Codinc

CodeMob: Programozás a felhasználók számára
2017. Október. Http://codemob.eu/ Szerzők:

Co-funded by the Erasmus+ Programme of the European Union

This publication has been co-funded by the European Commission's Ersamus+ Programme.

Tartalomjegyzék

Üdvözlünk a programozás kurzuson!

Mire a végére érünk, képes leszel megírni:

- egy játékot (pl. egy memóriajátékot) JavaScript nyelven
- csinálni hozzá GUI-t (grafikus felhasználói felületet) HTML5 és CSS3 használatával
- megérteni és visszafejteni az alapvető programozási megoldásokat

Mindenek előtt jegyzed meg, hogy sok ezernyi forrás áll rendelkezésedre online a témában. Itt most nem tudunk egy teljes körű kurzust adni neked. Viszont minden kérdésedre a programozással kapcsolatban egyszerű web-es kereséssekkel választ kaphatsz. Tekintsd ezt a kurzust bevezetőnek, mely lépésről-lépésre bemutatja a JavaScript-et. Jó szórakozást!

Miért tanuljunk JavaScript-et?

Ne keverjük össze a Java-val, a JavaScript lehetővé teszi számodra, hogy interaktív weboldalakat építs fel. A JavaScript alapvető web-es technológiává lépett elő a HTML és a CSS mellett, melyet a legtöbb böngésző program támogat. Így aztán kénytelen leszel megtanulni, ha be akarsz kerülni a web-es fejlesztések világába. Meg kell tanulnod, ha azt tervezed, hogy front-end (ügyfél oldali) fejlesztővé válsz, vagy ha a JavaScript-et back-end (háttér rendszer) fejlesztésre akarod használni.

Ezen kívül a JavaScript használata mostanra kiterjedt a mobil applikációk, az asztali alkalmazások és a játékprogramok világára is. Mindent összevetve rohamosan nő a népszerűsége és a megtanulása egy nagyon hasznos készséget jelent.

A szempontok (balról jobbra haladva):

- Barátságosság a kezdőkkel
- Skálázhatóság
- Fejlesztői közösség
- Karrier lehetőségek
- Jövőállóság

Forrás: http://www.bestprogramminglanguagefor.me/why-learn-javascript

Algoritmusok

A matematikában és a számítás tudományban az algoritmus műveletek zárt végű lépésről-lépésre végrehajtott sorozatát jelenti. Egy algoritmus kiszámolhat dolgokat, feldolgozhat adatokat és/vagy végrehajthat automatizálható feladatokat.

Itt egy egyszerű példa, milyen lehet egy algoritmus:

Kérdés, el tudok-e menni moziba!
- Igen
- Nem

Ha nem, akkor nem megyek moziba.
Ha igen, kérdés, van-e házi feladatom!
- Van
- Nincs

Ha van, akkor van-e időm megcsinálni időben a házi feladatot!
- Igen
- Nem

Ha nincs elég idő a házi feladatra, akkor nem megyek moziba.
Ha nincs házi feladatom vagy van elég idő megcsinálni, akkor már csak az a kérdés, hogy van-e elég pénzem a mozira!
- Igen
- Nem

Ha nincs, akkor egyértelmű, hogy nem megyek moziba.
Ha van elég pénzem, akkor mehetek moziba.

Az ábrán látható angol nyelvű ábra ezt az alogoritmust mutatja be.

Hogyan viszonyul egymáshoz az algoritmus és a program fogalma? Ránézésre egy algoritmus olyan, mint egy formula, ami kiszámít valamit. Egy program viszont az utasításoknak egy olyan sorozata, amit a számítógép megért és végrehajtja a kitűzött feladat megoldását adó algoritmust.

Próbáljuk megérteni egy példán keresztül:

Egyszerű algoritmus a téglalap területének kiszámítására
(T = szélesség szorozva a magassággal)
A program, ami kiszámolja a területet, így dolgozhat:
Bekéri a felhasználótól a szélességet
Bekéri a felhasználótól a magasságot
Használja a képletet (formula) a terület kiszámításához
Megjeleníti az eredményt

Bevezetés a HTML-be és a CSS-be

Egy weboldal elkészítéséhez a szükséges információkkal el kell látni a számítógépet, de nem elég csak begépelni a szöveget, ami meg fog jelenni az oldalon, hanem fontos tudni, hogyan akarjuk elhelyezni, elrendezni a szöveget, beszúrhatunk képeket, tudunk linkeket létrehozni, stb. Ahhoz, hogy mindezt elmagyarázzuk a számítógépnek, szükséges egy nyelvezet, amit a számítógép megért.

Sok programnyelv létezik, mint pl. a C++ és a Java. Ezek végtelenül összetettek és olyan embereknek szánták, akik már több számítógépes tudással rendelkeznek.

A HTML-t és a CSS-t kimondottan weboldalak készítésére találták ki úgy, hogy könnyen használhatóak legyenek. Mindkettőnek van egy speciális célja és ez a kettő így együtt, egymást kiegészítve eredményeznek, hoznak létre egy weboldalt.

- HTML: HyperText Markup Language („HiperSzöveg Jelölő Nyelv”) rövidítése, mely 1991-ben jelent meg. Ekkor indult a Web. A célja, hogy kezelje és összerendezze a tartalmakat. HTML-ben lett írva, amit az oldalnak meg kellett jelenítenie: szövegek, linkek (hivatkozások), képek … Pl. Cím, menü, szöveges tartalom …
  Ezen a kurzuson a HTML nyelv legfrissebb változatával foglalkozunk (HTML5), mely a jövőt jelenti és mindenkit csak bátorítani lehet a használatára.

- CSS: Cascading Style Sheets („Kaszkád Stílus Lapok”) rövidítése. Ez a nyelv csak a weboldal kinézetéért felel. A CSS-ben az tudod mondani, hogy cím legyen piros és aláhúzott, szövegeim legyenek Arial betűtípussal, a nevem legyen középre igazítva, a menü háttere meg legyen fehér, stb., stb. Ezzel a nyelvvel könnyen gyorsan ki tudjuk alakítani az oldalunk elrendezését.

A szerkesztőprogram

Az egyik kérdés, ami nyilvánvalóan felmerülhet, „Milyen szoftverrel tudom megcsinálni a weboldalamat?”

A Jegyzettömb (Notepad) nevű alkalmazás elég egy weboldal elkészítéséhez! De nhogy megkönnyítsük a dolgunkat inkább használjunk forráskód szerkesztőt, mint pl. a Notepad++ (az előnye, hogy automatikusan kiszínezi a HTML és a CSS kulcsszavait, ezáltal átláthatóbbá és olvashatóbbá teszi).

Böngészők

Mi az a böngésző?

A böngésző egy program, ami lehetővé teszi a weboldalak megtekintését. A feladata feldolgozni a HTML/CSS kódot, amit írtál és megjeleníteni, amit azok reprezentálnak. Ha a CSS-ed azt mondja „Címkék legyetek pirosak”, akkor a böngésző a címkéket pirosan fogja megjeleníteni.

A számos létező böngésző között az alábbiak a legfontosabbak:
Internet Explorer – Mozilla Firefox – Opera – Netscape – Konqueror (Linux-ra) – Lynx (szintén Linux-ra) – Apple Safari (Mac-re) – Google Chrome - stb.

CSINÁLD MEG AZ ELSŐ HTML DOKUMENTUMODAT

Mi is az a HTML?

Ahogy mondtuk korábban, a HTML egy jelölőnyelv, ami leírja a webdokumentumot (web oldalt).

- A HTML jelenti a Hiperszöveges Jelölőnyelvet
- Egy jelölőnyelv jelölő tag-ekből (címkék) áll
- A HTML dokumentum HTML tag-ekkel van leírva
- Minden HTML tag különböző részét írja le a dokumentum tartalmának

Egy kis példa HTML dokumentum:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Oldal Címe</title>
        </head>
        </body>

