



CodeBerry

A programozás alapjai

Bevezetés játékosan, közérthetően,
JavaScript nyelven.



Kedves Diákunk!

Ahogy a CodeBerry Programozóiskola végleg bezárja kapuit, szeretnénk megosztani veled búcsújándékként legkedveltebb kurzusunk anyagát. Ez a dokumentum egyúttal azonban több is, mint tananyag – emlék arról a kalandos utazásról, amelyet együtt tettünk meg a programozás világában.

2016. és 2024. között izgatott tanúi lehettünk annak, ahogy diákjaink megtettek első lépéseiket a programozás világában, és elkezdték felfedezni annak transzformatív erejét. Büszkék vagyunk arra, hogy a nyolc év alatt 200 000 tanulót érhettünk el 43 különböző országból, és hálásak vagyunk minden egyes diáknaknak, akik hosszabb távon is elköteleződtek mellettünk és a kódolás mellett.

Ez a dokumentum szenvedélyünk és elkötelezettségünk emléke. Bízunk benne, hogy – legalábbis még egy jó darabig – nem csak nosztalgikus értéke lesz, hanem támogathat a tanulásban, és új dolgokra is inspirálhat. A CodeBerry talán bezár, de a tanulás szerencsére sosem ér véget.

Köszönjük, hogy részese voltál ennek a történetnek.

Búcsúzik:

--

a CodeBerry hálás csapata.

U.i.: Jó szívvvel ajánljuk tananyagunkat akkor is, ha korábban nem JavaScriptet tanultál nálunk. A programozás alapvető koncepciói univerzálisak.

U.u.i.: Ezt a dokumentumot neked, diáknak szántuk, hálánk jeléül. Használ egészséggel, okozzon sok kellemes, fejtörésben gazdag percet, és kérünk, hogy ne kérj érte mástól pénzt.

U.u.u.i.: A technológia fejlődik, az internet változik, így a dokumentumban szereplő állítások idővel érvényüket veszíthetik, a hivatkozott linkek pedig sajnos eltörhetnek, ráadásul jelen formátum nem teszi lehetővé mindazt, amit a tanulófelületünkön megszokhattál. Reméljük, mindezzel együtt is értéket adhat számodra.

Tartalomjegyzék

Tartalomjegyzék.....	3
Kódszerkesztők.....	10
1. Kódszerkesztők.....	10
1.1. Miért használunk kódszerkesztőket?.....	10
1.2. Kódszerkesztő online I.....	10
1.3. Kódszerkesztő online II.....	14
1.4. Kódszerkesztő a gépeden.....	16
1.5. Ajánlataink.....	17
JavaScript alapok I. (adattípusok).....	19
1. Bevezetés.....	19
1.1. Mi az a JavaScript?.....	19
1.2. A .js fájl és a konzol.....	19
1.3. A JS Bin JavaScript-panele.....	21
2. A JavaScript adattípusai.....	22
2.1. A JavaScript adattípusai.....	22
3. Játék számokkal.....	23
3.1. Számtani (arithmetic) operátorok.....	23
3.2. Gyakorlás: A JavaScript mint zsebszámológép.....	24
3.3. NaN (Nem szám).....	25
3.4. JavaScript-kifejezések (expressions).....	26
3.5. Összetett műveletek JavaScripttel.....	27
3.6. Összehasonlító (comparison) operátorok.....	27
4. Játék szavakkal.....	29
4.1. String.....	29
4.2. Escaping.....	30
4.3. Új sor.....	32
4.4. Stringek összefűzése.....	33
4.5. Stringek összehasonlítása egymással.....	34
4.6. Eltérő adattípusok nem szigorú összehasonlítása.....	35
4.7. Eltérő adattípusok szigorú összehasonlítása.....	36
5. Összefoglalás.....	37
5.1. Mit tanultál eddig?.....	37
6. Teszt.....	38
6.1. Ellenőrizd a tudásod!.....	38
Olvass, keress, kérdezz.....	39
1. Tanuld meg kihúzni magad a gödörből.....	39
1.1. Olvass (Read).....	39
1.2. Keress (Search).....	39
1.3. Kérdezz (Ask).....	41
1.4. Összefoglalás.....	42
JavaScript alapok II. (változók).....	43
1. Bevezetés.....	43

1.1. Mi az a változó?.....	43
1.2. Miért használunk változókat?.....	44
1.3. Változó a nem ismert érték helyén.....	45
1.4. Újrafelhasználható kód változókkal.....	46
2. Változók létrehozása (deklarálása).....	47
2.1. Egy változó anatómiája.....	47
2.2. JavaScript-kulcsszavak (keywords).....	48
2.3. JavaScript-utasítások (statements).....	49
2.4. Kommentek a JavaScriptben.....	50
2.5. A változók elnevezésére vonatkozó szabályok és szokások.....	51
3. Értékek (adattípusok) a változókban.....	52
3.1. Az üres változó.....	52
3.2. Változók valódi értékekkel.....	53
3.3. Kifejezések változókban és változók kifejezésekben.....	55
3.4. Változók használata változókban.....	56
3.5. A változók értékének módosítása.....	57
4. Változók deklarálása a JavaScript legújabb verziójában.....	59
4.1. Új kulcsszavak: let és const.....	59
4.2. Változó deklarálása a let kulcsszóval.....	59
4.3. Változó deklarálása a const kulcsszóval.....	61
5. Összefoglalás.....	63
5.1. Mit tanultál eddig?.....	63
6. fejezet: Teszt.....	63
6.1. Ilecke: Ellenőrizd a tudásod!.....	63
JavaScript alapok III. (függvények).....	64
1. Bevezetés.....	64
1.1. Mik azok a függvények?.....	64
2. Függvények létrehozása és használata.....	66
2.1. Függvények deklarálása.....	66
2.2. Egy függvénydeklaráció.....	67
2.3. Függvények meghívása.....	68
2.4. Egy csepp kitekintés.....	69
2.5. Mi olvassa el a JavaScriptet a böngészőben?.....	69
2.6. Milyen sorrendben hajtódik végre a JavaScript-kód?.....	70
3. Rugalmas függvények paraméterekkel.....	72
3.1. Variációk egy téma.....	72
3.2. Paraméterek használata a függvényekben.....	73
3.3. Argumentum: a paraméter értéke.....	75
3.4. Hárrom megjegyzés a paraméterekről és az argumentumokról.....	76
3.5. Gyakorlás: szendvicsreceptfüggvény.....	77
4. A függvények eredménye.....	78
4.1. Return value (visszaadott érték) és side effect (mellékhatás).....	78
4.2. Gyakorlás: calculateRectangleArea függvény.....	79
4.3. Fókuszon a visszaadott érték (return value).....	79
4.4. A return kulcsszó.....	80
4.5. return = stop!.....	82
4.6. Gyakorlás: életkor-kalkulátor.....	82

5. Szerepkörök szétválasztása függvényekkel.....	83
5.1. Miért használunk függvényeket?.....	83
5.2. Szétválasztott területszámítás.....	84
5.3. Tetszőleges szélesség, tetszőleges hosszúság.....	86
5.4. Egymásnak passzolt argumentumok és a control flow.....	87
5.5. Gyakorlás: körök kerülete és területe.....	89
6. Scope-ok (területek).....	89
6.1. Globális és lokális scope-ok.....	89
6.2. Scope-ok a gyakorlatban.....	93
7. Összefoglalás.....	95
7.1. Mit tanultál eddig?.....	95
8. Teszt.....	95
8.1. Ellenőrizd a tudásod!.....	95
JavaScript alapok IV. (ciklusok).....	97
1. Bevezető a ciklusokhoz.....	97
1.1. Bevezetés – Ciklusok és Hófehérke dilemmája.....	97
1.2. Bevezetés – Átköltözés a JSBinbe és egy kis fájlszerkezet.....	97
1.3. Bevezetés – Pontosvessző és console.log.....	98
2. A while ciklus.....	100
2.1. Ciklusok – Bevezetés.....	100
2.2. Ciklusok – A while ciklus (while loop) felépítése.....	100
2.3. Ciklusok – Állítások megfogalmazása JavaScripttel.....	101
2.4. Ciklusok – A while ciklus alkalmazása a Hófehérke problémánakra.....	102
2.5. Ciklusok – A while ciklus, ami el tud számolni hétag.....	102
2.6. Ciklusok – A ciklust kontrolláló állításunk.....	103
3. A for ciklus.....	104
3.1. Ciklusok – A for ciklus (for loop).....	104
3.2. Ciklusok – A törpök, akik még alszanak és a ciklusok kombinálása.....	106
3.3. Ciklusok – Jövőbiztos ciklusok.....	108
4. Összefoglalás.....	110
4.1. Összefoglalás.....	110
JavaScript alapok V. (feltételek).....	112
1. Bevezető a feltételekhez.....	112
1.1. Bevezetés – Feltételek a JavaScriptben.....	112
2. Egyszerű feltételek.....	112
2.1. Egyszerű feltételek 1. – Egy online magazin és az if / else állítások.....	112
2.2. Egyszerű feltételek 2. – if és a barátja, else.....	113
2.3. Egyszerű feltételek 3. – prompt().....	114
2.4. Egyszerű feltételek 4. – alert().....	116
2.5. Egyszerű feltételek 5. – else if állítások és egy kiegészítő kérés.....	117
2.6. Egyszerű feltételek 6. – Egy kész megrendelés.....	119
3. Feltételek és ciklusok.....	120
3.1. Feltételek és ciklusok 1. – Bevezetés.....	120
3.2. Feltételek és ciklusok 2. – for, if / else és a páros-páratlan számok.....	121
3.3. Feltételek és ciklusok 3. – A szélerőmű-megrendelés.....	123
3.4. Feltételek és ciklusok 4. – Egy lehetséges megoldás.....	124
4. Összefoglalás.....	125

4.1. Feltételek (conditionals) – Összefoglalás.....	125
5. Teszt.....	126
5.1. Ellenőrizd a tudásod!.....	126
Programozóként gondolkodni.....	127
1. A Marson rekedve.....	127
1.1. Bevezetés.....	127
1.2. A három űrhajós dilemmája.....	127
1.3. Problémamegoldás programozó módra.....	128
1.4. A szükséges üzemanyag-mennyiség megállapítása.....	128
1.5. Annak kiszámítása, hogy mennyi üzemanyagot adhatunk le.....	129
2. A munkakörnyezet felállítása.....	130
2.1. A fájlok létrehozása.....	130
2.2. A JavaScript és HTML fájl összekapcsolása.....	130
2.3. A kapcsolat ellenőrzése.....	131
3. Problémamegoldás lépésről lépésre.....	132
3.1. Gondolkodj, mielőtt kódolsz.....	132
3.2. Az első megjegyzések.....	132
3.3. A jegyzetek véglegesítése.....	133
3.4. Az első lépés leprogramozása.....	134
3.5. Az első lépés visszaellenőrzése.....	136
3.6. A második lépés megoldása.....	137
4. Felhasználói felület.....	139
4.1. Felhasználói felületre márpedig szükség van.....	139
4.2. További megjegyzések hozzáadása.....	140
4.3. Értékek bekérése.....	141
4.4. Felesleges paraméterek eltávolítása.....	142
4.5. Az eredmények megjelenítése.....	143
4.6. Végső simítások.....	145
5. Hazaút.....	147
5.1. A számológép használata élesben.....	147
5.2. Összefoglalás.....	148
Az alapok gyakorlása.....	150
1. Bevezető a gyakorlófeladatokhoz.....	150
1.1. Gyakorlás rajzolással.....	150
1.2. A canvas elem.....	150
1.3. fillRect().....	152
1.4. fillStyle.....	153
1.5. moveTo() és lineTo().....	154
1.6. A canvasWidth és a canvasHeight.....	156
2. Bemelegítő feladatok.....	157
2.1. Egy magányos négyzet.....	157
2.2. Két átfedő négyzet.....	158
2.3. Zászló.....	159
2.4. Átlók.....	160
2.5. Egy háromszög.....	161
2.6. Felezővonalak.....	162
3. Ciklusok gyakorlása.....	163

3.1. DRY négyzetek.....	163
3.2. Sorok.....	164
3.3. Színskála.....	165
3.4. Háromszögalagút.....	166
3.5. Szivárványszínű téglalapok.....	167
4. Feltételek gyakorlása.....	168
4.1. Fizzbuzz négyzetek.....	168
4.2. Páratlan piramis.....	169
4.3. Mennyi háromszög.....	170
5. Függvények gyakorlása:.....	171
5.1. triangle().....	171
5.2. star().....	172
5.3. lineToCenter().....	173
6. Haladó gyakorlatok.....	175
6.1. A JavaScript telepesei.....	175
6.2. drawCheckeredPattern().....	176
6.3. Tetractys.....	177
JavaScript alapok VI. (tömbök).....	179
1. Mi az a tömb?.....	179
1.1. Bevezetés.....	179
1.2. Tömb írása.....	179
1.3. Tömb elmentése.....	180
1.4. Adatok tárolása tömbökben.....	181
2. Elemek hozzáadása és törlése egy tömb elejéről vagy végéről.....	182
2.1. Új elem hozzáadása egy tömb végéhez.....	182
2.2. Több új elem hozzáadása egy tömb végéhez.....	183
2.3. Új elemek hozzáadása egy tömb elejéhez.....	185
2.4. Az utolsó elem törlése.....	186
2.5. Az első elem törlése.....	188
2.6. Marcsi bakancslistája.....	189
2.7. Példamegoldás a bakancslistára.....	190
2.8. Összefoglalás.....	190
3. Egy tömb egy elemének megváltoztatása.....	191
3.1. Hozzáférés a tömb egy eleméhez.....	191
3.2. Egy tömb egy elemének megváltoztatása.....	192
3.3. Jancsi vendéglistája.....	193
3.4. Példamegoldás a vendéglistára.....	194
4. Elemek hozzáadása és törlése bárhonnan egy tömbből.....	194
4.1. Elem törlése bárhonnan.....	194
4.2. Több elem törlése bárhonnan egy tömbből.....	196
4.3. Új elemek hozzáadása bárhol egy tömbben.....	198
4.4. Kati könyvlistája.....	200
4.5. Példamegoldás a könyvlistára.....	200
4.6. Összefoglalás.....	201
5. Egy tömb összes elemének megváltoztatása.....	201
5.1. Egy tömb hossza.....	201
5.2. minden elem megváltoztatása egyesével.....	202

5.3. Az összes elem megváltoztatása for ciklussal.....	204
5.4. Hogyan változtatta meg a for ciklus a tömböt?.....	204
5.5. A tömb hosszának használata a for ciklusban.....	206
6. Összefoglalás.....	209
6.1. Összefoglalás.....	209
7. Teszt.....	209
7.1. Ellenőrizd a tudásod!.....	209
JavaScript alapok VII. (objektumok).....	211
1. Mi az az objektum?.....	211
1.1. Bevezetés.....	211
1.2. Objektum létrehozása.....	211
1.3. Objektum mentése.....	213
1.4. Tulajdonságok neve és tulajdonságok értékei.....	214
2. Objektumtulajdonságok elérése.....	216
2.1. Pontjelölés.....	216
2.2. Zárójeljelölés.....	217
2.3. Beágyazott objektumok.....	219
2.4. Összefoglalás.....	220
3. Objektumok tulajdonságainak megváltoztatása, hozzáadása és ellenőrzése.....	221
3.1. Objektum tulajdonságának megváltoztatása.....	221
3.2. Új objektumtulajdonság hozzáadása.....	223
3.3. Objektum tulajdonságának ellenőrzése.....	224
3.4. Összefoglalás.....	226
4. Objektummetódusok.....	227
4.1. Objektummetódus létrehozása.....	227
4.2. Objektum Metódus elérése.....	228
4.3. Oldalszám elmentése egy tulajdonságba.....	229
4.4. Az elolvasott oldalak számának kiszámolása.....	231
4.5. A kiírt üzenet megértése.....	233
4.6. A megfelelő üzenet megjelenítése.....	234
4.7. Összefoglalás.....	236
5. A this kulcsszó és a paraméterek használata metódusokban.....	236
5.1. Több könyv elmentése a katalógusba.....	236
5.2. A this kulcsszó használata.....	238
5.3. A page marker metódus frissítése.....	239
5.4. A könyvkatalógusod építése.....	240
5.5. Paraméterek használata objektummetódusokban.....	242
5.6. Összefoglalás.....	245
6. Végigiterálás egy objektumon a for...in használatával.....	246
6.1. Végigiterálás a katalógusodon a for...in használatával.....	246
6.2. Változó használata tulajdonság eléréséhez.....	247
6.3. Olvasatlan könyveid kiíratása.....	249
6.4. Az objektumok referenciátipusok.....	251
7. Újonnan szerzett tudásod gyakorlása.....	253
7.1. Tulajdonságot megváltoztató metódus létrehozása.....	253
7.2. A metódus példamegoldása.....	253
7.3. Idézetet hozzáadó metódus létrehozása.....	254

7.4. A metódus példamegoldása.....	255
7.5. A kölcsönvett könyveket listázó metódus létrehozása.....	255
7.6. A metódus példamegoldása.....	256
8. Összefoglalás.....	257
8.1. Összefoglalás.....	257
9. Teszt.....	258
9.1. Ellenőrizd a tudásod!	258

Kódszerkesztők

1. Kódszerkesztők

1.1. Miért használunk kódszerkesztőket?

Az elkövetkezendő leckék során megnézzük, mik is azok a kódszerkesztők és miért érdemes használni őket, ha programozásra adtad a fejed. Vágunk is bele!

Végső soron bármelyik program szövegből épül fel, amelyet valahol meg kell írni és utána fájlba menteni.

Elsőként eszünkbe juthat, miért ne használhatnánk egy egyszerű szövegszerkesztőt, mint a Jegyzettömb, vagy a Wordöt egy program megírására? A szövegszerkesztők tökéletesen alkalmasak egy egyszerű dokumentum megírásakor, azonban a programozáshoz sajnos kevés a tudásuk. Itt jönnek be a képbe a kódszerkesztők.

Kódszerkesztőnek nevezzük azt a speciális programot, amely segít nekünk a programok megírásában. Ezek a programok a kódok megírásán kívül rengeteg kiegészítő eszközzel is rendelkeznek, például:

- intelligensen kezelik a megírt kódokat, azaz
 - kiegészítik a megírt kódot, így nem kell minden karaktert beötvözni,
 - jelzik, ha helytelen kódot írtunk,
- le tudják futtatni a megírt kódokat, azaz a programot.

Emellett még a kód átláthatóságát is segíti a fontos kulcsszavak színes kiemelésével és a különböző részek elkülönítésével.

A kódszerkesztők ezáltal jelentősen gyorsítják a fejlesztéseket és megkönnyítik az életünket. Pontosan emiatt mutatjuk be neked a kódszerkesztők két nagy csoportját a következő leckék során:

- Online → az interneten található és a böngészőben futó kódszerkesztők,
- Offline → a gépeden futó kódszerkesztők.

Megjegyzés: A leckék során a célunk a kódszerkesztők bemutatása, viszont nem szeretnénk egy konkrét program mellett lándzsát törni. Rengeteg különböző kódszerkesztő létezik, a saját előnyeikkel és hátrányaikkal. Emiatt általánosságban fogunk a kódszerkesztőkről beszélni, viszont az utolsó lecke során teszünk néhány ajánlást, amelyek szerintünk az adott programnyelvekhez legalkalmasabbak. Nyugodtan próbálj ki más kódszerkesztőket is, mindenkinek más az ízlése és lehet, hogy neked más válik majd be.

1.2. Kódszerkesztő online I.

Elsőként nézzük meg az online kódszerkesztőket. Ezek olyan kódszerkesztők, amelyek az interneten találhatóak. A használatukhoz nem kell semmit telepítened a gépedre, mivel a böngészőből futnak. A legtöbbük ingyenesen használható, azonban előfordulhat, hogy be

kell majd regisztrálnod a használatuk előtt.

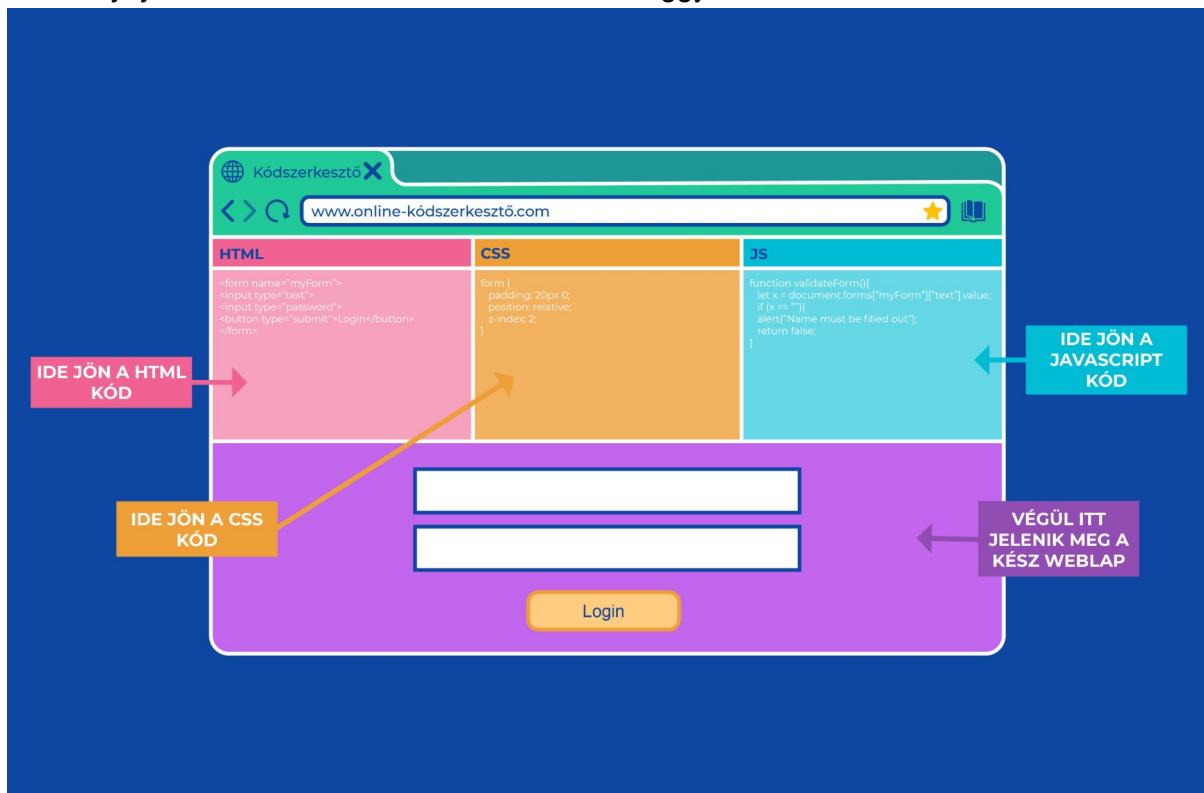
Az online kódszerkesztőket két csoportra fogjuk osztani:

- Csak weboldalak készítésére alkalmas kódszerkesztők
- Bármilyen típusú program fejlesztésére alkalmas kódszerkesztők

Ebben a leckében a weboldalak készítésére alkalmas kódszerkesztőket nézzük meg és még egy apróbb feladat is vár rád a lecke végén.

Megjegyzés: Fontos kiemelni, hogy az online kódszerkesztőket legtöbbször apróbb kódrészletek kipróbálására használják. A tényleges, összetett programokhoz asztali kódszerkesztőket célszerű igénybe venni.

A webfejlesztéshez kialakított kódszerkesztők leggyakrabban az alábbi módon néznek ki:



Megjegyzés: A kis panelek elrendezése bárhogyan személyre szabható és méretezhető. Egy-egy panel akár el is rejthető.

Ha látni szeretnéd a megírt kódod eredményét, akkor le kell futtatnod azt. Ezt a legtöbbször a Run gombbal tudod megtenni. Nézz körül a szerkesztőprogramodban, ott kell lennie valahol.

Feladat

Most hogy már tudod hogyan épülnek fel a weboldalak fejlesztésével foglalkozó online kódszerkesztők, így most adunk neked egy lehetőséget, hogy megtekintsd élesben is a működését.

Megjegyzés: A következőkben adunk neked egy jó pár sor számodra ismeretlen kódot. Teljesen rendben van ha még nem érted mit jelentenek ezek. A hangsúly most azon van, hogy képes legyél boldogulni egy még számodra ismeretlen eszközzel, úgy hogy a felépítéséről szóló tudás már a kezedben van.

Első lépésként nyisd meg [ezt az oldalt](#). Ha figyelmesen olvastad végig a leckét, akkor már észre is vettek, hogy rendelkezik minden részkellem, amelyeket a lecke elején bemutattunk.

Hogy ne csak az üres kód szerkesztőt nézegesd, lentebb találhatsz pár sor kódot, amelyeket a megfelelő helyre beillesztve egy szuper oldal fog eléd tárulni.

A **html** kód:

```
<div class="bg-image"></div>

<div id="welcome" class="bg-text">
  <h1>Üdv a CodeBerryben!</h1>
  <button id="start" class="btn default">Vágunk bele!</button>
</div>

<div id="congratulation" class="bg-text">
  <h1>Gratulálunk!</h1>
  <p>Sikerült megcsinálnod az első oldaladat!</p>
</div>
```

A **css** kód:

```
body, html {
  height: 100%;
}

* {
  box-sizing: border-box;
}

.bg-image {
  background-image: url('https://images.unsplash.com/photo-1506874101255-0b89005ee4ad?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYyNDk4MDkwMg&ixlib=rb-1.2.1&q=85');
  filter: blur(5px);
  -webkit-filter: blur(5px);
```

```
height: 100%;  
  
background-position: center;  
background-repeat: no-repeat;  
background-size: cover;  
}  
  
.bg-text {  
background-color: rgb(0,0,0);  
background-color: rgba(0,0,0, 0.4);  
color: white;  
font-family: Raleway, sans-serif;  
font-weight: bold;  
border: 3px solid #f1f1f1;  
position: absolute;  
top: 50%;  
left: 50%;  
transform: translate(-50%, -50%);  
z-index: 2;  
width: 80%;  
padding: 20px;  
text-align: center;  
}  
  
.btn {  
font-family: Raleway, sans-serif;  
font-weight: bold;  
border: 2px solid black;  
background-color: white;  
color: black;  
padding: 10px 20px;  
font-size: 16px;  
cursor: pointer;  
border-radius: 5px;  
}  
  
.default {  
border-color: #e7e7e7;  
color: black;  
}  
  
.default:hover {  
background: #e7e7e7;
```

```
}
```

```
#congratulation {
    display: none;
}
```

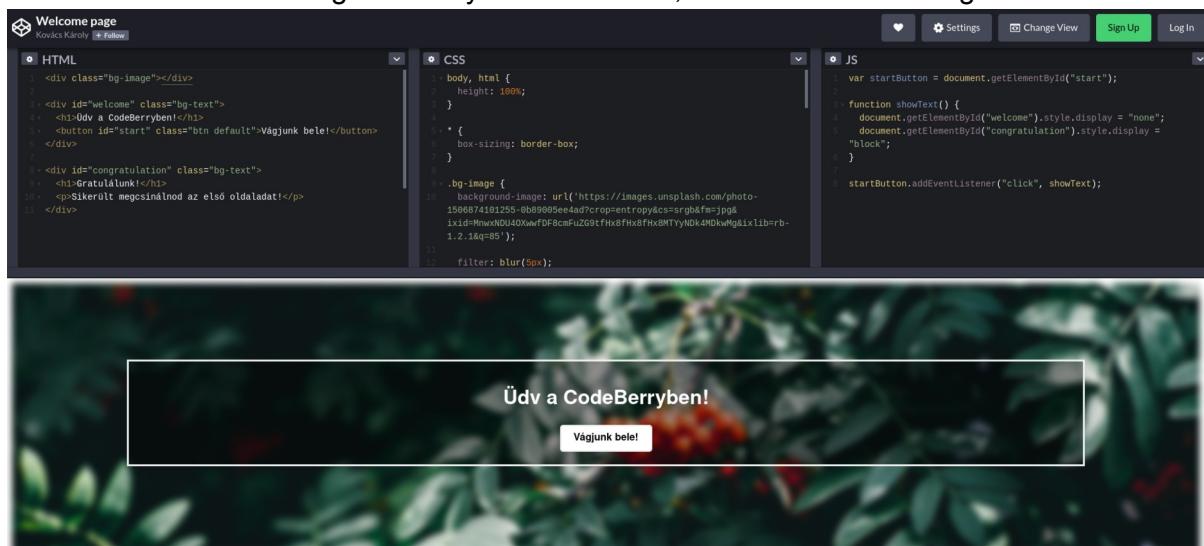
A javascript kód:

```
var startButton = document.getElementById("start");

function showText() {
    document.getElementById("welcome").style.display = "none";
    document.getElementById("congratulation").style.display =
    "block";
}

startButton.addEventListener("click", showText);
```

Ha sikerült minden a megfelelő helyre bemásolnod, akkor ezt az oldalt fogod látni:



1.3. Kódszerkesztő online II.

Az online kódszerkesztők másik nagy csoportjába azok a kódszerkesztők tartoznak, amelyeket nem csak weboldalak fejlesztéséhez készítettek. Ezekben például Python vagy Java programokat tudsz elkészíteni. Mivel itt nincsen szükség megjelenítésre, mint a weboldalak esetén, ezért általában kevesebb panelt fogsz látni.



A kódszerkesztők általában két részre vannak osztva:

- **Editor** → a szerkesztőfelület, ide írod a kódot
- **Output** → itt jelenik meg a kód eredménye. Ez a különböző programnyelvek esetén más-más elnevezésű lesz.

Feladat

Most, hogy már tudod, hogyan épülnek fel az online kódszerkesztők, adunk neked egy lehetőséget, hogy élesben is lásd a működésüket.

Első lépésként nyisd meg [ezt az oldalt](#). Ha figyelmesen olvastad végig a leckét, akkor már észre is vettek, hogy rendelkezik minden részükkel, amelyeket a lecke során bemutattunk.

Hogy ne csak az üres kódszerkesztőt nézegesd, másold be az alábbi kódsort a megfelelő panelbe. Előfordulhat, hogy a panelen belül találsz már valamilyen kódot. Ezeket nyugodtan töröl ki, most nem lesz rájuk szükség.

