JavaScript

Oh boi, this is where the fun begins. Az erre való Bibliát megtalálod ezen a linken.

Hogyan használjunk JavaScriptet?

Baromi sokféleképpen lehet, most megmutatom azt a fajtát amire nekünk szükségünk lesz. Szórakozásból és tesztelésképpen irogathatunk JS kódot a böngészőnk console-jában is. HTML-be kétféleképpen rakhatunk JS kódot, az egyik, hogy a `<body>` után egy:

```
<script src="scripts/myscript.js"></script>
```

tag-el behúzzuk a "scripts" mappából a myscript.js fájlunkat.

A másik pedig hogy közvetlen a `<script>` tagbe írunk JS kódot:

```
<script>
    var myname = "Earl";
    console.log("My name is "+myname);
</script>
```

De lehetőleg maradjunk annál hogy külön fájlba írjuk a kódjainkat.

Változók

Kezdjük az alapokkal.

Mi is az a változó? Minden programozási nyelvben vannak változók. Változóknak nevezzük azt, amiben valamilyen értéket vagy adatok tárolunk, legyen az szöveg, vagy szám stb. Háromféleképpen lehet változót létrehozni:

```
const sex = "Male";
var name = "Margit";
let age = 69;
```

Először koncentráljunk a `var`-al való létrehozásra. A `var` (mint variable) kulcsszóval hozunk létreváltozót, ezután megadjuk a nevét, majd egyenlőség jel után pedig megadjuk az értékét, utána ;-vel lezárjuk.

Hogyan nevezzük el a változókat?

Van rá egy jó kis leírás:

- A JavaScript változók tartalmazhatnak betűket, számokat, alsó vonást (_) és $ jelet.
- MINDIG betűvel kell hogy kezdődjenek
- Amúgy kezdődhet _ vagy $ jellel is, de ez nem ajánlott, hányadékul nézni.
- Case sensitive-ek a változó nevek, szóval a `var y` és `var Y` két különböző változónak számít

Változók létrehozása

JavaScript-ben a változóknak nincs fix adattipusok, szóval nem kell úgy megadni egy szám változók mint pl c#-ban hogy `int age = 2;`...

Itt elég `var age = 2;"-vel létrehozni egy szám változót, mert az egyenlőségjel után megadott érték fogja eldönteni a tipusát.

```
var age = 2;
var name = "Bela";
```

Számot úgy rakunk változóba, hogy csak simán leírjuk a számot, ha pedig szöveget szeretnénk belerakni, akkor az macskakörömmel tegyük, mint ahogy a fenti példában is látható.

Mi az a "let"?

A "let" abban különbözik a "var"-tól, hogy csak egyszer tudjuk létrehozni ugyan azzal a névvel, és block scopeja van.

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
// Ahogy látjuk itt is, nem tudtunk később véletlen még egy x nevű
változót létrehozni mert hibát kapunk
```

Viszont ha ugyan ezt próbáljuk "let" helyett "var"-al, akkor újra létretudjuk hozni a változót ugyan azzal a névvel, és felülírjuk a tartalmát. Tartalomban és felhasználásban egyébként nem tér el semmiben a "let" és a "var".

A másik fontos dolog amiben különbözik a "let", hogy ha egy blokkban létrehozzuk, az csak ott lesz használható, példa:

```
if (true) {
    let x = 2;
}

//az if-en kívül már nem fogjuk tudni elérni a let-et.
//ha "var"-t írnánk a "let" helyett, akkor eltudnánk érni
```

Mi az a "const"?

A "const" nagyon hasonlít a "let"-re, ugyan úgy block scopeja van, ugyan úgy csak egyszer tudjuk létrehozni, de emellé még az is hozzátartozik, hogy ha egyszer létrehozzuk és értéket adunk neki, többször

azt később nem tudjuk megváltoztatni! Példa:

```
const PI = 3.141592653589793;
PI = 3.14;      // Hibát fogunk erre kapni ha megakarjuk változtatni a már
megadott értékét
```

Operátorok

Operátorokkal kezeljük a változók értékeit. Sorban vegyük át őket:

Érték hozzáadó operátorok

"=" jel operátor

Sima egyenlőségjel operátorral tudunk adatot megadni egy változónak, példa:

```
var name = "Jani";
```

"+" jel operátor

Plusz jellel adunk össze dolgokat. Példa:

```
var number1 = 5;
var number2 = 3;
var total = number1 + number2; //a total értéke 8 lesz

var text1 = "My name is ";
var text2 = "Earl";
var total_text = text1 + text2; //"My name is Earl" lesz a total_text
értéke.

var everything = "Itt van minden amit létrehoztunk: " + text1 + text2 +
number1 + number2;
//"Itt van minden amit létrehoztunk: My name is Earl53" lesz az everything
változó tartalma.
//Miért lett 53 a vége és nem 8?
//Azért mert amikor szöveget kezdtünk el rakni a változóba, akkor már nem
szám lett a változó tipusa, hanem szöveg, és ha szöveget plusszozunk
egymás mellé, akkor azt csak simán egymás mellé rakja. Ha szöveg mellé
számot rakunk, akkor azt mellé rakja, mint egy szöveget.
```

"-" jel operátor

Minusz jellel vonunk ki számokat egymásból.

```
var number1 = 5;
var number2 = 3;
var total = number1 - number2; //a total értéke 2 lesz
```

"*" jel operátor

Csillag jellel szorzunk számokat.

```
var number1 = 5;
var number2 = 3;
var total = number1 * number2; //a total értéke 15 lesz
```

"/" jel operátor

Per jellel osztunk számokat.

```
var number1 = 5;
var number2 = 3;
var total = number1 / number2; //a total értéke 1.6666666666666667 lesz
```

"%" jel operátor

Százalék jellel maradékos osztást tudunk csinálni.

```
var number1 = 5;
var number2 = 3;
var total = number1 % number2; //a total értéke 2 lesz
```

++ és -- jel operátor

++ és -- jelekkel eggyel növelni vagy csökkenteni tudjuk a szám változó értékét.

```
var number1 = 1;
number1++;
console.log(number1); //most már azt fogod látni hogy 2 az értéke
```

Meglévő érték módosító operátorok

"+=" operátor

Ezzel a meglévő értékhez tudunk hozzáadni újabb értéket.

```
var number1 = 1;
number1 += 5;
console.log(number1); //most már azt fogod látni hogy 6 az értéke

var text1 = "Why are you";
text1 += " geh";
console.log(text1); //most már azt fogod látni hogy "Why are you geh" a
szöveg értéke
```

"-=" operátor

Ugyan az mint a hozzáadás, csak kivonás.

```
var number1 = 5;
number1 -= 2;
console.log(number1); //most már azt fogod látni hogy 3 az értéke
```

"*=" operátor

Szorozzuk a meglévő értéket.

```
var number1 = 5;
number1 *= 2;
console.log(number1); //most már azt fogod látni hogy 10 az értéke
```

"/=" operátor

Osztjuk a meglévő értéket.

```
var number1 = 5;
number1 /= 2;
console.log(number1); //most már azt fogod látni hogy 2.5 az értéke
```

"%=" operátor

Maradékosan osztjuk a meglévő értéket.

```
var number1 = 5;
number1 %= 2;
console.log(number1); //most már azt fogod látni hogy 1 az értéke
```

Összehasonlító operátorok

"==" operátor

Összehasonlító operátor. A két egyenlőségjel egyik és másik végén lévő értéket vagy változót összehasonlítja, és true vagy false értéket fog adni.

```
if (5 == 10) {
    console.log("igen");
} else {
    console.log("nem");
}

//Mint láthatjuk, 5 nem egyenlő 10-el, így nem fog lefutni az if függvényünk
```

"===" operátor

Ugyan az mint a == operátor, csak itt nem csak az értéket vizsgálja, hanem az adatnak a tipusát is.