                <h1>Az első fejezetem</h1>
                <p>Az első bekezdésem</p>
        </body>
</html>
```

A példa értelmezése:

- A <!DOCTYPE html> jelzi, hogy ez egy HTML5 dokumentum

- A <html> tag-ek (<html> és </html>) közötti rész adja a HTML dokumentumot
- A <head> tag-ek (<head> és </head>) közötti részbe kerülnek különböző információk a dokumentumról
- A <title> tag-ek (<title> és </title>) közötti rész adja a dokumentum címét
- A <body> tag-ek (<body> és </body>) közötti rész a dokumentum látható része
- A <h1> tag-ek (<h1> és </h1>) között szerepel egy fejezetcím
- A tag-ek (és) között pedig egy bekezdés tartalma szerepel

Használva ezt a leírót egy böngésző meg fog jeleníteni egy dokumentumot egy fejezetcímmel és egy bekezdéssel.

Még több információért nézz bele a teljes dokumentációba a W3Schools oldalon.

A számodra érdekes HTML elemek

Ebben a leckében az a feladatod, hogy létrehozz egy egyszerű HTML dokumentumot a következő tartalommal:

- Title
- Text paragraph
- Image
- Link

És ne felejtsd el kommentezni, amit csinálsz!

A komment egy speciális alakú HTML tag:

<!-- Ide kerül a komment – >

Bárhova be lehet illeszteni a forráskódba, ahova csak akarod és nincs hatása az oldalad működésére, de használhatod a kommenteket, hogy később is megértsd a kódodat és később is hozzá tudj nyúlni. Ha sok forráskódod van, ez nagyon jól használható. Figyelj! Bárki megnézheti a HTML kódodat, amit egyszer feltöltöttél a web-re. Csaj jobb-klikk az oldalon és kiválasztod az „Oldal forrásának megtekintése” opciót. Vagyis érzékeny információkat mint pl. jelszavakat ne írj be kommentbe.

Légy elővigyázatos a forráskódjaiddal.

Csinálj CSS fájlt és formázd meg a HTML dokumentumodat

Mi is az a CSS?

- A CSS jelenti a „Kaszkádolt Stíluslapokat”
- A CSS határozza meg, miként jelenjenek meg a HTML tag-ok a képernyőn, papíron vagy más média felületen
- A CSS sok munkát spórol meg. Egyben képes több weboldal kinézetét vezérelni.
- A külső stíluslapok a CSS fájlokban vannak tárolva

Miért használj CSS-t?

A CSS használatával definiálhatod a stílusát a weboldalaknak beleértve a dizájnt és az elrendezést a sokféle kijelzőn a különféle eszközökön számos képernyő felbontásnál.

A CSS megoldott egy nagy problémát

A HTML sosem akart jelöléseket tartalmazni a weboldal formázásához
A HTML csak a weboldal tartalmáért felel, lsd.:

<h1>Az első fejezetem</h1>
* Az első bekezdésem *

Amikor a <font> tag és a color attribútum bekerült a HTML 3.2-es szabványba, egy rémálom kezdődött el a webfejlesztők életében. A nagy webszájt-ok fejlesztése, ahol a fontok és színek bekerültek minden egyes lapba, nagyon hosszadalmas és drága folyamatokká váltak.

Megoldandó ezt a problémát a World Wide Web Konzorcium (W3C) létrehozta a CSS-t és kiszedték a formázásokat a HTML lapokból.

A CSS sok munkát megspórol

A stílus definíciók normális esetben külső .css fájlokba kerülnek.
Egy külső stíluslap fájllal meg tudod változtatni egy teljes webszájt kinézetét úgy, hogy csak egyetlen fájlt módosítunk.

Még több információért nézz bele a teljes dokumentációba a W3Schools oldalon.
https://www.w3schools.com/css/css_intro.asp

Külső stíluslap

Amikor egy böngésző beolvas egy stíluslapot, meg fogja formázni a HTML dokumentumot a stíluslapban megadott információknak megfelelően.

A CSS beillesztés 3 módja:

1. Külső stíluslap
2. Belső stíluslapba
3. Beágyazott stíluslap

Ebben a leckében külső stíluslapot fogunk beszúrni.
Egy külső stíluslap fájllal meg tudod változtatni egy teljes webszájt kinézetét úgy, hogy csak egyetlen fájlt módosítunk.

Minden oldalnak tartalmaznia kell egy hivatkozást a külső stíluslapra egy <link> tag-en keresztül, amit a <head> részbe (header-be) kell tenni.

Példa:

```
<head>
    <link rel=”stylesheet” type=”text/css” href=”mystyle.css”>
</head>
```

Bármilyen szövegszerkesztővel meg tudunk írni egy külső stíluslapot. A fájlnak nem kellene HTML tagokat tartalmaznia. A stíluslap fájlt .css kiterjesztéssel kell elmenteni.

Így néz ki a „myStyle.css”:

```
body {
  background-color: lightblue;
}
h1 {
  color: navy;
  margin-left: 20px;
}
```

Megjegyzés: Sose tegyünk szóközt egy tulajdonság értéke és mértékegysége közé (lsd. Margin-left: 20 px;). A helyes írásmód: margin-left: 20px;

Programozzunk JS-ben

A JavaScript története

A JavaScript nyelvet Brendan Eich (Mozilla Alapítvány) hozta létre 1995-ben a Netscape-re alapozva. A nyelv jelenlegi verziója az 1.8.5-ös, amely a 3. megvalósítása az ECMA262 szabványnak.

- 1995 decemberében a Sun és a Netscape bejelentette az első verzióját a JavaScript-nek, amire válaszul a Microsoft kifejlesztette a Jscript-et (ugyan az a nyelv más elnevezéssel, hogy elkerüljék a szabadalmi vitákat a Sun-nal).
- A Netscape 1996 novemberében benyújtotta a nemzetközi ECMA szervezethez a JavaScript-et szabványosításra
- DHTML (1996) egy bonyolult és szükségtelen kezdet volt
- Ajax (2000), megnövekedett az érdeklődés az új lehetőségeknek köszönhetően
- Json, alternatíva az XML-lel szemben, ami JavaScript-en alapul
- JavaScript keretrendszerek, kényelem, hatékonyság és egységesség, de egy nagyon nehéz választás
- node.js; eseményvezérelt és szerveroldali
  - NodeWebkit és Cordova

Mi is az a JavaScript?

- Script-nyelv (magas szintű programnyelv)
  - Interpretált (szemben a fordított (compile) nyelvekkel
- Kliens oldali nyelv (a felhasználó gépén fut, nem a szerveren)
  - Ámbár a node.js 2009-ben megváltoztatta ezt a jellegét
- Képes manipulálni a DOM fát
  - Létrehozni, módosítani és törölni HTML elemeket és attribútumaikat, vagy a CSS-t
  - Ki tud váltani, tud figyelni és feldolgozni eseményeket
- Semmi köze a Sun JAVA programnyelvéhez

Példa: http://www.piskelapp.com/ & a Polyfills

Telepítés

A szkriptek elhelyezhetőek a HTML lapban akár a <head>, akár a <body> szekcióban (csak a </body> lezáró tag előtt lehet), ill. lehetnek belső és külső scriptek. A type attribútum opcionális, nem kötelező., ugyanis ez az alapértelmezett értéke (a HTML kizárólag a JavaScriptet fogadja el).

```
<script type="text/javascript">
//<![CDATA[
...(Your JS code)
//]]>
</script>
<!If there is an external source the tag must be empty >
<script src="./js/script.js"></script>
```

JavaScript példa:

Próbáljuk meg megcsinálni az első JavaScript fájlunkat. Ebben a példában feldobunk egy egyszerű figyelmeztető ablakot. Ehhez be fogunk linkelni egy külső .js fájlt a .html dokumentumunkba.

Először is lépjünk bele abba a könyvtárba, amelyben az első .html és .css fájlunkat létrehoztuk. Hozzuk létre és mentsük le ide az új .js fájlt (a példa kedvéért) myscript.js néven.

Tegyük bele a következő szkriptet a .js fájlunkba:

```
alert("Én egy figyelmeztető ablak vagyok!");
```

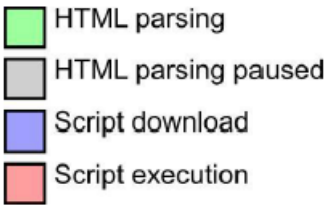
Végül linkeljük be a .html dokumentumunkba a myscript.js fájlt:

```
<!DOCTYPE html>
<html>
	<body>
		<script src="myScript.js"></script>
	</body>
</html>
```

A külső szkriptre mutató hivatkozást elhelyezheted a <body>-ban vagy a <head>-ben, ahogy a kedved tartja. A szkript annak megfelelően fog viselkedni, ahol el lett helyezve.

A szkriptek végrehajtási sorrendje – Késleltetett és aszinkron

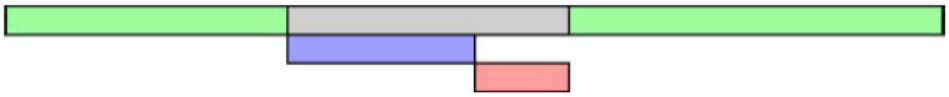
A szkriptek végrehajtásának időzítése függ a HTML lapon belül elfoglalt helyétől és a defer (késleltetés) és async (aszinkron) attribútumoktól.



Színmagyarázat:

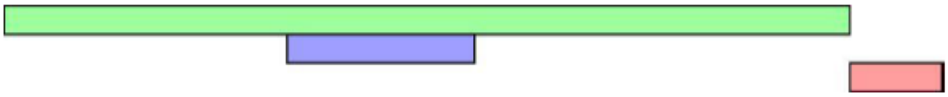
- Zöld: HTML feldolgozás
- Szürke: HTML feldolgozás szünetel
- Kék: Szkript letöltése
- Piros: Szkript végrehajtása

Normál végrehajtás



Általános működés: ha két szkript egymást követi, akkor a második addig nem fog futni, amíg az első véget nem ért, mivel több erőforrás is addig zárolva van.

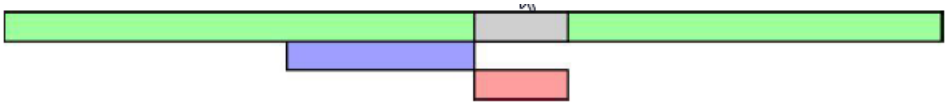
Késleltetés (Defer)



Sose használjuk a document.write()-ot vagy ehhez hasonlókat késleltetetten.

Mindig vegyük figyelembe, hogy a szkriptek végrehajtási sorrendjében ugyan az a prioritás. Elvileg ez a támogatott, bár az IE4-9 (Internet Explorer) verzióiban ez hibás.

Aszinkron (Async) (HTML 5)



Ez ideális egy olyan szkripteknek, melyek futási ideje, időzítése nem fontos (pl. Google Annalytics), így már a szkriptek végrehajtási sorrendje nem garantált.

Sose használd a document.write()-ot vagy ehhez hasonlókat aszinkron (async) módon.

Megjelenítés kitisztázása

Amikor a JavaScript olvassa majd újraírja vagy módosítja a DOM-ot, akkor a megjelenítés (layout) érvénytelenné válik és újra kell kalkulálni valamikor a jövőben. A böngészők próbálnak ezzel várni a keret (frame) lefutásának végéig, de ha mégis muszáj hamarabb ezt megtenni, annak súlyos hatásai vannak a teljesítményre.

Röviden a lényeg, hogy minél kisebb mértékben manipuláljuk a DOM-ot, vagy legrosszabb esetben is csoportosítsuk ezeket a módosításokat lehetőség szerint egy helyre.

Ha érdekel az a téma, akkor többet megtudhatsz a requestAnimationFrame-ről és a FastDOM-hoz hasonló könyvtárakról (lib): https://github.com/wilsonpage/fastdom

A JavaScript konzol

Könnyű elrontani a JavaScript programkódot, ah nincs hibakeresőnk (debugger). A debug-olás (ejtsd: dibágolás) a tesztelés egy folyamata, amivel megkeressük és elhárítjuk a bug-okat (ejsd: bágokat), azaz a hibákat, a számítógépes programokban.

Ha egy szkript nem működik (bug-os), először is megnézzük a hibát … Vagyis kell a JavaScript konzol, ami megjeleníti nekünk a hibaüzeneteket (a fájl és a sor száma a kódban, a programkód színezése, valós idejű szkript írás).

- **Firefox** F12>Console (JavaScript) vagy Konzol (JS)
- **Chrome** CTRL+SHIFT+I>Console

Hibakeresésre használható a console.log(…), amely a kapott paramétereket megjeleníti futás közben. Hozzá lehet adni a programkódhoz megállási pontokat (breakpoint, ejtsd brékpoint), de használhatod a „debugger” parancsot is.

A console.log() metódus

Ha a böngésző támogatja a hibakeresést, akkor használhatod a console.log() hívásokat megjeleníteni az értékeket a hibakereső ablakban.

Például:

```
<!DOCTYPE html>
<html>
	<body>

		<h1>Első weboldalam</h1>

		<script>
			a = 5;
			b = 6;
			c = a + b;
			console.log( c );
		</script>
	</body>
</html>
```

Megállási pontok beszúrása

A hibakereső ablakban be tudod állítani a megállási pontokat a JavaScript kódban.

Minden ilyen ponton a JavaScript kód végrehajtása meg fog állni, így megvizsgálhatjuk a JavaScript futás közbeni értékeit.

Az értékek vizsgálata után folytathatod a kód futtatását (általában egy „play”, lejátszás gomb megnyomásával).

A „debugger” kulcsszó

A „debugger” kulcsszó megállítja a JavaScript kód futását és meghívja (ha létezik) a debugging függvényt.

Ez a függvény ugyan úgy fog viselkedni, mintha beállítottuk volna a megállási pontot a hibakeresőben.

Ha a hibakereső nem elérhető, akkor a hibakereső utasításoknak nem lesz hatása.

Bekapcsolt hibakeresővel az alábbi kód meg fog állni a 3. programsor előtt.

```
var x = 15 * 5;
debugger;
document.getElementbyId(„demo”).innerHTML = x;
```

Tartsuk észben, hogy a console objektum nem része a szabványnak. Annak ellenére, hogy a Chrome-ban és Firefox-ban elég jól implementálták.

Például: Firefox Konzol API

Kommentek

A JavaScript kommentek érthetővé és jobban olvashatóbbá teszik a JavaScript kódot.

A JavaScript kommenteket használhatjuk egyes programrészek futásának megakadályozására (kikommentezés) például, amikor tesztelni akarunk a kódváltozatot fejlesztés közben.

1 soros kommentek

Az egy soros kommentek //-rel kezdődnek

Minden olyan szövegről, ami a // után a sor végéig szerepel, a JavaScript nem vesz tudomást (nem lesz futtatva).

Ez a példa minden programsor előtt tartalmaz egy egy soros kommentet:

```
// Fejléc módosítása
document.getElementById(„myH”).innerHTML = „Az első oldalam”;
// Bekezdés módosítása
document.getElementById(„myP”).innerHTML = „Az első bekezdésem”;
```

A következő példa pedig a programsorok végén tartalmaz a kódot magyarázó kommenteket:

```
var x = 5;        // Deklaráljuk x-et és értékül kapja az 5-öt
var y = x + 2;  // Deklaráljuk y-t és értékül kapja az x + 2-t (vagyis a 7-et)
```

Több soros kommentek

A több soros komentek /*-gal kezdőnek és */ zárja őket.

Minden szöveget a /* és a */ között a JavaScript figyelmen kívül hagy.

Az alábbi példakód tartalmaz egy több soros kommentet a programkód magyarázatára:

```
/*
Az alábbi programkód módosítani fogja
a fejlécet, melynek Id-ja „myH”
és módosítja a bekezdést, melynek ID-ja „myP”
a weboldalunkon belül.
*/
document.getElementById(„myH”).innerHTML = „Az első oldalam”;
document.getElementById(„myP”).innerHTML = „Az első bekezdésem”;
```

Kikommentezés a futás megakadályozására

A kikommentezés alkalmas a programkód futásának megakadályozására a forráskód tesztelése közben.

Tegyünk //-t a programsor elejére, hogy a sor futó kódból nem futó komment váljék.

Az alábbi példa mutat

```
//document.getElementById(„myH”).innerHTML = „Az első oldalam”;
document.getElementById(„myP”).innerHTML = „Az első bekezdésem”;
```

Ez a példa pedig azt mutatja, hogyan lehet több programsort kikommentezni egy komment blokkal:

```
/*
document.getElementById(„myH”).innerHTML = „Az első oldalam”;
document.getElementById(„myP”).innerHTML = „Az első bekezdésem”;
*/
```

Az egy soros kommentek tehát //-rel kezdődnek, míg a több sorosakat /* ... */ jelölések közé kell zárni. Alább egy összetettebb példát láthatunk arra, hogyan kell szabályosan beilleszteni JavaScriptet XHTML dokumentumba.

Mindig a saját igényeink szerint kommentezzünk, a függvények elvárt paramétereit, visszatérési értékeit. Vagy programrészletek deaktiválására is használható anélkül hogy törölni kellene azokat.

```
<script type="text/javascript">
        //<![CDATA[
        var elsoUzenetem = 'Helló Világ!';
        console.log(elsoUzenetem);
        function elsoFuggvenyem(){
           var proba = document.getElementById("proba");
           proba.style.color ="#990000";
        }
        elsoFuggvenyem(); //]]>
</script>
```

Miket ne csináljunk: Eval és JavaScriptet

Soha ne használjunk eval()-t, ami lehetővé teszi, hogy tetszés szerinti kódot futtassunk. Ez biztonsági kockázatot okoz.

A setInterval() függvény az 1. paraméterében átadott stringet (szöveg) eval()-lal lefuttatja mint egy programkódot a megadott bizonyos idő eltelte után.

```
setInterval( function(){myFunction(param1,param2);},5000);
```

Figyeljünk a szeparációra és ne keverjük a HTML és a JavaScript kódokat (hasonló okok miatt, mint a HTML és CSS esetében), mellőzzük a document.write(...)-ot és válasszuk szét a JavaScriptet és a megjelenítést.

```
<! Soha ne tedd ezt ... >
<a href="javascript:myFunction();">....</a>

<! ...és lehetőleg kerüld az ilyeneket >
<a title="Kattints valami JS funcióért"
    href="enablejs.html"
    onclick="myFunction(); return false;"
>
link
</a>
```

Matematikai műveletek

- *, /, +, - (Műveleti sorrend és zárójelezés)
- % (Modulo, vagyis maradékos osztás maradéka)
- +=, =, *= és /= (pl. x = x + 5;)
  - x++ és x-- (utólagos növelés vagy csökkentés: x = x + 1)
  - ++x és --x (előzetes növelés és csökkentés)

Konstansok

A konstansok hasonlóan viselkednek, mint a változók, egyszer deklarálva vannak, de az értékük utána többé nem tud változni. Gyakran használják előre definiált paraméterekként. Általános konvenció minden programnyelvnél, hogy a konstansok elnevezését csupa nagy betűvel írják a aláhúzás karakter kerül a szóközök helyére, aláhúzásokkal vannak szeparálva.
Figyelem! Az IE10 nem támogatja a konstansok használatát.

```
// Konstans
const AZ_EN_KEDVENC_SZAMOM = 9;

// Nem a konvenciótól lesz konstans, ez így nem is az
var AZ_EN_SZERENCSE_SZAMOM = 7;
```

Változók és változótípusok

A változók tekintetében a JavaScript típustalan (pontosabban dinamikusan típusolt) nyelv a PHP-hoz hasonlóan. A változók félreérthetetlen szintaxist követnek és érzékenyek a kis-nagy betűkre. Legyünk módszeresek és szigorúak, mert a változók futás közben változtathatják a típusukat.

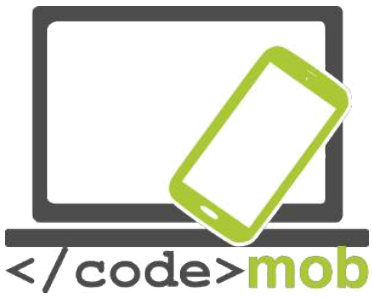
A „Lower Camel Case” írásmód használata a változónevek tekintetében egy általánosan elfogadott jó gyakorlat, ami azt jelenti, hogy a változónevet alkotó minden szó első betűje nagy, kivéve a legelső betűt és persze a többi karakter is mind kicsi. A JavaScript-nek van egy szemétgyűjtő mechanizmusa (Garbage Collector, ejtsd gerbidzs kollektor), ami megszünteti a már nem használt változókat (vagy amik NULL értékre lettek beállítva).

Ne használjuk a var a = new Array(); szintaxist és ne használjunk kötőjeleket (mínuszjel) a változónevekben (pl. var cause-An-Error = 5;).

```
var userIdNumber; // Változó deklaráció var kulcszóval
userIdNumber =5;   // Ugyan ennek a változónak az inicializálása

var userIdNumber =5; // Deklarációs és inicializálása egyszerre
initialization

var variable1 ="Hello World!";
var variable2 =42;

var variable1 ="Hello World!",
variable2 =42; // Többszörös deklaráció
```

Adattípusok

- Primitív, egyszerű típusok
  - String (karaktersorozat): „Helló Világ!”
  - Number (egész és lebegőpontos szám): 3,14 vagy 42
  - Boolean (logikai érték, igaz/hamis): true vagy false
  - Undefined (nem definiált, olyan változó, ami még nem kapott értéket, nem határozódott még meg a típusa)
  - null (Üres változó, nincs benne semmi, nem egyezik meg a 0 értékkel!)
- Objektumok
  - Függvények
  - Tömbök
  - Date, Math
  - RegExp (reguláris kifejezés)



Típuskényszerítés

Lehetőség van egy változó típusának módosítására, kényszerítésére (casting, ejtsd kaszting) pl. a Number(...) , a String(...) és a Boolean(...) vagy més metódusok használatával, mint a .split() vagy a .join().