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    return x / y

print("Válassz egy műveletet:")
```

```
print("\t1. Összeadás")
print("\t2. Kivonás")
print("\t3. Szorzás")
print("\t4. Osztás")

while True:
    choice = input("Válassz egyet(1/2/3/4): ")

    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Üsd be az első számot: "))
        num2 = float(input("Üsd be a második számot: "))

        if choice == '1':
            print("\t", num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print("\t", num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print("\t", num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print("\t", num1, "/", num2, "=", divide(num1, num2))
        break
    else:
        print("Rossz válasz.")
```

Ha sikeresen bemásoltad a fenti kódsortokat és lefuttatod a Run gomb segítségével, akkor egy egyszerű számológép fog megjelenni a jobb oldali panelben. Próbálgasd ki, miket tud!

1.4. Kódszerkesztő a gépeden

A kódszerkesztők azonban nem csak a böngészőben futhatnak, hanem a gépeden is. Ebben a leckében ezeket a kódszerkesztőket nézzük át.

A számítógépen futó kódszerkesztőket asztali kódszerkesztőnek hívjuk.

Amikor egy kódszerkesztő a gépeden fut, akkor sokkal több eszközzel fog rendelkezni, mint a böngészőből futó társa. Ez abból adódik, hogy a gépedet tudja használni és nem függ egy külső, interneten található szolgáltatástól. Ilyen plusz lehet például több fájl együttes kezelése, kódkiegészítés vagy a különböző típusú fájlok kezelése.

Pontosan emiatt **amikor komplikáltabb programot készítesz, célszerű áttérni egy asztali kódszerkesztőre.**

Amikor megnyitsz egy asztali kódszerkesztőt, akkor legtöbbször az alábbi kinézet jön veled

szembe:



A bal oldalon lévő panel mutatja az éppen aktuális projektet, a hozzá tartozó fájlokkal. Itt tudsz változatot hozni a fájlok között és új fájlokat létrehozni.

Minden esetben lesz egy szerkesztőfelület, ahova a konkrét kódot írhatod. Az időd nagy részében itt fogsz dolgozni.

Végül pedig van az Output panel. A feladata a megírt kód eredményének megjelenítése. Ez a megjelenítés lehet például egy konkrét weboldal, a konzol vagy akár egy kép előnézete is.

Megjegyzés: Az asztali kódszerkesztőkhöz akár kiegészítőket is fel lehet telepíteni, amelyek segítségével új eszközöket kap. Ilyen például egy megosztásra képes kis kiegészítő, amellyel más programozók is élőben láthatják és szerkeszthetik a kódokat.

Tipp: Asztali kódszerkesztő használata esetén célszerű lesz majd rendszerezned a programaidat, például külön mappákba rakni a különböző projekteket.

1.5. Ajánlataink

Ahogy megígértük az elején, az utolsó leckében ajánlunk neked pár konkrét kódszerkesztőt.

Megjegyzés: Ez a felsorolás a mi szubjektív listánk. Simán előfordulhat, hogy találsz egy másik kódszerkesztőt, és neked az fog beválni. Mindig azt a kódszerkesztőt használd, ami neked a legkényelmesebb és otthon érzed magad benne.

Online kódszerkesztők

Ha weboldalakat szeretnél készíteni, akkor a [CodePen](#) oldalát ajánljuk. Egyszerűbb weboldalakhoz tökéletes lesz, pont emiatt használtuk a korábbi lecke során.

Amennyiben az online kódszerkesztőknél maradsz és Pythonban vagy Javában szeretnél programozni, akkor a [Repl.it](#) oldalát ajánljuk. Sok különböző programnyelvet ismer és komolyabb eszközei is vannak.

Pythonban való programozás esetén még hasznos lehet a [Programiz](#) oldala is. Könnyen használható és gyors oldal, érdemes lesz kipróbálnod.

Offline kód szerkesztő

Amennyiben a gépeden szeretnél programozni, akkor több lehetőség áll előtted.

Programnyelvtől függetlenül az egyik legjobb asztali kód szerkesztő a [Visual Studio Code](#).

Mindegy, hogy weboldalakat szeretnél készíteni vagy egy Python programot megírni, a VSCode meg fog birközni velük.

Megjegyzés: Előfordulhat, hogy egyes, korábban készült leckéink még más, az ajánlataink között fel nem sorolt kód szerkesztőt használnak a mintamegoldások megjelenítésekor vagy a feladatok kiadásakor (pl. JS Bin). Ilyenkor bátran használd azt a kód szerkesztőt, ami számodra a legkényelmesebb, hiszen most már pontosan tudod, hogy végső soron nagyon hasonlóan működnek.

JavaScript alapok I. (adattípusok)

1. Bevezetés

1.1. Mi az a JavaScript?

Az internet hajnalán (nem is olyan régen) a statikus weboldalak urálták a világot. A böngésződ letöltött egy HTML- és egy CSS-fájlt, megjelenítette őket, és ezzel végzett is. Ha bármilyen változást szeretett volna megjeleníteni az oldal, egy újabb HTML- és CSS-fájlt kellett letölteni, és újra kellett renderelni (leképezni) az egész weblapot.

Ez a folyamat idő-, sávszélesség- és erőforrás-igényes volt. Kellett egy jobb megoldás.

És lett is: ma a dinamikus weblapok és a webappok korát éljük. Nincs olyan oldal a neten, ahol ne nyomhatnál meg egy gombot, ne írhatnál egy mezőbe, ne lenne valamiféle dinamikusan változó tartalom előtted. Ezt pedig túlnyomórészt a JavaScript (röviden JS) hajtja.

Amikor vásárolsz az Amazonon vagy bármelyik másik webshopban, elküldesz egy kérdőívet, regisztrálsz valahová, vagy egyszerűen csak lájkolsz valamit, JavaScripttel lépsz kapcsolatba. A programnyelv közvetlenül a böngészőben fut, és lehetővé teszi a weboldalak számára, hogy dinamikusan változzon a tartalmuk, anélkül, hogy újra kéne tölteni az egész lapot.

A következő leckékben ezzel a sokoldalú és szuper nyelvvel fogsz megismerkedni, és megtanulod, hogyan teheted az oldalaid dinamikussá és interaktívvá a látogatóid számára.

Előtte azonban megmutatjuk, hova írhatsz JavaScript-kódot!

1.2. A .js fájl és a konzol

Mielőtt beleugranál a kódolásba, egy kicsit beszéljünk arról, hány helyre lehet JavaScriptet írni, és hogy ezek között mi a különbség.

A .js fájl

Weboldalak készítésekor, amikor a saját számítógépeden fejlesztesz, a JavaScript-kódot egy `*.js` (gyakran `scripts.js`, `main.js` vagy `app.js`) fájlba írod, majd ezt belinkeled a HTML-kódba. Amikor a böngésző letölti a weboldal HTML- és CSS-fájljait, letölti mellé a JavaScriptet is, és lefuttatja azt. Ebben a modulban ilyen esettel még nem fogsz találkozni, ezért ugorjunk is a következő lehetőségre, a böngésző konzoljára.

A konzol

A JavaScript a böngészőben fut, ezért nevezzük kliensoldali programnyelvnek. Logikus, hogy minden böngészőben legyen egy felület, ahol közvetlenül érintkezhetsz a JavaScripttel.

Ezt az eszközt konzolnak nevezünk.

A konzol egy olyan felület, ahol:

- egyszerűen beírhatunk és futtathatunk JavaScript-kódot,
- másrészt fontos üzeneteket (például hibaüzeneteket) jeleníthet meg az adott weboldalon futó JavaScript a fejlesztők számára.

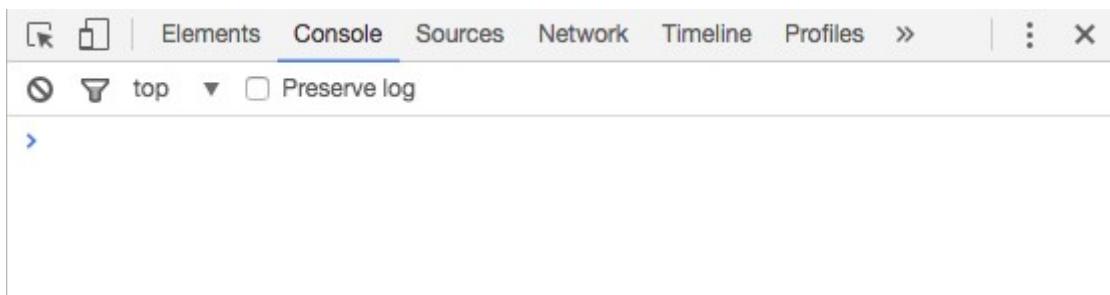
A konzolt bármelyik weboldalon megnyithatod és írhatsz bele JavaScript-kódot, de ez a weboldalnak csak az éppen letöltött változatát fogja megváltoztatni – nem változtatja meg a szerveren lévő állományokat. Azaz ha a konzolban változtatsz meg valamit, majd frissíted a weboldalt, a változtatásaid eltűnnek.

A konzol tanulás során kiváló arra, hogy JavaScript-kódokkal gyakoroljunk benne, fejlesztés során pedig hibakeresésre lehet használni.

Tipp: A Google Chrome-ban a konzolt a fejlesztői eszközök között éred el, amelyeket a `<kbd>Ctrl + Shift + I</kbd>` vagy Macen a `<kbd>Cmd + Alt + I</kbd>` billentyűkombinációval nyithatsz meg. Vagy kattints a böngésző jobb felső sarkában a hárompontos menüikonra (A Google Chrome személyre szabása és beállításai), majd azon belül a További eszközökre, és végül a Fejlesztői eszközökre.

Nyiss most egy új lapot a Chrome-ban, és írd be a keresőmezőbe ezt: `about:blank`. Ez a parancs egy üres oldalt hoz létre számodra. Nyisd meg a fejlesztői eszközöket, azon belül pedig a konzolt.

Ha ezzel megvagy, a konzolban látni fogsz egy balra nyíló kacsacsőt (`>`), amely után villog a kurzor. Ide írhatod a JavaScript-kódot, amit az Enter leütése után a böngésző lefuttat, és visszaadja a kód eredményét.



Próbáld ki! Másold be a konzolba a következőt: `20 + 19`.

A böngésző gyorsan kiszámolta, hogy ez 39!

Ellenőrző kérdés

(A továbbiakban az ellenőrző kérdésekben a válaszok alatt egyes esetekben a válaszokhoz tartozó, specifikus visszajelző üzeneteket találsz. A helyes válasz vastag betűvel van kiemelve.)

Mennyi 324 szorozva 245-tel? Használd a konzolt a válaszhoz!

- 569
 - Hmm, sajnos helytelen. Biztosan jó utasítást adtál a konzolnak?
- **79380**
 - Helyes válasz! Már tudod, hogyan használd a konzolt.
- 792380
 - Hmm, sajnos helytelen. Biztosan jó utasítást adtál a konzolnak?
- 79,380
 - Hmm, sajnos helytelen. Biztosan jó utasítást adtál a konzolnak?

1.3. A JS Bin JavaScript-panele

Korábban már megismerkedtél a JS Binnel, amelynek előnye, hogy automatikusan el is menti a munkádat, ha be vagy jelentkezve. A jelen kurzus során a JS Bin JavaScript-panelét és konzolját fogod használni, a többi ablakot (HTML, CSS és Output) viszont becsukhatod, rájuk nem lesz szükség.

[Nyiss most egy új bint](#) (így hívjuk a JS Binben a különálló projekteket)!

Másold be a JS-panelbe ugyanazt a parancsot (**20 + 19**), amit az előbb a böngésző konzoljába írtál be, és üss egy Enter-t. Mi történt? A cursor új sorba ugrott, de ezen kívül semmi egyéb.

A JS-panelben egy kicsit másképp kell megfogalmazni a parancsokat, mint a konzolban. Alapvető követelmény, hogy minden sort pontosvesszővel zárunk (**;**) – ezt már is írd a sorod végére. De még ezzel a kiegészítéssel sem történik semmi.

Ahhoz, hogy lefuttassuk ezt a parancsot, a JS Bin konzoljának jobb felső sarkában levő „Run” (Futtatás) gombra kell kattintani. Próbáld ki!

Még mindig semmi. Vagyis látszólag semmi. A JavaScript végrehajtotta az utasítást, amit adtál neki, azt azonban nem kérte, hogy a választ írja is ki. Ez bizony egy külön kérés. Nagyon fontos, hogy a kéréseinket minden pontosan, apró lépésekre lebontva fogalmazzuk meg, hiszen a számítógép csak azt csinálja, amit mi kérünk tőle.

Rengeteg módja van annak, hogy a JavaScript működésének eredményét megjelenítsük, de most kezdjük a legeslegegyszerűbbel, a **console.log()** parancssal.

- A **.log** utasítással azt kérjük, hogy legyenek kiírva dolgok.
- De milyen dolgok, és hova legyenek kiírva? A parancs előtagja, a **console** kifejezés a konzolra utal (hol), azaz itt fognak megjelenni a zárójelben (**()**) megadott dolgok (mit).

Próbáld ki! Másold a következő sort a JS Bin JS-panelébe, majd kattints a konzol ablakban a „Run” gombra.

```
console.log(20 + 19);
```

Ezúttal megkaptuk az eredményt. Szép munka!

Ha be vagy jelentkezve, akkor a fenti kódot a JS Bin automatikusan el is mentette számodra. minden bin saját, egyedi linket kap, így azt a böngésződ címsorából kimásolva meg tudod osztani a munkádat akárkivel.

A következő leckékben minden jelölni fogjuk, hogy az adott kódot a JS Bin JS-paneljére, vagy a böngésző konzoljába írd. Addig is jegyezd meg a `console.log()` módszert, még kelleni fog.

Mennyi 20 000 osztva 40-nel? Használd a JavaScript-panelt és a `console.log()` módszert!

2. A JavaScript adattípusai

2.1. A JavaScript adattípusai

A számítógépek világában az egyik alapvető építőelem **az adat**. Mivel a gépek minden információt adatként tárolnak és kezelnek, szükségük van arra, hogy különbséget tudjanak tenni a különböző adattípusok között.

A JavaScript nyelv hét alapvető adattípust (angolul data type-ot) különböztet meg:

1. **Number**: ebbe a típusba tartozik minden szám, a törteket is beleértve. A számok minden idézőjel nélkül szerepelnek a kódban: `3, 6, 2019, 43.25`.
2. **String**: ebbe a típusba tartoznak a szövegek, pontosabban bármilyen karaktersor. Egy string akármit tartalmazhat (betűt, számot, szóközt, szimbólumot); egészen addig string típusú értéknek számít, amíg a karaktersor aposztróffal (`'`) vagy idézőjellel (`"`) van körülvéve. Pl. `'Ez egy string.'`, `"Ez is 1 string."`, `'Söt, 3z is @z.'`
3. **Boolean**: ez az adattípus kétféle értéket vehet fel: `true` vagy `false` (magyarul igaz vagy hamis). Talán már ki is találtad, hogy segítségével igaz-vagy-hamis, igen-vagy-nem típusú kérdéseket tehetünk fel a kódban.
4. **Null**: ez egy úgynévezett „empty value” (azaz üres érték), amely a konkrét érték hiányát jelzi.
5. **Undefined**: a nullhoz nagyon hasonló empty value. Működésük szituációtól függ, éppen ezért inkább akkor fogjuk jobban elmagyarázni őket, amikor felbukkanak egy-egy leckében.
6. **Symbol**: a szimbólumok egyedi azonosítók, amelyek összetett kódolás során tudnak hasznosak lenni. Szóval erre egyelőre nem lesz szükséged.
7. **Object**: az objektum több egymáshoz kapcsolódó adat gyűjteménye. Egy későbbi modulban bővebben is megismerkedsz majd velük.

Az első hat típust **primitív adattípusoknak** is szokták nevezni. Ezek a JavaScript programnyelv legalapvetőbb elemei – olyanok, mint az emberi nyelvben a szavak.

Ebben a modulban a number, string, valamint a boolean adattípusokkal fogsz bővebben megismерkedni.

Megjegyzés: Az egyedi adatokra értékként (angolul value) szoktunk hivatkozni. Itt egy példamondat erre: „Minden string érték a string adattípusba tartozik.”

Ellenőrző kérdés

Milyen adattípusba tartoznak az alábbi értékek? 'húsz' "tizenkilenc" "20" '19' '2019' "2null19"

- number
 - Sajnos helytelen a válasz. A felsorolt értékek szám és szöveg keverékei, azonban mindenket körülveszi a szimpla vagy dupla idézőjel, emiatt pedig a string típusba tartoznak.
- string
 - Helyes válasz! A felsorolt értékek szám és szöveg keverékei, azonban mindenket körülveszi a szimpla vagy dupla idézőjel, emiatt pedig a string típusba tartoznak.
- undefined
 - Sajnos helytelen a válasz. A felsorolt értékek szám és szöveg keverékei, azonban mindenket körülveszi a szimpla vagy dupla idézőjel, emiatt pedig a string típusba tartoznak.
- null
 - Sajnos helytelen a válasz. A felsorolt értékek szám és szöveg keverékei, azonban mindenket körülveszi a szimpla vagy dupla idézőjel, emiatt pedig a string típusba tartoznak.

3. Játék számokkal

3.1. Számtani (arithmetic) operátorok

Még mielőtt a lecke címét olvasva megijednél, hogy belehúzunk a matekba, hadd nyugtassunk meg: alapszintű összeadás, kivonás, stb. magasságokban fogunk csak mozogni, és minden el fogunk magyarázni lépésről lépére.

Itt, az első leckében az operátorokról fogunk beszélni.

Az **operátorok** olyan karakterek, amelyek értékek között állva valamilyen műveletet hajtanak végre azokon, és ezzel egy új értéket hoznak létre. Az operátorok működési elvükben nagyon hasonlítanak a matematikai műveleti jelekre.

Olyannyira, hogy a műveleti jeleknek meg is van az operátor megfelelőjük a JavaScriptben:

- Összeadás: 5 + 2
- Kivonás: 5 - 2
- Szorzás: 5 * 2

- Osztás: 10 / 2
- Maradék (modulus): 12 % 5

Az első négy operátor pontosan úgy működik, mint ahogy sejted: összeadja, kivonja, megszorozza vagy elosztja a számokat egymással.

A maradék operátor (%), amit modulusnak is nevezünk) egy kicsit másképp működik. Előbb megnézi, hogy hányszor van meg a bal oldali számban a jobb oldali szám, majd visszaadja az osztás maradékát: 17 % 5 esetében a válasz 2, mert 17-ben háromszor van meg az 5, és 2 a maradék.

Megjegyzés: A JavaScriptben sok különböző operátor létezik, amelyeket funkcióik alapján csoportokba sorolunk. A fenti operátorokat arithmetic (azaz számtani) operátoroknak hívjuk, mivel matematikai műveletek elvégzésére jöttek létre. A további leckékben több másik operátorral is találkozni fogsz.

Ellenőrző kérdés

Az alábbiak közül melyik definíció illik a JavaScript-operátorokra?

- Az operátorok a JavaScriptben az adattípusok egy különleges csoportját alkotják.
- Az operátorok speciális parancsok, amelyekkel összeadási és kivonási feladatokat tudunk elvégezni a JavaScriptben.
- **Az operátorok speciális karakterek a JavaScriptben, amelyek két (vagy több) érték (value) között állva azokon valamiféle műveletet végeznek el, ezzel egy új értéket létrehozva.**
- Az operátorok speciális karakterek a JavaScriptben, amelyek két érték között állnak, és egy algoritmus alapján kiválasztják a kettő közül valamelyiket.

3.2. Gyakorlás: A JavaScript mint zsebszámológép

Mielőtt továbbhaladnánk, nézzük meg az operátorokat működés közben is!

Nyisd meg a böngésződ konzolját.

Tipp: A Google Chrome-ban a konzolt a fejlesztői eszközök között éred el, amelyeket a <kbd>Ctrl + Shift + I</kbd> vagy Macen a <kbd>Cmd + Alt + I</kbd> billentyűkombinációval nyithatsz meg. Vagy kattints a böngésző jobb felső sarkában a hárompontos menüikonra (A Google Chrome személyre szabása és beállításai), majd azon belül a További eszközökre, és végül a Fejlesztői eszközökre.

Írd be a konzolba a következő összeadást, és nyomd meg az Entert: 2019 + 1988.

Ha minden igaz, a konzol kiírta neked az összeadás eredményét válaszként. Szuper! Írd be az alábbi sorokat is egyenként, és mindegyik után üss egy Entert:

21342 - 5438

```
99 * 99  
9124 / 4  
25 % 3
```

Ahogy látod, a JavaScript kiváló zsebszámolóképként is használható. Elég menő, nem? De ennél még sokkal többre is képes, hamarosan meglátod!

Utolsó ujjgyakorlatként következzen egy kis varázslat, amellyel gyakorolhatod a számtani operátorok használatát. Kövesd pontosan a lépéseket, majd válaszd ki a helyes megoldást a lentie lehetőségek közül!

1. Nyisd meg a böngésző konzolját.
2. Gondolj egy számra 1 és 99 között, és írd fel valahova.
3. A konzol segítségével vonj ki egyet ebből a számból.
4. A konzol segítségével szorozd meg az eredményt 3-mal.
5. A konzol segítségével adj 12-t az eredményhez.
6. A konzol segítségével oszd el 3-mal az eredményt.
7. A konzol segítségével adj 5-öt az eredményhez.
8. A konzol segítségével vond ki az eredeti számot az előbb kapott eredményből.

Mennyi jött ki?

Ellenőrző kérdés

Válaszd ki az előbbi feladat eredményét!

- 6
 - Hmm, sajnos helytelen a válasz. Biztos, hogy minden lépést pontosan követtél, és nem hagytál ki semmit? Kérlek, ellenőrizd!
- 7
 - Hmm, sajnos helytelen a válasz. Biztos, hogy minden lépést pontosan követtél, és nem hagytál ki semmit? Kérlek, ellenőrizd!
- 8
 - Nagyon jó! A válasz mindig 8!
- 9
 - Hmm, sajnos helytelen a válasz. Biztos, hogy minden lépést pontosan követtél, és nem hagytál ki semmit? Kérlek, ellenőrizd!

3.3. NaN (Nem szám)

Az aritmetikai operátorok – az összeadás kivételével – csak szám (number) típusú értékeken működnek. Ha bármilyen más adattípuson próbálunk meg végrehozni egy számítási feladatot, a JavaScript azt fogja mondani, hogy „hé, ez nem egy szám”.

Ezt úgy teszi meg, hogy egy speciális értéket ad vissza: **NaN** (Not a Number, azaz szó szerint „nem egy szám”).

Próbáld ki ezt: írd be a konzolba, hogy '`'alma'` - 5', majd nyomd meg az Entert.

Ennek a kérésnek jól láthatóan nincs értelme, hiszen egy szövegből nem lehet kivonni ötöt – és ezt a JavaScript meg is mondja neked. Az érték, amelyet visszakapsz a konzolban, a **NaN**.

Néhány értelmetlen matematikai számítás is **NaN** eredményt ad, például ha 0-t próbálsz meg elosztani 0-val.

Megjegyzés: Ahogy a legelején említettük, az egyetlen kivétel a fentiek alól az összeadás-operátor (`+`). Rá a következő, stringről szóló fejezetben bővebben is ki fogunk térní.

Ellenőrző kérdés

Az alábbiak közül melyik művelet eredményez **NaN** értéket?

- `520 * 430`
- **0 / 0**
- `'az' + 'alma' + 'egy gyümölcs'`
- `0 * 0`

3.4. JavaScript-kifejezések (expressions)

Amikor két értéket kombinálsz egy operátorral, a JavaScript nyelv egy új építőköve, az expression (magyarul *kifejezés*) jön létre.

Expressionnek számít minden olyan kódrészlet, amely egy értéket (value) eredményez. Amikor ugyanis a böngésző futtatja a JavaScriptet, akkor sorról sorra megy végig rajta, és kiértékeli azt.

Például tegyük fel, hogy a böngésző ezt a sort találja egy JavaScript-fájl első sorában:

```
2 + 5
```

Mielőtt továbblépne, a böngésző kiértékeli ezt az expressiont, és a számítás eredményével, 7-tel helyettesíti be. A böngésző szemében ez a kifejezés egyetlen értékké egyszerűsödött.

Emlékszel, amikor azt mondtuk, hogy az értékek (values) olyanok a JavaScript számára, mint a szavak az emberi nyelvben? Építve erre a hasonlatra, **az expressionök (kifejezések) olyanok, mint a szókapcsolatok**. Még nem teljes mondatok, de már nagyobb és hasznosabb elemei a nyelvnek, mint a pusztta szavak.

Ellenőrző kérdés

Válaszd ki az alábbi definíciók közül azt, amelyik a kifejezéseket (expressions) írja körül!

- szavak és szótöredékek
 - Sajnos helytelen. A kifejezések az emberi nyelvek szókapcsolataihoz

hasonlóak.

- **értéket (value) létrehozó kódrészlet**
- egy összeadás
 - Sajnos helytelen. Nemcsak az összeadást végrehajtó operátor használata vezet új értékhez, hanem mindeneké.
- két kód együttese
 - Sajnos helytelen. A kifejezések (expressionök) egy ennél pontosabb definícióval bírnak.

3.5. Összetett műveletek JavaScripttel

Akárcsak a matekban, a JavaScriptben is kötött sorrendje van az összetett matematikai műveletek megoldásának. A sorrend mindenkor a következő:

1. **Parentheses (zárójelek)**: Először mindenkor a zárójelben szereplő műveleteket számolja ki.
2. **Exponents (exponenciális műveletek)**: Ezután a hatványozás és gyökvonás következik.
3. **Multiply és Divide (szorzás és osztás)**: Majd a szorzás és osztás kerül sorra.
4. **Add és Subtract**: Végül pedig kiszámolja az összeadásokat és kivonásokat.

Megjegyzés: Az egyenrangú műveleteket (például szorzás és osztás) a JavaScript balról jobbra haladva végzi el. Ahogy egyébként azt nekünk, embereknek is kell.

A folyamat teljesen ugyanaz, mint amit a matematikaórán tanultál, de ha másképp nem, a sorozat kezdőbetűiből képzett PEMDAS mozaikszóra gondolva könnyen megjegyezheted a sorrendet.

Ellenőrző kérdés

Mennyi az eredménye a következő összetett számításnak? $8 / 2 * (2+2)$

- 1
 - Sajnos helytelen a válasz. Ne feledd, a szorzás és osztás egyenrangú műveletek, így a JavaScript balról jobbra haladva végzi el őket.
- 0,8
- NaN
- 16
 - Helyes válasz! Gratulálunk, már tudsz összetett számításokat is végezni a JavaScript segítségével.

3.6. Összehasonlító (comparison) operátorok

Időnként előfordul majd, hogy összehasonlító jellegű kérdéseket szeretnél felenni a kódodban. Például „X nagyobb-e, mint Y?” vagy „W egyenlő-e Z-vel?”. Az ilyen kérdéseket a JavaScript képes kiértékelni, és igaz/hamis kijelentésekkel megválaszolni.

Ahhoz, hogy ilyen jellegű kérdést tegyünk fel, egy újabb operátorcsoportra, a cseppet sem meglepően elnevezett **összehasonlító (angolul comparison) operátorokra** van

szükségünk.

Megjegyzés: Egyesek *comparatorként* hivatkoznak ugyanezekre az operátorokra. Ezzel az elnevezéssel is találkozhatsz majd a világban.

Próbáld ki! Írd be a következőket soronként a böngésző konzoljába, és mindenkor után üss Enter-t. Milyen értékeket kaptál vissza?

```
10 < 1  
9 > 4
```

Ha minden jól ment, akkor az első állításra a böngésző a **false** (hamis), a másodikra pedig a **true** (igaz) választ adta vissza. A böngésző megnézte az állításaidat, és eldöntötte, hogy ezek igazak vagy hamisak-e.

Megjegyzés: Ha szemfűles vagy, akkor észrevehetted, hogy ezek az összehasonlítások valójában JavaScript-kifejezések (expressionök). Azonban ezúttal a két number (szám) típusú kiindulási érték nem egy harmadik számot, hanem egy boolean értéket eredményez az összehasonlító operátor miatt.

Az összehasonlító (comparison) operátorok listája

A JavaScriptben elérhető *comparatorok* matematikaóráról biztosan ismerősek lesznek, csak ott relációs jeleknek hívjuk őket:

- **5 > 2** → nagyobb, mint
- **8 < 4** → kisebb, mint
- **6 >= 2** → nagyobb vagy egyenlő
- **5 <= 10** → kisebb vagy egyenlő
- **4 == 4** → egyenlő
- **3 != 4** → nem egyenlő

A logika tehát a következő:

```
érték1 {összehasonlító operátor} érték2 → igaz-e az  
összehasonlítás
```

Próbáld ki te is! A böngésződ konzolját használva nézd meg, hogy igaz vagy hamis-e az alábbi három állítás, majd válaszd ki a helyes megoldást!

```
(48 - 8) >= (7 * 6 + 6)  
(81 / 9) <= (11 - 2)  
(8 * 8) != (6 * 4 + 40)
```

Ellenőrző kérdés

Melyik a helyes megoldás a fenti három állításhoz?

- igaz, igaz, hamis
 - Sajnos helytelen. Kérlek, ellenőrizd még egyszer a műveleteket!
- hamis, hamis, hamis
 - Sajnos helytelen. Kérlek, ellenőrizd még egyszer a műveleteket!
- **hamis, igaz, hamis**
 - Helyes válasz! Gratulálunk, már tudod használni a comparatorokat.
- hamis, igaz, igaz
 - Sajnos helytelen. Kérlek, ellenőrizd még egyszer a műveleteket!

4. Játék szavakkal

4.1. String

Míg a number (szám) adattípus csak számokat vehet fel értékként, addig a string adattípusba a karakterek bármelyike és bármilyen összetétele beletartozhat. Egy string állhat betűkből, számokból, de akár betűk, számok, szöközök és szimbólumok keverékéből is.

Erre a rugalmasságra azért van szükség, mert **a stringeket szövegek tárolására használjuk**.

Írd be a következőt a böngésződ konzoljába, és nyomj egy Enter-t:

```
Egy icipici pók felmászott az ereszen.
```

Mi történt? Hoppá, valami nincs rendben. A konzol egy piros hibaüzenetet írt ki: "**Uncaught SyntaxError: Unexpected identifier**".

Az történt, hogy a JavaScript nem értette, mit szeretnél. Te tudod, hogy egy darab szöveget, azaz stringet szerettél volna bevinni, de a JavaScriptnek szüksége van arra, hogy ezt jelezd valahogy számára. Erre az az eszközünk, hogy **a stringeket egyszeres (') vagy kétszeres (") idézőjelekkel vesszük körbe**.

```
'Ez egy valid string.'  
"És ez is egy valid string."
```

Megjegyzés: A két írásmód közül mi az egyszeres idézőjeleket (single quotes) ajánljuk. Ennek az az oka, hogy HTML-ben az attribútumok megadásakor kétszeres idézőjeleket (double quotes) kell használnunk, így amikor a két nyelv találkozik, a JavaScript jobban elkülönül a HTML-től, ami olvashatóbbá teszi a kódot.

Most próbáld ki újból a legelső feladatot ezzel a módosítással:

```
'Egy icipici pók felmászott az ereszen.'
```

Ha minden jól ment, a konzol visszaírta neked pontosan ugyanezt, vagyis sikerült megfogalmaznod az első stringedet. Gratulálunk!

Megjegyzés: minden, amit single quote-ok közé raksz, stringként viselkedik. Tehát ha számokat raksz idézőjelbe, a JavaScript a továbbiakban azt stringnek, egymás mögé fűzött számok sorának értelmezi majd. Míg single quote nélkül a 123 százhuzzonhármat jelent, addig a '123' a JavaScript szempontjából csak egy egyes, kettes és egy hármas egymás után.

Ellenőrző kérdés

Melyik a szintaktikailag helytelen string? Válassz egyet a felsoroltak közül!

- 'Csodásan nézel ma ki.'
 - Nem, nem – ez egy érvényes string.
- "Tedd vagy ne tedd. De ne próbáld."
 - Sajnos helytelen. Ez egy érvényes string.
- 'Ne add fel a reményt!'
 - Helyes válasz! A szöveget lezáró single quote hiánya megtöri a kódot, így pedig nem lesz a JavaScript számára értelmezhető.
- '“Miért vagy te Rómeó?”'
 - Akármilyen meglepő is, ez egy érvényes string. A turpisság abban rejlik, hogy a folyószövegekben használt, irodalmi idézőjelek nem egyeznek meg a programozásban használt double quote-okkal. A fenti stringet single quote-ok veszik körül, eggyel beljebb pedig irodalmi idézőjelek találhatók, ezért érvényesnek számít a string.

4.2. Escaping

Az előző leckében megtanultad, hogy nagyon fontos a stringek körüli idézőjelek helyesírása. Viszont mi történhet akkor, ha a szövegben magában is szerepel egy idézőjel?

Nézzük meg ezt közelebbről! Írd be a következő sort a böngésződ konzoljába, és nyomj egy Enter-t:

```
'A Starkok mindig azt mondják, 'Közeleg a télen.' – most igazuk is lett.'
```

Mi lett az eredmény? A konzol megtint Error-t (hibát) dobott. Miért? Mert a JavaScript szempontjából a fenti sor sehogy sem értelmezhető.

Szerinte:

- Megadtál egy stringet: 'A Starkok mindig azt mondják, '.
- Meg egy másikat: ' – most igazuk is lett.'

- Valamit pedig odabiggyesztettél a két string közé, ami se nem szám, se nem string: **Közeleg a tél..**

Mit lehetünk ilyenkor? Megmondhatjuk a JavaScriptnek, hogy hagyja figyelmen kívül a két belső idézőjelet. Ezt **escapingnek** hívjuk a programozásban. (A szó angolul *menekülést* jelent, és nem szoktuk lefordítani.) Az escape-elésre van egy speciális karakter, a fordított törtvonal (angolul *backslash*): \. Az ezt követő bármilyen karaktert a JavaScript figyelmen kívül hagyja a kód értelmezésekor – pontosabban úgy kezeli, mintha sima karakter lenne, speciális jelentés nélkül.

Próbáljuk újra az eredeti problémát ezzel az új tudással. Másold be az alábbi stringet a böngésződ konzoljába:

```
'A Starkok mindig azt mondják, \'Közeleg a tél.\' – most igazuk is lett.'
```

Szuper! A böngésző átugrotta a string közepén lévő idézőjeleket, és csak a két külsőt vette figyelembe. Így végre megértette, hogy egy stringben szeretnénk kiíratni vele az egész mondatot.

Megjegyzés: Az escaping nemcsak a stringeket jelölő, egyszeres vagy kétszeres idézőjelek esetében működik, hanem valamennyi speciális JavaScript-karakternél. Még a backslash esetében is, egyszerűen tegyél elé egy másikat, így: \\.

Mielőtt továbbmennénk, egy utolsó gondolat. A kezdeti probléma egy másik megoldása, ha egyszeres helyett kétszeres idézőjelekkel veszed körbe a stringed:

```
"A Starkok mindig azt mondják, 'Közeleg a tél.' – most igazuk is lett."
```

Ebben az esetben nem kell escape-elned a belső idézőjeleket, hiszen a JavaScript nem zavarodik össze, hogy hol kezdődik és végződik a string. Ugyanez fordítva is igaz lenne: használhatnál egyszeres idézőjeleket a string jelzésére, és akkor a stringen belül írhatnál escape-elés nélküli kétszeres idézőjeleket.

Megjegyzés: Ha kipróbálod a lehetséges megoldásokat a JS Bin Console (konzol) paneljén, látni fogod, hogy az applikáció egy picit furán viselkedik. A stringen belül használt kétszeres idézőjeleket escape-ellen jeleníti meg, a stringet pedig akkor is double quote-okkal körülveve írja ki, ha te single quote-okat használtál. Emiatt ne aggódj, ez nem a JavaScript alapértelmezett működése, hanem egy bug a JS Binben.

Feladat

Nyiss egy új bint a JS Binben. Írasd ki a következő szöveget a Console-panelre: **A ház nem hivatalos, ám közismert jelmondata: 'Egy Lannister mindig megfizeti az adósságait.'** A valódi jelmondatuk ezzel szemben: 'Halld

üvöltésem! '.

A programod a következő kritériumoknak feleljen meg:

- A JavaScript-panelen dolgozz.
- A szöveg kiíratásához használ a `console.log()` parancsot.
- Ügyelj a szöveg megfelelő escape-elésére.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4.3. Új sor

Nyisd meg a böngésződ konzolját (ha bezártad), és másold bele ezt a versikét:

```
'Szép cica, jó cica,  
Selymes a bundád.  
Kedves cica, álmos cica,  
Bújj hozzám.'
```

Mi történt? Egy újabb Error, igaz? Bár a Chrome nem árul el sokat arról, hogy mi a baj, mi megsúgjuk, hogy azért dobta vissza a stringed, mert escape-eletlen sortörések vannak benne, ez pedig a JavaScriptben nem megengedett.

Megjegyzés: Ne feledd, hogy a számítógépek szempontjából a sortörés, szóköz stb. is karakternek számít.

Oké, akkor escape-eljük a sortöréseket:

```
'Szép cica, jó cica,\nSelymes a bundád.\nKedves cica, álmos cica,\nBújj hozzám.'
```

Hát, az eredmény nem egészen az lett, amit szerettünk volna: **Szép cica, jó cica,Selymes a bundád.Kedves cica, álmos cica,Bújj hozzám.**

Szerencsére erre a problémára létezik egy eszköz a JavaScriptben, az új sor (newline) karakter: `\n`. Ahova kirakod a newline karaktert a stringedben, ott a böngésző egy sortörést fog renderelni, amikor megjeleníti a szöveget.

Próbáld ki!

