



JavaScript

Drugi dio



Pregled

- Conditional Statements
 - if/else, switch, ternarni operator
- Coercion
- Operator == i ===, tj. != i !==
- Petlje (loops)
 - while
 - for
 - break i continue
- Funkcije (functions), uvod
- Objekti, uvod
- Zadaci



Par bitnih stavki

- Bitne stavke za dobru saradnju
 1. Analitičko rješavanje i pristup problemu je ključna stvar
 2. Tehnička komunikacija jako bitna (implementirati kod na osnovu nečijeg objašnjenja, zato ćemo se dosta fokusirati i na teorijski dio JSa)
 3. Dobri pristupi i praksa je takođe bitan faktor (debugging, struktura koda, strpljenje i oslanjanje na dokumentaciju)
 4. Neverbalna komunikacija (pristojna i prijatna komunikacija)
 5. Iskustvo



Obnavljanje

- Čemu služi `console.log` i kako otvaramo konzolu pretraživača?
- Koja je razlika između deklaracije i inicijalizacije?
- Šta smo rekli, koja je razlika između tri načina deklarisanja varijabli **var**, **let** i **const**?
- Dobra praksa za imenovanja varijabli.
- Zašto dodajemo `script` element na samom dnu `body` elementa?
- Na koji način provjeravamo tip podatka?
- Kako deklariramo niz, a kako objekat?
- Šta će da se prikaže u konzoli ako odradimo `console.log(3 < 2 < 1)` i zašto?
 - Coercion i operator precedence



Conditional Statements

- Slično kao i kod drugih programskih jezika, tri vrste uslova
 - if
 - if...else
 - if...else if... else if.....else

```
if (condition1) {  
    block of code to be executed if condition1 is true  
} else if (condition2) {  
    block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    block of code to be executed if the condition1 is false and condition2 is false  
}
```



Conditional Statements, switch

- Expression se računa jednom, vrijednost izraza se poredi sa vrijednostima svakog slučaja
- Break može drastično da sačuva performanse
- Koristi === poređenje

Primjer

```
switch(expression) {  
    case x:  
        code block  
        break;  
    case y:  
        code block  
        break;  
    default:  
        code block  
}
```

```
switch (new Date().getDay()) {  
    case 4:  
    case 5:  
        text = "Soon it is Weekend";  
        break;  
    case 0:  
    case 6:  
        text = "It is Weekend";  
        break;  
    default:  
        text = "Looking forward to the Weekend";  
}
```



Conditional Statements, ternary

- Koristi se nekad za jednostavne provjere
 - `condition ? expr1 : expr2`
 - Ako condition vrati true, izvršiće se `expr1`, inače `expr2`
- Primjeri

```
var elvisLives = Math.PI > 4 ? 'Yep' : 'Nope';
```

- Pogledajte [članak](#) za više informacija

```
var stop = false, age = 23;

age > 18 ? (
    alert('OK, you can go.'),
    location.assign('continue.html')
) : (
    stop = true,
    alert('Sorry, you are much too young!')
);
```



Coercion, uvod

- Konverzija iz jednog tipa podatka u drugi
- U JS postoje tri tipa konverzije i to:
 - U string
 - U broj
 - U boolean
- Vrlo prisutno u JSu
- Postoje dva Coercion tipa:
 - Explicit Coercion
 - Implicit Coercion

Coercion, ToString

- Konverzija u String. Za eksplicitnu konverziju koristi se `value.toString()` metod ili `String(value)`

<code>null</code>	<code>"null"</code>	<code>[]</code>	<code>""</code>
<code>undefined</code>	<code>"undefined"</code>	<code>[1,2,3]</code>	<code>"1,2,3"</code>
<code>true</code>	<code>"true"</code>	<code>[null,undefined]</code>	<code>","</code>
<code>false</code>	<code>"false"</code>	<code>[[[],[],[]],[]]</code>	<code>",""</code>
<code>3.14159</code>	<code>"3.14159"</code>	<code>[,...]</code>	<code>"..."</code>
<code>0</code>	<code>"0"</code>	<code>{}</code>	<code>"[object Object]"</code>
<code>-0</code>	<code>"0"</code>	<code>{a:2}</code>	<code>"[object Object]"</code>

Coercion, ToNumber

- Konverzija u broj. Za eksplicitnu konverziju koristi se `parseInt(val)` ili `parseFloat(val)` ili `Number(val)`.

