



# JavaScript

Osamnaesti dio



# Pregled

- Decorator funkcije
  - Uvod
  - Object.defineProperty
- Memoization
- ES6
- Composition kod OOP
- Partial and Currying
- Regex




# Obnavljanje

- Iteratori
  - Čemu služe
  - Kako se kreiraju
- Generatori
  - Definisanje
  - Čemu služe
  - `async/await`



# Decorator funkcije

- **Decorator** funkcija je funkcije koja dodaje neke dodatne osobine funkciji
- Ovaj tip funkcije uveden u ES7 i obavezno se preporučuje konverzija u stariju verziju ES standarda, npr. pomoću babel-a
- Dekorator izgleda ovako 
  - Kao argument uzima funkciju myFunc i proširuje joj funkcionalnost
- Primjena:
  - Memoization (keširanje rezultata funkcije)
  - Pri setovanju kontrole pristupa (access control)
  - Provjere izvršavanja trajanja funkcija
  - Neko logovanje
  - try/catch

```
@myDecorator  
function myFunc(){  
    return;  
}
```



# Object.defineProperty

- Sintaksa: `Object.defineProperty(obj, prop, descriptor)`. Za više informacija [pročitati](#)
- Primjer

```
1 const object1 = {};  
2  
3 Object.defineProperty(object1, 'property1', {  
4   value: 42,  
5   writable: false  
6 });  
7  
8 object1.property1 = 77;  
9 // throws an error in strict mode  
10  
11 console.log(object1.property1);  
12 // expected output: 42  
13
```



# Object.defineProperty, descriptor keys

- Treći argument u defineProperty predstavlja descriptor za property/key objekta na koji se kači. Može da sadrži data descriptors i accessor descriptors. Dijelev sledeće ključeve:
  - **configurable**: true ako i samo ako je tip ovog property descriptor-a može biti izmijenjen i ako property može biti obrisani iz odgovarajućeg objekta. Default je **false**
  - **enumerable**: true ako i samo ako se property prikazuje tokom enumeracije svojstava na odgovarajućem objektu, default **false**
- Data descriptori imaju i ove opcione ključeve:
  - **value**: vrijednost uvezana sa imenovanim svojstvom. Može da bude bilo koja validna JS vrijednost, default **undefined**
  - **writable**: true ako i samo ako vrijednost odgovarajućeg propertya može da se izmijeni, default **false**
- Accessor descriptori imaju i ove opcione ključeve:
  - **get**: funkcija koja se ponaša kao getter za property ili undefined ako nema gettera. Kada se pristupa propertyu, poziva se ova funkcija bez argumenata i sa this koji je setovan na objekat pomoću kog je pozvan property. Return vrijednost će se koristiti kao vrijednost ovog propertya. Default je **undefined**
  - **set**: funkcija koja se ponaša kao setter za property ili undefined ako nema setter. Kada se propertyu proslijedi vrijednost, poziva se ova funkcija sa jednim argumentom.



# Object.defineProperty

- Još par napomena:
  - Ako descriptor ne sadrži value, writeable, get i set ključeve, smatra se da je data descriptor
  - Ako descriptor sadži oboje value ili writeable i get ili set ključeve, javlja se greška
- Imajte na umu da ovi atributi nisu obavezno descriptori sopstvenih svojstva. Naslijeđena svojstva će biti takođe razmotrena. Da biste bili sigurni da su default vrijednosti sačuvane, možete da odradite freeze Object.prototype i definišete sve opcije eksplicitno ili da pointer za proto setujete na null sa Object.create(null)
- Primjeri, code-1



# Decorator funkcija, nastavak(1)

- Sintaksa za kačenje dekoratora na metod:

`function myDecorator(target, key, descriptor)`

- target - objekat na koji se kači dekorator
  - key - property objekta (metod) čija se funkcionalnost unapređuje
  - descriptor - objekat kako definiše kako metod treba da se ponaša
- Primjer, code-2
- Primjer, code-3





# Memoize funkcija

- Služi za čuvanje među-rezultata funkcije.
  - Keširamo rezultate funkcije
- Čuva rezultate funkcija koje se pozivaju sa istim argumentima
- Pogledajmo jedan primjer (code-4)



# ES6

- Pogledajmo zajedno ovaj članak:
  - [Link](#)
- Kako konvertujemo novi kod u stari tako da bi ga ostali pretraživači podržali?
  - Babel
  - Da li to moramo ručno da radimo
    - Webpack
    - Gulp



# Composition

- Koncept klasičnog nasleđivanja (inheritance) ima mane
- Da bi ste dobro razumjeli zašto je nekad bolje koristiti Composition u odnosu na klasično nasleđivanje:
  - Pogledati [video](#)
  - Pročitati [članak](#)
- Inače, interesantan youtube kanal [fun fun function](#)



# Partial and Currying

- Za currying pročitati [članak 1](#), [članak 2](#) i [članak 3](#)
- Predlažem da pogledate:
  - [Video 1](#)
  - [Video 2](#)
  - [Video 3](#)



# Regex

- Ili regularni izrazi, koriste se dosta u praksi, a najčešće za validaciju unijetih podataka, replace dijelova stringa
- Kod JSa regularni izrazi su također objekti, kao i sve ostalo
- Kreiraju se na dva načina
  - `var re = /ab+c/;`
  - `var re = new RegExp('ab+c');`
- Samo smo ih kratko pomenuli ovdje, predlažem da pročitate [članak](#), a predlažem da pogledate:
  - [Video 1](#)
  - [Video 2](#)
  - [Video 3](#)



# Napredno

- Više o dekoratorima pogledati video:
  - [Link](#)
- Dodatno pročitati
  - [Članak](#)



Pitanja