```
'Szép cica, jó cica,\nSelymes a bundád.\nKedves cica, álmos cica,\nBújj hozzám.'
```

Végre úgy néz ki, mint egy aranyos mondóka egy óvodás könyvéből, igaz? Szuper!

Feladat

Nyisd meg az előző feladatban használt bined, amelyben a Lannisterek jelmondatát írtad ki a konzollal. Változtasd meg a kódod úgy, hogy a stringben legyen egy sortörés a két mondat között.

A programod a következő kritériumoknak feleljen meg:

- A JavaScript-panelen dolgozz.
- A szöveg kiíratásához használd a `console.log()` parancsot.
- Ügyelj a szöveg megfelelő escape-elésére.
- Az első mondat után legyen egy sortörés.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4.4. Stringek összefűzése

Alkalomadtán jól jön, ha össze tudunk fűzni két stringet eggyé. Ha két string közé teszel egy `+` (plusz) jelet, a JavaScript összefűzi neked azokat.

Próbáld ki! Írd be a böngésződ konzoljába a következő sort:

```
'Egy kismalac röf-röf-röf' + 'trombitálgat töf-töf-töf.'
```

A konzol szépen elnyelte a két különálló stringet, és összefűzte őket eggyé, amit aztán visszaadt.

Tipp: Fontos megjegyezned, hogy a **szóköz (space)** ugyanúgy egy karakter a JavaScript szempontjából. Ha szeretnéd, hogy az egyesített stringben legyen szóköz a két eredeti string határán, akkor ne feledd el belerakni valamelyik eredetibe. (Lásd a fenti példát, ahol az első string végén van egy szóköz.)

Megjegyzés: Ha szemfüles vagy, észrevehetted, hogy ez az a kivétel, amelyről a számítani operátorok kapcsán beszéltünk. Az összeadás (`+`) operátor nemcsak a számokon, hanem a stringeken is működik, egyedüliként a csoportjában.

Ellenőrző kérdés

Hogyan fűznéd össze a következő mondatokat a konzollal? Válaszd ki a helyes megoldást!
„A ház nem hivatalos, ám közismert jelmondata: 'Egy Lannister minden megfizeti az adósságait.'” és „A valódi jelmondatuk ezzel szemben: 'Halld üvöltésem!'”

- 'A ház nem hivatalos, ám közismert jelmondata: 'Egy Lannister minden megfizeti az adósságait.' ' + 'A valódi jelmondatuk ezzel szemben: 'Halld üvöltésem!''
 - Sajnos helytelen a válasz. Ha ezt írod a konzolba, hibát fog jelezni, mert nem escape-elte a stringen belüli single quote-okat. Gondold át a tanultakat, és próbáld meg újra!
- 'A ház nem hivatalos, ám közismert jelmondata: \'Egy Lannister minden megfizeti az adósságait.\' ' + 'A valódi jelmondatuk ezzel szemben: \'Halld üvöltésem!\'\''
 - Hmm, ez nem tűnik helyesnek. Ha ezt írod a konzolba, hiányozni fog egy szóköz a két mondat között. Gondold át még egyszer a tanultakat, és próbálkozz újra!
- 'A ház nem hivatalos, ám közismert jelmondata: \\'Egy Lannister minden megfizeti az adósságait.\' ' + 'A valódi jelmondatuk ezzel szemben: \'Halld üvöltésem!\'\''
- A ház nem hivatalos, ám közismert jelmondata: 'Egy Lannister minden megfizeti az adósságait.' + A valódi jelmondatuk ezzel szemben: 'Halld üvöltésem!'
 - Sajnos helytelen a válasz. Ha ezt írod a konzolba, hibát fog visszajelzni, mert nem tettek idézőjelbe a stringeket. Gondold át még egyszer a tanultakat, és próbáld meg újra!

4.5. Stringek összehasonlítása egymással

Lesz majd olyan, amikor két stringet kell összehasonlítanod, és megmondanod, hogy megegyeznek-e egymással vagy sem. Ehhez két – már ismert – comparison operátort használhatsz:

- egyenlő (*equal*) → ==
- nem egyenlő (*not equal*) → !=

Tegyük fel például, hogy kíváncsi vagy arra, hogy az alábbi két sor pontosan megfelel-e egymásnak:

"Ne akarj másnak látszani, mint aminek látszol, mert ha nem annak akarsz látszani, aminek látszol, akkor nem annak látszol, aminek látszani akarnál." – Alice
Tükörországban

"Ne akarj másnak látszani, mint aminek látszol. Mert ha nem annak akarsz látszani, aminek látszol, akkor nem annak látszol, aminek látszani akarnál." – Alice
Tükörországban

Hasonlítsd össze a kettőt az egyenlőség (*equal*) operátor segítségével:

```
'"Ne akarj másnak látszani, mint aminek látszol, mert ha nem annak akarsz látszani, aminek látszol, akkor nem annak látszol, aminek látszani akarnál." – Alice Tükörországban' == '"Ne akarj másnak látszani, mint aminek látszol. Mert ha nem annak akarsz látszani, aminek látszol, akkor nem annak látszol, aminek látszani akarnál." – Alice Tükörországban'
```

Mit adott vissza a konzolod eredményként? Hamis, azaz `false`. Miért?

Mert a második formájában az idézetnek van egy pont és egy nagybetű, amely az elsőben nem szerepel. Az összehasonlítás csak akkor ad vissza `true` (igaz) értéket, ha a két string tökéletesen, karakter pontosan megegyezik egymással.

A *not equal* (`!=`, nem egyenlő) operátor pontosan ugyanezen az elven működik. Írd be a következő kódot a konzolba:

```
'Batman and Robin' != 'Barney and Robin'
```

Mint várható volt, egy `true` (igaz) választ kaptál, hiszen a két string valóban nem felel meg egymásnak.

Megjegyzés: A többi comparatort (`<`, `>`, `<=`, `>=`) is használhatod stringek összehasonlítására. Ilyet azonban valószínűleg ritkán fogsz csinálni, mivel ezek azt vizsgálják, hogy a két string közül melyik helyezkedik el előbb az ABC-ben.

Ellenőrző kérdés

Az alábbiak közül melyik operátort használnál arra, hogy kiderítsd, azonos-e a két string?

- `==`
- `=`
- `!`
- `=!`

4.6. Eltérő adattípusok nem szigorú összehasonlítása

Az előző leckében láttad, hogy mi történik akkor, ha stringeket hasonlítunk össze comparison operátorokkal, a 3. fejezetben pedig kipróbáltad, hogyan viselkednek a number (szám) típusú értékek ugyanebben a kérdésben. De vajon mi történik akkor, ha eltérő adattípusú értékeket hasonlítunk össze?

Próbáljuk ki! Másold be az alábbi, számot stringgel összehasonlító kódot a böngésződ konzoljába, és üss egy Entert:

```
12 == '12'
```

Úgy tűnik, a JavaScript szerint ez az egyenlőség igaz (**true**).

Egyrészről védhető ez az álláspont, ha a szándékot vizsgáljuk – *elvégre* tizenkettő egyenlő 12-vel. Ugyanakkor, ha szigorúak akarunk lenni, akkor a bal oldalon egy szám áll, a jobb oldalon pedig egy szöveg. A kettő természeténél fogva nem lehet egyenlő.

A válasz abban rejlik, hogy a JavaScript egy nagyon megengedő nyelv, és az alapértelmezett comparison operátorai úgynevezett **non-strict** (nem szigorú) operátorok. Amikor egy ilyen non-strict comparatorral hasonlítunk össze két eltérő adattípusú értéket, akkor a JavaScript érzékeli az eltérést, és az értékeket automatikusan azonos típusúvá alakítja, mielőtt elvégezné az összehasonlítást.

Tehát a fenti szituációban a következő zajlott le a háttérben:

```
12 == '12' // Hm... ezek eltérő adattípusok. Hadd alakítsam át
őket!
12 == 12 // Na, erre már tudok válaszolni!
true
```

Ennek a jelenségnek neve is van a JavaScriptben, úgy hívjuk, hogy **type coercion** (kényszerített típusváltás).

Ökölszabályként jegyezd meg, hogy a type coercion kerülendő dolog. Programozás közben általában azt szeretnénk, ha a JavaScript szigorú lenne velünk szemben, és nem próbálná meg magától kitalálni, hogy „vajon mire gondolhatott a költő”.

Természetesen erre van eszközünk: a strict (szigorú) comparison operátorok.

4.7. Eltérő adattípusok szigorú összehasonlítása

A strict (szigorú) comparison operátorok – ahogy a nevük is sugallja – az összehasonlítandó értékek összes tulajdonságát szigorúan figyelembe veszik, és nem engedik, hogy type coercion történjen a háttérben.

Próbáld meg összehasonlítani az előbbi két értéket, ám ezúttal szigorú egyenlőségi (*strictly equal*) operátort használva (**==**):

```
12 === '12'
```

A konzol ezúttal azt mondja, hogy az állításod hamis (**false**), ahogy az várható.

Azt érdemes elvinned mindebből, hogy amikor JavaScript-kódot írsz, **értékek összehasonlítására mindig strict comparison operátorokat használj**. Ezzel elkerülök a háttérben történő type coerciót, amely váratlan, nehezen felkutatható hibákat tud okozni a programokban.

Megjegyzés: A *not equal* (nem egyenlő) operátornak is van strict verziója: `!==`.

Ellenőrző kérdés

A két alábbi adatot összehasonlítva mely két operátor használatakor kapnánk mindenkor azt a választ a konzolban, hogy `false`? Az adatpár: `100 + 20` és `'120'`.

- `==` és `===`
 - Sajnos helytelen. Habár a strict operátor eredménye valóban hamis, a non-strict operátor eredménye igaz.
- `!=` és `!==`
 - Sajnos helytelen. Habár a non-strict operátor eredménye valóban hamis, a strict operátor eredménye igaz.
- `!=` és `==`
 - Helyes válasz! Sikeresen megértetted a non-strict és a strict egyenlőségi operátorok közötti különbséget. Gratulálunk!
- `!==` és `==`
 - Sajnos helytelen. Mindkét operátor eredménye igaz.

5. Összefoglalás

5.1. Mit tanultál eddig?

Gratulálunk! Ha idáig eljutottál, az azt jelenti, hogy:

- A JavaScript 7 primitív adattípusából megismertél ötöt: `number`, `string`, `boolean`, `undefined` és `null`.
- Tudod, hogyan végezz matematikai műveleteket `number` típusú értékekkel:
 - Tudsza JavaScriptben összeadni, kivonni, szorozni és osztani, valamint tudod, hogyan kapd meg az osztás utáni maradékot (modulus).
 - Olyan eseteket is ismersz, amikor az eredmény nem egy szám (`NAN`).
- Tudod, hogy mi a matematikai műveletek megoldásának helyes sorrendje (PEMDAS), és meg tudsz oldani összetett matematikai műveleteket a JavaScript segítségével.
- Össze tudsz hasonlítani két számot comparison operátorok segítségével, és értelmezni tudod az eredményként kapott boolean értékeket (`true`, `false`).
- Tudod, hogy a stringekben szövegeket tárolunk, és te magad is tudsz szintaktikailag korrekt stringet írni.
- Képes vagy stringeken belül escape-álni a backslash (`\`) használatával.
- Tudod, hogyan tudsz sortörést hozzáadni a newline karakterrel (`\n`).
- Össze tudsz fűzni stringeket az összeadás-operátor (`+`) használatával.
- Tudod, mi a különbség a non-strict és strict comparison operátorok között, és megtanultad, hogy ökölszabályként érdemes minden strict comparison alkalmazni, hogy elkerüld a type coercion jelenségét.

Nem semmi lista, ügyes vagy!

A legfontosabb azonban az, hogy találkoztál a JavaScript első három építőkövével, és megértetted ezek szerepét a programnyelvben:

1. **Értékek** → egy egységnnyi adatok, olyanok, mint az emberi nyelv szavai.
2. **Operátorok** → két vagy több érték között állva azokon valamiféle műveletet hajtanak végre, és ezzel egy új értéket hoznak létre.
3. **Expressionök** (kifejezések) → minden olyan kódrészlet, amely értékké válik a kód kiértékelése során (pl. két érték egy operátorral összekötve). Olyanok, mint az emberi nyelvek mondattöredékei.

Készen állsz arra, hogy eggyel tovább lépjünk, és megismerkedjünk a változókkal.

Megjegyzés: Ha a fenti összefoglaló olvasása közben elbizonytalanodtál valahol, kérlek, ugorj vissza az adott fejezetre, és még egyszer olvasd át jó alaposan, mielőtt továbbhaladnál.

6. Teszt

6.1. Ellenőrizd a tudásod!

[10 kérdés, körülbelül 10 perc, kérdésenként 1 pont.](#)

Tipp: Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a „Küldés” (Submit) gombot a teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

Megjegyzés: A tesztet többször is kitöltheted.

Sok sikert!

Olvass, keress, kérdezz

1. Tanuld meg kihúzni magad a gödörből

Időnként el fogsz akadni a programozás során.

Ilyenkor ne ijedj meg, ez teljesen oké - a tapasztalt fejlesztők is elakadnak néha. A művészet az, hogy tudd, hogyan kell továbbelőlni ezeken az akadályokon.

Pontosan ezt fogjuk most megtanulni – a technika neve Read-Search-Ask, azaz **Olvass-Keress-Kérdezz**.

1.1. Olvass (Read)

A tanulás és a későbbi fejlesztői munkák során is többször lesz olyan, hogy a kódod valamiért nem fut le. Az oldalad nem úgy néz ki, ahogy kellene, valami nem történik meg, vagy épp más történik, mint aminek kellene. Ez a programozás természetes része, pánikra semmi ok.

Ilyenkor az első lépés, hogy nekiállsz olvasni.

Először a saját kódodat, sorról-sorra. Az esetek jó részében egy hiányzó karakter, egy be nem zárt zárójel vagy egy plusz szóköz okozza a gondot. A gépek ritkán megengedők az elírásokkal szemben. :) Ha valami nem működik, az első lépésed minden legyen az, hogy figyelmesen átolvasd a saját kódod.

Ha a kód átolvasása nem segít, olvasd el a feladatot, a dokumentációt vagy bármilyen írásos anyagot, amiből éppen dolgozol.

Ez nem csak erre a tanfolyamra igaz, fejlesztőként gyakran fogsz mások kódjával dolgozni, akár egész programokat átalakítani a saját céljaidra. Ilyenkor fontos, hogy olvasd el a másik által írtakat, mert lehet, hogy valami elkerülte a figyelmed vagy elsőre félreérte valamit.

Az első taktika lényege tehát: mielőtt bármilyen más csinálnál, olvasd el a rendelkezésedre álló információt. Az esetek nagy részében már ez meg fogja oldani a problémád. Ha mégsem, akkor folytasd a következő lépéssel.

1.2. Keress (Search)

Ezt most még valószínűleg nem tudod, de leendő fejlesztőként irtó jó helyzetben vagy: a választott szakmád a világ egyik legjobban dokumentált munkája. A programozók világszerte elköpesztő mennyiségű tudást halmoztak fel az interneten, amit – jól kereshetően – el tudsz érni.

Szinte biztos, hogy valaki, valahol már belefutott a problémádba, talált rá megoldást és ezt posztolta is valahol a neten. Épp ezért a Read-Search-Ask metódus második lépése minden-

a keresés.

Szóval, hogyan keres jól egy programozó?

1. Ne kérdéseket tegyél fel a Google-nek

Mivel a Google módszere a kulcsszó-egyezésekben alapul, azt érdemes megtippelned, hogy az általad keresett információ vajon hogyan szerepel a neten.

Mondjuk, hogy az a problémám, hogy nem tudom középre helyezni a **h1** tagben lévő főcímemet a weboldalamon. Megfogalmazhatnám így a keresésemet:

Nem jó helyen van a cím az oldalon, középre szeretném, mit csináljak?

Teljesen valid megfogalmazása a problémának, egy ember értené is, de a Google-ből ezzel ritkábban tudsz használható találatokat kinyerni. Ehelyett próbáld meg kitalálni, hogy milyen szavakat használhat a megoldást tartalmazó oldal:

*h1 HTML tag pozicionálása középre
(Position an h1 HTML tag to center)*

Ezzel rögtön az első két találatban megvan a megoldás. Az eredményes Google keresés egyik titka, hogy próbálj egy picit robotabbruk gondolkodni.

2. Fogalmazz röviden és pontosan

A fenti első példával nem csak az a baj, hogy emberi kérdés formájú, hanem az is, hogy nagyon sok szóval írja le a problémát. Kicsi az esélye, hogy valaki más pont ezeket a kulcsszavakat használja a megoldásnál.

Próbálj mindenkor legkevesebb szóval keresni és azok legyenek a lehető legjellemzőbbek a problémádra. A második példában pont ezért a lehető legkevesebb szóval írtuk le, hogy egy h1 taget szeretnénk középre rakni HTML-nyelvben.

3. Mellékeld a fontos kiegészítő információkat

Az internet nagy. Ha eredményesen akarsz keresni rajta, minél szűkebb területre érdemes lőnöd. Ennek az egyik módja, hogy megfelelő kiegészítő infókat adsz a Google-nek. A fenti második példában ilyen a "HTML" rész hozzáadása. Ezzel mondtuk meg a keresőnek, hogy minket azok a találatok érdekelnek, ahol szöveget HTML-oldalon helyeznek középre.

Ha a "h1 HTML tag" rész helyett mondjuk azt írtuk volna, hogy "szöveg középre helyezése" (ami továbbra is igaz lett volna a problémánkra), akkor valószínűleg rengeteg találatot kaptunk volna, ahol Wordben rakkák szöveget középre.

4. Ha van hibaüzeneted, használd

Ha az elakadásod során kapsz hibaüzenetet, azt mindenkor csatold a keresésedhez - az esetek 99%-ában ezzel pontos találatot kapsz.

Igen, ez látszólag ellentmond a második szabálynak. Azonban ebben az esetben a válasz, amit keresel, nagy valószínűséggel szintén tartalmazni fogja ugyanezt a hibaüzenetet, így pontosabb találatokat fogsz kapni.

Ha ezeket a szabályokat betartod, sokkal jobb keresési találatokat fogsz kapni a Google-tól. :)

Tipp: Egy forrást érdemes megjegyezned, ha programozással kapcsolatos kérdésed van. Ez a [StackOverflow](#) – a világ legismertebb informatikai kérdezz-felelek oldala. Ha problémád van, érdemes ott kutakodnod először.

1.3. Kérdezz (Ask)

Ha a kódod átolvasásával és egy alapos kereséssel sem sikerül rátalálnod a megoldásra, akkor jön a harmadik lépés: kérdezz.

Hol kérdezz

Kérdéseket sok helyen föl lehet tenni programozással kapcsolatban, de egyet ezek közül mindenki megjegyezhet:

Ez az előzőekben már említett [StackOverflow](#). Itt bármikor teheted fel kérdéseket, amit a közösség megválaszol. Ez főként a későbbiekben lesz majd hasznos, amikor már fejlesztőként dolgozol.

Hogyan kérdezz

Ahhoz, hogy segítséget kapj, tudnod kell jól kérdezni. Egy szépen megfogalmazott fejlesztői kérdés valahogy így épül fel:

"Azt szeretném, ha a kódom ezt csinálná..., helyette ez történik..."

Ez elemeire lebontva:

- Mindig írd le, hogy mi a pontos szituáció. Mi az elvárásod, ami nem teljesül? Mit szeretnél elérni, minek kéne történnie? Milyen nyelvben írod a kódot?
- Írd le, hogy mi történik ahelyett, amit várnál. Nem történik meg valami? Más történik, mint aminek kéne?
- Foglald össze, hogy milyen megoldásokat próbáltál meg eddig. Ez nem csak azért fontos, hogy a segítők tudják, hogy milyen ötletek nem működtek eddig, de sokszor ezen a ponton döbben rá az ember, hogy mit nem próbált még meg. Lehet, hogy ennél a lépésnél jóssz rá a megoldásra.
- Oszd meg a kódod vagy a problémás kódrészletet. Elméletben debuggolni nehéz, a segítőknek látnia kell, hogy mi a kód, amit megírtál.

Ha ezeket a szabályokat betartod a kérdezsnél, nagyobb az esélye, hogy gyorsan hasznos választ kapsz.

1.4. Összefoglalás

Legközelebb, ha elakadsz, emlékezz a Read-Search-Ask, Olvass-Keress-Kérdezz formulára:

- Először fésüld át a kódod és a dokumentációt.
- Aztán keress Google-ön, rövid, tiszta kulcsszavakat használva.
- Végül, ha nincs megoldás, tedd fel a kérdésed más programozóknak / tanulóknak.

Ne feledd, mindenki elakad. Tanuld meg, hogyan segíthetsz magadnak.

JavaScript alapok II. (változók)

1. Bevezetés

1.1. Mi az a változó?

Az értékek (*values*) és az operátorok után a JavaScript következő építőeleme a **változó** (angolul *variable*).

Mielőtt megijednél, hogy na most aztán valami nagyon bonyolult dologról lesz szó, nyugodj meg. A változókat egyáltalán nem nehéz megérteni, és már találkoztál is velük az életed során, csak lehet, hogy már elfelejtetted.

Hogy hol? A nyolcadikos matekórán.

Megjegyzés: Igen, megint előbújt a rettegett matematika szó, de szeretnénk megnyugtatni: továbbra sem fogunk matematikafeladványokon dolgozni. Csak egy könnyen érthető példa erejéig vesszük elő a témát.

Talán emlékszel, talán nem, de nyolcadikban a slágertéma a geometria volt. Állandóan három-, négy- és sokszögek kerületét, területét, átlóit stb. számoltatok. Az egyik legtöbbet előforduló szöveges feladat valahogy így hangzott:

„*Egy derékszögű háromszög két rövidebb oldala (a befogói) 18 és 24 cm hosszúak. Milyen hosszú a hosszabbik oldal (azaz az átfogó)?*”

A tanárod (ha jófej volt) ilyenkor megsúgta, hogy a Pitagorasz-tételre lesz szükséged a megoldáshoz, amely kimondja, hogy:

- bármely derékszögű háromszög átfogójának (c) négyzete
- megegyezik a befogók (a és b) négyzetösszegével.

Azaz képletek formában: $a^2 + b^2 = c^2$.

Talán már sejted, hogy hova tartunk mindezzel. **A fenti megoldóképletben az a, b és c betűk az egyenlet változói.** Egy konkrét számot jelölnek az egyenletben, amelyet be tudsz helyettesíteni a valós értékkel, ha tudod azt.

Például az eredeti szöveges feladatban tudjuk a és b értékét is, tehát behelyettesíthetjük velük a megfelelő változókat:

18² + 24² = c²

És innentől az egyenlet egyszerűen megoldható.

Ahogy látod, a matematikában a változó egy betű (vagy szimbólum), amely egy adott számértéket reprezentál.

A JavaScript-változók nagyjából ugyanígy működnek, három lényeges különbséggel:

1. Nem betűk vagy szimbólumok, hanem **szavak, egyértelmű kifejezések**.
2. Nemcsak számokat, hanem **bármilyen JavaScript-adattípust** (stringeket, booleant, objektumokat stb.) reprezentálhatnak.
3. Nem(csak) egyenletekben használhatjuk őket, hanem **mindenhol a kódban**.

Egy köznapiabb hasonlattal élve, a változók olyanok, mint a nevek. Ahogy a te neved reprezentál téged, úgy a változók adatokat reprezentálnak. Amikor létrehozunk egy változót, fogunk egy valamilyen JavaScript-értéket, és elnevezzük azt. Később ezen a néven hivatkozni tudunk az értékre.

Megjegyzés: A fenti hasonlat egy kitétellel igaz: míg a te neved mindig téged jelöl, **addig a változók felvehetnek új értéket**. Így ugyanaz a változónév reprezentálhat új adatot, míg a te neved mindig téged jelöl.

Ellenőrző kérdés

Az alábbi definíciók közül melyik illik a JavaScript-változókra? Válaszd ki a helyes választ!

- A JavaScript-változók szimbólumok vagy betűk, amelyek számokat reprezentálnak a kódban.
- A változók előre megadott értékek a kódban.
- A változók szavak, amelyek egy adott number vagy string értéket (value) reprezentálnak a kódban.
- **A változók szavak, amelyek egy adott értéket (value) reprezentálnak a kódban.**

1.2. Miért használunk változókat?

Ha a változók adott értékeket reprezentálnak, akkor felmerülhet benned a kérdés, hogy nem lenne-e egyszerűbb kapásból a konkrét értékeket használni. Minek teszünk plusz egy kört a változókkal?

Két okból:

1. **Nem mindig tudjuk a konkrét értéket.**
2. **Az értékek változhatnak.**

Hogy kézzelfoghatóbbá tegyük ezt számodra, lássunk egy apró gondolatkísérletet.

A benzinköltség-kalkulátor

Tegyük fel, hogy nyár van, és szeretnél elutazni vakációzni. A tengerpartra mennél autóval, és szeretnéd tudni, hogy mennyibe fog kerülni odafelé az útiköltség. Mivel épp programozni tanulsz, szeretnél írni egy programocskát, amely elvégzi helyetted a számítást.

Lássuk, hogyan!

1.3. Változó a nem ismert érték helyén

Tudod, hogy az úticéled **600 km**-re van, és ismered az autód átlagfogyasztását: **100 km-en 6,5 liter**. Először azt szeretnéd kiszámolni, hogy 600 km alatt összesen mennyi lesz a fogyasztás.

Nyisd meg a böngésződ konzolját, és másold bele az alábbi JavaScript-sort, amely ezt számolja ki:

```
600 * 6.5 / 100
```

39 liter – ennyit fog az autód összesen fogyasztani a partig. Az összköltséghez már csak meg kell szoroznod ezt a számot a benzin literárával... és itt elakadsz. Mivel nem tudod a benzin literárát, nem tudod befejezni a programod.

Megjegyzés: Ebben a gondolatkísérletben a Google sajnos nem létezik, így nem tudsz egyszerűen rákeresni a benzin árára.

Látod, itt jön jól, hogy léteznek változók a JavaScriptben. Bár nem tudod a pontos értéket, behelyettesítheted a **benzinara** (benzin ára) változót a programodba:

```
600 * 6.5 / 100 * benzinara
```

Így a programod logikája (a rész, amely a konkrét számítást végzi) készen áll. Ehhez a sorhoz nem kell már hozzájárulnod, egyszerűen csak megadod majd a **benzinara** változó értékének a literárat, ha megszerezted ezt az információt.

Megjegyzés: Ha megpróbáld a fenti sort lefuttatni a böngészőben, egy hibaüzenetet fogsz visszakapni: **Uncaught ReferenceError: benzinara is not defined**. Ez azért történik, mert a változókat előbb egy külön lépésben létre kell hoznunk, csak azután használhatjuk őket. Ezen most egyelőre ne aggódj, ettől még a kód, amit látsz, teljesen valid. Hamarosan pedig azt is megtanulod, hogyan kell létrehozni változókat.

Ellenőrző kérdés

Miért jó, hogy léteznek változók a JavaScriptben? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- **A változók lehetővé teszik számunkra, hogy úgy írunk programokat, hogy a kód által használt pontos értékeket még nem tudjuk.**
- A változók figyelmeztetnek arra, ha helytelen értéket akarnánk megadni a helyükön.
- A változók lehetővé teszik, hogy anélkül írunk kódot, hogy matematikai számításokat kéne végeznünk.
- A változók használatának nincs különösebb haszna. Mindegy, hogy azokat vagy konkrét értékeket használunk, ez leginkább egyéni ízlés kérdése.

1.4. Újrafelhasználható kód változókkal

A vakációd hatalmas siker volt, nagyon jól érezted magad a tengerparton. Eltelik egy fél év, és ahogy eljön a síszezon, elhatározod, hogy szeretnél ellátogatni a hegyekbe, csúszkálni egy kicsit.

Ismét autóval mennél, és ismét szeretnéd használni a kalkulátorod, amely így néz ki:

```
600 * 6.5 / 100 * benzinara
```

Ránézel a programodra, és megállapítod, hogy a **benzinara** változón kívül minden elavult benne. A hegyek nem 600, csak 350 kilométerre vannak tőled, és mivel megjavítattad az autód motorját, a fogyasztás visszaesett 5,2 literre.

Két út áll előtted:

- átírhatsz az értékeket magában a programban, vagy
- lecserélheted a konkrét számokat változókra, így többet nem kell hozzányúlnod a kódhoz.

Jó programozópalántaként az utóbbi opciót választod:

```
tavolsag * atlagfogyasztas * benzinara
```

Ebben a formájában már nemcsak egy konkrét útra használható, hanem bármilyen utazásához, bármilyen autóval és bármilyen benzinárák mellett kiszámolja az útköltséget. Így ahelyett, hogy újra és újra átírnád a programod minden alkalommal, amikor használni szeretnéd, elég megadnod a változók értékét egy felületen.

Megjegyzés: Képzeld el, hogy a kalkulátorod nem egy sorból, hanem 15 000-ból áll, és nem egy helyen szerepel benne a távolság értéke, hanem 450-en. Ezt átírni már nem egyszerű kényelmetlenség lenne, hanem komoly akadálya a fejlesztéseknek. Arról nem is beszélve, hogy minden egyes felhasználónak saját változatot kéne kódolnod a programodból, ami teljesen életszerűtlen.

Ellenőrző kérdés

Milyen fenntarthatósági előnnyel jár a változók használata? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- Változókat könnyebb létrehozni, mint értékeket.
- Egy változó több értéket is tárolhat, így kevesebb kódot kell írnunk.
- **Amikor egy adott érték megváltozik, nem kell újraírnunk a kódot mindenhol, ahol az az érték szerepel. Elég megváltoztatni a változót, amely tárolja az értéket.**
- Ha a változót egyszer megadtuk, utána az értéke sosem változhat meg, így nem kell aggódnunk, hogy elfelejtjük az értéket kódolás közben.

2. Változók létrehozása (deklarálása)

2.1. Egy változó anatómiája

Az előző fejezet végére jó eséllyel egy égető kérdésed merült fel: „**Rendben, de hogyan hozhatom létre a változókat?**”

A következő módon:

```
var theNameOfTheVariable = value
```

Lássuk elemenként:

- A `var` egy speciális JavaScript-kulcsszó (a *variable*, azaz változó angol szó rövidítése). Ezzel jelezzük a kódban, hogy itt egy változó következik.
- Ezután jön a változó neve (`theNameOfTheVariable`). Ezt a nevet fogjuk használni arra, hogy hivatkozzunk a változó által tárolt értékre. Kicsit később megbeszéljük az elnevezésre vonatkozó szabályokat, egyelőre annyit jegyezz meg, hogy a névnek minden angolnak kell lennie.
- Az egyenlőségjel (`=`) egy olyan operátor, amellyel eddig nem találkoztál. A neve *assignment* (hozzárendelő) operátor, és azt jelzi, hogy a bal és a jobb oldalán álló dolgok egymásnak felelnek meg. Jelen helyzetben azt jelöli, hogy innentől fogva a bal oldali változó a jobb oldali értéket reprezentálja.
- Végül a `value` (érték) konkrétan az az adat, amelyet reprezentálni szeretnél a változóval. Ahogy már említettük, ez lehet bármilyen adattípus (szám, string, boolean stb.).

Próbáld ki élesben! Nyisd meg a böngésződ konzolját, és írd be a következő sort:

```
var number = 9
```

Üss egy Entert. Mi történt?

A konzol visszadobott annyit, hogy `undefined`. Ennek örülünk: ez annak a jele, hogy sikerült deklarálnod a változót. Próbáld meg beírni a konzolba ezt:

```
number
```

Üss egy újabb Entert. Most mi történt?

A böngésződ emlékszik, hogy az imént dekláráltad a `number` változót, és a `9`-es számot tároltad el benne. Így amikor meghívted a változót, a tárolt információt, azaz a `9`-et adta vissza. **A böngésző szempontjából a `number` változónév és a `9`-es szám ugyanaz.**

Megjegyzés: Két kifejezés, amelyeket gyakran fogsz hallani programozóktól, a „deklarálj egy változót” (angolul: *declare a variable*) és „tárold el az értéket egy változóban” (angolul:

store/assign the value in/to a variable). Az első annyit jelent, hogy hozz létre egy változót, a második pedig azt, hogy add meg a változó értékét.

Ellenőrző kérdés

Az új tudásod birtokában tegyük valóban működővé az előző fejezetben bemutatott benzinköltség-kalkulátort. Nyiss meg egy új bint a JS Binen. A JS panel első sorában deklarálj egy `gasPrice` (benzin ára) változót, és az értékének adj meg `384`-et. Utána másold be ezt a sort a változód alá: `console.log(600 * 6.5 / 100 * gasPrice)`. A „Run” gomb megnyomása után mennyit kaptál eredményül a JS Bin konzoljában?

- 624,96
- 149,76
- **14 976**
- 29 952

2.2. JavaScript-kulcsszavak (keywords)

Ez egy kiváló alkalom arra, hogy egy pillanatra megállunk, és beszéljünk két új JavaScript-fogalomról:

- a **kulcsszavakról** (*keywords*) és
- az **utasításokról** (*statements*).

Kulcsszavak (keywords)

Az előző leckében említettük, hogy a `var` kulcsszónak számít a JavaScriptben, és azt üzeni a böngésző számára, hogy figyeljen, itt egy változó következik.

A `var` nem az egyetlen kulcsszó, jó pár másik is létezik, és a működési elvük ugyanaz: valamiféle funkciót jelölnek a JavaScriptben.

Ahogy haladsz előre a tanulmányaidban, szép sorban meg fogsz ismerkedni a kulcsszavakkal és viselkedésükkel. Addig csak jegyezd meg, hogy léteznek, és az értékek, illetve operátorok mellett ezek újabb elemi egységei a JavaScript nyelvnek.

Megjegyzés: A kulcsszavak egy fontos jellemzője, hogy ezek úgynevezett **foglalt szavak** (angolul *reserved words*). Kulcsszavakat nem adhatsz meg változók vagy függvények (róluk később lesz szó) neveként. Ha érdekel, itt [a kulcsszavak teljes listája](#) angolul.

Ellenőrző kérdés

Milyen gyakorlati jelentősége van a kulcsszavaknak kódolás közben? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- A kulcsszavak úgynevezett foglalt szavak, amelyekkel csak bizonyos funkciójú változókat nevezhetünk el.
- A kulcsszavak olyan szavak, amelyeket bárhol használhatunk a kódban, és akár több változót is elnevezhetünk velük.

- A kulcsszavak úgynevezett foglalt szavak, így csak azután használhatjuk őket változók vagy függvények elnevezésére, miután feloldottuk őket.
- **A kulcsszavak úgynevezett foglalt szavak, így ezeket nem használhatjuk változók és függvények elnevezésére.**

2.3. JavaScript-utasítások (statements)

A 2.1. leckében találkoztál egy ilyen sorral:

```
var number = 9
```

Ha szeretnénk folyószöveggel kiírni, hogy ez a sor mit mond a számítógépnek, akkor ezt jegyezhetnénk le:

Készíts egy „number” nevű változót,
és tárold el benne a kilences számot!

Ahogy látod, ez egy utasítás (angolul *statement*). minden JavaScript-program ehhez hasonló utasítások sorozata, amelyeket a gép sorról sorra haladva végrehajt.

Ha még emlékszel, az első fejezetben azt mondtuk, hogy a programnyelvek és az emberi nyelvek között sok hasonlóság létezik:

- Az értékek, operátorok és kulcsszavak hasonlóak az egyes szavakhoz.
- A például értékekből és operátorokból létrejövő kifejezések (*expressions*) pedig hasonlóak a mondattörökékekhez.

Megjegyzés: Emlékeztetőül – minden olyan kódrészlet expressionnek számít, amely értékké válik a kód értelmezése során.

E logika mentén az utasítások olyanok, mint az emberi nyelvek mondatai. Komplettek, önmagukban érthető jelentéstartalommal bírnak, és ha eleget egymásra halmozunk belőlük ügyesen, akkor szöveg (azaz jelen esetben program) lesz belőlük.

A JavaScript-utasítások a következő alkotóelemekből állhatnak:

1. értékek (*values*),
2. operátorok,
3. kulcsszavak (*keywords*),
4. kifejezések (*expressions*),
5. kommentek.

Megjegyzés: Ezek közül már mindegyikkel találkoztál a kommentek kivételével. Utóbbit nemsokára pótoljuk.

Ahogy a mondatok végét írásjel jelzi, úgy a JavaScriptben is van egy speciális karakter, a pontosvessző (;), amellyel az utasítások végét jelöljük:

```
var number = 9;
```

Megjegyzés: Kezdj el hozzászokni, hogy az utasítások végére kiteszed a pontosvesszőt. Az egyik leggyakoribb hiba, amelyet kezdők elkövetnek, hogy valahol lehagynak egy pontosvesszőt, és emiatt nem fut le az egyébként helyesen megírt program.

Ellenőrző kérdés

Milyen részekből állhat egy JavaScript-utasítás (*statement*)?

- értékek, operátorok, kulcsszavak és kommentek
 - Hmm, valami kimaradt... Nézd át még egyszer a leckét, keresd meg a hiányzó elemet!
- kulcsszavak, operátorok és kifejezések
 - Hmm, valami kimaradt... Nézd át még egyszer a leckét, keresd meg a hiányzó elemet!
- operátorok, kommentek, értékek és kifejezések
 - Hmm, valami kimaradt... Nézd át még egyszer a leckét, keresd meg a hiányzó elemet!
- **kommentek, kulcsszavak, operátorok, értékek és kifejezések**
 - Nagyon jó! Szépen összegezted, hogy miről is tanultál ebben a leckében.

2.4. Kommentek a JavaScriptben

Mielőtt továbbmennénk, egy pillanatra állunk meg, és beszéljünk picit a JavaScript-kommentekről, ha már az előző leckében előkerültek.

Esszenciáját tekintve a JavaScript-kód nem más, mint egy szövegfájl, amelyet a böngésző „elolvashat” és végrehajthat. minden sort, amelyet egy `.js` fájlba (vagy a konzolba vagy a JS Bin JS paneljére) írsz, a böngésző JavaScript-kódnak tekint, és megpróbálja azt végrehajtanı.

Megezik azonban az is, hogy szeretnél a fájlodban olyan szöveget elhelyezni, amelyet nem kódnak szánsz, és azt akarod, hogy a gép ignorálja végrehajtáskor. **Az ilyen esetekre találták ki a JavaScript-kommenteket.**

Kommenteket két módon hozhatsz létre egy JavaScript-fájlban:

```
// 1) Ez egy egysoros komment. Dupla perjel (forward slash) jelöli  
a sor elején.  
  
