JavaScript – Bevezetés

Mint ahogy tanultuk, alapvetően 3 fő részre tudjuk bontani a mai modern weboldalak felépítését:

Tartalom – HTML

Kinézet – CSS

Interakció, logika, adatkezelés: JavaScript (+ 'PHP')

A JavaScript (továbbiakban: JS) a HTML-lel és a CSS-sel ellentétben már valódi programozási nyelv (viszont semmi köze a Java-hoz, ne keverjük!)

JavaScript kódot többféleképpen tudunk létrehozni, egyelőre ezt a <script>...</script> tagek között fogjuk megtenni, a HTML oldalunk törzsében (= a <body> -ban, bár vannak olyan alkalmak, amikor ezt a <head> közé kell írnunk, mi egyelőre nem ott fogunk dolgozni).

írjuk ki a böngészőnkbe, hogy „Helló Világ!" Ezt többféleképpen tudjuk megtenni, próbáljunk ki hármat, nézzük meg, melyik funkció mire való:

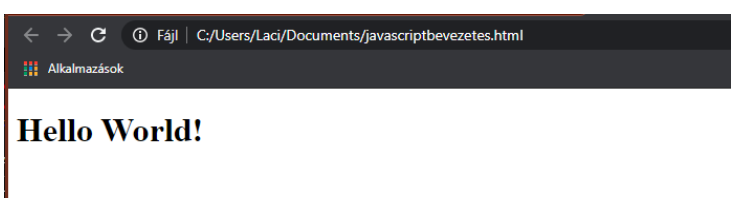
```
<body>

    <script>
        alert("Helló Világ!");
        document.write("Hello World!");
        console.log("Helló Világ!");
    </script>

</body>
```

Mint tapasztalható, a console.log() a konzolra írja ki a paraméterében kapott adatot (böngészőkben ezt az F12 bill.-el tudjuk előhozni), az alert() egy ablakot dob fel a tartalmával. (vegyük észre, hogy amíg nem OK-ozzuk le az ablakot, nem jelenik meg mögötte a „HELLÓ VILÁG" felirat! Ez azért van, mivel a sorrendiség itt is fontos, és a document.write() csak az alert után fog lefutni! Mi történik akkor, ha felcseréljük a sorrendjüket?)

Illetve ott van a document.write(), ami konkrétan futáskor beleír a dokumentumunkba (a HTML állományba, de nem írja felül azt), így akár más HTML tag-ekbe is be tudjuk ágyazni a tartalmát, pl: H1




```
document.write("<h1>Hello World!</h1>");
```

Változók

A JS a C#-től eltérően egy gyengén típusos nyelv (a c# erősen/szigorúan típusos nyelv). Azaz míg C#-ban minden értéknél definiálni kellett a változó típusát létrehozáskor (pl: double, string, int, bool, char stb…), addig JS-ben változót MINDEN esetben a 'var' (mint variable) utasítással hozzuk létre, és típust 'automatikusan kap', az értékétől függően, pl:

```
var length = 16;                                // Number
var lastName = "Johnson";                       // String
var x = {firstName:"John", lastName:"Doe"};     // Object
```

mint látható, alapvetően 3 főbb adat típus van: a **Numbe***r* (itt mindegy, hogy egész, nem egész számról beszélünk, a JS-t ez nem érdekli, a szám az szám), a **String**, illetve **Object**. (+ **Boolean** és az undefined típusok, de erről később 😊)

hozzuk létre a következő változót, és nézzük meg, mit ír ki a konzolunk!

```
var x = 16 + "alma";
```

eredmény: 16alma

nézzük meg a következőt is:

```
var x = 16 + 4 + "alma";
console.log(x);
```

eredmény: 20alma

illetve ezt is:

```
var x = "alma" + 16 + 4;
console.log(x);
```

eredmény: alma164

Látható, hogy a JS balról jobbra végzi a műveleteket, az első példánál egy Number és egy String típust adtunk össze, így lett '16alma' az eredményünk, a másodiknál viszont először két Numbert adtunk össze (16+4), és csak utána fűztük össze egy String típussal.

A harmadik példánál viszont elsőnek String + Number ('alma16') művelet volt, ezért utána a 4-es értéket nem matematikai számként kezeli (Number), hanem mint egy Stringet ('4'), ezért lesz az eredmény 'alma164'.

további példák: https://www.w3schools.com/js/js_datatypes.asp

Az összehasonlító operátorok:

Az '==' operátor (a c#-hoz hasonlóan) összehasonlít két (vagy több) értéket, és ha ez egyenlő, akkor True értékkel fog visszatérni, ellenkező esetben False értékkel. Vizsgáljuk meg/futtassuk le a következő utasítást:

```
var x = 10;
console.log(x == "10");
```

Az 'x' változónk 'Number' típusú, míg értelemszerűen a '10' String (szöveges) típusú érték, mégis TRUE lesz a kimenetünk a konzolon! Ez azért van, mert az '==' operátor CSAK AZ ÉRTÉKEKET HASONLITJA ÖSSZE, a típusukat NEM!

Ha egyező típusokat szeretnénk összehasonlítani, akkor használjuk Az '===' (3DB) operátort! Igy az előző példánk eredménye 'False' lesz!

Operator	Description	Example
==	Equal to	5 == 10 false
===	Identical (equal and of same type)	5 === 10 false
!=	Not equal to	5 != 10 true
!==	Not Identical	10 !== 10 false
>	Greater than	10 > 5 true
>=	Greater than or equal to	10 >= 5 true
<	Less than	10 < 5 false
<=	Less than or equal to	10 <= 5 false

Tömbök

A tömbök, mint adatszerkezet hasonlóan használható, mint c#-ban, de nem kell deklaráláskor megadni az elemszámot (és változhat is futáskor):

```
var autok = ["FIAT", "Volvo", "BMW"];
```

Ugyanúgy 0-tól kell indexelnünk, tehát az első eleme a tömbnek (fiat) a [0] indexű eleme, a második az [1] stb…

Továbbá elérhető a `.length property` (az 'autok' példánál maradva ez 3 lenne, mivel 3 elemű tömbről beszélünk).

Egy tömbben akár többféle típusú értéket is tárolhatunk, pl:

```
var autok = ["FIAT", "Volvo", 3.14, 4, true];
```

Függvények

Függvényeket a 'function' kulcsszóval tudunk létrehozni, ugyanúgy lehet nekik adni paramétert, és ugyanúgy lehet visszatérési értéke (return). pl:

```
<body>

    <script>

        function sajatFuggvenyem(szam1, szam2) {
            return szam1 * szam2;
        }

        console.log(sajatFuggvenyem(10, 5));

    </script>

</body>
```

Ciklusokra példák:

```
for (var i = 0; i < 10; i++) {
    document.write(i + ": " + i * 3 + "<br />");
}

var sum = 0;
for (var i = 0; i < a.length; i++) {
    sum += a[i];
}
```

```
var i = 1;
while (i < 100) {
    i += 2;
    document.write(i + ", ");
}
```

Elágazásra példák:

```
var status;

if ((age >= 14) && (age < 19)) {
    status = "Jogosult.";
}
else {
    status = "Nem jogosult.";
}
```

```
var text;

switch (new Date().getDay()) {      // mai nap
    case 6:                         // if (day == 6)
        text = "Saturday";
        break;
    case 0:                         // if (day == 0)
        text = "Sunday";
        break;
    default:                        // else...
        text = "Whatever";
}
```