<code>""</code>	<code>0</code>	<code>[]</code>	<code>0</code>
<code>"0"</code>	<code>0</code>	<code>["0"]</code>	<code>0</code>
<code>"-0"</code>	<code>-0</code>	<code>["-0"]</code>	<code>-0</code>
<code>" 009 "</code>	<code>9</code>	<code>[null]</code>	<code>0</code>
<code>"3.14159"</code>	<code>3.14159</code>	<code>[undefined]</code>	<code>0</code>
<code>"0."</code>	<code>0</code>	<code>[1,2,3]</code>	<code>NaN</code>
<code>".0"</code>	<code>0</code>	<code>[[[]]]</code>	<code>0</code>
<code>" "</code>	<code>NaN</code>	<code>true</code>	<code>1</code>
<code>"."</code>	<code>NaN</code>	<code>null</code>	<code>0</code>
<code>"0xaf"</code>	<code>175</code>	<code>undefined</code>	<code>NaN</code>



Coercion, ToBoolean

- Konverzija u Boolean tip. Za eksplicitnu konverziju koristi se Boolean(val).

Falsy	Truthy
""	"foo"
0, +0, -0	23
null	{ a:1 }
NaN	[1,3]
false	true
undefined	function(){..}
	...



Explicit Coercion

- Kada je jasno iz koda da se radi konverzija (coercion) onda govorimo o eksplicitnoj
- Pogledajmo par primjera (code-1)
- Do sad smo vidjeli samo explicit coercion primjere, u nastavku ćemo vidjeti primjere za implicit coercion

Implicit Coercion

- Javlja se kao neželjeni efekat neke druge operacije
- Pogledajmo primjere (code-2)
- Nikad ne koristite
 - `== true`
 - `== false`

```
1 "0" == null;           // false
2 "0" == undefined;      // false
3 "0" == false;          // true -- UH OH!
4 "0" == NaN;            // false
5 "0" == 0;              // true
6 "0" == "";             // false
7
8 false == null;         // false
9 false == undefined;    // false
10 false == NaN;         // false
11 false == 0;           // true -- UH OH!
12 false == "";          // true -- UH OH!
13 false == [];          // true -- UH OH!
14 false == {};          // false
15
16 "" == null;           // false
17 "" == undefined;      // false
18 "" == NaN;            // false
19 "" == 0;              // true -- UH OH!
20 "" == [];             // true -- UH OH!
21 "" == {};             // false
22
23 0 == null;            // false
24 0 == undefined;       // false
25 0 == NaN;             // false
26 0 == [];              // true -- UH OH!
27 0 == {};              // false
```

```
1 "0" == false;         // true -- UH OH!
2 false == 0;           // true -- UH OH!
3 false == "";          // true -- UH OH!
4 false == [];          // true -- UH OH!
5 "" == 0;              // true -- UH OH!
6 "" == [];             // true -- UH OH!
7 0 == [];              // true -- UH OH!
```

```
42 == "43";             // false
"foo" == 42;            // false
"true" == true;         // false

42 == "42";             // true
"foo" == [ "foo" ];     // true
```