/*  
2) Ez egy többsoros,  
úgynevezett block komment.  
Az elejét perjel + csillag,  
a végét csillag + perjel jelzi.  
A kettő közti szöveget a böngésző ignorálja.
```

```
 */
```

A kommentek tartalma bármi lehet (akár valid JavaScript-kód is), a böngésző ignorálni fogja azt a program futtatásakor.

Mire használjuk a kommenteket?

Programozóként a kommenteket leggyakrabban a kódod magyarázására fogod használni. Ahol nem egyértelmű magából a JavaScriptból, hogy mi történik, egy komment sok időt tud spórolni a munkatársaiddnak, akik próbálják megérteni a munkád. Arról nem is beszélve, hogy neked mennyit fog segíteni, ha sok hónap után kell visszatérned egy kód részletéhez, amelyet ugyan te írtál, de már rég elfelejtettek, hogyan működik.

Megjegyzés: A kódod commentelésére természetesen léteznek jó gyakorlatok, ezekkel szépen fokozatosan fogsz megismerkedni a tanulmányaid során.

A CodeBerry-ben kommentekkel két helyzetben fogsz találkozni:

1. Ha szeretnénk egy megjegyzést fűzni egy adott kódsorhoz valamelyik beszúrt kód blokkban.
2. Ha komplexebb példáknál nem írjuk meg a teljes kódot, csak az éppen fontos részt. A maradék helyeken kommenteket szűrünk be, amelyek leírják számmodra, hogy a kód ott mit csinálna.

Oké, ennyi volt a kitérő – még egy gyors feladat neked, aztán visszatérünk a változókhöz.

Ellenőrző kérdés

Az alábbi lehetőségek közül válaszd ki azt, ahol a komment szintaxisa helytelen!

- `// Ez egy komment.`
- `/* Ez is egy komment. */`
- `/* Mi több,
ez is az. */`
- `/* Sőt, ez is. */`

2.5. A változók elnevezésére vonatkozó szabályok és szokások

Az előző modulban már megtapasztalhattad, hogy a programnyelveknek nagyon pontos utasításokat kell adni, ha azt szeretnénk, hogy valóban azt csinálják, amit elvárunk. Az elgépeléseket és pontatlanságokat a JavaScript sem türi, ezért nagyon oda kell figyelni, mit és hogyan írunk.

A változók elnevezésének is vannak szabályai, amelyeket be **kell** tartanod. A hibás elnevezéseket mindenhol dupla felkiáltójellel (!! jelöltük:

1. A változónévben **lehet** betű, szám, dollárjel (\$) és alulvonás (_), de:
2. Nem lehet benne szóköz (!! `var no space`).
3. Nem kezdődhet számmal (!! `var 3isacrowd`).
4. Nem lehet benne speciális karakter (!! `var v@ria§le`).

5. Nem lehet reserved word (!! `var` `var`).

A változók érzékenyek a kis- és nagybetűkre (angolul *case sensitive*-ek), tehát `number` és `NUMBER` két különböző változó.

Ezen kívül vannak olyan elnevezési konvenciók – szokások –, amelyek nem kötelezőek, de mivel a programozói gyakorlat tapasztalatából születtek, általában javítják a kód átláthatóságát. Neked is érdemes már a kezdetektől betartani ezeket, hogy (jó) szokássá váljanak.

Ezek a szokások a következők:

1. A változó neve legyen angol.
2. A név utaljon a tartalmazott adatra.
3. Kerüld a betűket (pl. `x`) vagy rövidítéseket (pl. `sum2num`). Helyette írd ki a változó teljes, beszédes nevét.
4. Ha a változónév több szóból áll, akkor az első szót írd kisbetűvel, a többi szó első betűje pedig legyen nagy: `var moreThanOneWord`.

Megjegyzés: A négyes pontban bemutatott írásmódot **camelCase-nek** hívják. Ez a módszer a következő modulokban sokszor elő fog kerülni, szóval érdemes megjegyezned.

Ellenőrző kérdés

Az alább felsorolt, nagymacskákra vonatkozó változók közül melyik elnevezése a helyes? Válaszd ki a helyes megoldást!

- `var types = 'lion, cheetah, tiger';`
 - Emlékezz: a változó elnevezésének utalnia kell a tartalomra. Biztos, hogy nincs egy ennél specifikusabb a listában? Nézd meg újra!
- `var bigCats = 'lion, cheetah, tiger';`
 - Szuper, minden szabályt és szokást megjegyeztél! Mehetsz is a következő leckére!
- `var big cats = 'lion, cheetah, tiger';`
 - A változó elnevezése nem tartalmazhat szóközt. Nézd meg újra a listát, és keresd meg a helyes választ!
- `var BigCats = 'lion, cheetah, tiger';`
 - Majdnem jó, azonban a camelCase írásmód értelmében a változó nevének első szavát kisbetűvel kell írni, itt viszont naggyal szerepel. Keresd meg a helyes választ!

3. Értékek (adattípusok) a változókban

3.1. Az üres változó

A minden napokban gyakran előfordul, hogy programozóként nem tudod előre egy változó majdani értékét. Gondolj például a webshopok regisztrációs oldalaire, ahol a felhasználó megadja a nevét, szállítási címét, telefonszámát stb.

Ezeket a kódodnak fogadnia kell a másik oldalon, azaz kell hogy legyen változód előkészítve, de az még nem tartalmazhat semmit.

E célra az érték nélküli, úgynevezett **üres változókat** (angolul *empty variable*) használjuk. Ilyenkor – logikusan – nem adunk meg értéket a változónak, amikor deklaráljuk azt:

```
var userName;  
var userAddress;  
var phoneNumber;
```

Nézzük ezt meg ezt a böngésződ konzoljában! Másold be a fenti kódot!

A konzol annyit válaszolt, hogy **undefined**. Ezt már ismerjük, azt jelenti, hogy a böngésző elmentette a változót. Válaszd ki valamelyik változót, mondjuk a **userName**-et, és írd be a konzolba, majd üss Enter-t.

Megint azt kaptad vissza, hogy **undefined**, de ez most egy másik **undefined**.

Az előző, „Adattípusok” modulban bemutattuk neked az **undefined** értéket. Ha emlékszel, ott azt mondtuk, hogy ez egy úgynevezett üres érték (*empty value*), melyet a JavaScript olyan esetekben ad vissza, amikor nincs értelmes érték, amit visszaadhatna.

Azt is említettük, hogy az **undefined** szituációtól függően másat és másat jelenthet. Jelen esetben az üzenete pontosan az, amit a szó maga jelent: **undefined** a **userName** változód értéke, mert még nem adtad meg (nem definiáltad) az értékét.

Ellenőrző kérdés

Ha szeretnél létrehozni egy üres változót, amely az októberben született ismerőseid nevét fogja tárolni egy stringben, melyik a helyes szintaxis az alábbiak közül?

- **var friendsBornInOctober;**
- **var friendsBornInOctober = '';**
 - Sajnos nem. Ez nem egy üres változót, hanem egy üres stringet definiál. Próbáld meg újra!
- **var friendsBornInOctober**
 - Sajnos nem. Nézd meg tüzetesen a kódot! Mintha hiányozna valami a végéről, nem? Mit is mondtunk, hogyan zárul minden JavaScript-utasítás?
- **var friendsBornInNovember;**
 - Szintaktikailag helyes, de azt mondtuk, hogy a változó nevének utalnia kell a tartalmazott értékre. Próbáld újra!

3.2. Változók valódi értékekkel

A változók többsége persze nem üres, amikor létrehozzuk őket. A 2. fejezetben elsajátítottad

a változók deklarálásának elméletét, most lássuk, a gyakorlatban hogyan hozol létre változókat a különböző adattípusokkal.

Számok

A számokat már láttad, ezek egyszerűek. Az általános szintaxis a következő: `var` kulcsszó + változónév + assignment operátor + szám (idézőjelek nélkül!) + pontosvessző. Például:

```
var numberOfDwarfs = 7;
var numberOfSnowWhites = 1;
```

Stringek

A stringek pontosan ugyanezt a formulát követik, azzal a kivétellel, hogy idézőjelekkel kell körbevenned a stringet:

```
var welcome = 'Üdvözöljük a Mágiaügyi Minisztériumban!';
var warning = 'Kérjük, vigyázzon a fejére és pálcájára ereszkedés közben!';
```

Boolean

Ezt most még csak hidd el nekünk, később aztán magad is tapasztalni fogod: sokszor van olyan, hogy létrehozol egy változót, amelynek adsz egy igaz (`true`) vagy hamis (`false`) értéket. Az ilyen változó egyfajta kapcsolóként működik a programokban – a kód egy másik része valamiért átváltja az ellenkező boolean értékre, és ekkor történik valami.

Gyakorlati szempontból boolean-t tárolni nem nagy kaland, az előbb lefektetett alapelveket követi:

```
var switchOne = true;
var switchTwo = false;
```

Tipp: Figyelj arra, hogy ne tegyél idézőjeleket a booleanek köré! Hiába szavak, ezek nem stringek, így „csupaszon” kell hagynod őket.

Megjegyzés: Egyelőre ezt a három adattípust tanuljuk meg. Belátható időn belül nem fogsz `Undefined` és `null` típusú értékeket belátható időn belül nem fogsz változóban tárolni, az objektumokat (és a hozzájuk kapcsolódó tömböket) pedig később, külön modulokban beszéljük meg.

Ellenőrző kérdés

Az alábbi lehetőségek közül melyikben állnak helyesen deklarált változók?

- `var number = '42'; var string = this is a string; var boolean = 'true';`

- `var number = 42; var string = 'this is a string'; var boolean = true;`
- `var number 42; var string 'this is a string'; var boolean true;`
- `var number = 42 var string = 'this is a string' var boolean = true`

3.3. Kifejezések változókban és változók kifejezésekben

Emlékeztetőül: a JavaScriptben kifejezésnek (*expressionnek*) számít minden olyan kódrészlet, amely a kód futásakor valamilyen értékké értékelődik ki. Például:

```
10 + 10 // → A böngésző egy darab 20-as számmá fogja alakítani.  
'egy,' + ' megérett a meggy' // → A böngésző az 'egy, megérett a  
meggy' stringgé fogja alakítani.
```

Talán nem lepődsz meg azon, ha azt mondjuk, hogy a változók nemcsak egyes értékeket, de kifejezéseket is képesek tárolni:

```
var sumOfTwoNumbers = 10 + 10;  
var sumOfTwoStrings = 'egy,' + ' megérett a meggy';
```

Másold be a fenti két sort a böngésződ konzoljába, majd üss egy Entert, aztán hívd elő a frissen definált változókat. Figyeld meg, mi történik! A böngésző nem a konkrét kifejezést, hanem a kifejezés értékelésekor keletkező értékeket mentette el a változókba.

A dolog fordítva is működik – a kifejezésekben állhatnak változók is. Másold be az alábbi kódrészletet a böngésző konzoljába, és nézd meg, hogy mi történik:

```
var myName = 'Alex';  
'Helló, a nevem ' + myName;
```

A számítógéped behelyettesítette a `myName` változó értékét ('Alex'), kombinálta a két stringet, és visszaadta a végeredményként keletkező stringet: Helló, a nevem Alex.

Megjegyzés: Ha szemfüles vagy, észrevehetted, hogy az *expression* definíciója szerint a változók maguk is kifejezések, hiszen az értékükkel helyettesítődnek be, amikor lefut a program.

Feladat

Nyiss meg egy új bint a JS Binen! Hozz létre egy `ultimateAnswer` változót, amely a következő kifejezést tárolja el: $1890 / 90 * 2$. Utána írasd ki a konzollal a változót, előtte a következő szöveggel: 'A válasz az életre, a világmindenségre, meg mindenre: '.

A programod a következő kritériumoknak feleljen meg:

- A JavaScript-panelen dolgozz.
- A szöveg kiíratásához használ a `console.log()` parancsot.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.4. Változók használata változókban

Eddig láttad, hogy a változók tárolhatnak egyes értékeket és kifejezéseket (*expressionöket*) is. Készülj fel, most érünk a nyúlureg legaljára: arra is van mód, hogy **két vagy több változóval végrehajts valamilyen számítást, és a végeredményt elmentsd egy újabb változóban**.

Lássunk egy gyakorlati példát:

Biztosan találkoztál már olyan online cikkekkel, ahol a hírportál megkérdezi, hogy elmúltál-e 18 éves, mielőtt engedi megnézni a hírt, mert az oldalon valamilyen explicit tartalom szerepel. Ilyenkor általában a születési évedet kell beírnod, és a böngésző ez alapján kiszámolja, hogy elmúltál-e tizenyolc.

A háttérben futó kód (nagyon leegyszerűsítve) valahogyan így nézhet ki:

```
var userBirthYear;
var currentYear;

/*
Itt egy kódblokk, amely
feltölti a fenti két változót tartalommal.
Megkérdezi a felhasználó születési évét,
és lekérdezi az idei évet a netről.
*/

var userAge = currentYear - userBirthYear;

/*
Itt pedig egy másik kódblokk, amely
eldönti, hogy a userAge kevesebb-e, mint 18,
és ez alapján továbbengedje-e a felhasználót vagy sem.
*/
```

Ha szemfájles vagy, láthatod, hogy a `var userAge = currentYear - userBirthYear;` sor egy már ismert szerkezet. A `userAge` változóba egy kifejezést (expressiont) mentünk el. Az újdonság csupán annyi, hogy a kifejezés ezúttal teljes egészében változóból áll.

Ez egy gyakori minta a programozás során, ilyen kóddal még nagyon sokszor fogsz találkozni, és te magad is sok ilyet fogsz írni.

Feladat

Egy középiskolai osztályban a tanulók jegyei a legutóbbi irodalomdolgozaton a következőképp alakultak: 5, 5, 3, 5, 4, 2, 2. Nyiss meg egy új bint a JS Binen, és tárold el az osztály tanulmányi átlagát egy változóban, majd írasd ki ennek az értékét a konzolba.

A programod a következő kritériumoknak feleljen meg:

- A JavaScript-panelen dolgozz.
- A tanulók számát tárold el egy `studentCount` nevű változóban.
- A jegyeket add össze a JavaScript segítségével, és a kifejezést mentsd el egy `sumOfGrades` nevű változóban.
- A tanulmányi átlag kiszámításához oszd el a `sumOfGrades` változót a `studentCount` változóval, és a kifejezést tárold el egy `meanOfGrades` nevű változóban.
- A `meanOfGrades` értékének kiírásához használd a `console.log()` parancsot.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.5. A változók értékének módosítása

Az utolsó nagy koncepció, amelyet meg kell értened a változókkal kapcsolatban, hogy **az általuk tárolt értékeket megváltoztathatod**.

Hogy megértsd ennek a jelentőségét, gondolj a különböző internetes szolgáltatások (Facebook, Amazon stb.) profiloldalaira. Ezeken általában meg tudod adni a felhasználóneved, jelszavad, valódi neved, e-mail- címed stb. Az egyszerűség kedvéért feltételezzük, hogy ezeket az információkat az adott app különböző változókban tárolja.

Képzeld el, milyen használhatatlanok lennének ezek az oldalak, ha az először beírt információidon soha többé nem tudnál alakítani. Szerencsére a valóságban ez nincs így – egy változó értékét bármikor felülírhatod egy újabbal.

Próbáljuk ki ezt a gyakorlatban! Nyisd meg a böngésződ konzolját, és deklarálj egy `age` nevű változót. Értéknek add meg a saját korodat. A példában én is a sajátomat fogom használni:

```
var age = 25;
```

Szuper, a böngészőm tudja, hogy az `age` egyenlő `25`-tel. Én azonban szeretnék egy kicsit bőlcsebb lenni, és 5 évet hozzáadnék a koromhoz:

```
age = 30;
```

Figyeld meg, hogy nem kellett kirakni a `var` szócskát az `age` elő. Azért nem, mert a böngésző már tudja, hogy az `age` egy változó. Bontsuk fel picit a szintaxist:

- `age` → Deklarálja, hogy az `age` nevű változóval akarunk dolgozni.
- `=` → A már ismert assignment operátor azt mondja a böngészőnek, hogy „figyelj, a bal oldali dolog meg fog egyezni a jobb oldalival”.
- `30` → Az `age` változó új értéke.
- `;` → A JavaScript-utasítást (*statementet*) záró pontosvessző.

Adj hozzá te is öt évet az életkorodhoz a konzolban, majd írd be újra az `age` változót, és üss egy Enter-t. A változó új értéke fog megjelenni, ha minden jól csináltál.

Megjegyzés: Mindezek alapján egy új hasonlatot alkothatunk a változókra. Ha a JavaScript egy polip, akkor a polip karjai a változók, az értékek (*values*) és kifejezések (*expressions*) pedig kisebb-nagyobb falat sütik. (Ebben az analógiában a JavaScriptünk egy édesszájú polip.) A karok megfoghatnak egyes sütiket, de bármikor el is engedhetik azokat, hogy másikat fogjanak meg.

Fontos: az új értéknek nem kell ugyanabba az adattípusba tartoznia, mint a réginek. A fenti példában mondhattad volna azt is a konzolnak, hogy `age = 'harminc'`, és onnantól az `age` változó a '`harminc`' stringet tárolta volna.

Ellenőrző kérdés

Valéria örökbefogadott egy kutyust, akinek első körben a Brutus nevet szerette volna adni, úgyhogy deklarált egy `var dogName = 'Brutus'`; változót. Viszont pár nap után rájött, hogy az új kedvencéhez sokkal inkább illik a Füles név. Az alábbi négy lehetőség közül melyikkel tudná Valéria megváltoztatni a `dogName` változó értékét?

- `var dogName = 'Füles';`
 - Sajnos nem jó a válasz. Kérlek, nézd át újra a leckét!
- `dogName = 'Füles';`
 - Helyes válasz, már tudod, hogyan kell megváltoztatni egy változó értékét!
- `dogName = Füles;`
 - Sajnos nem jó a válasz. Emlékezz, mit tanultál a stringek formai követelményeiről!
- Nem lehet felülírni egy változó értékét.

- Sajnos nem jó a válasz. Kérlek, nézd át újra a leckét, és keresd meg a helyes választ!

4. Változók deklarálása a JavaScript legújabb verziójában

4.1. Új kulcsszavak: let és const

A JavaScript egy folyamatosan változó programnyelv, amelynek 2015-től kezdve évente jön ki újabb verziója. minden verzió újabb eszközöket ad a programozók kezébe, és újabb finomításokat végez a nyelven.

A tanulmányaid során a 6-os verziót, és azazzal behozott újításokat fogjuk bemutatni, időnként ehhez hasonló kitérőket beiktatva. Mivel ez a változat volt 2015-ben a nyelv eddigi utolsó nagyobb frissítése, amire a későbbi változatok is támaszkodnak, ezért jelentős eltérések nincsenek köztük.

Megjegyzés: A kezdeti verziókat a következő módon jelölték: ES1, ES2, ..., egészen ES5-ig. Az általunk bemutatott 6-os verziót még ES6 néven kezdték fejleszteni, de végül az ES2015 nevet kapta, mert 2015-től átálltak az évszámos jelölésekre (ES2015, ES2016, stb). Az ES rövidítés az **ECMAScript** betűszóból jön. Az ECMAScript az a technikai specifikáció, amely alapján a JavaScriptet megalkották. Némileg leegyszerűsítve: **az ECMAScript szabvány a tervrajz, a JavaScript pedig a tervrajz alapján épített, működő termék**. Ne aggódj ezen túl sokat, csak azért szerettük volna megemlíteni, hogy ne zavarodj össze, ha találkozol az ES vagy ECMAScript kifejezésekkel a saját a kutatómunkád során. Oké, vissza a változókhöz!

Az ES5 verzió csak a **var** kulcsszót ismerte változók deklarálására, ám a programozók több ponton is kényelmetlenségekbe ütköztek a használata során. Éppen ezért az ES2015 verzióban két újabb kulcsszó – a **let** és a **const** – is megjelent, amelyek ezeket a problémákat igyekeznek kiküszöbölni.

Lássuk, mik voltak a gondok a **var**-ral, és hogyan segítenek az új kulcsszavak!

4.2. Változó deklarálása a let kulcsszóval

Hosszú programok írása közben könnyen előfordulhat, hogy elfeledkezünk arról, már deklaráltunk egy változót, sőt értéket is adtunk neki. Így előfordulhat, hogy véletlenül felülírunk valamit, amit nem kellene, és mivel a JavaScript nem jelez hibát, nekünk erről akkor fogalmunk sincs.

Nézzük meg ezt a gyakorlatban! Másold be a következőket a böngésződ konzoljába, és minden sor után üss Entert!

```
var age = 25;  
var age = 30;
```

Minden simán ment, mindenkor azt láttad, hogy `undefined`, azaz a változó létrejött. Ha leellenőrzöd az `age` változó beírásával, hogy annak mi a tartalma, látod, hogy `30`.

Most egy pillanatra állj meg, és képzeld el, hogy hiába az azonos név, ez a két változó nem ugyanarra az adatra akart vonatkozni. Az első mondjuk a te életkorod, míg a második a kedvenc vörösborodé. Amit szerettél volna csinálni, az ez:

```
var myAge = 25;  
var ageOfMyFavoriteWine = 30;
```

Ehelyett amit elértél, az ezzel ekvivalens:

```
var myAge = 30;
```

Nem hoztad létre a második változót, ellenben fals adattal felülírtad az elsőt. Innentől a programodban van egy hiba (bug), amire csak később, hosszas kutatás után fogsz rájönni, amikor észreveszed, hogy valami nem stimmel az `age` változót használó számítások eredményével.

Megjegyzés: A példa abszurdnak tűnhet („Miért nevezném el ugyanúgy a saját koromat tároló változót, mint a vörösbor korát tartalmazót?”), de csapatunka és nagy kódbazisok esetén könnyebben megtörténhet, mint gondolnád. Amikor egy sok ezer soros programon dolgozol, nem fogod sorról sorra végigolvasni az egész kódot. Így van esélye annak, hogy te egy adott kontextusban ugyanazt a változónevet tartod logikusnak, amelyet egy társad egy másik kontextusban már felhasznált. Ahogy az is simán előfordulhat, hogy egyszerűen elfelejted, pontosan mit kódoltál hetekkel-hónapokkal ezelőtt, pár száz sorral arrébb.

Pontosan ennek az esetleges hibának a kiküszöbölésére jött létre a `let` kulcsszó. **A `let` azt még megengedi, hogy megváltoztasd egy meglévő változó értékét, de azt már nem, hogy teljesen újra deklaráld magát a változót.**

Nézzük meg, hogyan! Frissítsd a böngészőoldalt, hogy kitöröld a memóriából az eddigieket, majd másold be a következő sorokat, ügyelve arra, hogy mindegyik után üss Enter!

```
let age = 25;  
let age = 30;
```

Ha minden igaz, a következő hibaüzenetet kaptad, amikor megpróbáltad újra dekláralni az `age` változót a második sorral:

```
Uncaught SyntaxError: Identifier 'age' has already been declared
```

A böngésző szolt, hogy „Hé, már van egy ilyen nevű változód, ezt nem tudod még egyszer létrehozni!”. Ez az egyszerű hibaüzenet sok későbbi fejtöréstől kímélhet meg, ezért innentől

fogva érdemes inkább a `let` kulcsszót használnod a `var` helyett.

Megjegyzés: A tananyagainkban még sok helyen fogsz `var` kulcsszóval találkozni. Ez nem hiba, egyszerűen az az oka, hogy amikor azokat a leckéket írtuk, az ES6 – és így a `let` – még nem volt széleskörűen támogatott. Ezeken a pontokon a döntést rád bízzuk: ha szeretnéd, használj `let`-et `var` helyett, de maradhatsz is az eredeti kulcsszónál – a kódod ettől nem lesz helytelen.

Tipp: Ugyanitt szeretnénk megjegyezni, hogy a `var`-t még ne törlök ki a fejedből azzal a felkiáltással, hogy „szuper, ez elavult, akkor el is felejthetem”. Az átállás az új kulcsszavakra nem egyik napról a másikra fog megtörténni, így a minden napjaid során még sokáig fogsz találkozni a `var`-ral.

Ellenőrző kérdés

Válaszd ki a szintaktikailag helyes deklarációt!

- `let var = 'swimming';`
 - Sajnos nem jó, reserved word (`var`) ugyanis nem lehet a változó neve. Próbáld újra, keresd meg a helyes megoldást!
- `let = sport 'swimming';`
 - Sajnos nem jó a kulcsszavak, operátorok és adatok sorrendje. Próbáld újra, keresd meg a helyes megoldást!
- `let sport = 'swimming';`
 - Gratulálunk, helyes válasz! Már tudod, hogyan deklarálj változókat a JavaScript legújabb verziójában!
- `let = 'sport swimming';`
 - Hmm, valami nem stimmel a sorrenddel. Próbáld újra, keresd meg a helyes megoldást!

4.3. Változó deklarálása a `const` kulcsszóval

Az, hogy a `var` és `let` kulcsszavakkal létrehozott változók módosíthatóak (azaz megadhatsz nekik új értéket), időnként problémához tud vezetni.

Ennek kiküszöbölésére hozták létre a `const` kulcsszót, amely – ahogy a neve is mutatja – konstans, azaz állandó változókat hoz létre. Magyarul **ez a kulcsszó nem engedi, hogy bármikor is módosítsuk a vele létrehozott változó értékét**. Ami egyszer létrejött, az úgy marad – nem lehet sem szándékosan, sem véletlenül felülírni.

Próbáld ki ezt is! Nyisd meg a böngésződ konzolját, és másold be a következő sort:

```
const PI = 3.14159265358979323846;
```

Ezzel létrehoztál egy π (pi) konstanst. Próbáld meg újradeklárálni a konzolban:

```
const PI = 2;
```

Hasonló hibát dob vissza a gép, mint a `let` kulcsszónál. Eddig a `const` ugyanúgy viselkedik, mint a `let`: szól, ha véletlenül újra deklarálnánk a változót. De mi a helyzet, ha egyszerűen csak módosítani akarjuk, mint bármilyen más változót?

Másold be ezt a konzolba, és nyomj Enter-t:

```
PI = 2;
```

A konzol megint hibát dobott, ezúttal ezt: `Uncaught TypeError: Assignment to constant variable..` Szólt, hogy nem tudja végrehajtani a kért feladatot, mert épp egy `const` változó értékét szeretted volna módosítani, amit nem lehet.

A konstans tehát nem adta meg magát, ahogy az egy jól nevelt állandótól elvárható.

Tipp: `const`-tal deklarált változókat jóval ritkábban használunk, mint a másik két típust. Akkor érdemes ilyet bevezetni, ha biztos vagy benne, hogy egy érték semmilyen körülmények között nem fog változni, vagy nem akarod, hogy változzon, ezért szeretnéd megakadályozni az adat módosítását.

Megjegyzés: A `const` kulcsszóval deklarált változók nevét csupa nagybetűvel szokás írni, hogy ezzel is emlékeztess magad arra, itt egy konstanssal van dolgod. Ha több szóból áll a változó neve, a szavak között alulvonást használunk, például: `BIG_CATS`.

Ellenőrző kérdés

Válaszd ki a szintaktikailag helyes deklarációt!

- `const GOLDEN_RATIO = 1.61803398874989484820;`
 - Gratulálunk, helyes válasz! Már tudod, hogyan deklarálj állandó változókat a JavaScript legújabb verziójában!
- `const goldenRatio = 1.61803398874989484820;`
 - Hmm, valami nem stimmel a változó nevével. Próbáld újra, keresd meg a helyes megoldást!
- `const = GOLDEN_RATIO 1.61803398874989484820;`
 - Hmm, valami nem stimmel a sorrenddel. Próbáld újra, keresd meg a helyes megoldást!
- `const GOLDEN_RATIO = 1.61803398874989484820`
 - Hmm, valami nem stimmel az utasítás (*statement*) legvégével. Próbáld újra, keresd meg a helyes megoldást!

5. Összefoglalás

5.1. Mit tanultál eddig?

Gratulálunk! Ha idáig eljutottál, az azt jelenti, hogy:

- Tudod, mik azok a változók, és hogy miért érdemes azokat alkalmazni.
- Tudod, mik azok a JavaScript-kulcsszavak (*keywords*) és kifejezések (*expressions*).
- Helyesen el tudsz nevezni egy változót, hiszen ismered az elnevezési konvenciókat, így például a camelCase írásmódot.
- Tudsz érvényes változókat deklarálni a `var`, a `let` és a `const` kulcsszavakkal.
- Tudsz üres változókat deklarálni, és tudod, hogy ez mire jó.
- Tudod módosítani egy változó értékét.
- Tudod, hogyan tárolj el kifejezést egy változóban, és hogy változó is szerepelhet kifejezésben, sőt akár állhat a kifejezés csak változókból is.

Szép munka!

Most már készen állsz arra, hogy továbblépj, és megismerkedj a függvényekkel. Találkozunk a következő modulban!

Megjegyzés: Ha a fenti összefoglaló olvasása közben elbizonytalanodtál valahol, kérlek, ugorj vissza az adott fejezetre, és még egyszer olvasd át jó alaposan, mielőtt továbbhaladnál.

Tipp: Ha szeretnéd leellenőrizni a tudásod, mielőtt továbbmész, töltsd ki a következő oldalon található tesztet.

6. fejezet: Teszt

6.1. lecke: Ellenőrizzd a tudásod!

[10 kérdés, körülbelül 10 perc, kérdésenként 1 pont.](#)

Tipp: Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a "Küldés" (Submit) gombot az teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

Megjegyzés: A tesztet többször is kitöltheted.

Sok sikert!

JavaScript alapok III. (függvények)

1. Bevezetés

1.1. Mik azok a függvények?

Hogy könnyedén megértsd a JavaScript-függvényeket (angolul *functions*), gyere velünk egy apró gondolatkísérletre:

Képzeld el, hogy egy menő séf vagy, aki egy sikeres belvárosi étteremben dolgozik.

Szeretsz saját ételeket kitalálni, új összetevőkkel és módszerekkel kísérletezni. A különleges toszkán paradicsomlevesed olyannyira jól sikerült, hogy a konyhafőnök felrakta az étlapra, és a város hetek óta a csodájára jár.

Valahányszor megrendelik ezt az űrülten finom levest, felsóhajtász, összegyűjtöd a kezed alatt dolgozó szakácsokat, és elsortolod nekik a főzés menetét:

1. Pirítsd meg olívaolajon a hagymát és fokhagymát.
2. Tedd hozzá a felkockázott paradicsomot, sózd és borsozd meg, majd öntsд fel 2 dl vízzel.
3. Főzd kis lángon 15-20 percig.
4. Ha szétfőtt a paradicsom, add hozzá a száraz kenyeret, olívaolajat, bazsalikomot, és adj hozzá egy kis vizet.
5. Főzd még 5 percig.
6. Szörj rá részelt parmezánt.
7. Szolgáld fel ciabattával.

Valahol a 823. alkalom után elunod ezt az időrabló folyamatot, és úgy döntesz, ideje végre leírnod a toszkán paradicsomlevesed receptjét. Így legközelebb, amikor egy vendég megrendeli ezt a fogást, a szakácsaidnak nem kell téged zargatni. Elég előkapniuk a receptet, és önállóan képesek megfőzni a levest.

Ha ezt érted, akkor igazából érted a JavaScript-függvényeket is – **a függvények ugyanis pontosan ugyanazon az elven működnek, mint a receptek.**

Amikor egy programnak több ponton is végre kell hajtania ugyanazt a részfeladatot, nem írjuk meg a vonatkozó kódot újra és újra, hanem írunk egy függvényt.

Benne csoportosítjuk a részfeladatot megoldó JavaScript-utasításokat. Amikor legközelebb szükségünk van a kódrészletünkre, elég hivatkoznunk az azt tároló függvényre. A böngészőnk szépen kikeresi, és végrehajtja a függvényben leírt utasításokat, pontosan úgy, ahogy a szakácsok is végrehajtják a receptben leírt lépéseket.



Ellenőrző kérdés

A fentiek alapján mit gondolsz, mi a JavaScript-függvény definíciója?

- A függvény egy olyan JavaScript-kifejezés, amely végrehajt egy feladatot.
 - Hoppá, ez egy trükkös kérdés volt. A függvények nem kifejezések. Próbáld újra!
- A függvény utasítások sorozata, melyeket követve meg tudunk írni egy programot.
 - Sajnos nem ez a helyes válasz. Próbáld újra!
- **A függvény JavaScript-utasításokat (statementeket) fog össze, amelyek egy adott feladatot hajtanak végre, pl. kiszámolnak egy értéket. Ha egyszer létrehoztuk, később bárhol hivatkozhatunk rá a kódban.**
 - Valóban így van, jól figyeltél a lecke közben. Most pedig nézzük meg, hogyan épül fel egy függvény!
- A JavaScript-függvényekkel különböző matematikai képleteket tudunk kiszámolni.
 - Hmm, nem egészen. Ennél többre is képes. Próbáld újra!

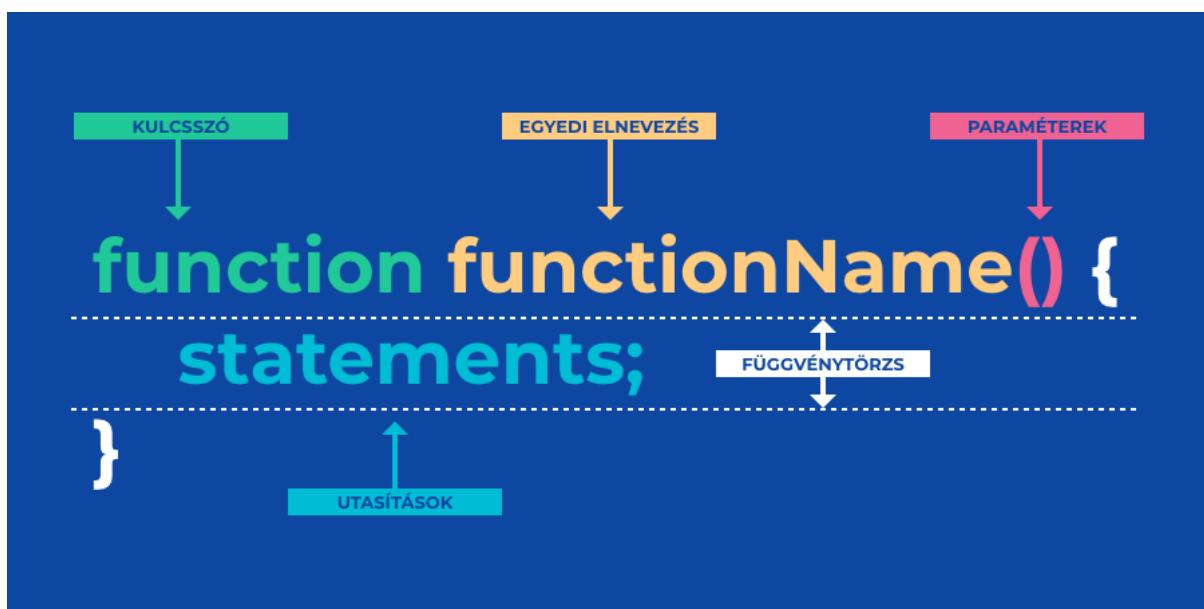
2. Függvények létrehozása és használata

2.1. Függvények deklarálása

Ahogy a receptírásnak is vannak bizonyos szabályai (pl. kigyűjtjük előre a hozzávalókat, az elkészítési lépéseket megszámazzuk, stb.), úgy a JavaScript-függvények megalkotásának is van specifikus módja. Most ezt fogjuk megtanulni.

Megjegyzés: A függvények létrehozásának több módja is van, most a legegyszerűbbel fogjuk kezdeni. Ezt a módszert szakkifejezéssel **deklarálásnak** hívjuk. Példamondat: „Juli deklarált egy új függvényt, hogy kiszámolja a négyzet területét.” Később kitérünk a többi lehetőségre is, ahogy relevánsá válnak számodra.

Egy függvénydeklaráció (angolul *function declaration*) így néz ki általános formában:



Nézzük meg együtt részletesen:

1. A sort a **function** **kulcsszóval** nyitjuk. Ezzel jelezzük, hogy egy függvény fog következni.
2. A kulcsszót a **függvény egyedi elnevezése** (**functionName**) követi. Ezek az elnevezések pontosan úgy működnek, mint a változónevek, amelyekről már hallottál.
3. A függvény egyedi nevét egy sima zárójel (**()**) követi (szóköz nélkül), amelybe úgynevezett **paraméterek** kerülhetnek. Hogy ezek mik, és mire használjuk őket, arra később visszatérünk.
4. A sor végén egy nyitó kapcsos zárójellel (**{**) kezdődik maga a **függvénytörzs** (*function body*), ahova azok az **utasítások** (*statements*) fognak kerülni, amelyeket szeretnénk a függvényvel végrehajtani. (A zárójelek és a kapcsos zárójel közé szóközt illik rakni.)
5. A függvény törzsében minden valid JavaScript-kódot írunk, így például az utasításokat pontosvesszővel zárjuk.

6. Az utolsó utasítást követően bezárjuk a függvény törzsét egy záró kapcsos (}) zárójellel. Ezt nem követi pontosvessző!

Megjegyzés: Ha megfigyeled, van egy bizonyos hasonlóság a változók és a függvények deklarálása között. Ahogy a változók deklarálásával egy értéket (vagy kifejezést) kapcsolunk egy változónévhez, úgy egy adott utasítássort kötünk a függvény nevéhez.

Ellenőrző kérdés

Válaszd ki az alábbi lehetőségek közül a helyesen deklarált függvényt!

- `someFunction = function() {}`
- `function someFunction{} () ;`
- `function someFunction() {}`
- `function = someFunction() {} ;`

2.2. Egy függvénydeklaráció

Nézzük meg, mi történik, ha az előző lecke alapján megpróbáljuk a toszkán paradicsomleves receptjét egy `tomatoSoupRecipe` függvénnnyé alakítani.

Nyiss meg egy új bint a JS Binen, és a JavaScript-panelre először írd le az üres függvénydeklarációt:

```
function tomatoSoupRecipe() {  
}
```

Aztán másold bele a receptet, hogy így nézzen ki:

```
function tomatoSoupRecipe() {  
    Pirítsd meg olívaolajon a hagymát és fokhagymát.  
    Tedd hozzá a felkockázott paradicsomot, sózd és borsozd meg,  
    majd öntsд fel 2 dl vízzel.  
    Főzd kis láncon 15-20 percig.  
    Ha szétfőtt a paradicsom, add hozzá a száraz kenyeret,  
    olívaolajat, bazsalikomot, és adj hozzá egy kis vizet.  
    Főzd még 5 percig.  
    Szórj rá részelt parmezánt.  
    Szolgáld fel ciabattával.  
}
```

Megjegyzés: Figyeld meg, hogy a függvénytörzs négy szóközzel be van húzva (indentálva van) magához a függvényhez képest. Ez egy formázási szokás, amely javítja a kódod olvashatóságát. Figyelj erre a jövőben.

Szuper, nyomd meg a Run gombot.

Egy hibaüzenetet kaptál, igaz? Persze, hiszen a recept lépései nem valid JavaScript-utasításokként fogalmaztuk meg. Javítsuk ezt meg!

Feladat

- Nevezd át a függvényt `printTomatoSoupRecipe`-re (magyarul `nyomtasdKiAParadicsomlevesReceptjet`).
- Oldd meg, hogy a böngésző minden egyes főzési lépést kinyomtasson a konzolba. Már ismered a JavaScript ehhez szükséges beépített metódusát.

Megjegyzés: A következő leckében megmutatjuk a megoldást, de kérlek, először próbálkozz egyedül. minden tudással rendelkezel a feladathoz.

Ha kész vagy, próbáld meg újra megnyomni a Run gombot. **Ha minden jól csináltál, akkor még mindig nem történik semmi**, viszont a konzolban már nem jelenik meg hibaüzenet.

Gratulálunk, sikerült deklarálnod életed első függvényét! A következő leckében megnézzük, hogy mire van szükséged ahhoz, hogy a függvényed végre is hajtsa a benne tárolt utasításokat.

2.3. Függvények meghívása

Ha minden jól csináltál az előző leckében, akkor a JS Bined JavaScript-paneljén a következő kód áll:

```
function printTomatoSoupRecipe() {  
    console.log('Pirítsd meg olívaolajon a hagymát és  
fokhagymát.');//  
    console.log('Tedd hozzá a felkockázott paradicsomot, sózd és  
borsozd meg, majd önts fel 2 dl vízzel.');//  
    console.log('Főzd kis lágon 15-20 percig.');//  
    console.log('Ha szétfőtt a paradicsom, add hozzá a száraz  
kenyeret, olívaolajat, bazsalikomot, és adj hozzá egy kis  
vizet.');//  
    console.log('Főzd még 5 percig.');//  
    console.log('Szórj rá reszelt parmezánt.');//  
    console.log('Szolgáld fel ciabattával.');//  
}
```

Ez egy helyesen deklarált függvény. Ha megnyomod a Run gombot, mégsem történik semmi. Miért?

Emlékezz: a függvények olyanok, mint a receptek. Ahogy a recept sem egyenértékű az étellel, amelyet elkészíthsz belőle, úgy a függvény is csak a futtatandó kód „tervrajza”.

A függvény deklarálásakor csupán annak létezését írjuk le – ez azonban még nem hajta

végre a függvénytörzsben (function body) tárolt kódot. Az abban szereplő utasítások csak akkor lesznek végrehajtva, **ha meg is hívod a függvényt**.

Megjegyzés: Angolul a függvény meghívása a következő: „*calling/invoking a function*”. A saját netes kutatómunkád során gyakran találkozhatsz majd ezzel a kifejezéssel.

Nézzük meg a gyakorlatban, hogyan tudsz meghívni egy függvényt!

Nyisd meg a bint, amelyben eddig dolgoztál, vagy nyiss egy újat, és másold be a JavaScript-panelre a kódblokkot a lecke tetejéről. Emlékeztetőül: ez eddig a **függvénydeklaráció**.

Ahhoz, hogy meghívunk egy függvényt:

- ki kell írnunk a függvény nevét,
- amelyet egy zárójel követ (szóköz nélkül).
- A sort – szokás szerint – pontosvessző (;) zárja.

Másold be az alábbi sort a függvénydeklarációd alá, és nyomd meg a Run gombot.

```
printTomatoSoupRecipe() ;
```

Siker! A toszkán paradicsomleves receptje megjelent a konzolban. Szép munka!

Megjegyzés: A függvény annyiszor fut le, ahányszor meghívod. Próbáld ki, hogy még kétszer bemásolod a `printTomatoSoupRecipe()`; függvényhívást (angolul *function call*) a bined aljára. Nyomd meg a Run gombot. Mi történik? A konzol háromszor kinyomtatta a receptet. Ez az, amiről a modul legelején beszélünk: a függvények lehetővé teszik, hogy egyszer írj meg egy adott kódrészletet, és utána annyiszor használd újra, ahányszor csak akarod.

Nyiss egy új bint, és írj meg egy függvényt, amely hasonló módon kinyomtatja a konzolba az egyik kedvenc ételed receptjét.

2.4. Egy csepp kitekintés

Mielőtt mélyebbre merülnénk a függvények világában, ez egy kiváló alkalom, hogy egy pillanatra megálljunk, és tisztázzunk két kérdést, amelyek a későbbiekben sokat fognak segíteni:

1. Pontosan mi olvassa el és értelmezi a JavaScriptet a böngészőben?
2. Milyen sorrendben olvassa el a gép a JavaScriptet, mi az a *control flow* (magyarul vezérlésáram)?

2.5. Mi olvassa el a JavaScriptet a böngészőben?

Az eddigi tanulmányaid során nagyvonalúan csak annyit mondtunk, hogy a „számítógéped” vagy esetleg a „böngésződ” olvassa el és értelmezi a JavaScriptet. A valóság ennél egy

leheletnyivel bonyolultabb.

A böngésződ valójában egy komplex szoftver, amely több, részfeladatokat ellátó egységből áll. A JavaScript interpretálásáért és futtatásáért is egy ilyen egység felel, amelyet **JavaScript-motornak** (angolul *JavaScript engine*) nevezünk.

Megjegyzés: A különböző böngészők különböző JavaScript-motorokat használnak, amelyeknek jópofa fantázianeveik vannak. A Chrome motorját például V8-nak, a Firefoxét SpiderMonkey-nak, míg az Edge-et és az Operát hajtó motort Chakrának keresztelték el.

Amikor azt mondjuk, hogy a „böngésződ” elolvassa és végrehajtja a JavaScriptet, akkor a valóságban azt értük alatta, hogy a böngésző JavaScript-motorja betölti, interpretálja és végrehajtja a kódodat.

Most, hogy már tudod, hogy mi az a JavaScript-motor, a jövőben képesek leszünk egy picit pontosabban beszélni arról, hogy mi történik az egyes kódok futása közben.

Ellenőrző kérdés

Mi az a JavaScript-motor?

- **A böngésző egyik specializált modulja, amely a JavaScript futtatásáért felel.**
- A böngésző egy része, amely a HTML, CSS és JavaScript futtatásáért felel.
- Egy a JavaScript cég által szponzorált Forma–1-es csapat.
- A böngésző azon része, amely JavaScript és különböző egyéb nyelvek (Java, Python, Ruby, PHP stb.) között fordít a weben.

2.6. Milyen sorrendben hajtódik végre a JavaScript-kód?

Ismerkedj meg egy új szakszavunkkal: control flow.

A control flow az a sorrend, amelyben a JavaScript-motor beolvassa az utasításaidat.

A legegyszerűbb esetben a control flow felülről lefelé halad, és sorról sorra hajtja végig a JavaScript statementeket.

Ez egészen addig igaz, amíg a motor nem fut bele valamibe, ami megváltoztatja a control flow-t, például egy függvénybe. Ilyenkor kiderül, hogy a JavaScript-motor rendelkezik egy szuperképességgel: **tud teleportálni**.

Vegyük az alábbi kódot:

```
var someVariable = 123;
var someExpression = 2 + 2;

function someFunction() {
    console.log('1. utasítás;');
    console.log('2. utasítás;');
```

```
    console.log('3. utasítás;');
}

someFunction();

console.log('Ez egy string. ');

someFunction();
```

Amikor lefuttatod a fenti programot, a control flow (az utasítások végrehajtásának sorrendje) a következőképp alakul:

1. A JavaScript-motor elindul legfelülről, és létrehozza a `someVariable` változót az **1. sorban**.
2. A motor lép egyet lefelé, létrehozza a `someExpression` változót, kiértékeli a `2 + 2` összeadást, és elrakja az eredményét, azaz a `4`-et a változóba.
3. A **4. sorban** a motor észleli a `someFunction` függvényt, de egyelőre nem csinál semmit, mert nem hívtuk meg a függvényt. Eddig a control flow az alapbeállítást követi: megy felülről lefelé.
4. A control flow első változása a **10. sorban** következik, ahol meghívjuk a `someFunction()` függvényt. A JavaScript-motor visszateleportál a **4. sorba**, ahol a függvénydeklaráció található. Végiglépdel a függvénytörzsön, és kilogolja a három stringet. Amikor a **8. sorban** eléri a függvény végét, visszateleportál a **10. sorba**, és folytatja az útját az alapértelmezett control flow szerint.
5. A **12. sorban** talál egy egyszerű `console.log()`-ot, ezt végrehajtja.
6. A **14. sorban** újabb függvényhívás (function call) szerepel, úgyhogy a motor ismét teleportál. A **4. sorban** megtalálja a `someFunction` függvényt, végigmegy a törzsön, végrehajtja, a függvény végén pedig visszateleportál a kiindulási helyére, azaz a **14. sorba**.
7. A motor észleli, hogy véget ért a program, és leáll.

Ahogy látod, **a függvények speciálisak abból a szempontból, hogy megtörök az alapértelmezett control flow-t.**

Megjegyzés: A későbbi tanulmányaid során még sok egyéb struktúrával is találkozni fogsz, amelyek mindenkorban a maguk különleges módján változtatják meg a control flow-t. minden ilyen esetben részletesen meg fogjuk beszélni, hogy mi történik és miért.

Tipp: A control flow-ról és annak viselkedéséről azért fontos tudnod, mert ez tesz képessé arra, hogy elolvass és értelmezz programokat. Amikor egy már megírt kódot próbálsz átlátni, mindig tedd fel magadnak a következő kérdést: „hogyan halad a control flow?”. Így képes leszel átlátni a program struktúráját, amely nem mutatná meg magát, ha egyszerűen csak megpróbáld fentről lefelé elolvasnai a kódot.

Oké, ennyi volt a kitekintésünk a JavaScript-motorokra és a control flow-ra. M, még egy gyors feladat, aztán visszatérünk a függvényekhez!

Ellenőrző kérdés

Hogyan alakul a control flow a következő JavaScript-programban? Milyen sorrendben megy végig a sorokon a JavaScript-motor?

```
1 var answer = 'This is an answer.';  
2  
3 function someFunction() {  
4     console.log('and');  
5 }  
6  
7 console.log(12 + 2);  
8  
9 someFunction();  
10  
11 console.log(42);  
12  
13 someFunction();  
14  
15 console.log(14 * 42);
```

- 1. → 3. → 7. → 9. → 11. → 13. → 15.
- 1. → 3. → 7. → 9. → 3. → 4. → 5. → 9. → 11. → 13. → 3. → 4. → 5. → 13. → 15.
- 1. → 3. → 4. → 5. → 7. → 9. → 3. → 4. → 5. → 9. → 11. → 13. → 3. → 4. → 5. → 13. → 15.
- 1. → 3. → 4. → 5. → 7. → 9. → 11. → 13. → 15.

3. Rugalmas függvények paraméterekkel

3.1. Variációk egy téma

Tegyük fel, hogy a fiktív éttermedben arra jutottál séfként, hogy a toszkán paradicsomlevesed nemcsak parmezánnal és ciabattával, hanem trappistával és zsömlével, sőt edamival és pogácsával is kiváló.

Szeretnél új recepteket írni ezekhez a levesvariációhoz. Séfként az egyetlen lehetőséged az lenne, hogy kétszer lemásolod a 90%-ban ugyanolyan levesreceptet, majd megváltoztatod a két-két sort bennük. Így végül három, picit különböző recepted lenne. Programozóként szerencsére van egy ennél sokkal jobb eszközöd: **a függvényparaméterek.**

Emlékeztetőül, így néz ki egy függvénydeklaráció általános formában:

```
function functionName() {
```

{}

Ha emlékszel, azt mondtuk, hogy az (egyelőre) üres zárójel a hely, ahová a paramétereket írhatod majd.

A paraméterekkel olyan adatokat tudsz definiálni, amelyek szeretnéd, ha dinamikusak, azaz változtathatóak lennének. Például a leves esetében szeretnénk, ha a sajt és a feltét egy-egy paraméter lenne. Így ugyanaz a függvény képes lenne kinyomtatni az összes receptet, csak meg kell mondanunk, hogy milyen sajttal és feltéttel szeretnénk a fogást.

Lássuk, hogyan néz ki ez a gyakorlatban!

Ellenőrző kérdés

Hova írhatjuk a paramétereket a függvénydeklarációban (function declarationben)?

- A függvény nevének helyére.
- A kapcsos zárójelek közé.
- **A zárójelbe, amely a függvény neve után következik.**
- A zárójelbe, amelyet be kell szúrnunk a függvény törzsébe.

3.2. Paraméterek használata a függvényekben

Mindenekelőtt nyisd meg a bint, amelyben a `printTomatoSoupRecipe` függvényeden dolgoztál. Ha nincs meg, akkor nyiss egy új bint, és másold bele a függvény kiinduló állapotát:

```
function printTomatoSoupRecipe() {  
    console.log('Pirítsd meg olívaolajon a hagymát és  
fokhagymát.');//  
    console.log('Tedd hozzá a felkockázott paradicsomot, sózd és  
borsozd meg, majd önts fel 2 dl vízzel.');//  
    console.log('Főzd kis lágon 15-20 percig.');//  
    console.log('Ha szétfőtt a paradicsom, add hozzá a száraz  
kenyeret, olívaolajat, bazsalikomot, és adj hozzá egy kis  
vizet.');//  
    console.log('Főzd még 5 percig.');//  
    console.log('Szórj rá részelt parmezánt.');//  
    console.log('Szolgáld fel ciabattával.');//  
}
```

Azt szeretnénk, ha a „Szórj rá részelt parmezánt.” és „Szolgáld fel ciabattával.” stringekben nem a konkrét *parmezán* és *ciabatta* szavak állnának, hanem ezeket a függvény képes lenne dinamikusan kitölteni az aktuális kérésünkkel.

Ha ez kezd úgy hangzani, mintha változókra lenne szükségünk, akkor jó helyen kapisgálsz.

A paraméterek gyakorlatilag úgy viselkednek a függvényen belül, mint a változók. Ennek fényében definiálunk – a már ismert elnevezési szabályok és konvenciók alapján – két paramétert a zárójelbe:

```
function printTomatoSoupRecipe(cheeseType, sideDish) {  
    ...  
}
```

Megjegyzés: A fenti kódblokkban a három pont (...) arra utal, hogy ott továbbra is ott van a kód, csak most szemléltetési szempontból elrejtettük, hogy a lényegi változásokra tudj koncentrálni. Most, hogy egyre nagyobb kódblokkokkal dolgozunk, ezzel a technikával időnként találkozni fogsz a leckék szövegében. Ilyenkor a saját kódodban ne cserél le a sorokat három pontra.

Már csak egy lépéstre vagyunk a győzelemről. Ugyanúgy, ahogy változókat helyettesítenénk be a stringekbe, helyettesítsük be az új paramétereinket a két utolsó `console.log()` utasításba:

```
function printTomatoSoupRecipe(cheeseType, sideDish) {  
    ...  
    console.log('Szörj rá reszelt ' + cheeseType + '-t.');//  
    console.log('Szolgáld fel ' + sideDish + '-val/vel.');//  
}
```

Megjegyzés: Figyeld meg, hogy a `console.log()` utasítások így most már egy-egy JavaScript-kifejezést fogadnak be, amelyek egyenként két-két stringből és egy-egy paraméterből állnak. Amikor a kód lefut és kiértékelődik, ezek a kifejezések kész stringekké fognak változni.

Ha megfigyeled, a paraméterek alapvető logikája teljesen megegyezik a változókéval. Először definiálod őket a zárójelen belül, hogy később aztán használhasd őket valahol a függvénytörzsben.

Hív meg a frissített függvényed! A deklaráció alá írd be a következő sort, majd nyomd meg a Run gombot:

```
printTomatoSoupRecipe();
```

Mi történt? A függvény szépen lefutott, a konzolban az első öt főzési lépés változatlanul megjelent, az utolsó kettő pedig így néz ki:

```
"Szörj rá reszelt undefined-t."  
"Szolgáld fel undefined-val/vel."
```

Szuper! A paramétereink szépen működtek, csak egyetlen dolog hibádzik: nem adtunk meg nekik konkrét értéket (*value*). Ezért – ahogy azt az üres változók is csinálják – **undefined** értéket vettek fel.

Ellenőrző kérdés

Az alábbi állítások közül melyik nem igaz a függvényparaméterekre?

- A paramétereket először definiáljuk a függvénydeklarációban lévő zárójelben, majd – a változókhöz hasonlóan – akárhhol felhasználhatjuk őket a függvénytörzsben, csak be kell helyettesítenünk a nevüket.
- A paraméterekre ugyanazok az elnevezési konvenciók és szabályok vonatkoznak, mint a változókra.
- Ha a paraméterekeknek nem adunk meg értéket, a helyükön **undefined** érték jelenik meg a kiértékelésük után.
- **Egy függvénynek csak egy paramétere lehet.**

3.3. Argumentum: a paraméter értéke

Jogos a benned felmerülő kérdés:

*„Ha nem **undefined** sajtot és **undefined** feltétel szeretnél enni a toszkán paradicsomlevesemhez, akkor hogyan adhatok értéket a paramétereimnek?”*

Talán nem fogsz meglepődni a válaszon: **a paraméterek értékeit a függvény meghívásakor tudod megadni a zárójelben.**

```
// Ez egy függvényhívás (function call):  
someFunction(/*Ide írhatod a paraméterek értékeit.*));
```

Megjegyzés: A függvényparaméterek értékeinek speciális neve van a JavaScriptben: **argumentum** (angolul *argument*). Amikor azt hallod egy fejlesztőtől, hogy „átadni egy argumentumot a függvénynek” (angolul „*passing an argument to a function*”), akkor arra gondol, hogy megadod a paraméter konkrét értékét a függvényhívásban.

Lássuk, hogyan működik ez a **printTomatoSoupRecipe** függvényünk esetében. Keresd meg a függvényhívást (function call) a függvénydeklaráció alatt, és írd át, hogy így nézzen ki:

```
function printTomatoSoupRecipe(cheeseType, sideDish) {  
    ...  
}  
printTomatoSoupRecipe('edami', 'pogácsa');
```

Nyomd meg a Run gombot. Ha minden jól csináltál, a konzolüzenet utolsó két sora így néz ki:

```
"Szörj rá reszelt edami-t."  
"Szolgáld fel pogácsa-val/vel."
```

Szép munka, megadtad életed első argumentumait!

Ellenőrző kérdés

Mik azok az argumentumok? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- **Az argumentum a konkrét érték, amelyet megadunk a függvény paraméterének a függvényhívás (function call) során.**
- Az argumentum a paraméter szinonimája. A két kifejezés ugyanazt jelenti.
 - Nem, sajnos ez egy tipikus tévedés az argumentumokkal kapcsolatban. Az argumentum a paraméter konkrét értékét jelöli, amelyet a függvényhívás során megadunk.
- Az argumentum (argument) az érték (value) szó szinonimája a JavaScriptben.
 - Sajnos nem. Igen, a paraméterek értékeit argumentumnak hívjuk, de a változók értékeit vagy az értékeket általánosságban nem. Az argumentum szűken, csak és kizárolag a paraméterek értékeire vonatkozik.
- Az argumentumok a függvények kvázi változói, amelyeket a függvénydeklaráció során adunk meg.
 - Sajnos nem. Összekeverte a paraméterrel. A paramétereket adjuk meg a deklaráció során, az argumentumok pedig a paraméterek konkrét értékei, amelyeket a függvényhívás során adunk meg.

3.4. Hárrom megjegyzés a paraméterekről és az argumentumokról

Mielőtt továbblépnénk a paraméterek témájáról, szeretnénk tenni három rövid megjegyzést, amelyek segítenek elmélyíteni a frissen megszerzett tudásod.

1. Egy függvény, akárhány paraméter

Egy adott függvénynek akármennyi paramétert megadhatsz. A határt csak az szabja meg, hogy hányszámú dinamikus adatra van szükséged a függvénytörzsben (function body).

Megjegyzés: A paramétereket (és meghívásnál az argumentumokat) minden esetben visszatérítővel (`,`) választjuk el.

Az érme másik oldala, hogy egy függvénynek nem feltétlenül kell hogy legyen paramétere – ahogy azt láthattad is a `printTomatoSoupRecipe` első iterációjában. Ettől függetlenül a zárójelnek minden esetben ott kell lennie a függvénydeklarációban.

2. A meghívásnál számít a sorrend

Amikor meghívsz egy függvényt, a JavaScript a sorrend alapján párosítja az értékeket a paraméterekkel.

Az első argumentumot fogja az első paraméterbe menteni, a másodikat a másodikba, és így tovább. Figyelj arra, hogy mi a paraméterek sorrendje, amikor meghívod a függvényt,

nehogy rossz helyre passzold be az adatod.

3. Mint a változóknál

A fejezetben eddig csak string értékeket adtunk meg argumentumként, de természetesen az összes többi adattípush is képes befogadni egy függvény. Például:

```
function someFunction(parameter) {  
    console.log(parameter);  
}  
  
someFunction('Ez egy string');  
someFunction(42);  
someFunction(true);  
someFunction(2 + 2);
```

Ökölszabályként, amit megadhatsz egy változó értékének, azt megadhatod egy paraméter argumentumaként is.

Ellenőrző kérdés

Az alábbi, függvényekkel kapcsolatos állítások közül melyik igaz? Jelöld meg a helyes megoldást!

- A függvényeknek akármilyen adattípusú argumentumot megadhatunk, kivéve booleant és objektumokat.
- **Egy függvénynek 0 és végtelen között akárhány paramétere lehet; a függvényhívásnál figyelnünk kell arra, hogy az argumentumokat a megfelelő sorrendben adjuk meg; és a paraméterek bármilyen adattípush elfogadnak argumentumként.**
- Egy függvénynek 1, 2 vagy 3 paramétere lehet; a függvényhívásnál az argumentumok sorrendje nem számít; és a paraméterek bármilyen adattípush elfogadnak argumentumként.
- A függvények csak 1, 2 vagy 3 paramétert fogadhatnak be, és az egyes paraméterek egymástól eltérő adattípusokat tudnak csak kezelnı.

3.5. Gyakorlás: szendvicsreceptfüggvény

Az eddigiekben elsajátítottad a függvények alapjait:

- ismered, hogyan kell helyesen deklarálni és
- meghívni őket, valamint
- azt is tudod, hogyan teheted őket rugalmassá paraméterekkel.

Ideje egy picit komolyabb gyakorlatnak, ahol elmélyítheted az új tudásod. Építs egy, a `printTomatoSoupRecipe`-hez hasonló függvényt, ám ezúttal szendvicsek készítéséhez nyomtasson ki receptet.

A programod a következő kritériumoknak feleljön meg:

- Egy darab függvényből álljon.
- A függvény neve legyen `printSandwichRecipe`.
- A függvény nyomtassa ki a konzolba a következő utasításlistát:
 1. Fogj két szelet kenyeret.
 2. Az egyik szeletre nyomj `undefined`-t.
 3. Helyezz a kenyérre két szelet `undefined`-t.
 4. Rakj rá egy szelet `undefined` sajtot.
 5. Koronázd meg pár szelet/karika/csipet ilyen zöldséggel: `undefined`.
 6. Borítsd be a szendvicset a másik szelet kenyérrel.
 7. Jó étvágyat!
- Az `undefined` értékek helyére a függvény helyettesítse be a megfelelő hozzávalókat.
 - A 2. sorba a szószt vagy öntetet.
 - A 3. sorba a húsfélét.
 - A 4. sorba a sajt típusát.
 - Az 5. sorba a zöldséget.
- Az adat az `undefined` értékek helyéreérkezzen a függvény paramétereiből.
- A paraméterek feleljenek meg az elnevezési szabályoknak és konvencióknak.

Sok sikert, ha kész vagy, találkozunk a következő fejezetben!

Megoldás: Amikor elvégezted a feladatot, [itt meg tudod nézni](#) az általunk alkotott mintamegoldást. Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat, mielőtt továbbmennél.

4. A függvények eredménye

4.1. Return value (visszaadott érték) és side effect (mellékhatás)

Minden függvény, amikor lefut, kétféle eredményt hozhat létre a kódban:

1. Visszaad egy **értéket** (angolul *return value*).
2. Létrehoz valamilyen **mellékhatást** (angolul *side effect*).

Az első eredmény, a return value visszaadása mindig megtörténik. Ha a függvény nem hoz létre konkrét adatot, akkor a return value `undefined`, azaz üres érték lesz, de a függvény akkor is visszaadja azt.

A mellékhatás ezzel szemben opcionális.

Mellékhatásnak hívunk minden olyan hatást, amelyet a függvény hozott létre – a return value kivételével. Ellentétben a gyógyszerekkel, a programozásban a mellékhatás nem feltétlenül rossz dolog, sőt. Időnként kifejezetten olyan függvényeket írunk, amelyeknek csak a mellékhatását szeretnénk felhasználni.

Megjegyzés: Te magad is írtál már ilyen függvényt. A `printTomatoSoupRecipe` függvény return value-ja `undefined` volt, és egyáltalán nem voltunk rá kíváncsiak. A mellékhatására – a konzolba kinyomtatott üzenetekre – annál inkább.

Az esetek jó részében persze igenis kíváncsiak vagyunk a return value-ra, és szeretnénk azt megkapni. Nézzük meg ezt az esetet egy új, négyzetek területét kiszámító függvényen keresztül!

Ellenőrző kérdés

Az alábbi állítások közül melyik igaz a függvények eredményére? Válaszd ki a helyes opciót!

- Egy függvény futtatásának két, egymást kizáró eredménye lehet: vagy visszaad egy értéket (return value) az öt meghívó kódnak, vagy lehet valamelyen mellékhatása (side effect).
 - Sajnos ez nem a helyes válasz. Két probléma is van vele. Egyrészt a return value visszaadása nem opcionális – ez mindenkor megtörténik, ha egy függvény lefut. Másrészről a kapcsolat nem vagylagos – egy függvénynek lehet mellékhatása amellett, hogy visszaad egy értéket.
- **Egy függvény futtatásának két eredménye lehet: kötelezően visszaad egy értéket (return value) az öt meghívó kódnak, illetve opcionálisan lehet valamelyen mellékhatása (side effect).**
- Egy függvény futtatásának az eredménye az, hogy kiszámol valamelyen értéket, és azt egy mellékhatásban közli a felhasználóval.
- A függvényeknek önmagukban nincs eredménye. Ahhoz, hogy eredményt adjanak vissza, szükséges egy return value-t és egy mellékhatást definiálnunk.

4.2. Gyakorlás: calculateRectangleArea függvény

Hogy gyakorolhass egy picit, próbáld meg önállóan megalkotni a függvényt, amelyet a következő leckékben példaként fogunk használni.

A kódodnak a következő kritériumoknak kell megfelelnie:

- A függvény neve legyen `calculateRectangleArea`.
- Fogadjon be egy `width` (szélesség) és `height` (magasság) paramétert.
- Számolja ki a megadott adatok alapján a négyzet területét.
- Nyomtassa ki az eredményt a konzolba az alábbi stringben: "A négyzet területe x négyzetméter." (az x helyén a számítás eredményével).

Sok sikert, ha kész vagy, találkozunk a következő leckében!

Megoldás: Amikor elvégezted a feladatot, [itt meg tudod nézni](#) az általunk alkotott mintamegoldást. Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat, mielőtt továbbmennél.

4.3. Fókuszon a visszaadott érték (return value)

Jelen helyzetben a `calculateRectangleArea` függvénynek két eredménye van:

- **Mellékhatás** (side effect): kinyomtatja az alábbi stringet a konzolba: "A négyzet területe x négyzetméter.".
- **Visszaadott érték** (return value): `undefined`.

Alakítsuk át úgy ezt a mellékhatás-fókuszú függvényt, hogy a visszaadott érték legyen a lényeg. Másold be az alábbi kódot a böngésződ konzoljába, és üss egy Enter-t:

```
function calculateRectangleArea(width, height) {  
    let area = width * height;  
}
```

Itt a függvény funkciója megváltozott. A feladata ezúttal az, hogy a felhasználó által beírt argumentumokat összeszorozza, az eredményt pedig egy `area` nevű változóba mentse el. Ez a függvény nem jeleníti meg szövegesen az eredményt, hiszen nem szerepel benne a `console.log()` utasítás.

Hogy lássuk is a számítás eredményét, írjuk be a konzolba külön a `console.log()` parancsot. A már ismert stringet fogjuk kiíratni, ezúttal azonban a pontos érték helyére egy függvényhívást helyettesítünk be. Másold be ezt a kódot a konzolba, majd üss egy Enter-t:

```
console.log('A négyzet területe ' + calculateRectangleArea(5, 7)  
+ ' négyzetméter.');
```

Habár a függvény tökéletesen működik, és a háttérben kiszámolta a végeredményt, mégsem írta azt ki a `console.log()` utasításra. Helyette az `undefined` érték szerepel a mondatban, mintha nem lenne értéke a függvénynek:

A négyzet területe undefined négyzetméter.

Ne aggódj, nem te csináltál valamit rosszul, egyszerűen szükséged van egy újabb JavaScript-eszközre: a `return` kulcsszóra.

Ellenőrző kérdés

Milyen értéket ad vissza egy függvény, ha nem specifikáljuk a return value-ját?

- Az utolsó változó értékét, amelyet elmentettünk a függvénytörzsben.
- A `null` vagy az `undefined` értéket.
- Egy boolean (`true` vagy `false`) értéket.
- **Az `undefined` értéket.**

4.4. A return kulcsszó

A függvények visszaadott értéke alapbeállítás szerint `undefined`. Függetlenül attól, hogy mi történik a function bodyban, egészen addig `undefined` értéket fogunk visszakapni,

amíg egyértelműen meg nem mondjuk a függvénynek, hogy mi legyen a return value.

Hogyan tudjuk ezt megtenni? A `return` kulcsszóval.

A `return` kulcsszót a függvénytörzsbe szúrjuk be, és az az érték követi, amelyet szeretnénk visszakapni.

Megjegyzés: Erre utal a `return` szó jelentése is: *visszaadni*.

Töltsd újra az oldalt, ahol a konzolod nyitva van, majd próbáld ki az alábbi kódot, amelyet ezúttal kiegészítettünk a `return` utasítással:

```
function calculateRectangleArea(width, height) {  
    let area = width * height;  
    return area;  
}  
  
console.log('A négyszög területe ' + calculateRectangleArea(5, 7)  
+ ' négyzetméter.');
```

Ezúttal a függvény eredménye szépen megjelent a stringben, éljen!

Megjegyzés: Ha úgy gondolsz a függvényekre, mint pici, zárt gyárakra, akkor a `return` kulcsszó a teherautó, amely az elkészített „terméket” ki tudja hozni a gyárból.

A `return` utasítás kifejezetten rugalmas, nemcsak változókat tud befogadni, mint a fenti példában, hanem értékeket, sőt kifejezéseket is. Ebből kiindulva az `area` változóra igazából nincs is szükség a `calculateRectangleArea` függvényben:

```
function calculateRectangleArea(width, height) {  
    return width * height;  
}
```

Megjegyzés: A `return` parancsnak nem kell kapcsolódnia a függvény belső logikájához. Akár olyan információkat is visszaadhatunk vele, amelyeknek semmi köze a függvényhez. Próbáld ki, helyettesítsd be az eredeti függvényben a `return` utáni `area` változónevet egy tetszőleges számra vagy stringre, és nézd meg, mi történik!

Ellenőrző kérdés

Mire való a `return` kulcsszó? Válaszd ki a helyes megoldást az alábbi lehetőségek közül!

- A `return` parancssal tudjuk megszabni, hogy milyen változót adjon vissza a függvény, amikor lefut.
- A `return` parancssal tudjuk megszabni, hogy mi történjen, amikor meghívjuk a függvényt.

- A `return` parancssal tudjuk megszabni, hogy milyen értéket adjon vissza a függvény, amikor lefut.
- A `return` parancssal tudjuk megszabni, hogy az `undefined` helyett mi legyen a függvény mellékhatása.

4.5. `return = stop!`

Mielőtt továbblépnénk, egy utolsó, nagyon fontos funkcióját meg kell ismerned a `return` kulcsszónak: **leállítja a függvény végrehajtását**.

Bárhol helyezzük el a függvényben, az utána következő utasítások (statementek) már nem lesznek végrehajtva.

Próbáld ki! Fogd meg az alábbi kódblokkot, és másold be a böngésződ konzoljába, majd üss egy Entert:

```
function printPoem() {  
    console.log('Egy – megérett a meggy,' );  
    return 'stop!';  
    console.log('Kettő – csipkebokorvessző,' );  
    console.log('Három – te vagy az én párom,' );  
}  
  
printPoem();
```

Mi történt? Az első `console.log()` még megérkezett, de rögtön utána a `return` leállította a függvényt, és kiugrott belőle.

Megjegyzés: Ahogy látod, a `return` kulcsszó egy újabb struktúra, amely megváltoztatja a control flow-t. Amikor a JavaScript-motor találkozik egy `return` parancssal, megáll, és visszateleportál ahhoz a kódrészlethez, amely meghívta az adott függvényt.

Ellenőrző kérdés

Hogyan foglalnád össze a `return` utasítás funkcióit?

- A `return` parancssal a függvény eredményét és melléktermékét írathatjuk ki.
 - Sajnos ez nem teljesen igaz. Próbáld újra, kérlek!
- A `return` parancssal bármilyen értéket kiírathatunk a függvény eredményeként.
 - Ez igaz, ám ezen kívül más, ennél fontosabb funkciói is vannak a parancsnak.
- **A `return` parancs leállítja a függvény végrehajtását, és visszaadja a benne specifikált értéket.**
- A `return` parancs leállítja a függvény végrehajtását, így a függvény inaktívvá válik.

4.6. Gyakorlás: életkor-kalkulátor

Hogy gyakorolhasd a mellékhatások nélküli függvények írását, illetve a `return` kulcsszó

használatát, itt egy újabb feladat: készíts egy életkor-kalkulátort!

A programod a következő kritériumoknak feleljen meg:

- Egy darab függvényből és egy `console.log()` utasításból álljon.
- A függvény neve legyen `calculateAge`.
- A függvény fogadjon be két paramétert:
 - A jelenlegi évet.
 - A felhasználó születési évét.
- A paraméterek legyenek a szabályoknak és konvencióknak megfelelően elnevezve.
- A függvény számolja ki a paraméterekből a felhasználó korát, és mentse el egy `age` nevű változóban.
- Az `age` változót adja vissza a függvény, mint return value-t.
- A függvény után egy `console.log()` utasítás nyomtassa ki a konzolba a te életkorodat úgy, hogy a `calculateAge` függvényt hívod meg magában a `console.log()`-ban.

Sok sikert, ha kész vagy, találkozunk a következő fejezetben!

Megoldás: Amikor elvégezted a feladatot, [itt meg tudod nézni](#) az általunk alkotott mintamegoldást. Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat, mielőtt továbbmennél.

5. Szerepkörök szétválasztása függvényekkel

5.1. Miért használunk függvényeket?

Az eddigi fejezetekben nagyrészt a függvények egy specifikus előnyét ismerhettek meg: azt, hogy **kevesebb kóddal érheted el ugyanazt az eredményt**, mivel nem kell újra és újra leírnod ugyanazt a (vagy egy nagyon hasonló) kód részletet.

Ez – önmaguk ismétlésének kerülése – olyannyira erős igény a programozók között, hogy a jelenségnek még neve is van. Az ilyen kódot **DRY** (Don't Repeat Yourself, azaz Ne Ismételd Magad) kódnak hívják.

Megjegyzés: A dry szó angolul azt jelenti, száraz, így az elnevezés egyfajta humoros kikacsintás is: „Írj száraz kódot!“.

Programozóként illik törekedni arra, hogy DRY kódot írj.

Ugyanakkor a függvényeknek van egy másik előnye is: lehetővé teszik a **szerepkörök szétválasztását** (angolul *separation of concerns*).

A szerepkörök szétválasztása – a DRY-hoz hasonlóan – egy fontos elv a programozásban. Azt mondja ki, hogy amikor megírunk egy programot, a kódot **válasszuk szét különálló szekciókra, amelyek mind egy-egy jól körülhatárolt szerepet látnak el**.

Ezek a szekciók aztán egymással kommunikálva, egymás működésére építve hozzák létre a kész, teljes szoftvert. A szétválasztás előnye, hogy a program átláthatóbb lesz, és könnyebb lesz fenntartani, mivel egy konkrét rész csak egy konkrét, viszonylag egyszerű feladatot lát el.

Ebben a fejezetben a szerepkörök szétválasztásáról fogunk beszélni, azért, mert ez az elv a modern szoftverfejlesztés egyik alappillére. Érdemes elsajátítanod a működését a gyakorlatban, mivel később úton-útfélen találkozni fogsz vele.

Ellenőrző kérdés

Milyen előnyökkel jár számunkra a függvények használata? Válaszd ki az alábbi lehetőségek közül a leg pontosabb leírást!

- A függvények használatának eredménye, hogy a kódunk kevesebb sorból fog állni, így gazdaságosabb lesz. Kevesebb memóriát, processzorkapacitást használ.
 - Ez igaz ugyan, de még nem a teljes válasz. Fussd át még egyszer a leckét, és próbáld meg újra!
- **A függvények használatával két alapelvet tudunk megvalósítani: a DRY kódöt és a szerepkörök szétválasztását. Az elvek betartásának eredményeként a programunk gazdaságosabb, könnyebben érthető és egyszerűbben fenntartható lesz.**
- A függvények használatának az az előnye, hogy a programon belüli egyes szerepköröket szét tudjuk választani egymástól. Ezzel a kódunk könnyebben olvasható és érthető lesz, és később egyszerűbb lesz fenntartanunk azt.
 - Ez való igaz, de a függvényeknek emellett van egy másik előnye is. Fussd át még egyszer a leckét, és próbáld meg újra!
- A függvények használatának nincs egyértelmű, objektív előnye. Ez inkább csak egy stílus kérdés. Vannak programozók, akik függvényekben szeretnek kódolni, és vannak, akik nem.
 - Ez sajnos nem igaz. Fussd át ismét a leckét, és próbáld újra megtalálni a helyes választ!

5.2. Szétválasztott területszámítás

Hogy lásd, hogyan működik a szerepkörök szétválasztása a gyakorlatban, vegyük elő az egyik korábbi programunkat, amely négyzetek területét számolta ki.

Nyiss egy új bint, és másold be az alábbiakat a JavaScript-panelre:

```
function calculateRectangleArea(width, height) {  
    return width * height;  
}  
  
console.log('A négyzög területe ' + calculateRectangleArea(5, 7)  
+ ' négyzetméter.');
```

Ez a program lehetne még elegánsabb. A `calculateRectangleArea` függvény rendben van, de a csupaszon álló `console.log()` utasítás furán néz ki. Az egyik lehetőségünk az lenne, hogy berakjuk a már létező függvénybe, valahogyan így:

```
function calculateRectangleArea(width, height) {  
    let area = width * height;  
    console.log('A négyszög területe ' + area + ' négyzetméter.');//  
}
```

Ez azonban sértené a szerepkörök szétválasztásának elvét. Ha megfigyeled, a fenti függvényben két, jól elhatárolt feladat jelenik meg:

1. A négyszög területének kiszámolása.
2. Az eredmény kommunikációja a felhasználó felé.

Ezeket két külön függvénnnyel lenne érdemes megoldanunk:

```
function calculateRectangleArea(width, height) {  
    return width * height;  
}  
  
function printRectangleArea() {  
    let area = calculateRectangleArea(5, 7);  
    console.log('A négyszög területe ' + area + ' négyzetméter.');//  
}
```

Látod a különbséget? Így a `calculateRectangleArea` semmi más nem csinál, csak számol, míg a `printRectangleArea` semmi más nem csinál, csak a konzolba nyomtatja az eredményt. A két függvény között van kapcsolat – a `printRectangleArea` a függvénytörzsben meghívja a `calculateRectangleArea`-t –, de ez a kapcsolat laza. Csak egy híd két külön, jól elhatárolt világ között.

Menj vissza a JS Binbe, és a program legalján hív meg a `printRectangleArea` függvényt. Nézd meg, mi történik.

Megjegyzés: Hogy hol van a határ a szerepkörök között, minden esetben döntés lesz, erre nincsenek szilárd szabályok. Jó gyakorlatok persze léteznek. A fenti kódban például az az elv jelenik meg, hogy érdemes elválasztani egymástól egy program logikáját (a szekciót, ahol a számítás történik) és az eredmény megjelenítését.

Ellenőrző kérdés

Hogyan lehet kiválasztani a szerepkörök határát, azaz azt, hogy mi kerül egy-egy függvénybe? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- Olyan programot kell írnunk, ahol az egyes függvényeknek vagy csak visszaadott

értéke (return value), vagy csak mellékhatása (side effect) van. A kettő nem keveredhet egy függvényben, ez adja a szerepkörök szétválasztásának alapját.

- Ugyan ez egy legitim programozási taktika, de semmiképp sem kötelező érvényű. Nyugodtan írhatsz olyan függvényt, amelynek return value-ja és side effectje is van, ha a helyzet úgy kívánja.
- A számításokat végző szekciót (a logikát) és az eredményeket megjelenítő részt mindenkor szét kell választanunk a programunkban.
 - Ez így nem igaz. A fenti állítás csak egy iránymutatás, amelyet nem kötelező betartanunk.
- **A szétválasztásra nincs kemény szabály, ez minden esetben döntés kérdése. Vannak iránymutatások, például érdemes arra törekedni, hogy a számítást végző és az eredményeket megjelenítő részeket válasszuk szét.**
- A szétválasztás alapja az, hogy a kódot én mint a programozó hogyan tartom a legszebbnek. Ez egy esztétikai kérdés.
 - Ennél azért vannak konkrétabb útmutatások. Olvasd el még egyszer a leckét, és próbáld újra!

5.3. Tetszőleges szélesség, tetszőleges hosszúság

Talán észrevettek, hogy az előző leckében létrehozott programunknak van egy komoly hibája. Történék bármi, mindenkor 5 méter széles és 7 méter hosszú négyszög területét számolja ki, mert ezt rögzítettük a 6. sorban lévő függvényhívásban:

```
function calculateRectangleArea(width, height) {  
    return width * height;  
}  
  
function printRectangleArea() {  
    let area = calculateRectangleArea(5,7);  
    console.log('A négyszög területe ' + area + ' négyzetméter.');//  
}  
  
printRectangleArea();
```

Ez meglehetősen haszontalanabbá teszi az eszközt. Ehelyett sokkal jobb lenne, ha – amikor meghívjuk a `printRectangleArea` függvényt – megadhatnánk neki argumentumokat, amelyeket a `calculateRectangleArea` fel tud használni a számításhoz. Lássuk, hogyan lehet ezt elérni!

A megoldás első lépése egyszerű: a `printRectangleArea` függvénynek kell egy `widthFromUser` és `heightFromUser` paraméter:

```
...  
function printRectangleArea(widthFromUser, heightFromUser) {  
    ...
```

```
}
```

```
...
```

Ne feledd, a paraméterek a függvénytörzsön belül változókként viselkednek. Bárhova behelyettesíthetők, akár a függvénytörzsön belül szereplő függvényhívás argumentumaiként is. Ezt tesszük mi is:

```
...
function printRectangleArea(widthFromUser, heightFromUser) {
    let area = calculateRectangleArea(widthFromUser,
heightFromUser);
    ...
}
...
```

Mivel a `printRectangleArea` most már képes argumentumokat fogadni, alakítsd át a függvényhívást a bin legutolsó sorában a következőre:

```
printRectangleArea(2, 4);
```

Láss csodát, a program működik! Na de mégis hogyan?

5.4. Egymásnak passzolt argumentumok és a control flow

Nem hibáztatunk, ha az előző lecke fekete mágiának tűnt. A függvények egymás között passzolatott argumentumait elsőre nem könnyű átlátni, főként, ha a programot felülről lefelé olvasva próbálod értelmezni.

Szerencsére azonban már ismersz egy technikát azokra a helyzetekre, amikor egy program fejtörést okoz:

Kövesd a control flow-t!

Lássuk, mi történik, ha lépésről lépésre végigsétálunk az úton, amelyet a JavaScript-motor jár be az alábbi program végrehajtásakor:

```
function calculateRectangleArea(width, height) {
    return width * height;
}

function printRectangleArea(widthFromUser, heightFromUser) {
    let area = calculateRectangleArea(widthFromUser,
heightFromUser);
    console.log('A négyszög területe ' + area + ' négyzetméter.');
```

```
}
```

```
printRectangleArea(2, 4);
```

Tipp: Az alábbi leírásban haladj sorról sorra, és közben kövesd a kódblokkban, hogy épp miről beszélünk.

A control flow a következőképp alakul:

- Az **1. sorban** a motor megtalálja a `calculateRectangleArea` függvényt, de még nem csinál semmit, mert nem volt függvényhívás (function call).
- Az **5. sorban** a motor megtalálja a `printRectangleArea` függvényt, de még nem csinál semmit, mert nem volt függvényhívás (function call).
- Az akció a **10. sorban** kezdődik. Itt történik egy `printRectangleArea(2, 4);` function call.
 - A motor megjegyzi az argumentumokat, majd visszateleportál az **5. sorba**, és a `widthFromUser` és `heightFromUser` paraméterek helyére behelyettesíti azokat: a **2-t** és a **4-et**.
 - A **6. sorban** a motor létrehozza az `area` változót, majd egy újabb function calltal talál: `calculateRectangleArea(widthFromUser, heightFromUser)`.
 - Mielőtt bármit csinálna, a motor felismeri, hogy a `widthFromUser` és a `heightFromUser` a `printRectangleArea` paraméterei, és behelyettesíti azokat az aktuális értékükkel. Így a függvényhívás valójában így néz ki: `calculateRectangleArea(2, 4)`.
 - Ezen a ponton – mivel function call történt – a motor felugrik az **1. sorba**.
 - Az itteni `width` és `height` helyére behozza a **2-t** és a **4-et**, hiszen ezekkel az értékekkel történt a function call a **6. sorban**.
 - A **2. sorban** a motor ismét behelyettesít: ezúttal a `calculateRectangleArea` aktuális argumentumaival, **2-vel** és **4-vel**.
 - Ezután végrehajtja a kifejezést, majd a `return` kulcsszó miatt visszaadja az eredményt (amely jelen esetben **8**), leállítja a függvényt, kilép belőle, és visszateleportál a **6. sorba**, ahonnan a függvényhívás történt.
 - A **6. sorba** a motor visszahozza a `calculateRectangleArea` return value-ját, azaz a **8-at**, úgyhogy a sor átalakul: `let area = 8;`.
 - A motor elmenti a **8-as** számot az `area` változóba.
 - A **7. sorban** a motor végrehajtja a `console.log()` utasítást a behelyettesített `area` változóval.
 - A **8. sorban** véget ér a `printRectangleArea` függvény, így a motor visszateleportál a **10. sorba**, ahonnan a függvény meghívódott.
- A **10. sor** végén a program véget ér, így a motor leáll.

Ha felülről lefelé próbálnád meg elolvasni és értelmezni ezt a programot, az egész elég zavarbaejtő lenne. Ha viszont követed a control flow-t, láthatod, hogy semmi más nem

történik, mint hogy a függvények szépen meghívják egymást, és a paramétereiken keresztül, illetve a return value segítségével adatokat közölnek egymással.

Pro tip: A control flow ily módon történő feltérképezése egy szuper hasznos eszköz minden fejlesztő kezében. Amikor nem világos számodra, hogy egy program hogyan csinálja, amit csinál, kezdd el követni a control flow-t, és lépésekkel össze tudod rakni a logikát.

5.5. Gyakorlás: körök kerülete és területe

Hogy gyakorolhasd a szerepkörök szétválasztását, illetve a függvények egymás közötti kommunikációját, ír egy programot, amely kiszámolja egy tetszőleges kör kerületét és területét a kör sugara alapján.

A programod a következő kritériumoknak feleljön meg:

- Álljon három függvényből:
 - egy számolja ki a kör kerületét,
 - egy a területét,
 - egy pedig kérje be a sugarat a felhasználótól, majd a számítások után nyomtassa ki az eredményt a konzolba.
- A függvények és paraméterek legyenek a szabályoknak és konvencióknak megfelelően elnevezve.

Megjegyzés: Szükséged lesz arra, hogy utánjárj, hogyan lehet kiszámolni egy kör kerületét és területét. Ez alkalmat ad arra, hogy elkezdj hozzászokni az ilyesfajta kutatómunkához. Programozóként gyakran kell majd utánjárnod különböző kérdéseknek az interneten.

Sok sikert, ha kész vagy, találkozunk a következő fejezetben!

Megoldás: Amikor elvégezted a feladatot, [itt meg tudod nézni](#) az általunk alkotott mintamegoldást. Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat, mielőtt továbbmennél.

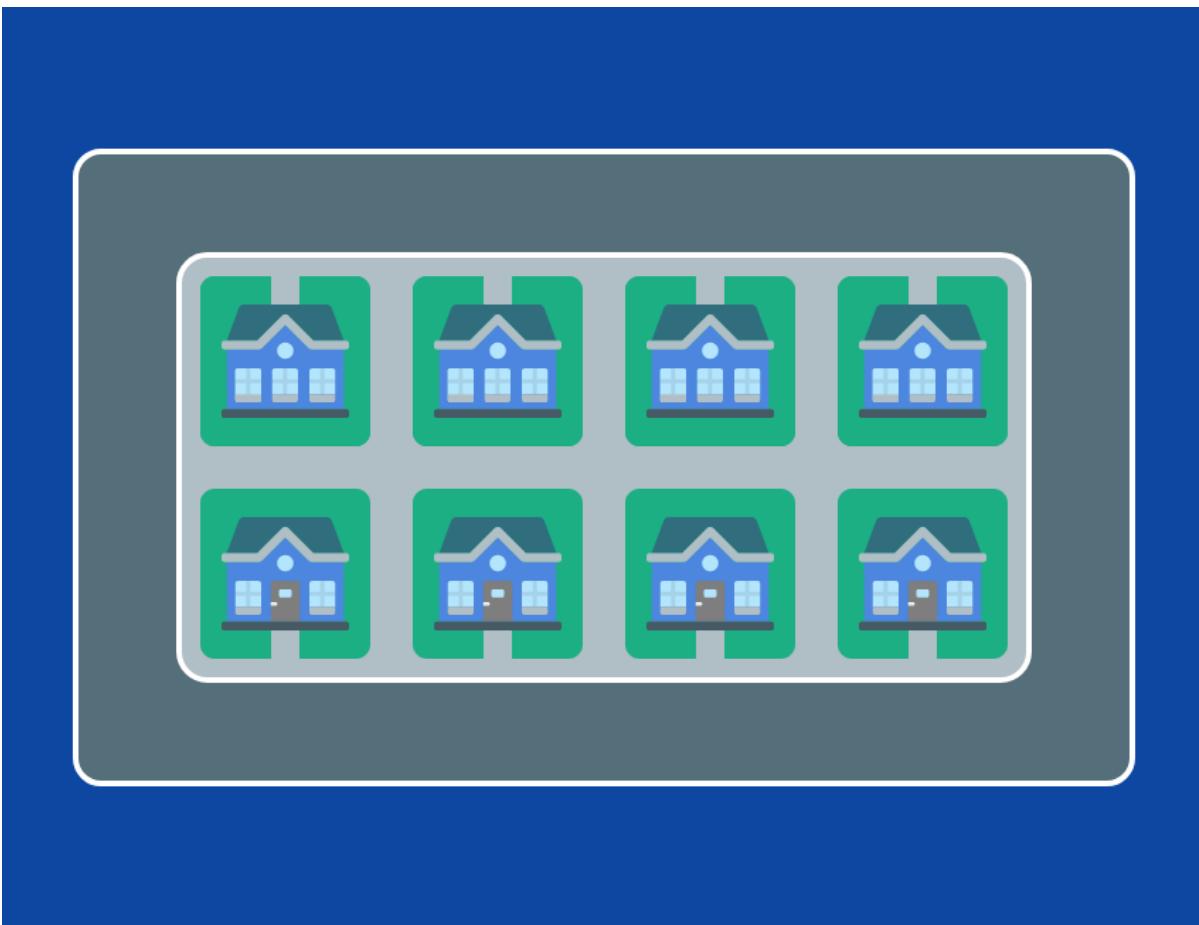
6. Scope-ok (területek)

6.1. Globális és lokális scope-ok

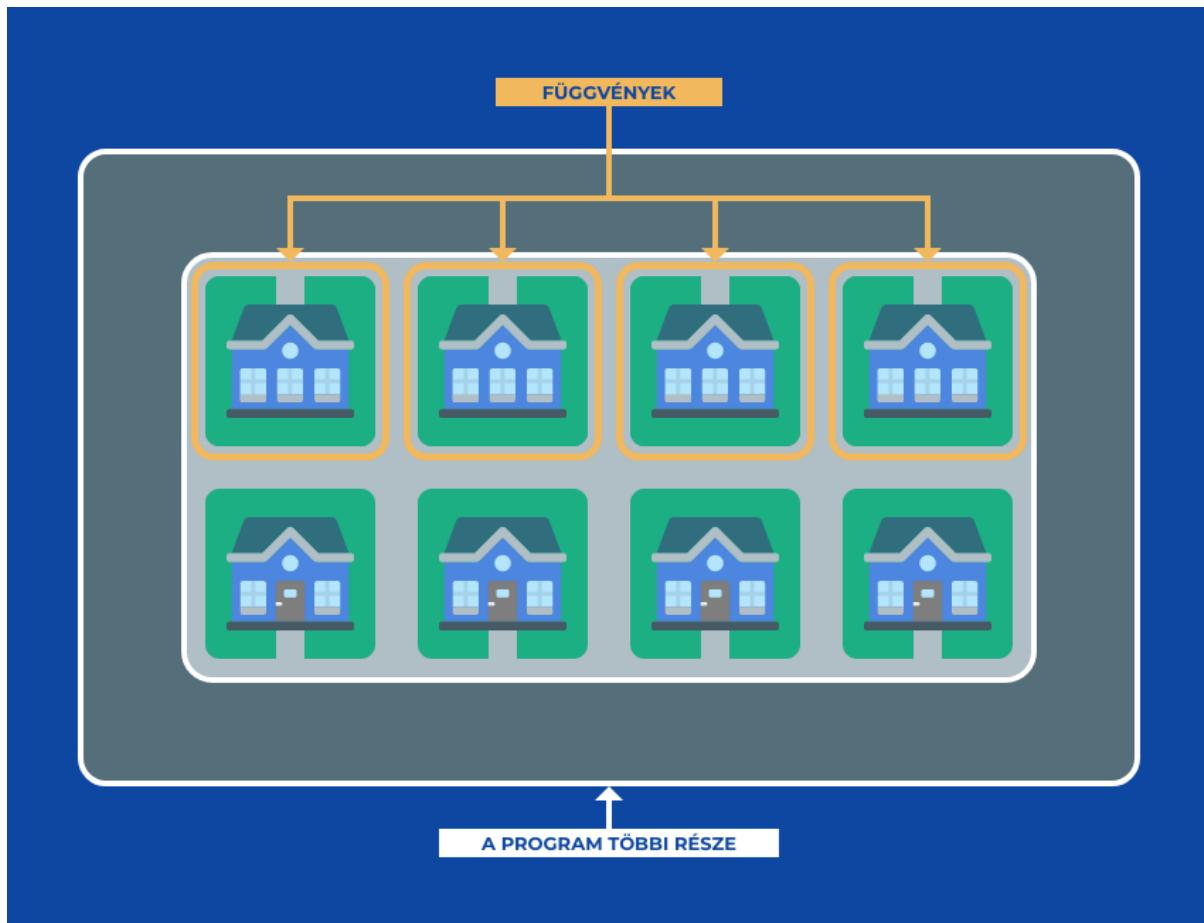
Az utolsó téma, amelyet érintenünk kell a függvényekkel kapcsolatban, a scope-ok (területek) jelensége.

Az már tiszta számodra, hogy a függvények külön kis kódblokkok, amelyek egy nagyobb kódban ülnek valahol. Amiről még nem beszélünk, az az, hogy miként viszonyulnak a kód többi részéhez.

Hogy ezt megértsd, képzelj el egy utcát, tele házakkal.

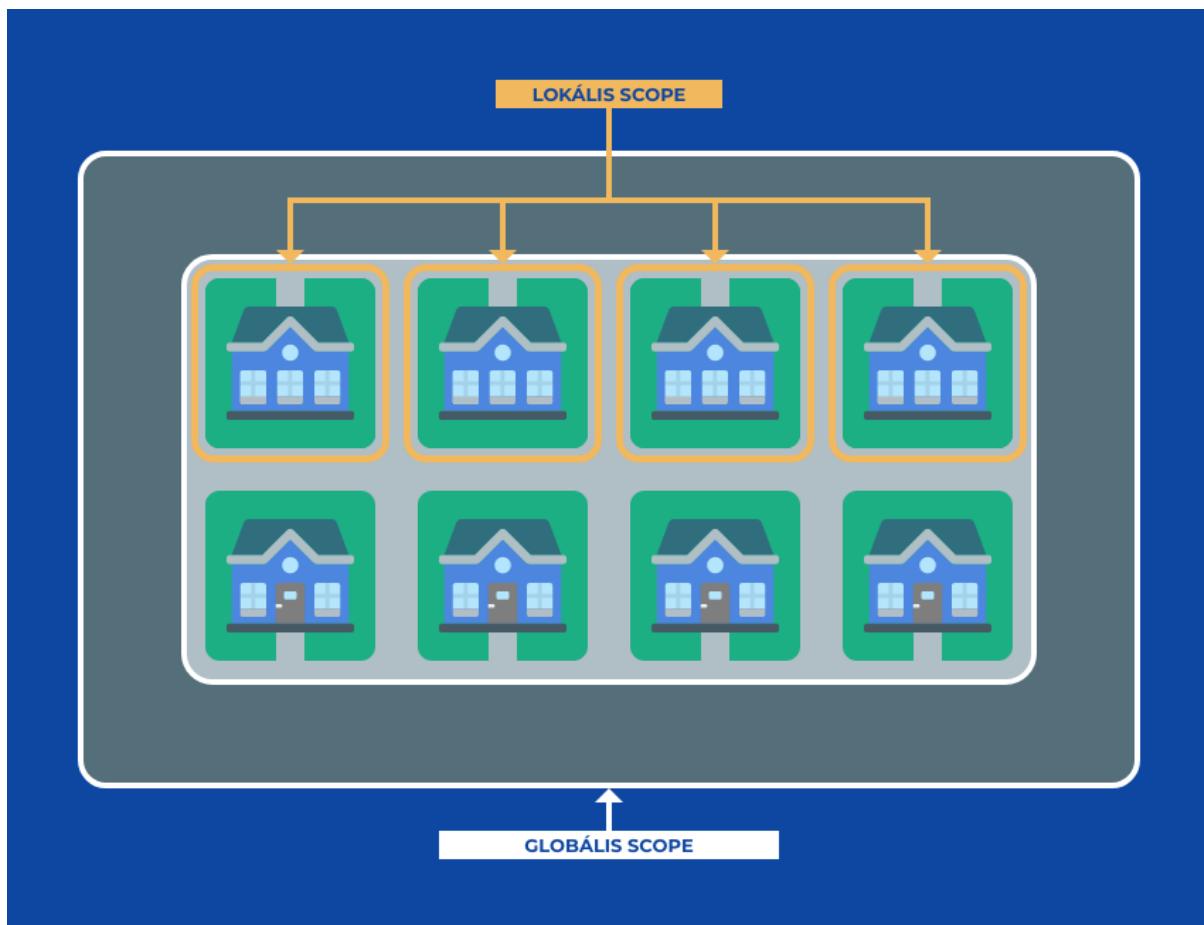


A házak a függvények. Az utca pedig, amelyen a házak állnak, a JavaScript-program többi része. Ketten együtt alkotják a teljes JavaScript-programot.



Ezekre a területekre az angol scope (terület vagy hatókör) szóval hivatkozunk. A lenti képen láthatod, hogy kétféle scope létezik:

- **Lokális (local scope)** → a függvények (a hasonlatban a házak).
- **Globális (global scope)** → a program többi része, minden, ami a függvényeken kívül található (a hasonlatban az utca).



Mindennek azért van jelentősége, mert a globális és lokális scope-ban létrehozott változók másként viselkednek:

- **A globális scope-ban definiált változók a programon belül bárhol elérhetőek, a lokális scope-okban is.**
- Ezzel szemben **a lokális változók csak abban a konkrét lokális scope-ban (függvényben) léteznek, ahol létrejöttek.** A program többi része nem éri el őket.

Ilyen szempontból a JavaScript hasonlóan működik a fenti képen ábrázolt szomszédsághoz:

- A házakból lehet látni az utcát, a rajta lévő tárgyakat és a többi házat,
- de egyik házból sem láthatsz át egy másik ház belsejébe.
- Az utcáról szintén lehet látni az utcát magát és a házakat, de nem láthatsz be a házakon belülre.

Tehát egy függvény:

- felhasználhatja a globális scope-ban létrehozott változókat,
- de nem férhet hozzá más függvényekben létező lokális változókhöz,
- és a saját lokális változói sem láthatók a program többi része számára.

A függvényekből kifelé lehet látni, de kintről a függvények belsejébe nem.

Ellenőrző kérdés

Mi különbözteti meg a lokális és globális változókat egymástól? Válaszd ki a helyes választ az alábbi lehetőségek közül!

- A lokális és globális változók között az a különbség, hogy előbbieket camelCase írásmód szerint nevezzük el, utóbbiakat pedig csupa nagybetűvel és alulvonással.
- A lokális változók a programon belül bárhol elérhetőek, a függvények belsejében is. A globális változók ezzel szemben csak a globális scope-ban léteznek.
- A lokális változók a programon belül bárhol elérhetőek, a függvények belsejében is. A globális változók ezzel szemben csak abban a konkrét scope-ban (függvényben) léteznek, ahol létrehoztuk őket.
- **A globális változók a programon belül bárhol elérhetőek, a függvények belsejében is. A lokális változók ezzel szemben csak abban a konkrét scope-ban (függvényben) léteznek, ahol létrehoztuk őket.**

6.2. Scope-ok a gyakorlatban

Hogy lásd a scope-ok működését a gyakorlatban is, itt egy rövid példa:

```
var variable = 10;

function localScope() {
    var variable = 20;
    console.log('local variable in the function: ' + variable);
}

localScope();

console.log('global variable: ' + variable);
```

Ez egy apró program, amelyben van egy globális scope (a program függvényeken kívüli része), és egy `localScope` nevű függvény, amely egy lokális scope-nak számít. A függvényben és a globális scope-ban is deklaráltunk egy `variable` változót. Meg tudod-e tippelni, hogy mit fog kiírni a konzol, ha lefuttatjuk ezt a programot?

Tipp: Először próbálj meg magadtól rájönni a válaszra, csak utána olvass tovább.

A konzol a következő sorokat fogja kinyomtatni:

```
local variable in the function: 20
global variable: 10
```

Miért? Mert a függvény egy külön világ. Ha ott deklarálunk egy változót, az csak ott létezik, és nem fog ütközni a globális scope-ban létrehozott változókkal. Akkor sem, ha ugyanaz a nevük. Ezen a ponton a JavaScript-motor ezt nem a globális `variable` újra deklarálásának

tekinti, hanem egy teljesen új, másik változó létrehozásának.

Megjegyzés: Felmerülhet benned a kérdés, hogy a második `console.log()` hogyan dönti el, hogy melyik `variable`-t nyomtassa ki. A válasz egyszerű: ez az utasítás a globális scope-ban van, és emlékezz, a globális scope-ból nem lehet belátni a lokális scope-okba. A második `console.log()` utasítás szempontjából csak egy `variable` létezik – a globális.

Mi történik akkor, ha egy pici változtatást eszközölünk a programunkon, és a `localScope` függvényben kivesszük a `var` kulcsszót a változó neve elől?