```
var nbr2Boo = Boolean(0);       // false
var str2Nbr = Number('12');     // 12
var boo2Str = String(true);     // "true"
```

Sztringek

- A sztringek összefűzése a '+' művelettel történik
- A sztringek megadása idézőjelek vagy aposztrófok között történik
    - A PHP-val ellentétben itt nincs különbség a kettőféle jelölés között
    - A speciális karakterek elé a sztringen belül \ jelet kell tenni (escape karakter, ejtsd: eszképelni kell)
        - \”, \n (újsor karakter), \\, …
        - Hosszú sztring több sorba törésekor a sorok végére \-t kell tenni
- A hossza elérhető a .length tulajdonságon keresztül
- A sztingeknek sok metódusai vannak
    - charAt()
    - indexOf()
    - substr, substring()
    - toLowerCase(), toUpperCase()
- Nem lehet egy sztringet úgy használni, mint egy tömböt

Példák: http://www.w3schools.com/js/js_strings.asp

Számok

- Ez csak egy típus (pedig egész és lebegőpontos), 64bites lebegőpontos ábrázolással kerül tárolásra
- Az „Infinity” és „-Infinity” konstansok nyilvánvalóan a pozitív és negatív végtelent jelképezik

- NaN-t (Not a Numbert, azaz nem egy szám) kapunk a matematikia képtelenségekre, mint pl. egy sztringgel való osztás.
- A .toSting() metódus szöveggé konvertálja a számokat
  - .toString(2): binárissá konvertálja (2-es számrendszer)
  - .toString(16): hexadecimálissá konvertálja (16-os számrendszer)
- A Math objektum számos funkcióval rendelkezik:
  - Math.random(): véletlenszám generálása 0-tól 1-ig (0 lehet, de 1 már nem)
  - Math.min(..., …) Math.max(..., …): Minimum és maximum meghatározása
  - Math.round(), Math.ceil() és Math.floor(): kerekítés, felkerekítés, lekerekítés
  - Math.sin(), math.cos(), Math.PI, …

Példák: http://www.w3schools.com/js/js_number_methods.asp

Feladat: Véletlenszerűen jelezzünk ki egy 0 és 1 közötti számot a konzolra, valahányszor újratöltik az oldalt (pl. F5 billentyűvel).

Tömbök

- A tömböket üres szögletes zárójelekkel [] lehet inicializálni, és ugyan ezeket lehet használni az egyes elemek elérésére.
  - **var** myArray = []; // Új üres tömb
- A legtöbb programnyelvhez hasonlóan 0-dik indexen kezdődik a tömb.
  - myArray[0] = 42;
- A tömb hosszát a .length tulajdonságon keresztül lehet elérni.
  - Sose módosul az a tömb, amit átadtál (?????)
- Figyelem! Ezek nem asszociatív tömbök!
- A tömbük rendelkeznek mindenféle metódusokkal:
  - .push(), .pop(), .splice(), ….
  - Lehet rendezni őket lexikografikusan (mint sztingeket abc sorrendben): .sort()
  - átadható a .sort()-nak komparátor függvény, ami szerint a rendezés történjuen
  - **.sort(function(a, b) {return a - b})**

Példák: http://www.w3schools.com/js/js_array_methods.asp

Feladat: Mutasd meg egy tömb utolsó elemét, miközben nem tudod a tömb hosszát.

Egy változó típusának ellenőrzése

Több metódus is rendelkezésünkre áll, amivel ellenőrizhetjük egy változó típusát.

- typeof myVariable
  - typeof myVar === 'undefined'
  - A NaN az szám típusú
  - Egy tömb az objektum típusú
  - Egy Date (vagyis dátum) objektum az objektum típusú
  - A null is objektum típusú
  - Egy nem definiált változó típusa 'undefined'

  Példák:

```
typeof „John”                  // „string”-el tér vissza
typeof 3.14                    // „number”-rel tér vissza
typeof false                   // „boolean”-nal tér vissza
typeof [1,2,3,4]               // „object”-tel tér vissza (nem „array”-jel, lsd. Lenti
                                  megjegyzés)
typeof {name: 'John', age:34}  // „object”-tel tér vissza
```

  Megjegyzés: A typeof operátor tömbök esetén is „obeject”-et jelezt, mert JavaScript-ben a tömbök is objektumok.

- isNaN(...)
- isFinite(...)
- Array.isArray(...);
- …

Változó vagy sem?

Egy függvényben egy változó a var kulcsszóval lokálisnak lesz deklarálva. Viszont nélküle pedig globálisnak. Röviden fogalmazva az a tanácsos, ha mindig deklaráljuk a függvények változóit (kerüljük a globális terület (global scope, ejtsd: globál szkóp) felüldefiniálását, ECMA6, …).

A lokális és globális változóknak más az élettartamuk: a lokális változók megsemmisülnek a függvény végrehajtásának végén (mármint a függvény saját lokális változói, az aktuális végrehajtási kontextusban), míg a globális változók a HTML lap lezárásig élnek.

Persze több finomság is van még ebben a kérdésben, de mindenképpen tartsuk észben, hogyan használjuk a var kulcsszót, amikor változót deklarálunk. Csak egy nem definiált változót lehet törölni.

Szkóp (scope)

JavaScriptben az objektumok és a függvények is változók.

A változóterület (scope, ejtsd: szkóp) a változók, függvények és objektumok egy halmazát jelenti, melyeket egy adott időpontban el tudsz érni. Ez egy függvényen belül változhat (függvény szkóp)

A szkóp mindig visszafele halad a lokálistól a globális felé, vagyis ha a JavaScript nem talál egy változót helyben, akkor keresni fogja fentebb (kintebb), a globális területen és végül hibát ad, ha nem talál semmit.

Emelő hatás (hoisting)

A JavaScript a változókat és a fgv-eket mindig azelőtt deklarálja, mielőtt bármilyen programkód lefutna: vagyis a programblokk elejére kerülnek, amelyen belül szerepelnek.

Tanácsos a változókat és a függvényeket a szkópjuk elején deklarálni.

Feltételes vezérlés

A feltételes utasítások lehetővé teszik különböző műveletek elvégzését különböző feltételek fennállása esetén.

A feltételes utasítások

A leggyakrabban, amikor programkódot írunk, arra van szükségünk, hogy különböző döntések alapján eltérő műveleteket hajtsunk végre.

A feltételes utasítást használhatod erre a kódodban.

JavaScriptben a következő feltételes utasítások állnak rendelkezésre:

- Használj if-et olyan programblokk végrehajtásához, ami akkor fut le, ha a megadott feltétel igaz.
- Használj else-t olyan programblokk végrehajtásához, ha az előző (if-nél megadott) feltétel hamis.
- Használj else if-et, ha az előző feltétel hamis és új feltételt akarsz szabni egy újabb programblokknak.
- Használj switch-et, ha számos alternatív programblokkot akarsz végrehajtani.

If … then … else …

Jegyezd meg a szóközt az else és az if között (és nem elseif, ahogy azt PHP-ban megszokhattuk).

```
if(money <0){
   status ="Le vagyok törve..."; }
elseif(money <10000){
   status ="Lehetne rosszabb is...";
}else{
   status ="Gazdag vagyok!";
}
```

Hármas művelet (ternary operator)

Egy tömör írásmódja az if … then szerkezetnek, használható változók feltételes deklarációjánál.

```
var admissibleAtI3 = (sex =="F") ? true : false;
```

Switch … case

Egy egyszerű módja többféle döntés mentén végighaladni.

Jegyezd meg azt, hogy egy switch-csel lehetőség van több esetet ráirányítani ugyan arra a kódrészletre. És a switch egyértelmű egyenlődági összehasonlítást (===) használ.

```
switch (newDate().getDay()){
   case1:
   case2:
   case3:
   default:
      text ="Várom a hétvégét!";
      break;
   case4:
   case5:
      text ="Fáradt vagyok és unott";
      break;
   case0:
   case6:
      text ="Játékidő! :)";
}
```

Logikai műveletek

- == (különbözik az = -től), !=
- ===, !==: értéket és a típust is összehasonlítja
- >, >=, <, <=
- && (és)
- || (vagy)
- ! (nem, vagyis tagadás)
  - kapcsolo=!kapcsolo //igaz hamissá válik és a hamis igazzá

A JavaScript, mint minden programnyelv megáll a kifejezések kiértékelésével, ha már egyértelmű annak végeredménye. && (és) operátorokkal összekötött feltételek esetén

megáll az első hamis értéket adó részénél a feltételnek. Ez lehetővé teszi, hogy először ellenőrizzünk egy változót és utána dolgozzunk vele az adott kifejezésben anélkül, hogy hibát kapnánk, mert nem létezik az a változó.

A JavaScriptben nincs XOR (kizáró vagy) operátor, de könnyű helyettesíteni egy egyszerű függvénnyel.

Lássuk az alábbi igazságtáblázatot:

	&& (AND, és)	|| (OR, vagy)	XOR
IGAZ, IGAZ	IGAZ	IGAZ	HAMIS
IGAZ, HAMIS	HAMIS	IGAZ	IGAZ
HAMIS, IGAZ	HAMIS	IGAZ	IGAZ
HAMIS, HAMIS	HAMIS	HAMIS	HAMIS

Feladat: Készíts egy feltételt 2 paraméterrel, ami helyettesíteni tudja az XOR-t.

