuf Wuf';
     }
}

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

```
class GoldenRetriever extends Dog {
     public String fetch() {
          return 'Ball!';
     }
}

var goldenRetriever = new GoldenRetriever();

golderRetriever.fetch();
golderRetriever.bark();
```

OOP u JavaScriptu

Baziran na **prototip** načinu nasljeđivanja. JavaScript nema klase!



Prototype objekt

Svaki JavaScript objekt ima drugi JavaScript objekt (ili null, ali to je rijetko) povezan s njim.

Ovaj drugi objekt poznat je kao **prototip**, a prvi objekt nasljeđuje svojstva od prototipa.

```
var person = {
        name: 'John',
        surname: 'Doe',
        <prototype>: Object { … }
};

console.log(person);
```

Probajte pokrenuti ovaj kod, pogledajte rezultat u konzoli.

Prototype !== Prototype

Jedan od najnerazumljivijih koncepata u JavaScriptu za shvatiti jest da JS objekti imaju **2 prototype svojstva**

1. Prvi se vodi pod imenom __proto__ ili <prototype>, referenca je na drugi objekt koji je jednak konstruktor prototype-u i nasljeđuje njegove metode. NE SMIJE SE MIJENJATI.

1. Drugi je objekt koji možemo slobodno mijenjati i dodavati metode. Ime mu je najčešće prototype, ali može biti i proizvoljno.

OOP JavaScript - 1.dio - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Napredni JavaScript

OOP JavaScript - 2.dio

Primjer: OOP u JavaScriptu

Za razliku od, npr. Jave, JavaScript nema definirana privatna, javno dostupna, zaštićena ili privilegirana svojstva objekata, ali mogu se ostvariti kroz principe closure-a.

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

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

Konstruktor

Konstruktor je funkcija dizajnirana za inicijalizaciju novostvorenih objekata.

Kritična značajka poziva konstruktora je da se prototip svojstvo konstruktora koristi kao prototip novog objekta.

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

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

Riječ “new”

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

1. Stvara se potpuno novi prazni objekt
2. Novi prazni objekt povezuje se sa svojstvom prototipa te funkcije
3. Isti novi prazni objekt postaje this za kontekst izvršenja tog poziva funkcije
4. Ako ta funkcija ništa ne vraća, onda implicitno vraća ovaj objekt.

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

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

Riječ “this” u konstruktor funkciji

U funkciji koja je invocirana riječju new (konstruktor funkciji), riječ **this** se odnosi na samu funkciju. Metode i varijable koje su deklarirane sa riječju this pripadaju objektu te funkcije.

Ne zaboravite!
Kod “normalnih” funkcija riječ this se odnosi na globalni objekt.

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

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

Call, bind, apply

Svaka funkcija ima metode pozivanja, vezivanja i primjene (.call, .bind, .apply).
Ove se metode mogu koristiti za postavljanje prilagođene vrijednosti "this" u kontekst izvršenja funkcije.

Bind vam omogućuje da postavite ovu vrijednost sada, dok vam omogućuje izvršavanje funkcije u budućnosti, jer vraća novi objekt funkcije.

	time of function execution	time of this binding
function object f	future	future
function call f()	now	now
f.call() f.apply()	now	now
f.bind()	future	now

PRO TIP!

Prototip metode

Metode objekta je najoptimalnije definirati na samom prototipu konstruktor funkcije.

Zašto?
Jer u suprotnom svaki put kad pozovemo konstruktor objekta te metode će se ponovno definirati, a neće biti različite od svih drugih u drugim objektima kreiranim s istim konstruktorom.

```
function Dog() {
     this.hasSat = false;
}

Dog.prototype.bark = function() {
     return 'Wuf Wuf';
}

Dog.prototype.sit = function() {
          this.hasSat = true;
}

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

JavaScript OOP - 2.dio - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 15min

Napredni JavaScript

OOP - vježba

Napredni JavaScript

Obrasci programskog koda

Obrazac (pattern)

Rješenje za česte probleme u programiranju. Rješava:

1. Potrebu za ponavljanjem istog koda na više mjesta
2. Kompliciran kod pojednostavljenjem abstrakcije
3. Probleme komunikacije tima pružanjem zajedničkog jezika opisivanja sličnih problema

Tipovi obrazaca

Dizajn obrasci
GoF obrasci za OOP jezike (Java, C++)

Obrasci u kodu
Vezani za specifični programski jezik

Antiobrasci (antipattern)
Obrasci koji kreiraju više problema nego što rješavaju

Dizajn obrasci za OOP

Singleton

Factory

Iterator

Decorator

Strategy

...

U OOP-u Singleton podrazumijeva samo jednu instancu klase. U JavaScriptu je objekt literal već singleton (**nema klasa!**):

