Head First JavaScript

Variables

Varijable su case-sensitive; npr. dollar ≠ Dollar.
Varijable mogu započeti sa slovom, _ ili $.
Varijable koje započinju sa $ su najčešće rezervirane za JavaScript library.

Globalne varijable se uvijek zapisuju na početku file-a i njima se može bilogdje pristupiti. Dok lokalne, su dostupne samo unutar scope-a.

Ako se nekoj novoj varijabli unutar funkcije, ne doda **var** keyword, onda se ona ponaša kao lokalna iako je tek u funkciji instancira. Njoj se onda može pristupiti svugdje!

Varijable kojima se inicijalizira vrijednost, dodijeli im se **undefined** vrijednost.

```
var globalVar = "Globalna varijabla";
function funDeklaracija(){
  var globalVar = "Lokalna varijabla";

  function inner(){
    return globalVar;
  }
  return inner;
}
```

```
var innerFunc = funDeklaracija();
var ispis = innerFunc();
console.log(ispis); //Ispisat ce se "Lokalna varijabla"
```

💡 **Lexical scope -** JavaScript's rules for scoping are based purely on the structure of your code

---

Closure - funkcija zajedno sa referenciranim enviroment-om

Functions

Funkcija se sastoji od keyword-a function, nazivaFunkcije i parametra.

Kada se učitava web-stranica, skripta se dva put očitava: prvi put pronalazi i pamti funkcije, drugi put se učitava aktualni kod.

Ako se neki parametar funkcije, naziva isto kao i globalna varijabla. Onda se ona ponaša kao lokalna varijable, pa ne učitava vrijednosti, niti ih dodijeljuje toj globalnoj varijabli, već se ponaša kao lokalna.

Funkcije mogu također vraćati vrijednost pomoću return keyword-a.

Internet preglednici prvo pregledavaju kod za funkcijske deklaracije, a onda tek učitava varijable/kod.

Važno je dobro razumijeti razliku između funkcije deklaracije i funkcijskog izraza.

Funkcije deklaracije, kao što je prethodno rečeno, tjeraju *browser* da radi u pozadini. Deklaracije se definiraju prije ostalog koda.

Function expression stvara funkciju koja rezultira kao referencu i na nama je da koristimo ili ne.
Ovakve funkcije rezultiraju tako što se stvara manje koda, lakše je za održavat, fleksibilnije i manje programiranja. Function expression se definira/čita kad i varijable, odnosno in real time.

```
var primjer = function(parametar){
  var a = 1;
```

```
  return a + parametar;
}
```

Funkcije treba tretirati kao varijable!

First Class Functions

> Vrijednost koja može biti tretirana kao svaka druga vrijednost u programskom jeziku, uključujući i mogućnost da se dodijeli varijabli, proslijedi kao argument i return-a od funkcije.

- **You can assign functions to variables**
- **You can pass functions to functions**
- **You can return functions from functions**

!! **Hoisting -** function declarations create functions that are defined everywhere in the code!

Constructors

Konstruktori su način za stvaranje puno objekta istog tipa.

```
function Dog(name, breed, weight) {
    this.name = name;
    this.breed = breed;
    this.weight = weight;

    this.bark = function(){
        console.log("Woof woof!");
    }
}
var fido = new Dog("Fido", "Golden retriever", 20); //Stvara se novi objekt tipa Dog
```

Unutar konstruktora, nemoraju se nužno samo deklarirati varijable objekta, već mogu i metode, na isti način kao i varijable.

Prototypes

Konstruktore možemo smatratati kao funkcije, ali s obzirom da znamo da se radi o JavaScript, funkcije su i objekti. To nam omogućava da pristupimo **prototype** stavci, koja nam dopušta da stvaramo prototipe, kako bi drugi objekti ga mogli naslijeđivat. To želimo jer nam to omogućava efikasno korištenje memorije.

> *Kada bismo nastavili raditi objekte tipa **Dog** na prethodno zapisani način, tada bi se za svaki objekt stvarala npr. referenca na funkciju bark(). Drugim riječima, jedan te isti kod, zapisao bi se u memoriju za svaki objekt tipa Dog.*

```
function Dog(name, breed, weight) {  //Ovo ce svaki pas imati specificno za sebe
    this.name = name;
    this.breed = breed;
    this.weight = weight;
}
Dog.prototype.bark = function (){ //Ovo ce svaki pas naslijediti
  console.log("Woof!");
}
```

Treba spomenuti, da prototipske funkcije/stavke, možemo override-at.

Kako ispitati je li funckija/stavka u instanci ili u prototipu?
Možemo koristiti **hasOwnProperty** koju svaki objekt naslijeđuje i koja vraća true ili false, ovisno je li property definiran u objektu.

```
//Kako override-at funkcije, poput toString()?

Dog.prototype.toString() = function(){
  return this.name;
}
```

Arrays

Arrays se deklariraju na način da im se dodijeli **[ ]** koje onda nazivamo array literal. Koji ima **length 0.**
Array se također može deklarirati na **new Array(veličina);** koji

Mogu spremati bilokoji tip ili kombinaciju tipova, ali onda moramo znati smisao array-a.

