# Zapiski iz pouka Osnove programiranja

Matej Blagšič

21. december 2017

# Kazalo

# JavaScript

## 1 Osnovno

JavaScript je jezik, ki je integriran v neko okolje, kot je npr. HTML.

```
<script>
<!-- KODA -->
</script>
```

Obmmočje delovanja js kode je:

```
<script src = "/..pot do datoteke/imedatoteke.js"><\script>
```

Operator **console.log();** izpiše vneseno vrednost/spremenljivko v konzolo, ki je dostopna v brskalniku z ukazom Ctrl + Shift + J

# 2 PRAVILA

- angleške črke
- desetiška števila
- podčrtaj (_) ločuje besede
- začetek stavka ne sme biti število
- loči velike in male črke(je case-sensitive)
- nedovoljena uporaba razerviranih izrazov/funkcij(&, = ...)
- "navednice"označujejo dobesedno navajanje/znak
- primer:

```
console.log(a); <!-- Izpiše vrednost spremenljivke -->
console.log("a"); <!-- Izpiše znak a -->
```

# 3 Razhroščevanje

V konzoli brskalnika lahko najdemo tudi orodje za razhroščevanje(debugging). V temu orodju se označi vrstica kode, ki jo želimo opazovati, kako se izvaja. Ob strani imamo tudi predalčnik "watch", kjer lahko nastavimo, katero spremenljivko želimo opazovati("add expression"). Nato osvežimo stran in se nam program ustavi na izbrani vrstici. Lahko najdemo ikone za ustavitev programa("||"), zraven pa štep"ikono. Ta nam izvede program en korak naprej. Tako lahko opazujemo, kako program deluje in kako se naše spremenljivke spreminjajo.

# 4 Spremenljivke

```
var = a;
```

Z enačajem se definira vrednost ali izraz spremenljivki na desni strani(var). Definicija se **VEDNO** konča s podpičjem.

Operator "="definira spremenljivko in ji priredi neko vrednost ali izraz.

```
a = 31;
```

Spremenljivki a priredimo vrednost 31.

## 4.1 Tipi spremenljivk

Operator **typeof()** vrne tip spremenljivke/vrednosti.

### Številski(number)

VREDNOSTI: 41, 2.15, Nan, infinity
primer:

```
console.log(typeof (13)) <!-- v konzoli se nam izpiše "number"-->
```

### Boolov(boolean)

VREDNOSTI: TRUE, FALSE primer:

```
console.log(typeof TRUE) <-- v konzoli se nam izpiše "boolean"-->
```

### Znakovni niz(string)

VREDNOST: "jabolko" –> string primer:

```
console.log(typeof ("jabolko")) <!-- v konzoli nam izpiše "string"-->
```

**Določen tip(undefined)**

VREDNOST: undefined

## 4.2 Pretvorbe tipov

```
Number(); --> pretvori v številko
Boolean(); --> pretvori v boolean
String(); --> pretvori v znakovni niz
```

**Boolean -> number**

`FALSE` –> 0

`TRUE` –> 1

**String -> number**

**"42"** –> **42** če uporabljamo operatorje × ali ÷ ali −

**"5x"** –> **NaN** ni število, ker se pretvori tudi string x, ki ni število

**"5"+5** –> **"55"** če uporabljamo operator +

Operator + pričakuje enak tip spremenljivke na obeh straneh, zato pretvori number v string in ju zlepi kot dva stringa.

**Number -> string**

**42** –> "42"

# 5 Izrazi

Izraz je del kode, ki razreši vrednost. Lahko vrednost dodeli spremenljivki ali pa jo ima sam.

Na primer: Izraz `x = 7` dodeli vrednost 7 spremenljivki x.

Pomembna lastnost izrazov je PREDNOST ali *precedence* operatorjev med seboj. To pomeni, da program bere operatorje glede na njihovo prioriteto oz. prednost.

Na primer: V izrazu `a + b * c` ima operator * prednost pred operatorjem +, zato program najprej zmnoži števili b in c in nato sešteje vsoto s številom a.

Operatorji istega tipa/prednosti se pa izvajajo v vrstnem redu iz leve proti desni(*associativity*).