```
...
function localScope() {
    variable = 20;
    console.log('local variable: ' + variable);
}
...
...
```

Most mit nyomtat majd ki a konzol szerinted?

Tipp: Először próbálj meg magadtól rájönni a válaszra, csak utána olvass tovább.

A konzolban az eredmény:

```
local variable: 20
global variable: 20
```

Miért? Azért, mert a függvényekből viszont látunk kifelé a globális scope-ra.

Amikor a JavaScript-motor megtalálja a függvényen belül a `variable` változót, először megpróbálja megnézni, hogy a lokális scope-ban van-e deklarálva ilyen változó. Ha nincs, akkor ugrik egyet kifelé, és megnézi, hogy a globális scope-ban van-e. Ez esetben volt, így megváltoztatta annak az értékét. Ez az oka, hogy a második `console.log()` utasítás is 20-at írt ki eredményül.

Ellenőrző kérdés

Mi a fő szabály a scope-okkal kapcsolatban? Válaszd ki a helyes lehetőséget az alábbiak közül!

- **A lokális scope-okból (függvényekből) kifelé lehet látni, de kintről a függvények belsejébe nem. A lokális scope-on belül hozzáférhetünk globális változókhöz, de a globális scope szempontjából a lokális scope-ok változói nem léteznek.**
- A lokális scope-okba (függvényekbe) befelé lehet látni, de a függvényekből kifelé nem. A lokális scope-on belül nem férhetünk hozzá globális változókhöz, de a globális scope szempontjából a lokális scope-ok változói is léteznek.

- A lokális és a globális scope között az a különbség, hogy máshogyan kell bennük elnevezni a változókat.
- Mindig figyelnünk kell arra, hogy ha egy változónévet már felhasználtunk a globális scope-ban, azt nem használhatjuk újra lokális scope-okban.

7. Összefoglalás

7.1. Mit tanultál eddig?

Gratulálunk! Ha idáig eljutottál, az azt jelenti, hogy:

- tudod, hogy a függvények a nagyobb programon belüli kisebb programok, utasítások sorozata, amelyet végrehajt a program.
- tudod, milyen részekből áll egy függvény, és hogyan kell azt deklarálni.
- tudod, hogyan kell működésbe hozni (meghívni) egy függvényt.
- tudod, mik azok a paraméterek és argumentumok, valamint azt is, hogyan kell ezeket megadni.
- tudod, hogy bármely függvénynek kétféle eredménye lehet (visszaadott érték és mellékhatás), és hogyan hívda elő az előbbi a függvényt lezáró `return` parancs segítségével.
- tudod, hogy a JavaScript-motor a kód beolvasásakor és végrehajtásakor milyen sorrendiséget követ (control flow).
- tudod, hogy a program változói helyzetüktől függően eltérő (globális vagy lokális scope-hoz (területhez) tartoznak.
- tudod, miért érdemes szerepek szerint szétválasztani a programot kisebb szekciókra függvények segítségével.

Most már készen állsz arra, hogy továbblépj. Találkozunk a következő modulban!

Megjegyzés: Ha a fenti összefoglaló olvasása közben elbizonytalanodtál valahol, kérlek, ugorj vissza az adott fejezetre, és még egyszer olvasd át jó alaposan, mielőtt továbbhaladnál.

Tipp: Ha szeretnéd leellenőrizni a tudásod, mielőtt továbbmész, töltsd ki a következő oldalon található tesztet.

8. Teszt

8.1. Ellenőrizd a tudásod!

[10 kérdés, körülbelül 10 perc, kérdésenként 1 pont.](#)

Tipp: Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a „Küldés” (Submit) gombot a teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

Megjegyzés: A tesztet többször is kitöltheted.

Sok sikert!

JavaScript alapok IV. (ciklusok)

1. Bevezető a ciklusokhoz

1.1. Bevezetés – Ciklusok és Hófehérke dilemmája

Az előző leckék során megismerkedtél az értékekkel, és megtanultad azokat változókban tárolni, valamint különféle módokon felhasználni. Amikor a változókról szóló részt foglaltuk össze, említettem, hogy azért használjuk ezeket, hogy a kódunk könnyen fenntartható és módosítható legyen.

Ebben a fejezetben ezen az úton lépdelünk tovább, és megtanulunk takarékos kódot írni a ciklusok (loops) segítségével, közben pedig írunk egy kis programot, amely segít Hófehérkének számon tartani a török hollétét.

1.2. Bevezetés – Átköltözés a JSBinbe és egy kis fájszerkezet

Fájszerkezet

Ezt a projektet már a JSBinben fogod építeni, így érdemes röviden felidézni, hogy is illeszkedik a JavaScript egy weboldal szerkezetébe. Ez már biztos ismerős:

- A **HTML**-t tartalmazó szövegfájl írja le az oldal szerkezetét,
- A **CSS**-t tartalmazó szövegfájl írja le az oldal kinézetét,
- A **JavaScript**-et tartalmazó fájl teszi dinamikussá a tartalmat.

Ezek a fájlok **.html**, **.css** és **.js** kiterjesztéssel végződnek, az elnevezésük pedig jó esetben utal a tartalmukra. Amikor komplexebb weboldalakat fogsz építeni, több ilyen fájlt fogsz különböző könyvtárakban tárolni, ezért össze kell linkelned őket.

A HTML-kódod **<head></head>** tagjei közé kell majd belinkelned a megfelelő CSS- és JS-fájlokat a következő módon:

```
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script src="scripts.js"></script>
</head>
```

Megjegyzés: A fájlok neve természetesen bármí lehet, a fentiek csak példák.

Amikor a felhasználó böngészője elkéri a szervertől a weboldalt felépítő fájlokat, végigmegy a HTML-en, és a **<head>**-ben látott fájlok tartalmát lekérve a webszervertől szintén felhasználja az oldal megjelenítésénél.

Mindez a JSBinben

A JSBinben egyszerűbb dolgod lesz, nem kell kötögetned semmit sehol, az alkalmazás automatikusan megcsinálja helyetted mindezt a háttérben.

Annyi a teendőd, hogy az adott nyelvet a megfelelő panelre írd be, HTML-t a HTML-panelre, JS-t a JS-panelre, és így tovább. Fontos: a JSBinben is van konzol, de ne ide írd a kódod, hanem a JS-fülre.

Megjegyzés: A JSBin konzolpanelje majd például a hibakeresésben lehet a segítségedre a későbbiekben.

Feladat

Hozz létre egy új bint a JSBinben, és nevezd el „hofeherke”-nek.

1.3. Bevezetés – Pontosvessző és console.log

Oké, ideje, hogy belevessük magunkat a munkába. A megrendelőnk – Hófehérke – szeretne egy olyan programot, amely megmondja, hogy a török épp a bányában vannak-e vagy sem.

Tipp: Ebben az egész leckében érdemes bezárnod a HTML-, CSS- és Output-füleket, mert nem fogjuk ezeket használni, illetve nyisd ki a Console-fület, mert az viszont fontos lesz. Emellett érdemes kiszedned a pipát az Auto Run JS dobozból.

Első lépésként írunk egy pár sor kódot, ami kilistázza, hogy a fiúk épp a bányában dolgoznak. Másold be a következő kódot a „hofeherke” bined JS-paneljére:

```
'A(z) ' + 1 + '. törpe a bányában.'  
'A(z) ' + 2 + '. törpe a bányában.'  
'A(z) ' + 3 + '. törpe a bányában.'  
'A(z) ' + 4 + '. törpe a bányában.'  
'A(z) ' + 5 + '. törpe a bányában.'  
'A(z) ' + 6 + '. törpe a bányában.'  
'A(z) ' + 7 + '. törpe a bányában.'
```

Ez eddig egyszerű string concatenation, semmi olyasmi, amit ne láttunk volna, szuper. Nyomd meg a „Run” gombot a Console-panelen. O-ó, valami nincs rendben – semmi sem történt.

Két dolog hiányzik, a pontosvesszők (`;`) és a `console.log()` parancs.

Pontosvessző (`;`)

A `.js` fájlba kódolni kicsit más, mint a konzolba. Itt szükség van arra, hogy az egymást követő, JavaScriptben megfogalmazott állításaid valahogy elválaszd egymástól, különben a böngésző nem fogja érteni, hogy mit akarsz.

Ennek az elválasztásnak a pontosvessző az eszköze. Amikor JavaScriptet írsz, a parancsaid végén mindenkor legyen ott a pontosvessző. Egészítsük ki ezzel a fenti kódot:

'A(z) ' + 1 + ' . törpe a bányában.' ;
'A(z) ' + 2 + ' . törpe a bányában.' ;
'A(z) ' + 3 + ' . törpe a bányában.' ;
'A(z) ' + 4 + ' . törpe a bányában.' ;
'A(z) ' + 5 + ' . törpe a bányában.' ;
'A(z) ' + 6 + ' . törpe a bányában.' ;
'A(z) ' + 7 + ' . törpe a bányában.' ;

Megjegyzés: A konzolban is nyilván alkalmazhatsz pontosvesszőt az állítások elválasztására, csak ott általában egy-egy állításnyi kódot használ szegyszerre, ezért nincs rá szükség.

A console.log() parancs

Ha most megnyomod a „Run” gombot, még mindig nem történik semmi. Pontosabban csak úgy tünik neked, mintha nem történne semmi – valójában a géped lefuttatja a JavaScriptet, amit írtál, csak nem jeleníti meg sehol.

Miért? Mert nem mondta meg neki, hogy jelenítse meg. :)

Itt jön be a `console.log()` módszer. Ez a parancs „kinyomtatja a konzolba” azt, ami a két zárójel között található. Egészítsük ki ezzel a kódunkat:

```
console.log('A(z) ' + 1 + '. törpe a bányában.');?>
console.log('A(z) ' + 2 + '. törpe a bányában.');?>
console.log('A(z) ' + 3 + '. törpe a bányában.');?>
console.log('A(z) ' + 4 + '. törpe a bányában.');?>
console.log('A(z) ' + 5 + '. törpe a bányában.');?>
console.log('A(z) ' + 6 + '. törpe a bányában.');?>
console.log('A(z) ' + 7 + '. törpe a bányában.');
```

Cseréld ki a JavaScript-paneleden lévő kódot ezzel a fenti blokkal, és nyomd meg a „Run” gombot. Ha minden jól ment, a konzol szépen kiírta neked, hogy mind a hét törpe a bányaiban van.

Ezen túl azonban még egy rakás gond van ezzel a megoldásunkkal – nagyon sok benne a feleslegesen ismétlődő kód, és ráadásul elkövettük az úgynevezett „hard coding” hibáját, amikor konkrét értékeket írunk a kódba változók helyett. Ez a két dolog együtt nehezen fenntarthatóvá teszi ezt a programot.

I ássuk mit lehet ezzel kezdeni

Feladat

Van egy „Population chart” nevű táblázat [ennek a Wikipédia-cikknek](#) az alján, amely a világ pandapopulációjának a változását ismerteti. Hozz létre egy új bint a JS Binen, és írj egy kis programot, amely kilistázza, hogy melyik évben hány panda élt vadon.

A programod a következő kritériumoknak feleljen meg:

- A program a konzolban jeleníti meg az információkat a `console.log()` segítségével.
- A bejegyzések az **év**: **populáció** formátumot kövessék, pl. **2003: 1596**.
- A számok adattípusa legyen szám (nem pedig string).
- Használj stringösszefűzést az utasításokban.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2. A while ciklus

2.1. Ciklusok – Bevezetés

Az előző leckében felvetettünk két problémát a programmal kapcsolatban, amit Hófehérkénék készítünk:

- túl sok sorból áll, amelyek ráadásul ismétlődnek, sokat kell kódolni,
- nehezen fenntartható, mert konkrét értékek szerepelnek benne – ha valamit változtatni szeretnénk, mindenhol át kéne írni kézzel a számokat.

Szerencsére mindkét gondra megoldást jelentenek a **ciklusok**. :)

A ciklusok olyan kódblokkok a JavaScriptben, amelyek újra és újra automatikusan lefutnak, amíg egy bizonyos állításunk igaz. Két formájuk létezik:

- a **while** ciklus (while loop)
- és a **for** ciklus (for loop).

Először a **while** ciklussal ismerkedünk meg.

2.2. Ciklusok – A while ciklus (while loop) felépítése

Elméletben a **while** ciklus a következő módon működik:

```
while (amíg ez az állítás igaz) {  
    addig ez a kód fusson le;  
}
```

Ha a zárójelek között megfogalmazott állításunk igaz, a kapcsos zárójelek közötti kód le fog

futni. Ha nem igaz, akkor a böngésző kihagyja a blokkot.

Tipp: A while szó angolul azt jelenti, amíg. Ezért is kapta a ciklus a nevét – amíg ez igaz, addig csináld ezt.

Feladat

Melyik zárójel közé kell írni a ciklust kontrolláló állítást? Kapcsos vagy sima?

2.3. Ciklusok – Állítások megfogalmazása JavaScripttel

A zárójelek közé bármi kerülhet, amit a JavaScript állításként tud értelmezni. Pár példa:

- Ha azt mondod, hogy `while (true)`, akkor a kódblokk örökké futni fog.
- Ennek megfelelően a `while (false)` ciklus soha nem fog lefutni.
- A `while (x == 1)` ciklus addig fog futni, amíg az `x` értéke egyenlő 1-gel.

És így tovább.

Megjegyzés: A `true` (igaz) és a `false` (hamis) úgynevezett boolean értékek a JavaScriptben, ahogy erről már beszélünk korábban.

Ez az állítás azért nagyon fontos, mert ez fogja kontrollálni a ciklusodat.

Ha nem jól adod meg, akkor a kódod vagy soha nem fog lefutni, vagy bekerülhet egy örök ciklusba (*infinite loop*), ahol az állítás soha nem válik hamissá, így a program nem tud kiszabadulni a ciklusból, folyamatosan csak futtatja a blokkot újra és újra.

Megjegyzés: A korábbi leckékben tárgyalt összehasonlító operátorok itt nagyon hasznosak lesznek.

Feladat

A feladatod, hogy írj egy szintaktikailag helyes `while` ciklust.

A programod a következő kritériumoknak feleljen meg:

- Egyetlen `while` ciklusból álljon.
- Addig fusson a ciklus, amíg az `y` változó nagyobb vagy egyenlő, mint `10`.

Megjegyzés: A ciklusod még nem fog lefutni, de emiatt ne aggódj. Hamarosan eljutunk a hiányzó részekhez. Most csak a feltételre és a `while` ciklus szerkezetére koncentrálj.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.4. Ciklusok – A while ciklus alkalmazása a Hófehérke problémánkra

Azt szeretnénk, ha egy darab ciklus kilistázná a konzolba azt, hogy minden hét törpe a bányában van. Hogy állíunk neki?

Tudjuk azt, hogy nagyából mi a blokk, amit futtatni akarunk:

```
while (állítás, amit még nem tudunk) {  
    console.log('A(z) ' + érték-amit-még-nem-tudunk + '. törpe a  
    bányában.');//  
}
```

Ezzel önmagában még nem vagyunk sokkal beljebb. Két dologra van most szükségünk:

- egy változóra, amit be tudunk illeszteni a `console.log()` parancsba
- egy állításra, ami kontrollálni fogja a loop-ot.

A változónk deklarálása

Ha emlékszel, akkor az előző leckében kézzel írtuk be mindenhol a törpék sorszámát. Most arra lenne szükségünk, hogy itt egy olyan érték legyen, amit mi valahol máshol szabadon változtathatunk. Ez egészen úgy hangzik, mint egy változó. :)

Deklaráljuk a `dwarfCounter` (törpeSorszám) változót, és kezdeti értéknek adjunk neki egyet. A kódunk most így néz ki:

```
var dwarfCounter = 1;  
while (állítás, amit még nem tudunk) {  
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában.');//  
}
```

Feladat

Melyek azok a szabályok, ami szerint elneveztük a `dwarfCounter` változót?

2.5. Ciklusok – A while ciklus, ami el tud számolni hézig

Oké, eggyel beljebb kerültünk, de ez a kód még nem fogja nekünk végigszámolni a törpéket, csak mindig kiírja, hogy az első törpe a bányában van. Valahogyan meg kell azt oldanunk, hogy a változó értéke minden lefutás után eggyel nőjön.

A `variableName++` utasítás egyet hozzáad a változó értékéhez, míg a `variableName--` egyet elvesz belőle. A `++` operátort használva tudunk írni egy olyan kódblokkot, amely kiírja az aktuális törpét a konzolba, majd hozzáad egyet a `dwarfCounter` változóhoz.

```
var dwarfCounter = 1;  
  
while (állítás, amit még nem tudunk) {
```

```
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában.');
    dwarfCounter++;
}
```

Szuper, most már minden egyes ciklusban eggyel nagyobb sorszámú törpét fog kilistázni a kódunk. Most már csak egy kontrolláló állításra van szükségünk.

Feladat

Mit írnánk a kódba, ha azt szeretnénk, hogy minden ciklus után duplázódjon meg a törpök száma?

2.6. Ciklusok – A ciklust kontrolláló állításunk

Eddig van egy ciklusunk, amely kiírja az aktuális törpét, majd hozzáad egyet a változóhoz, és ezt ismétli. Mi azt szeretnénk, hogy álljon meg a 7. törpénél, hisz nincs több.

Az állításunk tehát a következő: a ciklus fússon, amíg a törpék sorszáma (`dwarfCounter`) el nem éri a hetet. Ez kóban így néz ki:

```
var dwarfCounter = 1;
while (dwarfCounter <= 7) {
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában.');
    dwarfCounter++;
}
```

Kész a while ciklusunk. Gyorsan menjünk végig rajta még egyszer a böngésző szemével:

- `var dwarfCounter = 1;` → a `dwarfCounter` változó értéke egy
- `while (dwarfCounter <= 7)` → amíg a `dwarfCounter` kisebb vagy egyenlő, mint 7
- `console.log('A(z) ' + dwarfCounter + '. törpe a bányában.');` → írd ki a konzolba ezt a stringet
- `dwarfCounter++` → és adj hozzá a `dwarfCounter` változóhoz egyet
- ha kész, menj vissza a ciklus elejére, és ellenőrizd, hogy az állítás még igaz-e – ha igen, futtasd le a kódot, ha nem, akkor megállíthatsz

Másold be a fenti kódot egy új bin JS-paneljére, és futtasd le. Ha minden igaz, ugyanazt az eredményt kell kapnod, mint amikor kézzel beírtunk 7 sornyi `console.log()` parancsot.

Így azért egyszerűbb, nem? :)

Feladat

Nyiss egy új bint a JS Binen, és írj egy kis visszaszámító programot. A program 10-től 0-ig számolja vissza a másodperceket, a hátramaradó időt a következőképpen kiírva a konzolba: „X másodperc a kilövésig.”

A programod a következő kritériumoknak feleljen meg:

- Szerepel benne egy `while` ciklus.
- Szerepel benne egy `seconds` nevű változó, amely a fennmaradó idő tárolja el.
- A `seconds` változót használja a ciklus kontrollálására.
- A fennmaradó időt minden iteráció során kiírja a konzolba.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3. A for ciklus

3.1. Ciklusok – A for ciklus (for loop)

Az előző részben megismerkedtél a ciklusok egyik nagy típusával, a `while` ciklussal. Most a társáról, a for ciklusról (`for` loop) lesz szó.

Ahogy látni fogod, a `for` ciklussal el tudsz végezni minden, amit a `while` ciklussal tudtál, plusz tud olyasmit is, amit az előző nem. Fejlesztőként gyakrabban fogsz `for` ciklusokkal találkozni, úgyhogy vágunk is bele az ismerkedésbe.

A for ciklus felépítése

A `while` ciklus után a `for` ciklust könnyű lesz megértened. Elméletben a következőképp néz ki:

```
for (var i = 0; i < 3; i = i + 1) {  
}
```

A `for` után megmondjuk, hogy a ciklus hogyan működik majd. Ami a `{ }` jelek közé kerül, az hajtódik végre többször.

Például:

```
for (var i = 0; i < 3; i = i + 1) {  
    alert('Hello!');  
}
```

Ez a ciklus párszor kiírja a felhasználónak a „Hello!” feliratot egy felugró ablakban.

A `for` utasítás után zárójelek között megadunk 3 paramétert:

1. Honnan indul a számláló

2. Mi a feltétel, amíg fut a ciklus
3. Mi történik a számlálóval minden lefutás után.

A fenti példában:

1. `i = 0` (a számláló neve i, 0-ról indul)
2. `i < 3` (addig futunk, amíg i kisebb, mint 3)
3. `i = i + 1` (minden ciklus lefutáskor i értékét növeljük eggyel)

A paramétereket ; (pontosvessző) választja el egymástól

Nézzünk egy másik példát:

```
for (var i = 0; i < 3; i = i + 1) {  
    alert(i);  
}
```

Ez esetben a ciklus a következőképp fut:

(minden ciklus lefutást „iteráció”-nak nevezünk egyébként)

1. iteráció: `i = 0`. Kisebb-e i, mint 3? Igen. Akkor még futhatunk `alert(i);` - kiírjuk i-t, azaz 0-t.

Ciklus vége, tehát `i = i + 1`, azaz i értéke mostantól nem `0`, hanem `0 + 1`, azaz `1`.

2. iteráció: Kisebb-e i, mint 3? Igen, mert i éppen `1`, akkor még futhatunk `alert(i);` - kiírjuk i-t, azaz 1-et.

Ciklus vége, tehát `i = i + 1`, azaz i értéke mostantól nem `1`, hanem `1 + 1`, azaz `2`.

3. iteráció: Kisebb-e i, mint 3? Igen, mert i éppen `2`, akkor még futhatunk `alert(i);` - kiírjuk i-t, azaz 2-t.

Ciklus vége, tehát `i = i + 1`, azaz i értéke mostantól nem `2`, hanem `2 + 1`, azaz `3`.

4. iteráció: Kisebb-e i, mint 3? Nem, mert i éppen `3`, tehát ciklus vége.

Tehát a fenti ciklus elszámolt nekünk 0-tól 2-ig.

Megjegyzés: Az alert() egy hasonló, beépített parancs a JavaScriptben, mint a console.log(). Ez egy felugró ablakot hoz elő, amelynek a tartalma a zárójelek közé írt rész lesz.

A for ciklus és a törpéink

Elsőre a `for` ciklus bonyolultabbnak tűnik, mint a `while`, mert más a logikája, mint amit emberként megszoktunk. Az információ nem a végrehajtás sorrendje szerint rendeződik, hanem típus szerint:

- a zárójelek közé jön minden olyan kód, ami a ciklust kontrollálja,
- A kapcsos zárójelek között pedig csak a futtatandó kódblokk marad.

De ha már tudod ezt a különbséget, akkor igazából a `for` ciklust könnyebb olvasni, mint a `while` ciklust. Az eredmény pedig mindenkoránál ugyanaz.

Az előző leckében megírt `while` ciklusunk így nézne ki `for` ciklusként:

```
for (var dwarfCounter = 1; dwarfCounter <= 7; dwarfCounter++) {  
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában.');//  
}
```

- Megadtuk a `dwarfCounter`-t, mint a számláló nevét és a kezdeti értékét (1).
- Utána megfogalmaztuk a feltételt.
- Utána megadjuk, hogy minden kör végén inkrementálja eggyel a változót.
- És végül a kódblokkba beírjuk a `console.log()` parancsot, ami kiírja a törpéinket.

Másold be egy új binbe a `for` ciklussal megírt verziót, és futtasd le. Ahogy látod, ugyanazt az eredményt adja, mint a `while` ciklussal megírt.

Megjegyzés: Az inicializáló kódot (a számláló nevét és értékét megadó részt) csak első alkalommal futtatja le a ciklus, minden további körben figyelmen kívül hagyja. Ha nem így lenne, akkor az minden egyes körben visszaállítaná a kontrolláló változót a kiindulási értékre.

Feladat

Nyiss egy új bint, és alakítsd át az előző feladatban írt visszaszámláló programodat egy `for` ciklussá. A programmal kiírt szöveg maradjon ugyanaz: „X másodperc a kilövésig.”

A programod a következő kritériumoknak feleljön meg:

- Egyetlen `for` ciklusból áll.
- Szerepel benne egy `seconds` nevű változó, amely a fennmaradó idő tárolja el.
- A `seconds` változót használja a ciklus kontrollálására.
- A fennmaradó időt minden iteráció során kiírja a konzolba.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.2. Ciklusok – A török, akik még alszanak és a ciklusok kombinálása

Nincs minden törpe a bányában, és nincsenek minden egy helyen. Jelenleg például Szundi, Hapci és Morgó (az 5-7. török) még otthon alszanak. Oldjuk meg, hogy ezt a kódunk ki tudja írni.

Egy lehetséges megoldás lenne, hogy leírunk hét sornyi `console.log()` parancsot, de

ennél már okosabbak vagyunk. Használunk inkább két ciklust a feladatra:

- egyet, ami kiírja a törpöket 1-től 4-ig és
- egy másikat, ami kiírja őket 5-től 7-ig.

Törpék 1-től 4-ig

Nyiss egy új bint, és írj egy `while` ciklust, amely kilistázza az első négy kis bányászt a következő szöveggel:

„A(z) X. törpe a bányában.”

Tipp: Továbbra is használd a `dwarfCounter` változónevet. Ha elakadnál, [itt egy kis segítség](#), de kérlek, ne másold ki. Próbáld önállóan megoldani. :)

Törpék 5-től 7-ig

Szuper! A következő törpöket egy `for` ciklussal fogjuk kiíratni. Másold be a következő kódot az előbb megírt while ciklusod alá:

```
for (dwarfCounter; dwarfCounter <= 7; dwarfCounter++) {  
    console.log('A(z) ' + dwarfCounter + '. törpe még otthon  
alszik.');
```

Futtasd le most az egész kódot. Ha minden jól csináltál, a konzol szépen kiírta, hogy ki dolgozik, és ki alszik éppen.

Ami fontos ebből a példából – a böngésző a kódodat felülről kezdi el „olvasni”. Amikor egy ciklushoz ér, megáll és addig futtatja a ciklust, amíg a kontroll állítás igaz, és csak utána lép tovább a következő kódsorra. Ugyanaz történik logikailag, mintha kiírnád kézzel az összes blokkot egymás alá, csak sokkal kevesebb energiába kerül így. :)

Feladat

Nyiss egy új bint a JS Binen. Írj egy kis programot, amely a képzeletbeli Skylark-völgy nyúlpopulációjának a változását követi egy éven keresztül.

- Az első hónap elején a kezdőpopuláció 30 nyusziból áll.
- Az első hat hónap során minden hónap végére megduplázódik a populáció, ezt követően pedig minden hónapban megháromszorozódik.

Írasd ki a konzolba ezeket a változásokat minden hónapra, a következő szöveggel: „A Skylark-völgy nyulainak száma X a(z) Y. hónap végén.”

A programod a következő kritériumoknak feleljén meg:

- Két `for` ciklusból áll.
- Szerepel benne egy `monthCounter` változó, amely a hónapok számát tárolja, valamint egy `rabbitPopulation` változó, amely a nyuszik számát tárolja.
- A `monthCounter` változó kontrollálja a ciklusokat.

- A program minden iteráció során írjon ki a konzolba egy üzenetet, amely megmondja, hogy az aktuális hónap végén mi a nyuszik száma.
- A program álljon le a 12. hónap végén.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.3. Ciklusok – Jövőbiztos ciklusok

Már egész szuper a kódunk, egy dologra viszont még nem készültünk föl vele – mi van, ha több törpe is érkezik még Hófehérkékhez? Vagy mi történik, ha nem az utolsó három törpe alszik otthon, hanem az utolsó négy?

Ezekben az esetekben újra kéne írnunk egy csomó sort a programunkban, mert konkrét számokkal oldottuk meg eddig a feladatot. Változtassunk ezen, és amit lehet, cseréljünk ki változókra, hogy könnyen fenntartható legyen a kódunk.

Itt tartunk most

Eddig ezt a kódot írtuk meg:

```
var dwarfCounter = 1;

while (dwarfCounter <= 4) {
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában van.');
    dwarfCounter++;
}

for (dwarfCounter; dwarfCounter <= 7; dwarfCounter++) {
    console.log('A(z) ' + dwarfCounter + '. törpe még otthon
alszik.');
}
```

Amit lehet, most cseréljünk le ebből változókra.

totalDwarves

Először is egyáltalán nem biztos, hogy nem érkeznek még török, ha egyszer elterjed a híre, hogy Hófehérke milyen jófej fönök. Készüljünk fel erre, és cseréljük le a fix 7-es számot egy **totalDwarves** (összesTörpe) változóra, amelynek a jelenlegi értéke 7.