```
if (5 == "5") {
    console.log("igen két egyenlőségjellel");
}

if (5 === "5") {
    console.log("igen háromegyenlőségjellel");
}

//Mint láthatjuk, két egyenlőség jellel látjuk a konzolba a feliratot
//Három egyenlőség jellel már nem, mert szám 5-t hasonlítunk össze szöveg 5-el, és mivel itt már eltér az adattipusa, nem teljesül true-ra a vizsgálat
```

"!=" operátor

Nem egyenlő!

```
if (5 != 10) {
    console.log("yes");
}

if ("Labda" != "Gömböc") {
    console.log("yesyes");
}
```

```
//Látni fogjuk a konzolba hogy kiírta a programunk hogy "yes", mivel 5 nem
egyenlő 10-el, és azt is hogy "yesyes" mivel a két szöveg nem ugyan az.
```

"!==" operátor

Nem egyenlő az érték, vagy nem egyezik a tipus.

```
if (5 !== "10") {
    console.log("yes");
}

//látni fogjuk a konzolba azt hogy "yes".
```

">" operátor

Nagyobb mint...

```
if (5 > 2) {
    console.log("yes");
}

//látni fogjuk hogy "yes", mivel 5 nagyobb mint 2
```

"<" operátor

Kissebb mint...

```
if (2 < 5) {
    console.log("yes");
}

//látni fogjuk hogy "yes", mivel 2 kissebb mint 5
```

">=" operátor

Nagyobb vagy egyenlő...

```
if (5 >= 2) {
    console.log("yes1");
}

if (5 >= 5) {
    console.log("yes2");
}
```

```
if (5 > 5) {
    console.log("yes3");
}

//Azt hogy "yes3" már nem fogjuk látni, mivel 5 nem nagyobb 5nél, de
felette azt hogy "yes2" látni fogjuk, mert 5 nagyobb VAGY egyenlő 5-el.
```

"<=" operátor

Kissebb vagy egyenlő...

```
if (2 <= 5) {
    console.log("yes1");
}

if (2 <= 2) {
    console.log("yes2");
}

if (2 < 2) {
    console.log("yes3");
}

//Szintén nem fogjuk látni azt hogy "yes3", mert 2 nem kissebb 2-nél,
viszont azt hogy "yes2" látni fogjuk, mert 2 kissebb VAGY egyenlő 2-vel.
```

Logikai operátorok

Mind tudjuk, pl egy `if ()` függvény akkor fog lefutni, amikor a zárójelbe "true" érték kerül. A felső operátorok is "true" vagy "false" értéket adnak vissza összehasonlításkor, viszont fontos még ismernünk a logikai operátorokat, hogy egyszerre több dolgot tudjunk összehasonlítani.

"&&" operátor

"ÉS" operátor, azaz a két oldalán levő dolgot nézi, és csak akkor "true" értéket, ha mindkét oldalon lévő dolog igaz.

```
if (5 > 2 && 2 != 3) {
    console.log("yes");
}
```

Itt most láthatjuk hogy "yes", mivel "ÉS" operátort használtunk arra, hogy 5 nagyobb mint 2 (true) és 2 nem egyenlő 3-al (true), azaz true és true az igaz, így lefut az "if" függvényünk.

"||" operátor

"VAGY" operátor, az akkor fog true-t adni, ha a két oldalán legalább az egyik dolog igaz.

```
if (5 > 2 || 2 == 3) {
    console.log("yes1");
}

if (5 < 2 || 2 == 3) {
    console.log("yes2");
}
```

Láthatjuk azt a konzolba hogy "yes1" mivel az 5 nagyobb mint 2 (true), viszont a 2 nem egyenlő 3-al (false), de mivel "VAGY" logikai operátort használtunk, elég volt hogy a kettőből 1 legyen true, így lefutott az "if"-ünk. A másik esetnél viszont egyik vizsgálatunk se lett igaz, így nem láthatjuk hogy "yes2" mivel mindkét oldalon "false" értéket kaptunk, így nem lett "true" a "VAGY" logikai operátorunk, így nem futott le az az "if".

Függvények és funkciók

"if" vizsgálat

Már a nevéből is láthatjuk hogy "HA" függvény, ami akkor fog lefutni, "HA" valami igaz. Elnézést hogy előtte már csomószor használtam az "if" vizsgálatot és csak most beszélek róla, de midlife crysis-om lett attól hogy előtte magyarázzam el ezt vagy utánna... Hát most jó lesz így is.