```
var a = true;  // Ellenőrizd mind a 4 kombinációban: true, true, false, false
var b = true;                    // true, false, false, true
if(...){
   console.log('XOR eredménye igaz');
}else{
  console.log('XOR eredménye hamis');
};
```

Ciklusok

```
// FOR ciklus
for(var i =0; i <10; i++) {...};

// Optimalizált FOR ciklus
for(var i =0, len = myArray.length; i <len; i++) {...};

// Végigiterálni egy objektum mezőin
for(prop in obj){
   // prop
   // obj[prop]
};
```

```
// Egyszerű while ciklus, ciklus amíg szerkezet
var i = 0;
while(i <10){
   // ...
   i++; // Enélkül a növelés nélkül végtelen ciklus lenne
};

// Legalább egyszer lefutó ciklusmaggal, hátul tesztelő ciklus
do{
   // ...
   i++;
} while(i <10);
```

A böngészőknek van egy maximális időkorlátjuk a JavaScript kódok futtatásához.

Break, Continue és a címkék

A break és a continue egy kontrollálási lehetősget ad neked a ciklusok felett.

A címke pedig lehetővé teszi kijelölni egy adott pontot a szkriptben. A címkét a break vagy a continue előtt kell megadni.

- A break befejezi a ciklust
  - A break címkével képes több egymásba ágyazott blokkból is kilépni (többszörös ciklusok)
- A continue kihagyja az ciklus aktuális iterációjának hátralevő részét és a következő iterációra lép.

Feladat: Csinálj egy tennivaló listát, amin 3-szot végigmész mint egy tömbön (a getElementById-t és az innerHTML-t használd).

Try … catch

A try-catch blokk képes ellenőrizni és felismerni egy kódelem által generált hibát.

```
try{
   // Egy programkód blokk, ami generálhat hibákat.
}
catch(e){
   // Kód blokk, ami le tudja kezelni a lehetséges hibát az e változóban
```

```
}
finally{
   // Kódblokk, ami mindig lefut az eredménytől függetlenül
   // Normál lefutás és visszatérés vagy bármilyen hiba bekövetkezése
(eldobása) esetén
}
```

A throw kulcsszó fenn van tartva arra, hogy hibát kiváltsunk (akár saját típusút is).

```
throw 404;
throw new Error(„Hibás életkor!”);
```

Függvények

A függvényeknek számos előnyük van:

- Egybecsomagolják a kódot
- Fejleszti az átláthatóságot, olvashatóságot
- Újrahasznosítható és újraírható
  - Általánosítható a kód (ugyan az program, de különböző típusokra)

function sum(arg1, arg2) {return arg1 + arg2};

Módszeresen állítsuk össze a kiszolgáló függvényeket, melyek mindig adjanak vissza valamit, aminek a típusa egyértelmű.

A túl sok átadott argumentum figyelmen kívül hagyásra kerül, míg a deklarált de át nem adott paraméterek automatikusan definiálatlan (undefined) értéket vesznek fel. A függvényeken belül csináljuk meg a saját ellenőrzéseinket az argumentumokon, melyek egy tömbhöz hasonló objektumként elérhetőek (.length az argumentumok száma és [] zárójelekkel indexelhető.)

JavaScript függvény szintaxis

A JavaScript függvényeket a function kulcsszóval definiáljuk, ami a függvény neve követ () zárójelekkel.
A függvények neve tartalmazhat betűket, számokat, aláhúzást és dollár jelet (mint a változóknál).
A zárójelek között lehet felsorolni a paraméterek neveit, melyeket vesszővel kell elválasztani.
(parameter1, parameter2, …)

A függvény által végrehajtott programkódot {} kapcsos zárójelek közé kell elhelyezni.

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

A függvény paraméterek nevei a függvény definícióban (függvény fejléc) vannak felsorolva.
A függvény argumentumok a valódi értékeket jelentik, melyek átadásra kerülnek a függvénynek, amikor meghívásra kerül.

Függvényhívás és visszatérés

A függvényhívás

A kód a függvényen belül akkor fog lefutni, ha valami (vagy valaki) valahonnan meghívja a függvényt:

- amikor egy esemény váltja ki (pl. amikor a felhasználó kattint)
- amikor JavaScript kódból történik a meghívása
- automatikus hívás (önhívás)

Függvényből visszatérés

Amikor a JavaScript elér egy return utasítást, a függvény futása megszakad és visszatér oda, ahonnan a függvényt meghívták és az azt követő utasítással folyatódik a

program futása. A függvény gyakran visszatér valamiféle eredménnyel, a visszatérési érték visszaadásra kerül a függvény meghívójának.

Példa: Számoljuk ki két szám szorzatát és adjuk vissza az eredményt:

```
var x = myFunction(4, 3); // A függvény meghívásra kerül és az eredmény az x változóba kerül
function myFunction(a, b) {
   return a * b; // a függvény visszatér az a és b szozatával
}
```

Az eredmény az x-ben lesz: 12

Hívás és függvény referencia

Egy változó lehet egy hivatkozás (referencia) egy függvényre a neve által. A hívás úgy történik, hogy () zárójelek között átadásra kerülnek az argumentumok.

A Magas Szintű Függvények olyan függvények, melyek paraméterként más függvényeket fogadnak vagy más függvényekkel térnek vissza.

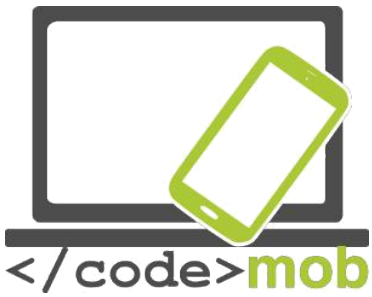
Névtelen (anonymus) függvények

A névtelen függvényekre nem tudunk a nevükkel hivatkozni, de referencia készíthető róluk, mely bekerülhet egy változóba és a referencia használatával meg lehet őket hívni.

```
var myFunction = function(message) { alert(message); }; myFunction('Ez egy teszt');
// Megjeleníti: : Ez egy teszt
```

Bezárás (closure)

Ez a téma elég JS specifikus és túl összetett kérdés ahhoz, hogy egy ilyen bevezető anyagban foglalkozzunk vele. De legalább annyit értsünk, hogy amikor egy function szót látunk egy másik függvényen belül , a belső függvény látja a külső függvény lokális változóit (olyan, mintha lenne a globális és lokális szkópok között egy regionális szintű szkóp).



Példa: http://www.w3schools.com/js/js_function_closures.asp

Objektumok

A String, Number vagy Boolean típusoktól eltérően az objektumok több értéket tartalmazhatnak kulcs-érték párok formájában.

```
var 1 ={};// Új objektum
var person = {
   firstName: "David",
   lastName: "Collignon",
   age: 39
};
console.log(person.firstName);
console.log(person['firstName']);
```

Névtelen objektum

Egy objektumot hasonlóan egyéb adattípusokhoz, nem kötelező elnevezni. Gyakori használati eset erre, amikor konfigurációs objektumot adunk át egy osztálynak paraméterül.

```
$('.bxslider').bxSlider({mode:'fade', captions:true});
```

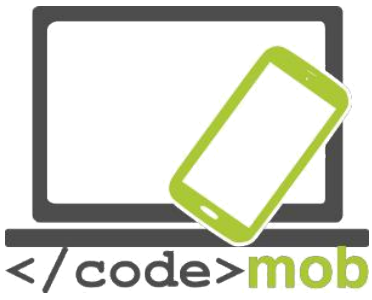
Közérdekű objektumok

Számos jól használható objektum áll közvetlenül rendelkezésre: document, window, Math, stb.

- Objektum: pl. document
  - Mező: pl. .innerHTML vagy .textContent
  - Metódus: pl. .getElementById()
  - Kulcsszó: this

```
document.getElementById("demo").innerHTML ="Hello World!";
```

A this kulcsszó az objektumra önmagára hivatkozik a kódban. Vagyis olyan, mintha az objektum önmaga lenne. A this-t részletes meg lehet ismerni a jQuery kapcsán.



Értékadás értékkel vagy referenciával

A JavaScriptben az összetett adattípusok (tömbök, objektumok) értékadása referenciálisan történik és nem érték szerint. Az igényeknek megfelelőne ezért szükséges lehet megírni egy scriptet, ami le tud másolni egy táblázatot vagy egy objektumot („mély” másolás, klónozás).

```
// Rövid példa klónozásra
JSON.parse(JSON.stringify(obj)) // csak ha nincs fgv.
var newObject = jQuery.extend(true,{},oldObject);
var newArray = jQuery.extend(true,[],oldArray);
```

A DOM

Mi is az a DOM?

A DOM egy W3C (World Wide Web Konzrocium) szabvány.

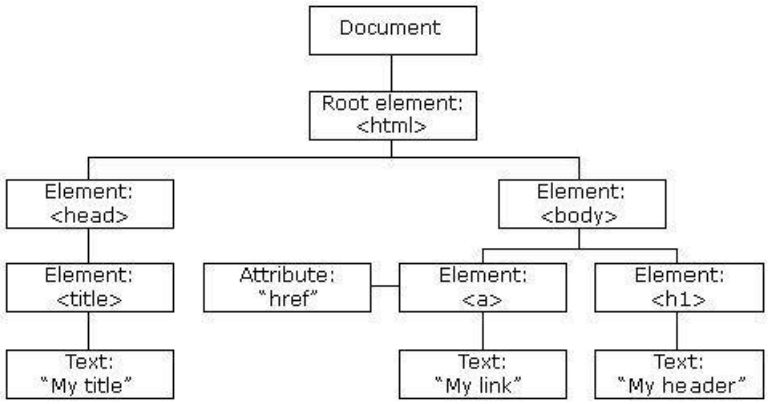
A DOM definiálja a sztenderd módját a dokumentumok eléréshez:

„A W3C Document Object Model (DOM) egy platform- és nyelvfüggetlen interfész,a mi lehetővé teszi a programoknak és szkripteknek, hogy dinamikusan hozzáférjenek és módosítsák a tartalmát, struktúráját és a stílusát a dokumentumoknak.”

A W3C szabvány 3 részből áll össze:

- DOM mag – szabványos modell minden fajta dokumentumhoz
- XML DOM – szabványos modell XML dokumentumokhoz
- HTML DOM – szabványos modell HTML dokumentumokhoz

A JavaScript képes bejárni, olvasni és módosítani a Document Object Model-t (DOM).



Feladat: a fenti ábrát megcsinálni HTML lapként.

A document objektum és a HTML egy szeletének kiválasztása

A document objektum reprezentálja a teljes HTML dokuemntumot (gyökérpont), miután betöltődött a böngészőablakba. Több metódust is biztosít egyes részeinek eléréshez mint ahogy ez a példa is mutatja:

```
var node1 = document.getElementById("demo"); // #demo
var node2 = document.querySelector("p");          // Az első p megtalálása
```

A getElementById és a querySelector az illeszkedő ID-vel tér vissza és mindig csak az első találattal. Ha nincs találat, akkor null-lal tér vissza.