# 6 Operatorji

## 6.1 Aritmetični

| | |
|---|---|
| $+, -$ | Unarni/Binarni |
| $\times, \div$ | |
| % | Ostanek pri deljenju |

## 6.2 Primerjalni

Ti operatorji vračajo le $TRUE$ ali $FALSE$.

| | |
|---|---|
| $>, >=$ | Večje, večje ali enako(pomembno je zaporedje znakov!!) |
| $<, <=$ | Manjše, manjše ali enako |
| $==$ | Je enako |
| $!=$ | Ni enako |

Prav tako velja tudi: `"5" == 5` -> $TRUE$

## 6.3 Logični

Ti operatorji vračajo le $TRUE$ ali $FALSE$.

| | |
|---|---|
| && | Logični IN |
| \|\| | Logični ALI |
| ! | Negacija |
| = | Priredilni operator –> spremenljivki na levi strani priredi vrednost na desni strani. |

Prednost in red izvajanja:

**Aritmetični** –>

**Primerjalni** –>

**Logični** –>

**Priredilni** <–

Bljižnice(shorthands):

`x = x + izraz` => v spremenljivko x shranimo vsoto spremenljivke asdasdasdsadasdasdasdasd

`x += izraz` => okrajšan zgornji stavek

`x -+ izraz` spremenljivki x odštejemo vrednost izraza

`x++` => spremenljivki x se vrednost poveča za 1

`x--` spremenljivki x zmanjšamo vrednost za 1

## 6.4 Vejični

Ima še nižjo prioriteto, kot priredilni operator

## 6.5 Pogojni

```
pogoj ?  ce_je_true :  ce_je_false
```

V prvi del pred vprašajem se vnese pogoj in nato izraz, ki se prebere, če je pogoj izpolnjen, po dvopičjem pa sledi izraz, če pogoj ni izpolnjen.

Primer:

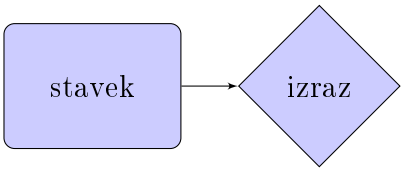
```
var x = 3;
var y = 0;
x > 2 ? y = 1 : y = 2;
console.log(y); --> konzola v tem primeru izpiše 1
```

## 6.6 Primerjalni

Črke: Črke primerja po abecedi, velike črke so pred malimi. Primerja se od prve do zadnje.

# 7 Stavki

## Diagram poteka




Obstaja prazen stavek, ki vsebuje le podpičje. Podpičja so neobvezna v Javascriptu, a jih je vseeno dobro uporabljati.

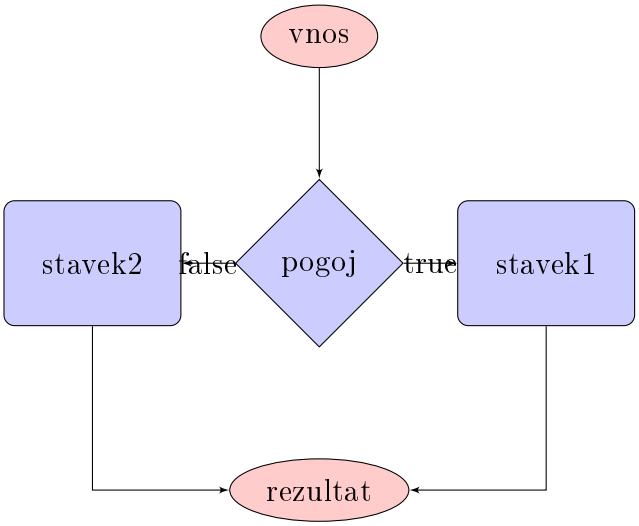
Primer stavka:

```
{
stavek1;
stavek2;
...
stavekN;
}
```

## 7.1 Stavek if/else

```
if (pogoj) stavek1 else stavek2
```

Po pogoju, sledi glede na rezultat pogoja **LE EN STAVEK!!** Za več kot en stavek, se uporabi zaviti oklepaj.


vnos
stavek2
false
pogoj
true
stavek1
rezultat

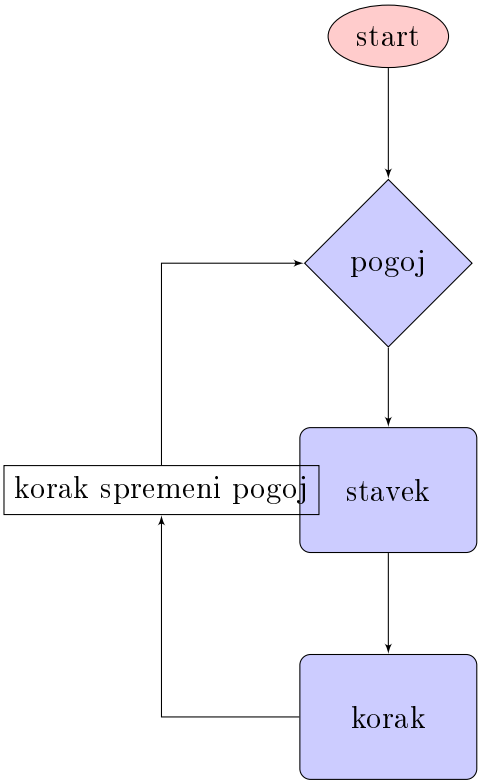
Primer stavkov: Računanje idealne teže s podanim podatkom o spolu in višini. Izvozi podatek idealne teže.

```
<script>

    var teza;
    var visina;
    var spol;

    spol = prompt("Vnesi spol (m/z)");

    if(spol == "m"){
        teza = 48 + (visina - 150) * 0.9;
    }
    else{
        teza = 43 + (visina - 150) * 0.7;
    }
    console.log("Tvoja idealna teža je" + teza + "kg");

</script>
```