```
if (2 > 1) {
    console.log("yes");
}
```

Mint láthatjuk kiírta a konzol azt hogy "yes", mivel teljesült az "if" vizsgálat. Miért és hogyan? Az "if ()" vizsgálatunk úgy működik, hogy a zárójelbe "true" értéket vár el, hogy lefusson. Ugye most a példakódban egy vizsgálatot írtam bele, hogy 2 nagyobb-e mint 1, ami azt adja vissza hogy "true" így teljesül az "if" és a benne lévő kód lefut.

```
if (2 < 1) {
    console.log("yes");
} else {
    console.log("Hát nem");
}
```

A fenti példában azt láthatjuk, hogy hamis értéket adtam meg az "if"-ben, azaz 2 nem kissebb mint az 1, így "false" értéket kapott az "if", így ami az "if"-ben van benne kód, nem futott le, viszont ha megnézed ott egy "else" ág, onnan kiírta hogy "hát nem". Miért? Azért, mert ha "if else" függvényt írunk, akkor ha az "if"-ben beírt dolog nem teljesül, akkor az "else" ágban lévő kód fog lefutni.

```
var number1 = 5;
var number2 = 3;

if (number1 < number2) {
    console.log("number1 nem kissebb mint number2");
} else if (number1 == number2) {
    console.log("Hát nem egyenlő");
} else {
    console.log("egyik feltételnek se felelt meg");
}
```

Azt fogjuk látni hogy egyik feltétel se teljesült. Az "if"-et bolondíthatjuk még "else if" ággal is, hogy ha az első "if"-be nem felelne meg, utána írhatunk még további vizsgálatot, és ha az se felel meg, utána még egy else-t is írhatunk ha akarjuk, hogy mindenképpen történjen valami.

"switch" vizsgálat

Switch-case vizsgálatot akkor érdemes használni, amikor véletlenül írtunk egy ilyen "if else if" gányolmányt:

```
var name = "Jani";

if (name == "Béla") {
    console.log("Ő Béla");
} else if (name == "Jani") {
    console.log("Ő Jani");
} else if (name == "Zoli") {
    console.log("Ő Zoli");
} else if (name == "Árpád") {
    console.log("Ő Árpád");
} else if (name == "Pisti") {
    console.log("Ő Pisti");
} else {
    console.log("fuj baszki");
}
```

.....na most hogy visszatértem miután kezet mostam ezután a borzalom megírása után, megmutatnám nektek, hogy ilyen esetekben miért használjunk inkább switch-case-t. A fenti undormányt átírom szép switch-case-be:

```
var name = "Jani";

switch (name) {
    case "Béla":
        console.log("Ő Béla");
        break;
    case "Jani":
        console.log("Ő Jani");
        break;
```

```
    case "Zoli":
        console.log("Ő Zoli");
        break;
    case "Árpád":
        console.log("Ő Árpád");
        break;
    case "Pisti":
        console.log("Ő Pisti");
        break;
    default:
        console.log("na ez már nem fúj!");
}
```

Na most ha megpróbáljátok mindkettő kódot lefuttatni, ugyan azt a működést fogjátok tapasztalni, de ha megnézitek a kódot, ebben az esetben a switch-case mennyivel elegánsabb és átláthatóbb, nem?! Működése az, hogy a "switch (variable)" zárójelébe írjátok azt a változót, aminek az értékét akarjátok vizsgálni. A Switch-case olyan mintha minden megadott adattal "==" operátoros vizsgálatot végezne. A "case"-ekbe megtudjuk adni az értéket hogy mivel kell megfelelnie, ez lehet szöveg macskaköröm között, vagy csak simán szám hogy: `case 2:` és akkor a 2-es számot fogja vizsgálni. A legvégén a "default:" pedig olyan mint az "if else"-ben a sima "else" ág.

"for" ciklus

A "for" ciklust arra szoktuk használni, ha valamilyen műveletet, egy adott mennyiségszer kell lefuttatni. Az alábbi példakódban azt láthatod, hogy 0-tól 99-ig, azaz 100szor lefut ez a kód, és ahol éppen jár, annak a számát kiírja a konzolba, egymás után.