```
var person = {
     name: 'John Doe'
}
```

Dizajn obrasci za OOP

Singleton

Factory

Iterator

Decorator

Strategy

...

Koristi se za kreaciju objekata. Ima sljedeće funkcije:

1. Obavlja ponavljajuće operacije prilikom postavljanja sličnih objekata
2. Nudi način stvaranja predmeta bez poznavanja određene vrste (klase) u vrijeme sastavljanja

https://itnext.io/easy-patterns-simple-factory-b946a086fd7e

Module obrazac

Koristi se za kreiranje JS paketa, tj. samostalnih dijelova koda, koji se mogu dodavati, zamijeniti, micati po volji bez opasnosti da se nesto poremeti u ostatku aplikacije/web stranice.

Par pojmova koje bi bilo dobro proučiti prije implementacije module obrasca:

- imenski prostor (namespace)
- samopozivajuće anonimne funkcije
- privatni i povlašteni članovi objekta
- deklaracije paketa o kojima ovisimo

Export/import paketa

```
// Module-first.js
var module1 = {
     x: 1,
     y: function(){
          console.log('Hello')
     }
};

module.exports = module1;
```

```
// Module-second.js
var a = require('./Module-first');
var module2 = {
     y: function(){
          a.y();
     }
};
```

Module obrazac - primjer

```
var myModule = (function($) {
  'use strict';

  var _privateProperty = $('#textarea').text('Hello World');

  function _privateMethod() {
    console.log(_privateProperty);
  }

  return {
    publicMethod: publicMethod,
  };
}(jQuery));
```

Obrasci programskog koda - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Napredni JavaScript

ES6 standard - 1.dio

ES6 (ECMAScript 6)

ES6 ili ECMAScript 2015 standard je specifikacija jezika ECMAScript koji definira standard za implementaciju JavaScripta.

Za sve preglednike koji ne podržavaju ovaj standard koristimo Babel transpiler koji pretvara kod u prijašnju verziju JavaScripta.

ECMAScript 2015 (ES6) - OTHER

Support for the ECMAScript 2015 specification. Features include Promises, Modules, Classes, Template Literals, Arrow Functions, Let and Const, Default Parameters, Generators, Destructuring Assignment, Rest & Spread, Map/Set & WeakMap/WeakSet and many more.

Usage | % of all users ?

Global | 91.21% + 5.15% = 96.36%

Current aligned | Usage relative | Date relative | Apply filters | Show all ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mo
		2-5	4-20	3.1-7	10-12.1	3.2-6.1				
	2 12-14	6-53	21-50	7.1-9.1	15-37	7-9.3		2.1-4.3		
6-10	2 3 15-17	2 54-67	2 51-75	10-12	2 38-60	10-12.1		4.4-4.4.4	7	12-12.
1 2 11	2 3 18	2 68	2 76	12.1	2 62	12.3	all	2 67	10	2 46
	2 76	2 69-70	2 77-79	13-TP		13				

“use strict”

Svrha direktive **"use strict"** je:

- naznačiti da je kod koji slijedi (u skripti ili funkciji) strogi kod
- strogi način rada olakšava pisanje "sigurnog" JavaScript-a
- strogi način rada prethodno naznačenu "lošu sintaksu" pretvara u stvarne pogreške u kodu.

ES6 novosti

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

...

Let deklaracija

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

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

```
var x = 10;

{

  let x = 2;

}
```

Koja je vrijednost varijable x unutar zagrada?

const deklaracija

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

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

Template literali

Umjesto konkatenacije stringova pomoću + operatora, koristimo backtickove.