```
var nodeList1 = document.getElementsByClassName("demo"); // Az összes demo
var nodeList2 = document.getElementsByTagName("p");          // Az összes p
var nodeList3 = document.querySelectorAll("p");                  // Az összes p
```

Ezek a metódusok elemek listájával (node lista) térnek vissza. A lista hossza a .length tulajdonságban található, az elemeket a [] szögletes zárójelekkel lehet indekszelni, így könnyű végigmenni rajta egy ciklussal. De ennek ellenére a lista nem egy tömb.

Ne feledjük, egy komplex CSS szelektorhoz jobban használhatóak a speciális programkönyvtárak, olyanok mint pl. a jQuery, mint a .querySelector metódus akár a teljesítményt, akár a használhatóságot nézzük. Néhányan, mint John Reisig is, kritizálják ezen könyvtárak megvalósítását.

Olvasni és módosítani a HTML-t

Mi az a HTML DOM?

Itt találsz egy jó megfejtést a kérdésre: https://css-tricks.com/dom/

Könnyen és gyorsan tudod módosítani a tartalmát és az attribútumait a HTML tag-eknek az .innerHTML (olvasható és írható is) és az attribútumok getter/setter metódusai segítségével.

```
// Get & Set
var href = document.getElementById("myLink").getAttribute("href");
document.getElementById("myLink").setAttribute("href",newValue);  // Ellenőrizni,
hogy létezik-e
```

```
var hasH = document.getElementById("myLink").hasAttribute("href"); //Törlés
document.getElementById("myLink").removeAttribute("href");

// HTML beillesztése
document.getElementById("demo").innerHTML ="New text"; // Szöveges tartalom
hozzáadása document.getElementById("demo").textContent ="New text";
```

Jegyezzük meg a különbséget a .innerHTML és a .textContent között. Az előbbi HTML-t fogad és azt fel is dolgozza, míg az utóbbi csak szöveges tartalom. A TextContent gyorsabb és biztonságosabb.

A CSS módosítása

Az összes CSS tulajdonság elérhető a style tulajdonságon belül pozicionált szintaxissal.

- A tulajdonságok elnevezéseit Lower Cammel Case írásmóddak kell írni (fontsize → fontSize)
- Az értékek mindig sztringek (pl. nem 250, hanem „250px”)
- Egy új érték hozzáadása beillesztéssel történik
- A pozícióhoz az .offsetTop és az .offsetLeft használatos (és nem a jobb és nem a lent)
- A teljes szélesség és magasság (helykitöltés, keret) a .offsetWidth-en és a .offsetHeight-en keresztül adható meg

```
var element = document.getElementById("demo");
element.style.backgroundColor = "green";// camel Cased
element.style["backgroundcolor"] = "green"; // standard CSS
var element = document.getElementById("demo");
element.className =element.className +"otherclass"; // space
```

A JavaScript általában hozzáfér a beágyazott stílushoz. Ha vissza akarod kapni a CSS stílust egy külső stíluslapról vagy egy style tag-ből, akkor használd a .getComputedStyle() metódust.

```
var style = window.getComputedStyle(document.querySelector('nav'),null);
var bgc = style.getPropertyValue('backgroundcolor'); //rgb(255, 191, 0)
```

A DOM módosítása

Van néhány metódus, ami képes létrehozni, beszúrni új HTML tag-eket a DOM-ba.

- **document.createElement("htmlTag"):** Létrehoz egy HTML tag-et

- **document.createTextNode("Hello world"):** Szöveges tartalom létrehozása
- **.removeChild(childToBeRemoved):** Eltávolít egy alárendelt elemet
- **.appendChild(newContent):** Új elem hozzáadása a szülő elem alatti utolsó elem után
- **.insertBefore(newElement, currentElement):** Új elem beszúrása adott elem elé
- **.replaceChild(newElement, currentElement):** Adott elem lecserélése

Néhány használható metódus a DOM bejárásához

- **.parentNode** A felettes elem
- **.children[index]** Alárendeltek tömbje indekszelve 0-tól
- **.nextElementSibling** A következő testvér elem (közös szülő)
  - Ne keverjük össze a .nextSibling-gel, amely a node-ok és a kommentek közötti tartalommal tér vissza
- **.previousElementSibling** Az előző testvér elem
- **…**

Próbáld ki magad!

JavaScript HTML DOM elemek:
http://www.w3schools.com/js/js_htmldom_elements.asp

Módosítani a HTML tartalmat: http://JavaScript HTML DOM - Changing HTML

Módosítani a CSS-t: http://www.w3schools.com/js/js_htmldom_css.asp

A jellemzők és metódusok teljes listája:
http://www.w3schools.com/jsref/dom_obj_all.asp

Pédák – W3Schools
[1] - https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementcreate
[2] - https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementcreate2

[3] - https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementreplace

A Screen objektum

A screen objektum információkat biztosít a felhasználó képernyőjéről. Ezek az információk nem hozzáférhetőek egy szerveroldali szrkiptnyelvből.

- screen.width és screen.height
  - screen.availHeight és screen.availWidth (a taskbar nélküli méret)
- screen.colorDepth

A Navigator objektum

A Navigator objektum információkat biztosít a felhasználó böngészőjéhez.

- **navigator.userAgent** A böngésző teljes neve
- **navigator.platform** A felhasználó op.rendszere
- **navigator.onLine** A felhasználó online vagy offline van
- **navigator.language** A böngésző nyelve
- **navigator.geolocation** Geolokáció a felhasználói megállapodás alapján
- **navigator.cookieEnabled** Ha a felhasználó elfogadta a cookie-kat
- ...

Példa: http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_nav_geolocation

Események

Bármilyen eseményt (klikk, dupla klikk, fölévitel, hiba, átméretezés, ...) rá lehet akasztani a DOM egy elemére a következő módon:

element.addEventListener(event, function, useCapture);

document.getElementById("myBtn").addEventListener("click", doStuff);
function doStuff(**ev**){...}

Az event nevű paraméterben egy string kerül, ami a HTML attribútomból jön, csak le kell szedni az elejéről az 'on'-t (pl. <a href="" onClick="…" /> → click).

Az események teljes listáját itt találod:
http://www.w3schools.com/jsref/dom_obj_event.asp

Igazából a HTML események olyan dolgoka, amik megtörténhetnek a HTML egy-egy elemével.

Amikor a JavaScriptet egy HTML lapon használjuk, a JavaScript képes reagálni ezekre az eseményekre. Az események akciók vagy esetek, melyeket leprogramozol, ami által a rendszernek meg tudod mondani, mi a kívánatos válasz, reakció ezekre. Például ha a felhasználó rákattint egy gombra a weboldalon , megjeleníthetsz egy információs ablakot válaszként.

A Web esetében az események a böngésző ablakán belül váltódnak ki és arra törekszik, hogy egy specifikus elemhez kötődjön – ami lehet egy egyszerű elem vagy elemek egy halmaza, a böngésző aktuális fülén lévő HTML dokumentum, vagy lehet egy teljes böngészőablak is. Egy nagy halom esemény lehet ami kiváltódhat, példák:

- A felhasználó egy konkrét elemre kattint vagy föléviszi az egeret egy konkrét elemnek
- A felhasználó megnyom egy gombot a billentyűzeten
- A felhasználó átméretezi vagy bezárja a böngésző ablakot
- A weboldal betöltődése befejeződött
- Az űrlap elküldésre került
- A videó lejátszása el lett indítva, meg lett állítva vagy be lett fejezve
- Bármiféle hiba keletkezett

Minden eseményhez tartozik egy eseménykezelő, ami egy kódblokk általában a fejlesztő által definiálva, amely lefut, amikor az esemény kiváltódik. Amikor egy kódblokkot úgy definiálunk, hogy az válaszként fusson le egy eseményre, olyankor azt úgy hívjuk, hogy beregisztrálunk egy eseménykezelőt. Néha az eseménykezelőket úgy is hívják, hogy esemény figyelők. A figyelők elég jól cserélgethetőek a céljainknak megfelelően, miközben szigorúan együttműködnek. A figyelő figyel az esemény kiváltódására és a kezelő a programkód, ami válaszként lefut.
Egy egyszerű példa:
A következő példában van egy gombunk, amit ha megnyomunk, akkor a háttérszín véletlenszerűen módosul:

```
<button>Szín módosítása</button>
```

A JavaScript pedig így néz ki:

```
var btn = document.querySelector('button');

function random(number) {
   return Math.floor(Math.random() * number);
}

btn.onclick = function() {
   var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
   document.body..style..backgroundColor = rndCol;
}
```

Ebben a kódban egy a gombra mutató referenciát teszünk egy változóba, amit btn-nek nevezünk, amihez a Document.querySelector() függvényt használjuk. Ezen felül definiálunk egy függvényt, ami véletlen számot generál. A harmadik része a kódnak az eseménykezelő. A btn változó egy <button> elemre mutat és az ilyen típusú objektumoknak számos eseménnyel rendelkeznek, melyek kiváltódhatnak és melyekre az eseménykezelők figyelhetnek. Mi most a kattintás eseményre figyelünk, amit az onclick jellemzőbe beállított eseménykezelő kezel le, ami jelen esetben egy anonymus függvényt jelent, ami tartalmazza a kódot, ami generálja véletlenszerűen az RGB színeket és beállítja a <body>-nak a háttérszínt.

Ez a kód akkor fog futni, amikor a kattintás esemény kiváltódik a <button> elemen, azaz amikor a felhasználó rákattint a gombra.

Kód és eredmény: https://codepen.io/pen/

Eseménykezelő tulajdonságok

Visszatérve az előző példához:

```
var btn = document.querySelector('button');

btn.onclick = function() {
    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';
    document.body..style..backgroundColor = rndCol;
}
```

Az onclick tulajdonság az eseménykezelés célját szolgálja ebben az esetben. Ez lényegileg ugyan olyan tulajdonsága a gombnak, mint bármely másik, ami elérhető. (mint a btn.textContent vagy a btn.style), de abból a szempontból speciális, hogy némi programkód kerül bele, az, amely lefut, amikor a gomb eseménye kiváltódik.

Esemény menedzser

Az Event objektum minden DOM eseményt meg tud testesíteni. Az esemény objektumot argumentumként megkapja az eseménykezelő függvény, ami lekezeli az adott eseményt.

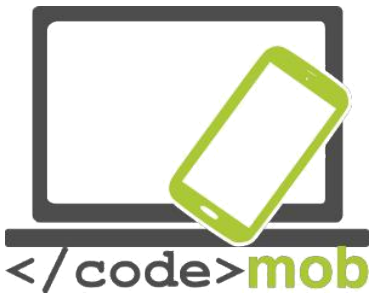
Az Event-nek több jellemzője és metódusa:

- event.target: Az objektum, ahonnan kiindult az esemény
- event.currentTarget: Az objektum, amit megcélzott az esemény
- event.preventDefault(): Megszakítja a normál viselkedését az eseménynek
- event.stopPropagation(): megakasztja az esemény tovaterjedését

Felszállás és elkapás

Az elkapás a lemenő irányt (a kintebbi elem felől a bentebbi felé halad), míg a felszállás a felmenő irányt (mint a halász aki lemerült majd felemelkedik) figyeli.

A useCapture alapértelmezett értéke hamis (false), és az IE9 csak a felbugyogó eseményeket támogatja.



Eseménykezelők eltávolítása

Csak használjuk a .removeEventListener() metódust, bár az anonymus függvényeket nem lehet eltávolítani. Adalék ehhez, hogy minden eseménykezelőt külön-külön is el lehet távolítani. Két különböző eseménykezelője lehet ugyan annak az eseménynek attól függően, hogy felfelé száll éppen vagy lefelé merül.

```
var e =document.getElementById("myDIV");
e.addEventListener("mousemove",myFunction);
e.removeEventListener("mousemove",myFunction);
```

Kvíz

Itt egy kis kvíz kérdéssor, amit most kitölthetsz.
A válaszokat megtalálod a végén. Próbálj nem csalni!

Kérdések

Bevezető

1. Amikor a programod nem működik, az első lépés, hogy megnézed a hibaüzeneteket. Hogy tudod megnyitni a web konzolt Firefox és Chrome alatt?

- F12 (Chrome és Firefox)
- Open Menu > Developer > Web Console (Firefox)
- Ctrl + Shift + K (Firefox)
- Ctrl + Shift + I (Chrome)
- Ctrl + C (Chrome és Firefox)

2. Deklarálhatsz egy változót var kulcsóval, vagy anélkül. Melyiket kellene használnod?

- Mindkettőt, nincs különbség
- A var kulcsszóval deklarált változó a helyi szkópba kerül, míg var nélküliek a globális szkópba. Mindig használjunk var kulcsszót, hogy ne szemeteljük össze a globális szkópot.

- A var kulcsszóval deklarált változó a globális szkópba kerül, míg a var nélküliek a helyi szkópba. Sose használjunk var kulcsszót, hogy ne szemeteljük össze a globális szkópot.

3. Miért írjunk mindig kommenteket

- Nagyon fontos, hogy megérthessük a programkód kiemelt pontjait, akár ha a későbbiekben visszatérsz egy régi projektedhez, akár ha egy csapat tagjaként dolgozol a szoftveren.
- A kommentelés kidobott idő, senki sem ír ilyeneket.
- Ez egy konvenció, hogy időlegesen hatályon kívül helyezzük a programkód egyes részeit, anélkül hogy ki kéne őket törölni és levesznének.

Feltételek

1. Van-e logikailag különbség egy if...else szerkezet és 2 egymás követ if használata között, ha azok feltételei egymás ellentétei.

- Nincs különbség, de az if...lese messzemenően jobb választás, mert átláthatóbb és kisebb a hibázási lehetőség benne.
- Igen, van különbség, nem ugyan azok.

2. Hogyan lépkedsz végig egy tömb elemein, ha nem tudod a tömb hosszát?

- A tömb .length tulajdonságát használom
- A tömb .length() metódusát hívom megadott
- A count() függvényt használom

3. Melyik a hatékonyabb a következő kódrészletek közül?
```
for (var i = 0, len = a.length; i < len, i++) { /* ... */ }
for (var i = 0; i < a.length, i++) { /* ... */ }
```

- Egyformák
- A len változó csak egyszer értékelődik ki, nem minden iterációban
- Az iterációk száma különbözik

Függvények

1. Fontos-e a „divide” nevű függvények argumentumainak sorrendje (az a függvény, ami visszatér az első és a második paraméter hányadosával)?

- Igen, módosítja a végeredményt
- Nem

2. Mi különbség a függvényhívás és a függvény referencia között?

- A függvény hívás (a függvény nevét követő zárójelekkel) a függvény kódjának azonnali futtatását kéri a hivás helyén. A függvény referencia (pl. egy call back-ben) egy egyszerű letárolását jelenti egy függvény nevének későbbi felhasználásra.
- Nincsen semmi különbség, ha a függvények nincsenek paraméterei.

3. Mi a fenntartott kulcsszó, ami megadja, milyen értékkel térjen vissza a meghívott függvény?

- Return
- Pass
- Exit

DOM

1. Melyik metódusok nem az első vagy az összes előfordulást célozzák?

- document.querySelector()
- document.querySelectorAll()
- document.getElementById()
- document.getElementByTagName()
- document.getElementByClassName()
- document.getElementByName()

2. A következő kódban milyen argumentumot kap a callback?
document.getElementById('myID').addEventListener('click', myCallBack);

- Magát az eseményt, ami aktiválta a callback-et
- Nincs argumentum átadva a callback-en keresztül

3. Hogyan tudod elkérni az értékét egy HTML kontrollnak (UI), mint pl. egy szöveges beviteli mezőnek?

- A .value tulajdonságon keresztül, ami visszatér a 'value' attribútum tartalmával.
- A .getValue() metóduson keresztül, ami visszatér a 'value' attribútum tartalmával.
- A ..val() metóduson keresztül, ami visszatér a 'value' attribútum tartalmával.

CSS

1. Milyen CSS szelektor illeszkedik minden olyan paragrafusra,a mi közvetlenül egy div alá tartozik?

- Div > p
- Div p
- Div, p

2. Milyen CSS szelektor illeszkedik arra, hogy az első paragrafus függetlenül attól, hogy hány testvére van a HTML node-ok között.

- p:nth-of-type(1)
- p:first-child
- p:nth-child(1)

3. Ha két CSS szelektor (.red és #blue) ugyan azt az elemet célozzák meg, milyen lesz a színe?

- Kék
- Piros
- Rózsaszín
- Attól függ, melyik van később deklarálva

Válaszok

Bevezető

1. Amikor a programod nem működik, az első lépés, hogy megnézed a hibaüzneteket. Hogy tudod megnyitni a web konzolt Firefox és Chrome alatt?

- F12 (Chrome és Firefox)
- Open Menu > Developer > Web Console (Firefox)
- Ctrl + Shift + K (Firefox)
- Ctrl + Shift + I (Chrome)

2. Deklarálhatsz egy változót var kulcsóval, vagy anélkül. Melyiket kellene használnod?

- A var kulcsszóval deklarált változó a helyi szkópba kerül, míg var nélküliek a globális szkópba. Mindig használjunk var kulcsszót, hogy ne szemeteljük össze a globális szkópot.

3. Miért írjunk mindig kommenteket

- Nagyon fontos, hogy megérthessük a programkód kiemelt pontjait, akár ha a későbbiekben visszatérsz egy régi projektedhez, akár ha egy csapat tagjaként dolgozol a szoftveren.
- Ez egy konvenció, hogy időlegesen hatályon kívül helyezzük a programkód egyes részeit, anélkül hogy ki kéne őket törölni és levesznének.

Feltételek

1. Van-e logikailag különbség egy if...else szerkezet és 2 egymás követ if használata között, ha azok feltételei egymás ellentétei.

- Nincs különbség, de az if...lese messzemenően jobb választás, mert átláthatóbb és kisebb a hibázási lehetőség benne.

2. Hogyan lépkedsz végig egy tömb elemein, ha nem tudod a tömb hosszát?

- A tömb .length tulajdonságát használom

3. Melyik a hatékonyabb a következő kódrészletek közül?
```
for (var i = 0, len = a.length; i < len, i++) { /* ... */ }
for (var i = 0; i < a.length, i++) { /* ... */ }
```

- A len változó csak egyszer értékelődik ki, nem minden iterációban

Függvények

1. Fontos-e a „divide” nevű függvények argumentumainak sorrendje (az a függvény, ami visszatér az első és a második paraméter hányadosával)?

- Igen, módosítja a végeredményt

2. Mi különbség a függvényhívás és a függvény referencia között?

- A függvény hívás (a függvény nevét követő zárójelekkel) a függvény kódjának azonnali futtatását kéri a hivás helyén. A függvény referencia (pl. egy call back-ben) egy egyszerű letárolását jelenti egy függvény nevének későbbi felhasználásra.

3. Mi a fenntartott kulcsszó, ami megadja, milyen értékkel térjen vissza a meghívott függvény?

- Return

DOM

1. Melyik metódusok nem az első vagy az összes előfordulást célozzák?

- document.querySelector()
- document.getElementById()

2. A következő kódban milyen argumentumot kap a callback?
```
document.getElementById('myID').addEventListener('click', myCallBack);
```

- Magát az eseményt, ami aktiválta a callback-et

3. Hogyan tudod elkérni az értékét egy HTML kontrollnak (UI), mint pl. egy szöveges beviteli mezőnek?

- A .value tulajdonságon keresztül, ami visszatér a 'value' attribútum tartalmával.

CSS

1. Milyen CSS szelektor illeszkedik minden olyan paragrafusra,a mi közvetlenül egy div alá tartozik?

- Div > p

2. Milyen CSS szelektor illeszkedik arra, hogy az első paragrafus függetlenül attól, hogy hány testvére van a HTML node-ok között.

- p:nth-of-type(1)

3. Ha két CSS szelektor (.red és #blue) ugyan azt az elemet célozzák meg, milyen lesz a színe?

- Kék

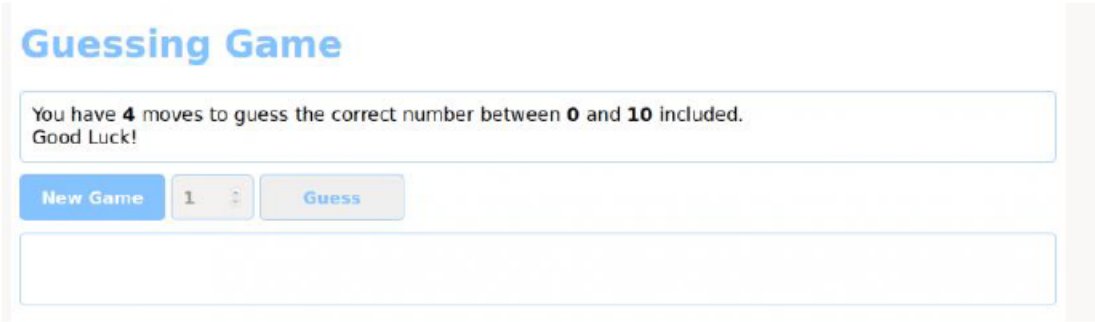
Kódolási labor: Számkitalálós játék

Nos miután megértettük a HTML/CSS/JavaScript alapjait, itt az idő egy kis mókára, írjuk meg az első játékprogramunkat.

Ebben a leckében újra fogjuk írni a számkitalálós játékot, ami azt jelenti, hogy létre fogunk hozni

1. egy HTML fájlt, ami kirak minden elemet, amire a játékhoz szükség van
2. egy CSS fájlt, ami stílust fog adni a játéknak (ahogy szeretnéd)
3. egy JavaScript fájlt, ami minden függvényt tartalmaz, ami a játékhoz kell

Ez egy egyszerű kinézet (nem szükségszerű) a játék számára:



A képernyő felépítése:

Játék címe: SZÁMKITALÁLÓS JÁTÉK

Játékismertető: 4 próbálkozásod van, hogy kitaláld a megfelelő számot 0 és 10 között. Sok szerencsét!

Gomb: Új játék
Legördülő lista: 1 10
Gomb: Tipp

A következő oldalon találsz a megfoldáson lépésről-lépésre végigvezető útmutatót, ha szükséges van rá …

Vágjunk bele

1. Szükségünk lesz legalább egy numerikus kiválasztóra és egy gombra, ami ellenőrzi az eredményt. De ezen kívül kelleni fog néhány 'konténer', melyek rejtettek vagy üresek és melyek elvégzik az eredmények naplózását.

Azon el lehet gondolkodni felhasználói és programozói szempontból, hogy szükség van-e az 'Új játék' gombra. Ha gondolod a feladat végén módosíthatod úgy a játékot, hogy a játékos feladhassa a játékot.

2. El kellene gondolkodni ennek a kis játék megvalósításának lépésein. Ezek fogják adni a kódod különböző blokkjait. Ha egy probléma túl nehéznek tűnik, akkor újra elő kell venned és kisebb részekre kell bontani.

Ha egy kódrészletet/logikát többször is fel akarunk használni, akkor a legjobb, ha kiszervezzük egy függvénybe különösen figyelve az induló paraméterekre, melyek változhatnak.

Mindig próbáljunk változókat használni a forráskódba beégetett értékek helyett. A változókkal könnyen lehet változtatni a játék paramétereit, mint pl. a próbálkozások számát. És ezen felül olvashatóbbá, konfigurálhatóbbá és fenntarthatóbbá válik a kód.

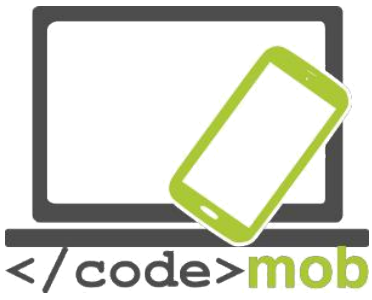
Mindig kommentezd fel a kódodat (megérteni a kód mögötti logikát) a jövőbeni önmagadnak ;) és a csapttársaidnak.

3. getRandomIntegerBetween() egy egyszerű példafüggvény arra, hogyan lesz könnyen újrafelhasználható és átlátható a kód egy darabja.

A valódi véletlen számok generálását nem tekinthetjük egy triviális problémának a számítástudományban. A számítógépek determinisztikusak, ami azt jelenti, hogy ha felteszed ugyan azt a kérdést mindig ugyan azt a választ fogod kapni. Valójában sz ilyen gépeket speciálisan és nagy gonddal úgy készítik, hogy kiküszöböljék a véletlent az eredményekből. Nézz utána a pszeudo-véletleneknek a Google-ön.

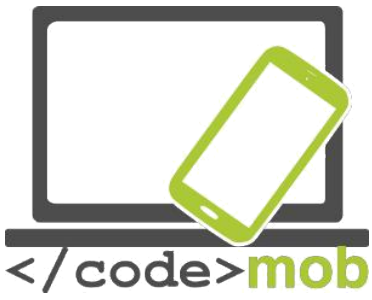
http://engineering.mit.edu/ask/can-computer-generate-truly-random-number

A checkResponse() függvényben láthatjuk, hogyan kell visszatérni egy függvényből anélkül, hogy bármit is visszaadna. Zárjuk rövidre a „játék vége” ellenőrzésnél, amikor a játékot már megnyerte a felhasználó.



Végezetül vegyük át azokat a webes technológiákat, a JavaScriptet, mellyel elérjük és módosítani tudjuk a HTML DOM fát (Document Object Model), pl. a document.getElementById('usertGuess') és a resultP.innerHTML vagy a CSS-t. Mindezek relatíve egyszerű technológiák, melyeket ötvözünk.

A következő oldalon található néhány iránymutatás a struktúrára és a függvényekre ...



Struktúrák és függvények