```
for (var i = 0; i < 100; i++) {
    console.log(i);
}
```

A felépítése, a következő. Nézzük meg a "for" ciklus utáni zárójelet: `var i = 0;` ez a kezdőértékünk, létrehozunk egy változót, általában "i" szokott a neve lenni, de amúgy lehet bármi. Ez jelenleg 0-tól indul, onnan kezdi a számlálást. `i < 100` ez pedig a feltétel, hogy meddig fut a ciklusunk, jelenleg megadtuk azt hogy amég "i" kissebb mint 100, addig fusson. Végül `i++`-t láthatunk a végén, itt pedig az történik, hogy amikor lefutott a ciklus, eggyel növeli az "i"-nek az értékét.

Ezután a "for" ciklusban láthatjuk azt, hogy én mindig kiiratom az aktuális értékét az "i" változónknak.

```
for (var i = 0; i < 100; i++) {
    if (i % 2 == 0) {
        console.log(i);
    }
}
```

Játszunk egy kicsit a benne lévő "i" változóval. A fenti kódban írtam egy olyat, hogy menjen végig 100szor a for ciklusunk, és minden páros számos irasson ki. Láthatjuk hogy a for ciklusba tettem egy "if" vizsgálatot, ahol a már beszélt "%" jeles operátorral megnézem hogy az adott számot elosztom kettővel, mennyi lesz a maradék. Ha 0 a maradék, akkor tudom hogy páros, szóval kiiratom a számot, ha nem 0, azaz páratlan szám lesz, akkor nem csinálok semmit.

function

JavaScript-ben a function-ök azaz a funkciók, végtelenül hasznos dolgok. Mivel ha mondjuk van egy kód halmazunk, ami csinál valamit, és sokszor szeretnénk használni, nem kell azt a kódot többször leírnunk a kódunkba, hanem elég betenni egy funkcióba, amit később meghívunk, esetleg még értéket is adunk neki.

```
function kisCica() {
    console.log("Kiscica");
}

kisCica();
```

Ez a kód azt fogja csinálni, hogy csináltunk egy funkciót, aminek az a neve hogy "kisCica" (fontos hogy zárójelet írtunk utána, mivel így lesz egy funkció). Majd ebbe beleírtuk a kódunkat, amit leszeretnénk futtatni. és utána "kisCica();" kóddal meghívtuk a függvényünket.

Most a függvényünk csak csinál valamit, de tudunk olyat is, hogy visszaadjon valami értéket! Ezt így tudjuk megvalósítani:

```
function kisCica() {
    console.log("Kiscica");
    return "yes";
}

var something = kisCica();
console.log(something);
```

Na most elmagyarázom hogy mit csinál ez. A funkciónkban a "return" kulccsszó vissza ad valami értéket, jelen esetben egy szöveget, hogy "yes". Ezt pedig én betöltöttem egy változóba. Szóval létrehoztam egy "something" változót, aminek az egyenlőség jel után a "kisCica();" függvényt meghívtam, ami vissza ad egy "yes" nevű értéket, na meg kiiratja a konzolba hogy "Kiscica".

Most pedig csináljunk olyant is, hogy átadunk egy értéket a függvénynek, azzal csináljon valamit, és adja vissza! Csináljunk egy egyszerű szám összeadó függvényt:

```
function sum(number1, number2) {
    var total = number1 + number2;
    return total;
}
```