## 7.2 Stavek for

```
for (start; pogoj; korak) stavek
```

For stavek izvaja določen stavek, dokler je pogoj uresničen. V for stavek se vnese štartni parameter. Ta se preveri v pogoju in se spreminja po koraku.




Npr: začnemo s številom i, katerega vrednost je 0. Če hočemo stavek ponoviti 4-krat, potem bomo povečevali naš i do števila 3(štetje se začne s številom 0) s korakom i(i++). To pomeni, da vsakič ko se bo izvedel stavek(več stavkov z {}) v for stavku, se po glede na nastavljen korak spremenil pogoj za 1 več(i++). Torej je v drugem(1-tem) krogu pogoj i = 1 in tako naprej.

```
for(var i = 0; i < 4; i++) console.log("Zdaj se izvajam v " + i +"-tem krogu");
```

## 7.3 Stavek while

```
while (pogoj) stavek;
```

## 7.4 Stavek do/while

```
do stavek while (pogoj);
```

Stavek se izvaja, ki je vnesen za operatorjem *do*, dokler je nek pogoj izpolnjen. Potem ko je pogoj v while zanki izpolnjen, se *do* zanka zaključi.

## 7.5 Switch

Switch stavek vzame določen izraz in ga primerja drugim vrednostim. S katerokoli se strinja, potem izvrši stavke, ki so asociirani s to vrednostjo.

```
switch (izraz){
    case vred1: stavki1 <-- če je pravilen, se vsi izvrsijo od kle naprej
    case vred2: stavki2

    case vredN: stavkiN
    default: privzetiStavki <-- stavek, ki se izvrsi ce se nobeden ne
}
```

Primer:

```
izraz === vred1 -> true -> stavki1
izraz === vred1 -> false -> izraz === vred2 ...
```

Če je vrednost *vre*1 enaka izrazu, potem se izvršjo vsi stavki naprej, vključno default. To je problem, ker če želimo, da je pogoj le en pravilni in da se izvrši le stavek zanj, potem moramo vnesti *break* po stavku pogoja.

# 8 Objekti

`Math.PI` V tem primeru, je Math objekt, ki ga mi vstavljamo v kodo. In "PI"je njegova lastnost/postopek oz. kateri del objekta Math želimo. V tem primeru nam objekt Math.PI vstavi število $\pi$

## 8.1 Postopki(methods)

To so deli Math objekta, npr: Math.abs();

**abs()** absolutna vrednost

**max()** največji od vnesenih(array) npr: `Math.max(array[ ])`

**pow()** potenca, npr: `Math.pow(3,2)` = $3^2$ = `9`

**random()** naključno število od `[0, 1)`

**sqrt()** kvadratni koren

**round()** klasično zaokroževanje

**ceil()** zaokroževane navzgor

**floor()** zaokroževanje navzdol

**sin(), cos(), tan()** kotne funkcije

## 8.2 Objektni tipi

`spr = new` [`tip-objekta`]`(parameter)` spr je nov objekt z imenom špr". Z ukazom new deklariramo vrsto oz. tip objekta. Tako dobimo:
`spr.lastonst() ali spr.postopek()`

### Array

`var a = new Array();` V oglatem oklepaju so elementi v žbirki", katere indeks se začne z 0. Torej podatek a[0] je prvi oz. nič-ti člen v zbirki. V našem primeru, je Array konstruktor, saj konstruira nov objekt.

`ali var a = [ ];` je samo definicija nove spremenljivke, ki postane array. To ni objekt.

`a[i] = i-ti člen array-a "a[ ]"`

Postopek
**indexOf()** nam izpiše želeni člen array-a. indexOf je postopek oz. lastnost of objekta array.
Primer: `var a = new Array(1,25,25,6,d,64); a.indexOf(4) = "d"`

Če postopek indexOf() ne najde želenega člena, bo izpisal -1.