```
const ime = 'Ivan';

const pozdrav = `Moje ime je ${ime}`;

console.log(pozdrav);
```

Zadatak: Nađite backtick znak na svojoj tipkovnici.

“Spread” operator (...)

Omogućava nam dekonstrukciju niza na njegove elemente.

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

“Rest” operator (...)

Omogućava nam definiciju funkcije sa nepoznatim brojem argumenata.

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

ES6 module - 1.dio - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Napredni JavaScript

ES6 standard - 2.dio

Za one koji žele znati više

https://www.w3schools.com/js/js_es6.asp
https://medium.com/beginners-guide-to-mobile-web-development/introduction-to-es6-c4422d3c5664

ES6 novosti

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

“Arrow” funkcije

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

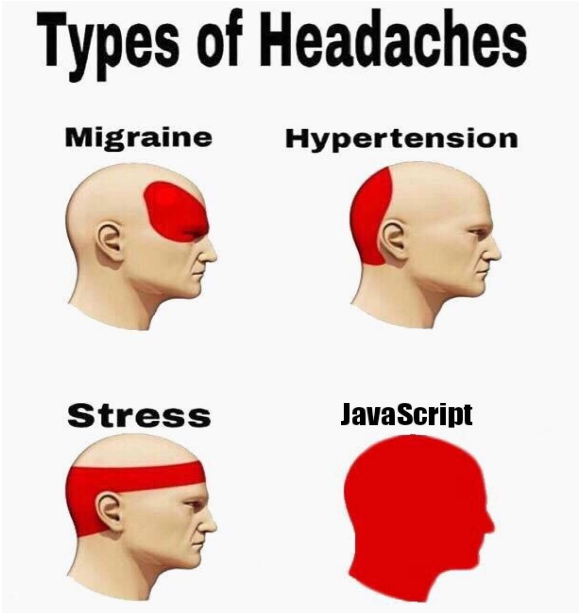
Napišite ovu funkciju
u ES5 notaciji!

```
const bark = () => ‘Wuf wuf’;

forEach(myArray, element => console.log(element));

forEach(myArray, element => {
          console.log(this);
          return element;
);
```

Klase

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

```
class Dog {
          ...
}
```



Klase

Također, imamo par novi izraza koje nam pomožu u radu sa ”klasama”.

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

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

Prepišite Dog i GoldenRetriever klase iz OOP - 2.dio lekcija sa novim class izrazima!

Obećanja (Promises)

Obećanje je objekt koji će rezultirati jednom vrijednosti koja može biti ili riješena ili neriješena (odbačena).

```
new Promise((resolve, reject) => {
  ...
  if (...) {
    return resolve;
  }
  return reject;
})
.then()
.catch();
```

Egzekutor funkcija mora vratiti ili resolve ili reject callback funkciju.

.then reagira na stanje fufilled promise objekta
.catch reagira na stanje rejected

Obećanja (Promises) - primjer

Promises se mogu nositi sa negativnim ishodima akcije:

```
function getX() {
           return Math.random() >= 0.5;
}

var wait = new Promise((resolve, reject) => {
  const x = getX();

  if (x) {
    return setTimeout(resolve, 1000);
  }
  return setTimeout(reject, 1000);
})
.then(() => console.log('Yay!'))
.catch(() => console.log('Oh no!'));
```

Obećanja (Promises) - primjer