```
var total_function = sum(2,3);
console.log(total_function);
```

A fenti funkciónkba azt csináltuk, hogy a "sum()" zárójelébe megadtunk két változót, amit később majd elvárunk amikor meghívjuk a funkciónkat. Jelen esetbe két változót adtam meg, amiknek az a neve hogy number1 és number2. Ezeket a változókat tudjuk később használni a funkciónkba. Én most azt csináltam itt, hogy a két számot összeadtam egy "total" változóba, majd ezt vissza return-öljük. Később pedig egy "total_function" nevű változóba meghívtam ezt a funkciót, ahol megadtam neki hogy a 2 meg 3 számot adja össze, és ugye mint fentebb is említettem, vissza ad egy értéket, jelen esetbe az összeadott két számot, ami 5 lesz, és ezt hogy lássuk, "console.log"-al kiírjuk.

Adattípusok

Előzőleg már beszéltünk a számokról, és a szövegekről, nézzünk meg még két új dolgot.

Boolean

A boolean két értékű adattipus. Azaz vagy "true" vagy "false". Ezek megfelelője lehet még az "1" és a "0", mivel programozási nyelvekben az 1 az true-t jelent, a 0 false. De maradjunk a "true" és "false" értékeknél, mivel ezek a boolean-ok.

```
var ezigaz = true;
var eznemigaz = false;

if (ezigaz) {
    console.log("igaz bizony.");
}
```

Ez kifogja írni hogy "igaz bizony" mivel az "ezigaz" változónak true az értéke, és ugye az if azt várja el.

Amikor egy vizsgálatot csinálunk, mint például:

```
var valami = 3 > 2;
console.log(valami);
```

akkor azt látjuk, hogy "true" lett a "valami" változó értéke, azért, mert megadtunk egy vizsgálatot neki hogy három nagyobb-e mint kettő, arra visszakapta azt hogy "true" és ez került a változóba.

Tömb

A tömb is egy adattípus, arra jó, hogy egy változóban egyszerre több adatot tároljunk el.

```
var tomb = ["Kiscica", "Kiskutya", "Bicikli"];
```

Így hozunk létre egy tömböt, szögletes zárójellel, és vesszővel választjuk el benne az adatokat. Természetesen nem csak szöveg kerülhet bele, bármilyen adatot bele írhatunk. Szöveget, számot, booleant stb...

Hogyan tudunk adatot kiolvasni belőle? Hát értelem szerűen egyszerre csak egy dolgot akarunk kiolvasni belőle, azt így tudjuk megtenni:

```
var tomb = ["Kiscica", "Kiskutya", "Bicikli"];
console.log(tomb[0]);
```

Ezzel a konzolba azt fogjuk látni hogy "Kiscica". A `tomb[0]` változó megjelenítésével megjelenítettük a tömb első elemét. Azért 0, mert PHP-ban a számolást 0-tól kezdjük, azaz a tömbnek a 0.-ik eleme az első, az 1 a második és így tovább.

Hogyan tudjuk okosan kiiratni a tömb minden elemét? Nyilván nem akarjuk kézzel megirogatni hogy `tomb[0]; tomb[1];` stb... Használjuk az előzőleg tanult for ciklust!

```
var tomb = ["Kiscica", "Kiskutya", "Bicikli"];

for (var i = 0; i < tomb.length; i++) {
    console.log(tomb[i]);
}
```

Ha ezt lefuttatjuk, szépen sorban kiírja a tömb összes elemét. Hogyan működik ez? A for ciklusba megadtuk azt, hogy addig fusson amekkora a tömb mérete, ezt úgy adtuk meg hogy a `tomb.length` paranccsal megkaptunk számba hogy mennyi elem van a tömbbe, jelen esetbe ez 3, szóval 3szor fog lefutni a for ciklus. Utánna pedig ugye már tanultuk hogy a for ciklusban az `i` változó növekszik mindig, hogy éppen hol jár a ciklus. Szóval ahogy előbb írtam hogy `tomb[0]`-val jelenítem meg a tömb első elemét, ezt a számot kell növelni. Szóval logikusan beírjuk hogy `tomb[i]` így ahogy fut a for ciklus, mindig az aktuális elemet fogja kiírni.