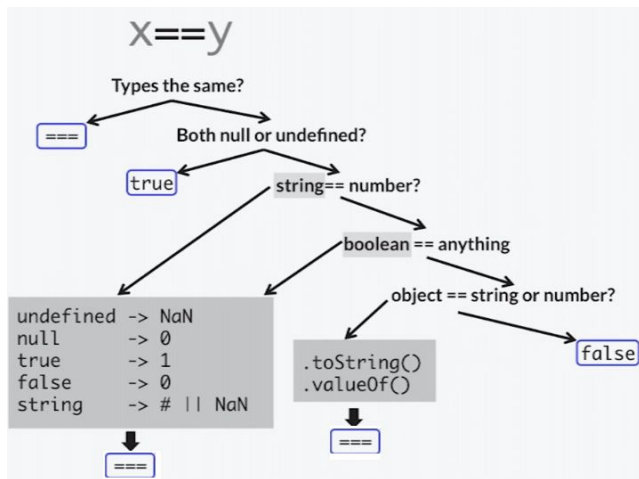


Coercion, dodatni materijal

- Za više detalja pročitati [članak](#)
- Još interesantnih primjera možete [ovdje](#) pogledati
- Za još više informacija o Coercion, pogledati i ovaj [članak](#)
- Da li vam je sada jasan primjer `console.log(3 < 2 < 1)`
 - Coercion i operator precedence

Operatori == i ===, tj. != i !==

- Kad kažemo da sa == JS radi poređenje vrijednosti lijeve i desne strane, dok sa === radi poređenje vrijednosti i tipa, nismo skroz precizni
- Kad koristimo == omogućavamo coercion, sa === to onemogućavamo





Algoritam za ==

1. If `Type(x)` is the same as `Type(y)`, then
 - a. Return the result of performing `Strict Equality Comparison x === y`.
2. If `x` is **null** and `y` is **undefined**, return **true**.
3. If `x` is **undefined** and `y` is **null**, return **true**.
4. If `Type(x)` is Number and `Type(y)` is String, return the result of the comparison `x == ToNumber(y)`.
5. If `Type(x)` is String and `Type(y)` is Number, return the result of the comparison `ToNumber(x) == y`.
6. If `Type(x)` is Boolean, return the result of the comparison `ToNumber(x) == y`.
7. If `Type(y)` is Boolean, return the result of the comparison `x == ToNumber(y)`.
8. If `Type(x)` is either String, Number, or Symbol and `Type(y)` is Object, return the result of the comparison `x == ToPrimitive(y)`.
9. If `Type(x)` is Object and `Type(y)` is either String, Number, or Symbol, return the result of the comparison `ToPrimitive(x) == y`.
10. Return **false**.

- Primjeri (code-3)

- Not equal
- Loose equality
Often gives "false" positives like "1" is true; [] is "0"
- Strict equality
Mostly evaluates as one would expect.



Petlje (Loops)

- Slično kao kod većine programskih jezika, izvršava se sve dok je određeni uslov ispunjen

While

```
while (condition) {  
    code block to be executed  
}
```

For

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

- **for... in** i **for... of** ćemo vidjeti naknadno vidjeti
- Sa **break** izlazimo iz petlje, a sa **continue** preskačemo određeni korak u petlji
- Primjeri (code-4)



Funkcije (Functions), uvod

- Slično kao i kod ostalih programskih jezika ali imaju dosta više dodatnih osobina u odnosu na ostale jezike. O svim ovim osobinama ćemo govoriti kako budemo prolazili kroz gradivo, a neke od njih su:
 - Parametar funkcije može da bude funkcija (callback)
 - Funkcija može da vrati funkciju (higher order functions, closure)
 - U varijablu se može smjestiti funkcija (function expression)
 - Funkcija može biti odmah pozvana čim se kreira (IIFE)
 - Vrijednost property-a objekta može da bude funkcija (metod)
 - Element niza može biti funkcija
 - Funkcija može da bude property
 - Arguments objekat
 - Anonimne funkcije
- Zašto su sva ova svojstva moguća ?
- Šta je razlika između parametra i argumenta funkcije?