```
setTimeout(function(){
  console.log('Yay!')
}, 1000)
```

```
var wait = new Promise(function(resolve, reject){
  return setTimeout(resolve, 1000);
}).then(function() {
  console.log('Yay!')
});
```

Prepišite gore napisani Promise sa arrow funkcijama!

ES6 module - 2.dio - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 25min

Napredni JavaScript

Rad s API-jima

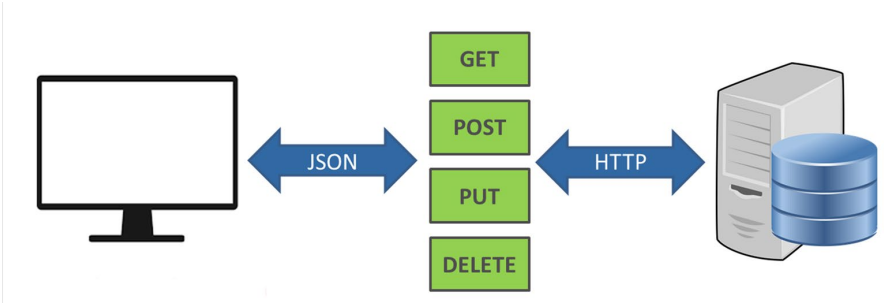
Za one koji žele znati više

https://www.taniarascia.com/how-to-connect-to-an-api-with-javascript/
https://restfulapi.net/
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
https://www.w3schools.com/xml/ajax_intro.asp

API

Application **P**rogramming **I**nterface

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



HTTP request

Akcija	HTTP Metoda	Opis
Create	POST	Kreira novi resurs
Read	GET	Vrati resurs
Update	PUT/PATCH	Ažurira postojeći resurs
Delete	DELETE	Briše resurs

REST(ful) API

REpresentational **S**tate **T**ransfer

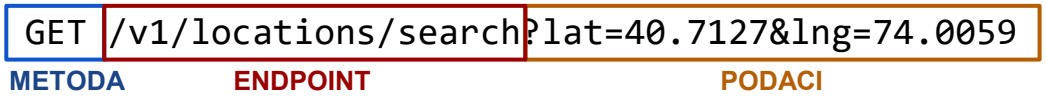
Skup pravila kojih se programeri pridržavaju prilikom kreiranja API-ja. Jedno od tih pravila kaže da biste trebali biti u mogućnosti dobiti dio podataka (koji se zove **resurs**) kada se povezujete na određeni **URL**.

```
GET /v1/locations/search?lat=40.7127&lng=74.0059
```

Povezivanje na REST API

1. Krajnja točka (endpoint)
2. Metoda
3. Zaglavlja (headers)
4. Podaci (ili tijelo) (data/body)

Koristit ćemo **JSON format** za prijenos podataka između servera i aplikacije!



AJAX

Asynchronous **Ja**vaScript and **X**ML

Kombinacija **XMLHttpRequest** objekta ugrađenog u preglednik (za traženje podataka s web poslužitelja) i HTML DOM-a (za prikaz ili upotrebu podataka).

AJAX

Sa AJAX-om možemo bez učitavanja

1. Zatražiti podatke s poslužitelja
2. Primati podatake s poslužitelja
3. Poslati podatke poslužitelju

```
var request = new XMLHttpRequest();
request.open('GET', endpoint, true);

request.onload = function () {
  // Pristupi podacima koje je server vratio
};