### Date

`var danes = new Date();` S tem je spremenljivka danes datumski objekt

V konstruktor Date() lahko kot parameter vnesemo milisekunde. Tako nam izpiše ven datum in čas po vnešenih milisekundah po 1. Januar 1970, ko se je začel šteti **UNIX** čas računalnikov.

Primer: Če vnesemo v Date(milisekunde) 1000, potem nam izpiše 1. Jan. 1970 00:00:01, saj je to ena sekunda po začetku štetja. Ker pa ta program poženemo na računalnikih na različnih delih sveta, nam Javascript upošteva drugačen čas, tako da v sloveniji nam prišteje 1 uro.

Konstruktorji:

**Date()** izpiše datum in uro v določenem časovnem območju

**Date(milisekunda)** izpiše datum kot milisekunde po začetku štetja

**Date(leto, mesec, dan)** nastavi se datum

Postopki:

**getFullYear()** izpiše leto = 2017

**getMonth** izpiše mesec = 0(jan), 1(feb), 2(mar) ...

**getDate()** vrne datum = 1-31

**getHours(), getMinutes(), getSeconds()** izpiše uro, minuto, sekundo

**setFullYear(), setHours()** nastavi se ura, leto ...

**String**

`var besedilo =` To generira spremenljivko string
`var besedilo = new String()` To pa generira objekt String

Konstruktorji:

**String** (besedilo)

Postopki:

**indeksOf()** isce clen v stringu, clen je lahko tudi string

**length** vrne dolzino stringa

**charAt(i)** vrne znak na i-tem mestu

**substring(prvi, zadnji)** izlušči podstring med znaki prvi in **DO** zadnji

**toUpperCase()**

**toLowerCase()**

# 9 Funkcije

```
function myFunnction(par1, par2, ...)
{
    //koda
    return vrednost;
}
```

v `function(par1, par2, ...)` so par1, par2 formalni parametri(spremenlljikve)

Vhodni podatki so določeni podatki, ki jih funkcija potrebuje, da se izvede in da "izpljune"ven vrednost.

Funkcijo se kliče: `myFunction(par1, par2, ..., parN)` , kjer so par1, par2,... dejanski parametri.

Primer funkcije:

```
<head>
    <script>
    var crta = function(dolzina){
        var i;
        var c = "";
        for (i = 0; i < dolzina; i++){
            c += "-";
        }
        console.log(c);
    }
    </script>
</head>


<body>
    <script>
        crta(7);
        crta(12);
    </script>
</body>
```

Najprej se izvede funkcija `crta(7);` z vnešenim parametrom, ki je definiran v glavi strani. Nato pa se izvede `crta(12);`, ki pa kliče isto funkcijo, le da je vhodni parameter drugečen.

Lahko dodamo `return vrednost` je funkcija, ki vrne vrednost želene premenljivke. Tako lahko shranjujemo vrednosti funkcije v spremenljivko, da ne rabimo vstavljati dolge kode v našo glavno kodo.

```
var spr;
spr = myFunction(dejanski parametri);
```

v zgornjem primeru, nam funkcija `myFuntion()` vrne neko vrednost, ki jo vstavimo v spremenljivko.

Primer:
`myFunction()` vzame na primer neka števila, in vrne vrednost največjega.

`console.log(myFunction(var1, var2, var3, var4, var5,...));` v konzolo izpišemo vrednost, ki jo izpljune funkcija `myFunction` z vnešenimi podatki var1, var2, var3 ...

### Območja

**Globalno območje (Global scope)** je območje, kjer definiramo določene spremenljivke in veljajo za celoten **body** kode. Torej če je spremenljivka $x$ definirana v **body** kode, potem velja povsod v kodi.

**Lokalno območje (Local scope)** je območje, kot funkcija, kjer delujejo **lokalne spremenljivke**, ki delujejo le na temu območju in nikjer drugje. Torej, če je spremenljivka $x$ definirana v funkciji, potem njena vrednost ni enaka enaki spremenljivki $x$ v celotni kodi izven funkcije.

## 10 Klient

Uporabniški del Javascripta. So načini, s katerimi lahko združimo uporabniški del HTML in Javascript.

Funkcija `document.getElementById("ime");` lahko vnesemo objekt iz HTML v Javascript. Tako lahko ureamo HTML datoteke tako, da nam ni treba spreminjati HTML datoteke.

Prav tako lahko spremnijamo in pregledujemo lastnosti HTML objektov.
Funkcije

**document.getElementById()** Vnesemo objekt iz HTML v Javascript v spremenljivko

**<id_objekta>.innerHTML** nam vrne vrednost objekta v HTML datoteki.