Funkcije, uvod, deklaracija


- Deklaracija funkcije sastoji se od riječi **function**, imena (opciono), **parametara** i **bloka** gdje se definiše kod. Najčešće sadrže i **return**

```
function name(parameter1, parameter2, parameter3) {  
    code to be executed  
}
```

- Funkcija može biti pozvana kad je neko/nešto pozove
 - Kada se desi neki događaj (npr. klik na neko dugme)
 - Kada se pozove ručno sa ()
 - Automatsko pozivanje (self invoked, IIF)
- **Parametri** funkcije se ponašaju kao lokalne varijable unutar funkcije, definišu se kao ime (bez var, const ili let)
- **Function expression** se pominje kada funkciju dodijelimo promjenljivoj
- Primjeri (code-5)

Objekti, uvod

- Složen tip podatka (sastoji se od više primitivnih tipova) i predstavlja abstrakciju nekog realnog entiteta
- Objekti se sastoje od svojstava koja ih opisuju i metoda pomoću kojih obavljamo neke akcije nad objektima

Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

- Kako biste napisali JS kod za jedan ovakav model



Objekti, nastavak

- Objekti su takođe varijable koje mogu sadržati više vrijednosti
 - key: value
 - Ovi parovi čine svojstva (properties) objekta
- **Svojstvima** objekta je moguće pristupiti na dva načina
 - `object.propertyName`
 - `object["propertyName"]`
 - Što mislite koja je razlika između ova dva načina za pristupanja vrijednostima svojstava?
- **Metode** su akcije koja mogu biti obavljene nad objektom
 - Takođe, kod JSa, funkcija koja se čuva kao property objekta
 - `object.methodName()` za pristupanje metodi objekta
 - Bez () šta dobijamo?



Objekti, primjer

- This je specijalni objekat i odnosi se na kreatora “vlasnika” funkcije (person u ovom slučaju)
- Puno više ćemo govoriti o this kada budemo pričali o OOP u JSu

```
var person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```



Objekti i for...in

- Sintaksa `for (variable in object) { ... }`
 - variable je promjenljivo
 - Ovom petljom prolazimo kroz sva svojstva objekta
 - Isprobajte da šampate sve vrijednosti svojstava iz prethodnog primjera



Zadaci

- Napisati funkciju koja uklanja sve falsy vrijednosti iz niza
 - Koristiti splice ili napraviti novi niz koji sadrži truthy vrijednosti iz starog niza, koristi `arr.push(elem)` za dodavanje novog elementa u niz
- Napisati funkciju koja vraće najmanji indeks na čijoj poziciji treba da se doda vrijednost (drugi argument) u niz (prvi argument) koji treba da ostane sortiran
 - `fun([1,2,3,4], 1.5) -> 1`
 - `fun([20,2,3], 19) -> 2` jer kad se niz sortira (2,3,20) element 19 može da se doda između 3 i 20, tj. na indeksu 2
- Napisati funkciju koja razbija niz (prvi argument) na grupe dužine zadate kao drugi argument funkcije i vraća novi 2D niz.
 - `fun([1,2,3,4], 3) -> [[1, 2, 3], [2, 3, 4]]`
- Napisati program koji za 10 elemenata niza (brojevi od 1 do 10) generiše parove tako da se niti jedan od elemenata koji je već u paru ne ponovi više ni u jednom paru. Npr. (1,4), (2,3), (5,8) su ok, ali (1,6) nije jer je 1 već u paru sa 4, (2,5) takođe nije jer je 2 u paru sa 3, a 5 u paru sa 8.
 - Koristiti `Math.floor()` - zaokružuje na donje cijelo i `Math.random()` - vraće decimilan broj između 0 i 1 (uključujući 0, ne uključujući 1)
 - `Math.random() * 10` - vraće brojeve od 0 do 9,99999....
 - Može vam pomoći i `array.splice(element_index, 1)` u kombinaciji sa `arr.indexOf(item)` [ako ne nađe item u arr vrati -1]
 - `["a","b","c"].splice(2, 1) => ["a","c"]`



Pitanja