request.send();
```

Napredni JavaScript

Rad s API-jima - vježba

Rad sa API-jima - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 45min

Napredni JavaScript

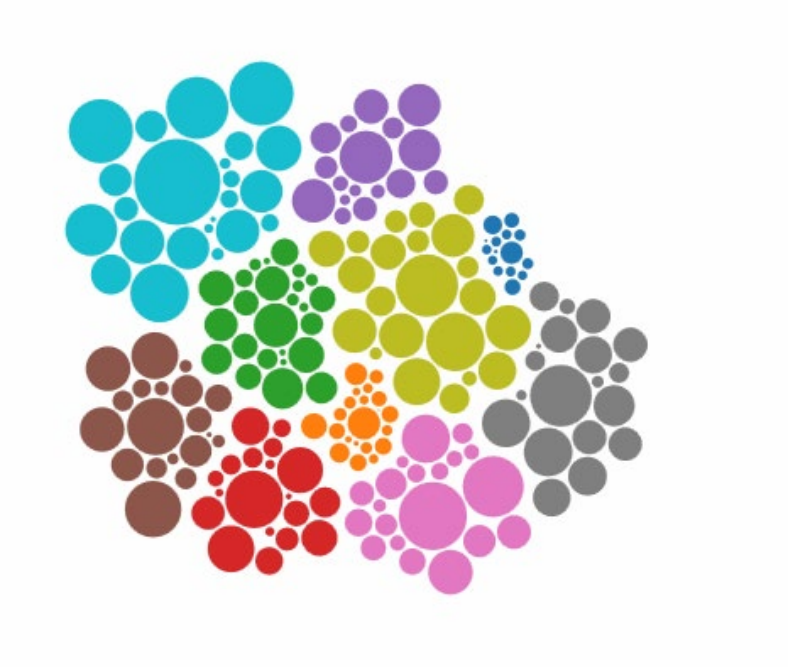
Animacije

Za one koji žele znati više

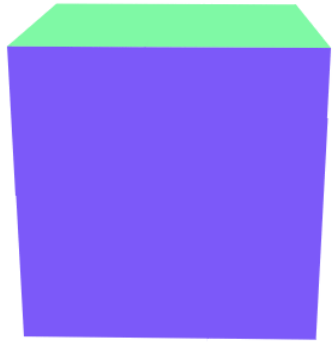
- https://cssauthor.com/javascript-animation-libraries/
- https://css-tricks.com/myth-busting-css-animations-vs-javascript/
- https://developers.google.com/web/fundamentals/design-and-ux/animations/css-vs-javascript
- https://animejs.com/documentation

Mo.js - animation engine library

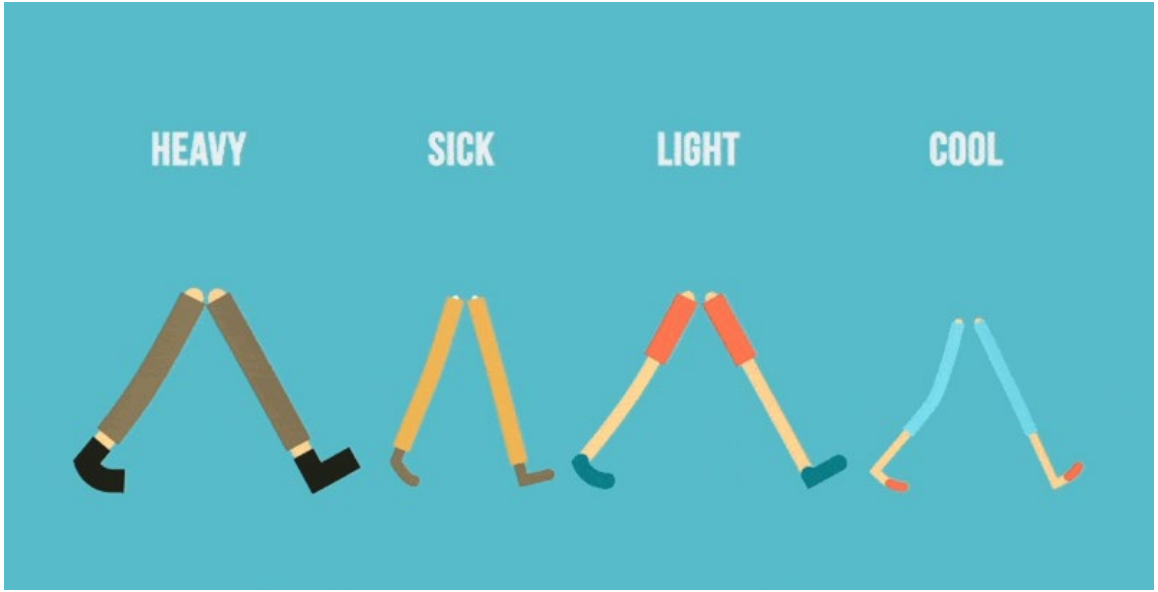
D3.js - za animiranje infografika



three.js - animacija 3D objekata



Greensock - animation engine library



JavaScript animacija vs. CSS animacija

Ako možete nešto animirati u CSS-u, animirajte u CSS-u.

Zašto?

1. CSS kod neće skršiti stranicu ako nešto krene po zlu.
2. CSS koristi browser hardware ubrzanja kad je moguće, JS nema tu privilegiju (obično).
3. JS kod je “skup”, tj. skuplji od CSS za obraditi.
4. Ako se oslanjate na biblioteku za animiranje, može vam se ukinuti održavanje te biblioteke.

JavaScript animacija vs. CSS animacija

```
var boxElement = document.querySelector('.box');

var animation = boxElement.animate([
  {transform: 'translate(0)'},
  {transform: 'translate(40px, 30px)'}
], 2000);
```

```
.box {
  animation-name: movingBox;
  animation-duration: 2000ms;
}
@keyframes movingBox {
  0% {
    transform: translate(0, 0);
  }
  100% {
    transform: translate(40px, 30px);
  }
}
```

Elementi animacije

Element	**Primjeri motoda**
Timeline	Delay, Stagger, Duration, Direction, Autoplay
Kontrole animacije	Play, Stop, Pause, Reverse
Easing efekti (ublažavanje)	Linear, Spring, Cubic bezier, Elastic
Keyframes	Definicija vrijednosti na zadanom keyframe-u

Animacije - vježba

Pratite upute u Upute.txt
Trajanje vježbe: 30min