Arrayu se dodijeljuju vrijednosti na dva načina:

1. mojArray[1] = 15;
2. mojArray.push(15);

Ako dodijeljujemo vrijednosti na 1. način, onda moramo paziti da ne preskačemo mjesta jer onda dolazi do **sparse array.**

Sparse array je array koji ima samo par vrijednosti dodijeljeno, a prostor između njih je prazan, odnosno dodijeljuje se vrijednost **undefined.**

Objects

```
var object = {
  name: "ball",
  isRound: true,
  width: 15
};
```

Objekt se kreira na način da mu se dodijeli **{ }.**
Objekt kreiran na taj način, naziva se **object literal** i proslijeđuje Object objekt. *To možemo ispitati sa instanceof.*

Da pristupimo svojstvu objekta trebamo samo dodati točku nakon naziva objekta ili staviti naziv svojstva u **[”naziv”].**

```
var name = object.name;
var width = object["width"];
```

Da bi obrisali svojstvo u potpunosti, kao prefix svojstvu objekta dodajemo keyword **delete.**

```
delete object.width;
```

Ako pristupamo nekom svojstvu koji ne postoji u objektu, kao rezultat dobit ćemo **undefined**.
Osim svojstva, objekti su korisni jer mogu imati i metode, kojima se pristupa kao svojstvo. Jedina razlika je što se na kraju svojstva dodaje **()**.

```
var car = {
  name: "Cadillac",
  mileage: 12500,
  started: false,
  start: function(){
    this.started = true;
};
```

Varijable kojima su dodijeli objekt, ne stvaraju kopiju već se stvara referenca na objekt!
Drugim riječima, ako se varijabli A dodijeli objekt i zatim se doda neko svojstvo. To svojstvo će se direktno mijenjati na objekt, a ne samo na varijablu A.

Kada pozivamo **typeof** na neku našu klasu koju smo definirali našim konstruktorom, tada će uvijek rezultat od typeof biti object, sa kojim nažalost nemožemo saznati o kojemu objektu se radi.

Naime, postoji work-around. Keyword **instanceof** može ispitati da li se radi o našoj klasi koju smo naveli. To je jedna od stvari koji keyword new radi u pozadini stvaranja objekta. Funkcionira na način da uspoređuje konstruktore.

```
if(ferrari instanceof Car){
  //code
}
```

Built-in objects

Object	Korištenjem Object konstruktora možemo kreirati objekte.
RegExp	Koristimo ovaj konstruktor za kreiranje regular expression objekte, koji nam dopuštaju da pretražujemo za patterne, čak i one kompleksne, u tekstu.

Math	Ovaj objekt ima svojstva i metode za izvršavanje matematičkih funkcija.
Error	Kreira standardne error objekte koji su korisni za hvatanje greški u kodu.

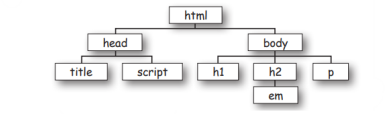
Kako saznati za sva svojstva nekog objekta?

Postoji tako zvana **for in** petlja, koja iterira po svim svojstvima objekta.

```
for (var prop in car) {
 console.log(prop + ": " + car[prop]);
}
```

DOM

DOM ili Document object model koji se stvara kada se stranica učitava. DOM je **tree** oblika koji nam omogućuje da pristupamo podacima koji su na stranici.



Kod koji mijenja DOM, **mora se izvršavati tek nakon što učitana stranica jer inače neće raditi!**

Kada želimo mijenjati neki HTML text ili nešto odmah pri učitavanju stranice, važno je da to funkciju dodijelimo **window.onload** kako bi se ta funkcija izvršila odmah čim se stranica učita. Ako to ne napravimo, taj dio koda se neće izvršiti što kasnije može stvarati probleme.

Types, Equality, Conversion

undefined je vrijednost dodijeljena varijabli koja još nema vrijednost.

typeof je operator koji vraća tip kojega je ta varijabla.

```
var subject = "Just a string";
var subjectType = typeof subject;
console.log(subjectType);
```

Razlika između **undefined i null** je ta što null se vraća samo kada se traži neki objekt koji ne postoji, a undefined kada objekt nije inicijaliziran.

NaN (Not A Number) je vrijednost koja se dodijeli prilikom nedozvoljenih ili nepoznatih matematičkih operacija.
NaN je za svaki različit odnosno, NaN ≠ Nan.
isNan je funkcija koja provjerava je li rezultat NaN.

```
var a = 0 / 0;
var b = 'b' * 5;
if(b == NaN) //false
if(isNaN(b)) //true
"true" == true; //false
```

```
undefined == null; //true
1 == ""; //"" => 0 --> 1 == 0 => false
```

Falsey, Truthy

Samo 5 stvari su falsey, ostale su truthy!:

- **undefined is falsey**
- **null is falsey**
- **0 is falsey**
- **The empty string is falsey**
- **NaN is falsey**

Events

Skoro svakom eventu se kao argument šalje **target,** odnosno objekt na kojem je preplaćen taj event. *Npr. ako na svim slikama se pretplatimo na onclick event, onda on kao argument šalje sliku kao target.*
Javascript nam omogućava da zanemarujemo parametre ako ih nećemo koristiti.

```
var image =
   document.getElementById("imgOne");
image.onclick = prikazi;

function prikazi(eventObj){
  alert(eventObj.target);
}
```

Javascript nam omogu

Math

Math.random() - daje vrijednosti između 0 i 1, ali nikada 1. To znači da ako kažemo Math.random() * 5, dobit ćemo broj između 0 i 4.99, ali nikada 5.

Math.floor() - zaokružuje na glavu broja. 4.99 će zaokružit na 4.