```
// Visszatér egy egész számmal a megadott minimum és maximum értékek között
    function getRandomIntegerBetween(min, max)

 // Aktiválja a felhasználói felületet egy új játékhoz
    function activateUI()

// Véga a játéknak: deaktiválja a felhasználói felületet
    function deactivateUI()

    function init(){
        // Maximális érték beállítása
        // Naplózás törlése
        // Új véletlenszám előállítása
        // Csalás ;)
        activateUI();
    }

// A válasz ellenőrztése
    function checkResponse()

    // A játékos elhasználta-e a rendelkezésére álló probálkozásokat

    // A játékos eltalálta a véletlenszámot

    // Minimális és maximális értékek, amik között választani kell

    // Megmutatni a játékszabályokat

    // A játkos játékának naplózása

    // Felhasználói felület

    // Tudsz mondani egy optimális stratégiát, amivel mindig meg lehet nyerni ezt a
játékot?
```

Referenciák és források

JAVASCRIPT

[EN] JavaScript
http://www.w3schools.com/js/default.asp
http://www.w3schools.com/js/js_performance.asp http://eloquentjavascript.net
https://developer.mozilla.org/en-US/en%C2%ADUS/docs/Web/JavaScript
https://developer.mozilla.org/en/docs/Web/API/Event
https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

[EN] JavaScript stílus útmutató
http://www.w3schools.com/js/js_conventions.asp

JavaScript teljesen NEM-programozóknak
http://www.webteacher.com/javascript/

Defer & async [EN]
http://www.growingwiththeweb.com/2014/02/asyncvsdeferattributes.html

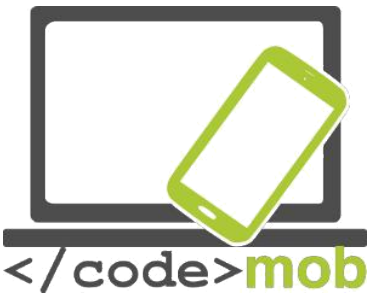
Modernizr
http://modernizr.com/

Mi az var kulcsszó a függvényeknél és mire használjuk?
http://stackoverflow.com/questions/1470488/what-is-the-purpose-of-the-var-keyword-and-when-to-use-it-or-omit-it

[EN] A népszerűtlen 'this' kulcsszó
https://www.sitepoint.com/mastering-javascripts-this-keyword/

[EN] ECMA 6 A JavaScript jövője
http://es6-features.org/#Constants

A JavaScript Source egy kiváló forrása copy-paste (kijelöl és átmásol) programkódok tonnáihoz.
JavaScript példakódok és mind szabadon felhasználható.
http://www.javascriptsource.com/



HTML és CSS

HTML5 Útmutató
http://www.w3schools.com/html/default.asp

CSS3 Útmutató
http://www.w3schools.com/css/default.asp

És egy halom más a Google-ö,, YouTube-on és az egész web-en!
Köszönet David Collignon-nak a hathatós segítségéért és megjegyzéseiért a kurzussal kapcsolatban!