```
var totalDwarves = 7;
var dwarfCounter = 1;
```

```
while (dwarfCounter <= 4) {  
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában van.');//  
    dwarfCounter++;  
}  
  
for (dwarfCounter; dwarfCounter <= totalDwarves; dwarfCounter++) {  
    console.log('A(z) ' + dwarfCounter + '. törpe még otthon  
alszik.');//  
}
```

Tipp: Figyeld meg, hogy a for ciklus kontroll állításában lecseréltük a `<= 7` kifejezést, `<= totalDwarves`-ra. Ezzel most már elég átírni a kód elején lévő változót, ha növekszik vagy csökken az összes törpék száma.

workingDwarves

Oldjuk meg azt is egy változóval, hogy ki az aki alszik, és ki az aki dolgozik:

```
var totalDwarves = 7;  
var workingDwarves = 4;  
var dwarfCounter = 1;  
  
while (dwarfCounter <= workingDwarves) {  
    console.log('A(z) ' + dwarfCounter + '. törpe a bányában van.');//  
    dwarfCounter++;  
}  
  
for (dwarfCounter; dwarfCounter <= totalDwarves; dwarfCounter++) {  
    console.log('A(z) ' + dwarfCounter + '. törpe még otthon  
alszik.');//  
}
```

Mi történt itt? A while ciklusunkban a `dwarfCounter <= 4` állítást lecseréltük `dwarfCounter <= workingDwarves`-ra. Mostantól a ciklus elején be tudjuk állítani, hogy hányas sorszámú törpék dolgoznak, és a program automatikusan alvóként listázza a többieket.

Tipp: Próbáld ki, hogy átírod a `workingDwarves` változó értékét `5`-re, a `totalDwarves` értékét pedig `10`-re. Nézd meg, mi történik.

Ezzel kész vagyunk a ciklusunk jövőbiztosá tételevel. Ha megfigyeled, most már sehol sem használunk konkrét számokat, csak változókat, így ha bármit át kell alakítanunk, sokkal egyszerűbb dolgunk lesz. :)

Feladat

Képzeld el, hogy egy városi buszjáratokat üzemeltető céget vezetsz. A vezető gépész

minden reggel megvizsgálja a garázsban álló autóbuszokat. Szeretnéd, ha egy alkalmazásban rögzítené a buszok összdarabszámát és az üzemképes buszok számát, hogy egy ehhez hasonló listát kapj:

```
A(z) 1. busz üzemképes, sofőrre vár.  
A(z) 2. busz üzemképes, sofőrre vár.  
A(z) 3. busz üzemképes, sofőrre vár.  
A(z) 4. busz üzemképes, sofőrre vár.  
A(z) 5. busz üzemképes, sofőrre vár.  
A(z) 6. busz üzemképes, sofőrre vár.  
A(z) 7. busz üzemképes, sofőrre vár.  
A(z) 8. busz üzemen kívül van, szervizre vár.  
A(z) 9. busz üzemen kívül van, szervizre vár.  
A(z) 10. busz üzemen kívül van, szervizre vár.
```

A te feladatod, hogy megírd ezt az applikációt.

A programod a következő kritériumoknak feleljen meg:

- Két `for` ciklust használ.
- Szerepel benne egy `totalBuses` változó, amely a buszok összdarabszámát tárolja.
- Szerepel benne egy `workingBuses` változó, amely az üzemképes buszok számát tárolja.
- Szerepel benne egy `busCounter` változó, amely az aktuális busz sorszámát tárolja.
- Az első ciklus az üzemképes buszokra vonatkozó üzeneteket írja ki, a második ciklus pedig az üzemen kívüli buszokra vonatkozó üzeneteket.
- Az aktuális busz sorszáma a `busCounter` változóból származzon.
- Úgy írd meg a programot, hogy az akkor is helyesen fussen le, ha megváltozik a buszok összdarabszáma és/vagy az üzemképes buszok darabszáma.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4. Összefoglalás

4.1. Összefoglalás

Ebben a fejezetben a `while` és a `for` ciklussal ismerkedtünk meg. A következőket sajátítottad el:

Pontosvessző és `console.log()`

- A JS-kódban (ha nem a konzolba írsz, hanem `.js` fájlba) pontosvesszővel kell elválasztani a sorokat egymástól (`;`)
- A konzolba a `console.log()` parancssal tudsz kiíratni eredményeket, például

```
console.log('Ez egy példa string.')
```

A while ciklus általános felépítése

```
while (amíg ez az állítás igaz) {  
    addig ez a kód fusson le  
}
```

A for ciklus általános felépítése

```
for (var i = 0; i < 3; i = i + 1) {  
    ez a kód fusson le  
}
```

A for ciklus logikája

A **for** után megmondjuk, hogy a ciklus hogyan működik majd. Ami a **{** és **}** jelek közé kerül, az hajtódiák végre többször.

Például:

```
for (var i = 0; i < 3; i = i + 1) {  
    alert('Hello!');  
}
```

Ez a ciklus párszor kiírja a felhasználónak a „Hello!” feliratot egy felugró ablakban.

A zárójelek között adjuk meg a ciklust kontrolláló paramétereket:

1. A ciklusszámláló kezdő értéke.
2. A feltétel, amely meghatározza, hogy meddig fusson a ciklus.
3. Mi történik a számlálóval minden lefutás után.

A fenti példában:

1. **i = 0** (a számláló neve **i**, 0-ról indul)
2. **i < 3** (addig futunk, amíg **i** kisebb, mint 3)
3. **i = i + 1** (minden ciklus lefutáskor **i** értékét növeljük eggyel)

A paramétereket **;** (pontosvessző) választja el egymástól.

JavaScript alapok V. (feltételek)

1. Bevezető a feltételekhez

1.1. Bevezetés – Feltételek a JavaScriptben

Az előző leckékben már szó esett arról, hogy fejlesztőként szeretünk minél világosabb, rövidebb és fenntarthatóbb kódot írni. Törekszünk arra, hogy a programunk minél többféle eshetőségre választ adjon, miközben a lehető legkevesebb sorból álljon. Minderre az egyik legjobb eszköz a feltételes állítások (conditional statements) használata a JavaScriptben.

A következő leckékben megtanuljuk, hogy miként lehet feltételes állításokat megfogalmazni egy gép számára, majd alkalmazzuk ezt a tudást a ciklusok felturbózására. Mindeközben pár megrendelést is teljesítünk egy online magazin, illetve egy szélerőműfarm számára.

Tipp: Ismét a JSBin JavaScript-paneljén fogunk dolgozni, ezért érdemes bezárnod minden a JavaScript- és Console-fülek kivételével. Az Output-panelen kiveheted az „Auto-run JS” melletti pipát is.

2. Egyszerű feltételek

2.1. Egyszerű feltételek 1. – Egy online magazin és az if / else állítások

Egy online magazin megrendelt tőlünk egy kis programot, amellyel azt tudják ellenőrizni, hogy a látogatóik megtekinthetik-e a korhatáros tartalmaikat. A kérésük az, hogy az általunk írt program:

- kérdezze meg, hány éves a látogató,
- ha 18 éves vagy idősebb, írja ki, hogy minden oké,
- ha pedig fiatalabb, írja ki, hogy sajnos még nem olvashatja el az adott cikket.

Az eddigi JavaScript-tanulmányaidból már sejtheted, hogy a feladat megoldásához szükségünk lesz változókra és összehasonlító operátorokra, plusz pár olyan dologra, amit most fogunk megtanulni:

- az `if` / `else` állításokra,
- a `prompt()` metódusra,
- valamint az `alert()` metódusra.

Tipp: Utóbbiról már volt ugyan szó, de most részletesebben is kitérünk rá.

Vágunk is bele!

2.2. Egyszerű feltételek 2. – if és a barátja, else

Az **if** angol szó azt jelenti magyarul, hogy „ha”, az **else** pedig ebben a kontextusban azt, hogy „egyébként”:

- ha (**if**) a megfogalmazott állításunk igaz, a böngésző lefuttatja az első kódblokkot,
- egyébként (**else**) pedig a második kódblokkot futtatja le a böngésző.

Kódban ez így néz ki:

```
if (A) {  
    X;  
} else {  
    Y;  
}
```

Ahol **A** egy feltétel, amely ha igaz, akkor lefut az **X** jelölésű kódblokk, ha pedig nem igaz, akkor lefut az **Y** jelölésű kódblokk.

Az **if / else** állítások (angolul **conditional statements**) lehetővé teszik azt, hogy a program megvizsgáljon egy adott helyzetet, majd az eredménynek megfelelő választ adjon.

Nézzünk erre egy rövid példát

```
var number1 = 4;  
var number2 = 15;  
  
if (number1 < number2) {  
    console.log(number1 + ' kisebb, mint ' + number2);  
} else {  
    console.log(number1 + ' nagyobb, mint ' + number2);  
}
```

Másold be a fenti kódot egy új bin JavaScript-paneljére, nyisd ki a Console-fület, és nyomd meg a „Run” gombot. Mi történik?

A konzol kiírta, hogy **"4 kisebb, mint 15"**. Való igaz. Próbáld meg felcserélni a két változó értékét a kód elején, aztán újra lefuttatni a programot. Most mi történt?

A konzol megváltoztatta az üzenetét, és kiírta, hogy **"15 nagyobb, mint 4"**. Ismét igaz. :)

Nézzük meg, hogy mi történik a kódban, lépésről lépéssre:

- Az első lépésben deklaráltunk két változót: **var number1 = 4** és **var number2 = 15**.

- Aztán jön az állításunk: ha `number1 < number 2`, akkor írja ki, hogy `number 1` kisebb, mint `number2`.
- minden egyéb esetben írja ki azt, hogy `number1` nagyobb, mint `number2`.

Gratulálok, megírtad az első `if / else` állításodat JavaScriptben!

Tipp: Próbáld meg azt, hogy egyenlő értéket adsz a két változónak, és úgy futtatód le a programot. Mi történt? A konzol kiírt valami ehhez hasonló butaságot: „**4 nagyobb, mint 4**”. Mégis miért? Nos, ha belegondolsz, ez az `if / else` állításunk szempontjából teljesen logikus. Az egyenlőség a program számára csak egy újabb „else” (minden egyéb) eset, ezért az ehhez kapcsolódó állítást írja ki. Csak két lehetséges állapotot határoztunk meg: `number1 < number2` és minden egyéb eset. Semmit sem mondtunk arról, hogy mi legyen, ha a két szám egyenlő, ezért a programunk furcsán, de logikusan működik. Ne aggódj, nemsokára tanulunk erre egy megoldást. :)

Feladat

Ideje nekiállni az online magazinnak szánt program megírásának: értesítsd a látogatókat, hogy megtekinthetik-e a tartalmat.

A programod a következő kritériumoknak feleljen meg:

- Mentsd el a látogató életkorát egy `age` nevű változóba. Adj neki egy általad választott értéket.
- Ha `age` kisebb mint 18, a konzol írja ki ezt: "Sajnáljuk, de ez a cikk csak 18 éven felüliek számára érhető el.".
- Ha `age` nagyobb vagy egyenlő mint 18, a konzol a következőt írja ki: "Köszönjük! Elolvashatod ezt a cikket.".

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.3. Egyszerű feltételek 3. – `prompt()`

Szuper, ha minden igaz, akkor kész vagy az `if / else` állítással, amely valahogy így néz ki:

```
var age = 15;

if (age < 18) {
    console.log('Sajnáljuk, de ez a cikk csak 18 éven felüliek
számára érhető el.');
} else {
    console.log('Köszönjük! Elolvashatod ezt a cikket.');
```

{}

Ez remek, de még messze vagyunk a kész terméktől. Ez a program egyelőre nem tud kommunikálni a felhasználóval, és az üzenetet a konzolba írja ki, amelyet senki sem nyit meg (a szemfüles webfejlesztőkön kívül). Meg kell tanítanunk a kis programunknak, hogy hogyan kérje el valakinek az életkorát, és miként írjon ki egy üzenetet úgy, hogy azt valóban lássák is. Itt jön be a `prompt()` és az `alert()` metódus.

`prompt()`

A `prompt()` a JavaScript egyik beépített metódusa. Meg tud jeleníteni egy üzenetet és egy szövegmezőt a felhasználó számára, akinek a válasza aztán értékként eltárolható.

Nézzük meg egy példán keresztül. Nyiss meg egy új bint, és másold be az alábbi kódot a JS-fülre:

```
prompt('Szia! Hogy hívnak?');
```

Nyomd meg a „Run” gombot.

A böngésződ feldobott egy kis ablakot, benne a **Szia! Hogy hívnak?** kérdéssel, egy szövegmezővel, valamint egy OK és egy Mégse gombbal. Írd be a neved, és nyomd meg az OK-t. Mi történt?

Semmi. Az ablak eltűnt, és kész. Ahhoz, hogy az itt megadott információt használni is tudjuk, szükségünk van egy változóra, amely tárolja a felhasználó által beírt választ.

Alakítsd át a fenti kódot erre:

```
var name = prompt('Szia! Hogy hívnak?');
```

Nyomd meg újra a „Run” gombot. Mi történt? Még mindig semmi, de ez valójában már nem semmi. Ugyan nem látható, de a böngésző a háttérben stringként eltárolta a választ a `name` változóban.

```
var name = prompt('Szia! Hogy hívnak?');
console.log(name);
```

Ha most lefuttatod a kódot, és megválaszolod a `prompt()` ablakban megjelenő kérdést, a böngésző kiírja a nevedet a konzolba. Menő, nem? :)

Tipp: Ha a felhasználó nem ír semmit a szövegmezőbe, vagy megnyomja a Mégse gombot, akkor a `prompt()` metódus a `null` értéket adja vissza. Az `undefined` és a `null` közötti különbségre még visszatérünk a későbbi leckékben.

Feladat

Módosítsd az online magazinnak szánt programodat: kérd el a látogatók életkorát, és a válaszuknak megfelelően értesítsd őket.

A programod a következő kritériumoknak feleljön meg:

- Mentsd el a látogató életkorát egy `age` nevű változóba. Az értékét a `prompt()` metódussal add meg, a következő kérdéssel elkérve a látogató életkorát: **Hány éves vagy?**.
- Ha `age` kisebb mint 18, a konzol írja ki ezt: "**Sajnáljuk, de ez a cikk csak 18 éven felüliek számára érhető el.**".
- Ha `age` nagyobb vagy egyenlő mint 18, a konzol a következőt írja ki: "**Köszönjük! Elolvashatod ezt a cikket.**".

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.4. Egyszerű feltételek 4. – alert()

Az `alert()` beépített metódust röviden már érintettük korábban, de most foglalkozzunk vele egy pici behatóbban is.

Ez a metódus megjelenít egy üzenetet a felhasználó számára egy kis felugró ablakban. Nézzük meg ezt is egy példán keresztül.

Másold be az alábbi kódot egy új binbe, és futtasd le:

```
alert('Helló-belló! :)');
```

Ha minden igaz, felugrott egy ablak, benne a kis üdvözlettel.

Ahogy a `console.log()`, az `alert()` is mindenféle értéket képes befogadni a zárójelek között: számot, stringet, változót, valamint ezek kombinációját is.

Annak érdekében, hogy ezt jobban megértsük, módosítsuk egy pici az előző leckében írt kódunkat. Ha még emlékszel, ez egy olyan kis program volt, amely megkérdezte a nevedet:

```
var name = prompt('Szia! Hogy hívnak?');
console.log(name);
```

Alakítsuk át ezt a következőre:

```
var name = prompt('Szia! Hogy hívnak?');
alert('Örülök, hogy találkoztunk, ' + name + '! Én vagyok a
böngésződ.');
```

Futtasd le. Milyen kis udvarias a böngésződ, nem? :)

Feladat

Módosítsd az online magazinnak szánt programodat: cseréld le a `console.log()` metódusokat `alert()` metódusokra.

A programod a következő kritériumoknak feleljen meg:

- Mentsd el a látogató életkorát egy `age` nevű változóba. Az értékét `prompt()` metódussal add meg, a következő kérdéssel elkérve a látogató életkorát: **Hány éves vagy?**.
- Ha `age` kisebb mint 18, egy felugró ablakban jelenjen meg ez az üzenet: **"Sajnáljuk, de ez a cikk csak 18 éven felüliek számára érhető el."**.
- Ha `age` nagyobb vagy egyenlő mint 18, egy felugró ablakban jelenjen meg az alábbi üzenet: **"Köszönjük! Elolvashatod ezt a cikket."**.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.5. Egyszerű feltételek 5. – else if állítások és egy kiegészítő kérés
Ha minden jól ment, akkor kész is vagy az eredeti megrendeléssel. Írtál egy programot, amely:

- megkérdezi a látogató életkorát, és
- a megadott információ alapján megjeleníti a megfelelő választ.

A kódod valami ilyesmi:

```
var age = prompt('Hány éves vagy?');

if (age < 18) {
    alert('Sajnáljuk, de ez a cikk csak 18 éven felüliek számára
érhető el.');
} else {
    alert('Köszönjük! Elolvashatod ezt a cikket.');
}
```

Azonban pont mielőtt átadnád a megrendelőnek a befejezett kódodat, jön egy kiegészítő kérés a magazintól. Szeretnék, ha az éppen 18 éves olvasóknak kedveskedne a program egy vicces kis üzenettel: **Pont 18? Gratulálunk, üdv a felnőttek között! Jó szórakozást a cikkhez!** Ne aggódj, sokszor fog még érkezni pontosítás a megrendelésekhez a munkád vélt elvégzése után.

Ideje visszaülnünk a tervezőasztalhoz.

Amikor két feltétel nem elég – az else if állítás

A helyzetünk a következő:

- a programunk tudja kezelni a 18 éven aluliakat
- és a 18 éven felülieket,
- de szükségünk van egy harmadik állapotra, amely a pont 18 évesekre vonatkozik.

Itt jön be az **else if** állítás, amely a következő logikára épül:

```
if (A) {  
    X;  
} else if (B) {  
    Y;  
} else {  
    Z;  
}
```

Ez a kód a következőt mondja a böngészőnek: ha A állítás igaz, akkor futtasd le X-et. Ha nem igaz A, nézd meg, hogy igaz-e B, és ha igen, akkor futtasd le Y-t. minden egyéb esetben futtasd le Z-t.

Az **else if** állítás lehetővé teszi, hogy kettőnél több feltételes állapotot is definiálunk.

Tipp: Akárhány **else if** ágat létrehozhatsz, nem csak egyet. Ezeken a böngésző sorban fog végigmenni.

Hogy lássuk ezt a gyakorlatban, térjünk vissza a korábbi leckékben írt „Melyik szám a nagyobb?” programunkra. Ha még emlékszel, így nézett ki:

```
var number1 = 4;  
var number2 = 15;  
  
if (number1 < number2) {  
    console.log(number1 + ' kisebb, mint ' + number2);  
} else {  
    console.log(number1 + ' nagyobb, mint ' + number2);
```

```
}
```

Mint láttuk, ez a kód csak addig működik jól, amíg a két változó értéke nem egyenlő. Amennyiben ugyanis egyenlők, a konzol butaságot ír ki. Javítsuk ezt meg egy `else if` állítással:

```
ar number1 = 4;
var number2 = 15;

if (number1 < number2) {
    console.log(number1 + ' kisebb, mint ' + number2);
} else if (number1 == number2) {
    console.log(number1 + ' és ' + number2 + ' egyenlők.');
} else {
    console.log(number1 + ' nagyobb, mint ' + number2);
}
```

Most próbáld ki azt, hogy megadod ugyanazt az értéket minden két változónak. Mi történt?

A konzol a helyes üzenetet írja ki, mert az `else if` ág lépett életbe.

Tipp: Talán észrevettek, hogy a két szám összehasonlításakor nem sima egyenlőségjelet (`=`) használtunk, hanem duplát (`==`). Ez fontos, mert még a szimpla egyenlőségjellel új értéket adhatunk egy változónak, a dupla egyenlőségjellel azt ellenőrizhetjük, hogy két érték egyenlő-e egymással (equality). A későbbiekben találkozol majd három egyenlőségjellel is (`====`), mely a teljes azonosságot (identity) vizsgálja.

Alakítsd át ismét az online magazinnak szánt programodat, hogy a megfelelő üzenetet írja ki a 18 évesek számára.

2.6. Egyszerű feltételek 6. – Egy kész megrendelés

Gratulálok, elkészültél az első, egyszerű feltételekre épülő JavaScript-programoddal. Szép munka!

Egy példamegoldás

```
var age = prompt('Hány éves vagy?');

if (age < 18) {
    alert('Sajnáljuk, de ez a cikk csak 18 éven felüliek számára
érhető el.');
} else if (age == 18) {
    alert('Pont 18? Gratulálunk, üdv a felnőttek között! Jó
```

```
szórakozást a cikkhez!');");
} else {
    alert('Köszönjük! Elolvasható ez a cikket.');
}
```

A következő fejezetben kombinálni fogjuk az egyszerű feltételeket a ciklusokkal, és egy szélerőmű farmnak építünk egy irányító programot. Készen állsz? Akkor hajrá!

3. Feltételek és ciklusok

3.1. Feltételek és ciklusok 1. – Bevezetés

A programozásban az egyik legfontosabb és legelterjedtebb jó gyakorlat úgynevezett **DRY kódot írni**. A DRY betűszó a Don't Repeat Yourself (ne ismételd magad) kifejezést takarja, és azt jelenti, hogy egy probléma megoldásakor minden próbáld meg a lehető legrövidebb kódot megírni, a lehető legkevesebb ismétlődéssel. Így átláthatóbb, olvashatóbb és könnyebben karbantartható lesz a programod.

A DRY kód írásának egyik nagyszerű eszköze a **feltételes állítások** és a **ciklusok** kombinálása.

Gyors emlékeztető a ciklusokról

A **while** ciklus általános felépítése:

```
while (amíg ez az állítás igaz) {
    addig ez a kód fusson le
}
```

A **for** ciklus általános felépítése:

```
for (var i = 0; i < 3; i = i + 1) {
    ez a kód fusson le
}
```

A zárójelek között adjuk meg a ciklust kontrolláló paramétereket:

1. A ciklusszámláló kezdő értéke.
2. A feltétel, amely meghatározza, hogy meddig fusson a ciklus.
3. Mi történik a számlálóval minden lefutás után.

A kapcsos zárójelek (**{** és **}**) közé kerülő kód újra és újra végre lesz hajtva.

Az eddigi tudásunkkal minden ciklusunk csak egyféle feladatot tudott végrehajtani. A következőkben megnézzük, hogy a feltételes állítások segítségével hogyan lehet egy

ciklussal több feladatot is megoldani.

Miért tanácsos DRY kódot írnod fejlesztőként?

3.2. Feltételek és ciklusok 2. – for, if / else és a páros-páratlan számok

Nézzük meg a ciklusokkal kombinált feltételek képességeit egy példán keresztül.

Azt szeretnénk, ha lenne egy **for** ciklusunk, amely kilistázza a számokat 1-től 20-ig a konzolba, és eldönti róluk, hogy párosak-e vagy páratlanok. Lássunk neki!

A kezdeti for ciklusunk

Kezdjük az alapoknál: első körben csak írunk egy **for** ciklust, amely elszámol 20-ig, és kinyomtatja a számokat a konzolba.

```
for (var i = 1; i <= 20; i++) {  
    console.log(i + ' egy szám.');//  
}
```

Másold be ezt a kódot egy új binbe, nyisd ki a Console-fület, és futtasd le a kódot a Run gombra kattintva. A program kinyomtatja a számokat, ahogy azt vártuk. Eddig szuper.

Páros és páratlan számok

Az biztos, hogy szükségünk lesz egy **if** / **else** állításra, amely megmondja a böngészőnek, hogy a páros vagy a páratlan számokhoz tartozó mondatot nyomtassa ki. Előbb azonban meg kell tanítanunk a programunkat különbséget tenni a páros és páratlan számok között.

Emlékszel még a modulus operátorra?

Az első JavaScript-modulban tanultál a különböző számtani műveletekről, köztük a modulusról is, amely a maradékot adja vissza egy osztás után.

Két példával:

- **6 % 2 = 0** (mivel 6-ot 2-vel osztva 0 a maradék)
- **5 % 2 = 1** (mivel 5-öt 2-vel osztva 1 a maradék)

Ebből láthatod, hogy a modulus segítségével különbséget lehet tenni páros és páratlan számok között. Egy páros számot 2-vel osztva a maradék 0 lesz, a páratlan számok esetében pedig 1.

Az if / else állításunk és a ciklus összerakása

Most már nincs akadálya, hogy megírjuk a véleges programunkat.

Először kezeljük a páratlan számokat:

```
for (var i = 1; i <= 20; i++) {  
    if (i % 2 != 0) {  
        console.log(i + ' egy páratlan szám.');//  
    }  
}
```

Az **if** állításunk tehát a következő: ha **i**-t 2-vel osztva a maradék nem 0, akkor **i** egy páratlan szám. Igaz? Igaz. :)

Adjuk hozzá a páros számokat kezelő **else** részt:

```
for (var i = 1; i <= 20; i++) {  
    if (i % 2 != 0) {  
        console.log(i + ' egy páratlan szám.');//  
    } else {  
        console.log(i + ' egy páros szám.');//  
    }  
}
```

Másold be a fenti kódot egy új binbe, és futtasd le. Ha minden igaz, a konzol kiírja a számokat a helyes mondatokkal.

Ezzel megírtad az első feltételekkel kiegészített **for** ciklusodat. Szép munka!

A 10 a legszebb páros szám – egy **else if** ág

A 10 egy különösen szép szám, nem gondolod? Alakítsd át a kódodat úgy, hogy amikor a ciklus eléri a 10-et, akkor a következő mondatot írja ki a konzolba: **A 10 a legszebb páros szám..**

Tipp: Ehhez egy **else if** ágra lesz szükséged. Ha nagyon elakadnál, [itt egy példakód](#), de kérlek, először próbáld meg önállóan megoldani. Abból sokkal többet lehet tanulni.

Feladat

Írj egy programot, amely a konzolba kiírja a számokat 1-től 100-ig. Ha a szám osztható 3-mal, írja mellé, hogy „kutya”. Ha a szám osztható 5-tel, írja mellé, hogy „cica”. Ha szám osztható 3-mal és 5-tel is, írja mellé, hogy „egér”.

A programod a következő kritériumoknak feleljen meg:

- Egy **for** ciklusból álljon.
- A cikluson belül használj **if**, **else if** és **else** ágakat.
- A feltételes állításokon belül használd a **console.log()**-ot.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.3. Feltételek és ciklusok 3. – A szélerőmű-megrendelés

Most, hogy megismerkedtél a ciklusok és a feltételes állítások alapjaival, ideje egy kis önálló munkának.

A megrendelés

Egy szélerőműfarm felkért rá, hogy építs nekik egy kontrolláló programot, amely a következőket tudja:

- Adjon egy állapotjelentést minden egyes turbináról, és számítsa ki az adott turbina bekapcsolásának a pillanatában összesen termelt energiamennyiséget (pl. a 3. turbinánál számítsa ki az első 3 turbina által termelt összenergiát).
- Összesen 25 szélturbinájuk van.
- A mai napon az első 10 turbina teljes fordulaton üzemel, egyenként 2000 MWh-t termelve. A következő 10 turbina fél fordulaton üzemel, egyenként 1000 MWh-t termelve. Az utolsó 5 turbina jelenleg nem működik.

A programra vonatkozó kérések:

- Listázza ki az első 10 turbinát a konzolba, a következő üzenettel: "A(z) X. számú szélturbina teljes fordulaton működik, 2000 MWh-t termelve. A farmon termelt összenergia jelenleg Y MWh."
- Listázza ki a következő 10 turbinát a konzolba, a következő üzenettel: "A(z) X. számú szélturbina fél fordulaton működik, 1000 MWh-t termelve. A farmon termelt összenergia jelenleg Y MWh."
- Listázza ki az utolsó 5 turbinát a konzolba, a következő üzenettel: "A(z) X. számú szélturbina nem működik, 0 MWh-t termelve. A farmon termelt összenergia jelenleg Y MWh."
- A feladat megoldásához egyetlen `for` ciklust használj, és az üzenetekben változókat használj számok helyett.

Tippek

- Valószínűleg szükséged lesz egy `allTurbines`, egy `turbinePower`, egy `sumPower` és egy `turbineCounter` változóra a programban.
- Ne feledd a feltételes állításokat: `if`, `else if` és `else`.

Ez egy kihívást jelentő feladat, de ha elvégezted az eddigi JavaScript-leckéket, akkor minden szükséges tudással rendelkezel a megoldásához. Adj magadnak időt, próbálkozz, és ha kell, nézd át a korábbi leckéket. Menni fog. :)

Ne feledd, a Google remek barát, ha kódolás közben elakadsz.

Hajrá! :)

Tipp: +1 tipp: Ha végképp nem boldogults, akkor a következő oldalon találsz egy lehetséges megoldást.

3.4. Feltételek és ciklusok 4. – Egy lehetséges megoldás

Példakód

```
var allTurbines = 25;
var turbineCounter = 0;
var turbinePower = 0;
var sumPower = 0;

for (var i = 0; i < allTurbines; i++) {
    if (i < 10) {
        turbineCounter++;
        turbinePower = 2000;
        sumPower += turbinePower;
        console.log('A(z) ' + turbineCounter + '. számú szélturbina teljes fordulaton működik, ' + turbinePower + ' MWh-t termelve. A farmon termelt összenergia jelenleg ' + sumPower + ' MWh.');
    } else if (i < 20) {
        turbineCounter++;
        turbinePower = 1000;
        sumPower += turbinePower;
        console.log('A(z) ' + turbineCounter + '. számú szélturbina fél fordulaton működik, ' + turbinePower + ' MWh-t termelve. A farmon termelt összenergia jelenleg ' + sumPower + ' MWh.');
    } else if (i < 25) {
        turbineCounter++;
        turbinePower = 0;
        sumPower += turbinePower;
        console.log('A(z) ' + turbineCounter + '. számú szélturbina nem működik, ' + turbinePower + ' MWh-t termelve. A farmon termelt összenergia jelenleg ' + sumPower + ' MWh.');
    } else {
        console.log('Valami váratlan történt.');
    }
}
```

Megjegyzések

- A legutolsó **else** ágat gyakran alkalmazzuk hibakezelésre.

- A változókat deklarálhatsz a cikluson kívül és belül is. Utóbbi esetben minden egyes lefutásnál újra dekláralja őket a program.
- A ciklusszámláló (**i**) kapcsán ne felejtsd, hogy a programozás nulla alapú, azaz 0-tól számolunk, nem 1-től.
- A **variableName++** egyet hozzáad a változó értékéhez, a **variableName1 += variableName2** pedig ugyanazt teszi, mint a **variableName1 = variableName1 + variableName2**. Mindkét technika segít a DRY kód írásában.

4. Összefoglalás

4.1. Feltételek (conditionals) – Összefoglalás

Az if / else állítások logikája

```
if (A) {  
    X;  
} else {  
    Y;  
}
```

Ha az **A**-val jelölt állítás igaz, akkor lefut az **X** kód blokk. Ha nem igaz, akkor az **Y** kód blokk fut le helyette.

Az else if állítás

```
if (A) {  
    X;  
} else if (B) {  
    Y;  
} else {  
    Z;  
}
```

Ez a kód a következőt mondja a böngészőnek: ha **A** állítás igaz, akkor futtasd le **X**-et. Ha nem igaz **A**, nézd meg, hogy igaz-e **B**, és ha igen, akkor futtasd le **Y**-t. minden egyéb esetben futtasd le **Z**-t.

Ciklusok és feltételek

A feltételes állításokat kombinálhatjuk a ciklusokkal, így egy ciklus képes ellátni több különböző feladatot.

Egy példa:

```
for (var i = 1; i <= 20; i++) {
```

```
if (i % 2 != 0) {  
    console.log(i + ' egy páratlan szám.');//  
} else {  
    console.log(i + ' egy páros szám.');//  
}  
}
```

`prompt()`

A `prompt()` a JavaScript egyik beépített metódusa. Meg tud jeleníteni egy felugró ablakot, ahova a felhasználó beírhat valamilyen információt, amely aztán elmenthető egy változóba.

`alert()`

Ez a beépített JavaScript-metódus megjelenít egy felugró ablakot a felhasználó számára.

5. Teszt

5.1. Ellenőrizd a tudásod!

[13 kérdés, körülbelül 13 perc, kérdésenként 1 pont.](#)

Tipp: Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a „Küldés” (Submit) gombot a teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

Megjegyzés: A tesztet többször is kitöltheted.

Sok sikert!

Programozóként gondolkodni

1. A Marson rekedve

1.1. Bevezetés

Szia! Üdvözünk a „Programozóként gondolkodni” modulban!

Ebben a projektben megtanulhatod, hogyan közelítik meg a komplex feladatokat a programozók. Mindezt egy Mars-expedíció keresztül fogjuk szemléltetni, ahol te leszel az egyetlen programozó a legénység tagjai közt, és egy egyszerű JavaScript program segítségével kell majd magadat és a társaidat kihúznod a csávából.

Ezalatt első kézből szerezhetsz tapasztalatot a standard fejlesztési ciklusról, és egy csomó szoftverfejlesztéssel kapcsolatos legjobb gyakorlatról is szó esik majd. A projekt végére minden eszközöd meg lesz ahhoz, hogy bármilyen új projektet programozói szemlélettel közelíts meg.

Remélem, te is annyira izgatott vagy, mint mi! Vedd elő a képzelőrődet, és vágunk is bele a vörös bolygón játszódó történetünkbe!

1.2. A három űrhajós dilemmája

2070-et írunk.

Földünk energiatartalékai már szinte teljesen kiadtak a 20. és a korai 21. század kizsákmányolásának következtében.

Egy tudósokból és mérnökökből álló csapat a maradék erőforrásokból egy szerény űrhajót épített, amivel három merész űrhajós – Ádám, Anna és *te* – repülhet el a Marsra, alternatív energiaforrások után kutatva.

Az űrsikló gond nélkül hagyta el a Földet, de mivel a landolás már sajnos nem ment ennyire simán, az egyik hajtómű komolyabban megsérült aközben. Ennek eredményeként, az újonnan megszerzett energiaforrással együtt az űrhajó már túl nehéz ahhoz, hogy a tönkrement hajtómű nélkül is fel tudjon újból szállni.

Egy kis fejtörés után Ádám azt javasolta, hogy az üzemanyag egy részét hagyjátok hátra, hogy az űrhajó így könnyebb legyen. Ez elméletben működhett is, de Anna észrevette, hogy a landolás során az üzemanyagszint-mérő is meghibásodott, és így nem lehet tudni, hogy mennyi üzemanyagtól szabadulhattak meg úgy, hogy mindenkorban vissza tudjatok jutni a Földre.

Mivel te vagy az egyetlen programozó a Marson, Ádám és Anna tőled várják a megoldást.

Szerencsére mint minden, most is számíthatsz a számítógéped és a józan eszed segítségére. És nem is kell ennél több, hogy mindhárman biztonságban hazajussatok!

1.3. Problémamegoldás programozó módra

Bármilyen problémával is találod magad szemben, a lényeg, hogy azt bontsd kisebb elemekre, és így több kisebb problémát kell csak megoldanod, külön-külön.

Alapvetően minden programozó így közelíti meg a problémákat a munkája során, és mi is ezt fogjuk tenni a küldetésünk teljesítéséhez.

Vegyük példának a jelenlegi helyzetet: **te és az űrhajós társaid a Marson rekedtetek, és szeretnétek valahogy visszajutni a Földre.**

Ez egy összetett probléma.

De most nézzük meg ugyanezt a problémát kisebb elemekre bontva, néhány logikus kérdés segítségével:

1. **Miért rekedtetek a Marson?** → A plusz súly és a tönkrement hajtómű miatt az űrsikló túl nehéz ahhoz, hogy fölszálljon.
2. **Hogyan tudnátok ezt a problémát megoldani?** → Megszabadulhattak a fölösleges üzemanyagtól.
3. **Mennyi üzemanyagot kell ehhez leereszteni?** → Még nem tudjuk, kellene egy számológép, amivel ki tudjátok ezt számolni.
4. **Mire van szükséged ahoz, hogy létrehozz egy ilyen számológépet?** →
 - a. Némi matekra.
 - b. Aztán ebből a matekból egy kis JavaScript-programot kell majd építeni.

Látod? Egy pár egyszerű kérdés megválaszolásával egyetlen hatalmas probléma helyett már két kisebb, kézzelfoghatóbb problémával kell csak megküzdened:

- Kitalálni a mateket.
- Felépíteni a megfelelő JavaScript számológépet a matek alapján.

A következő lépésekben az első kisebb problémát fogjuk megoldani, méghozzá ugyanezzel a lebontásos technikával.

Megjegyzés: Most gyors egymásutánban kétszer is láttad a **matek** szót. Végy egy nagy levegőt, és ne menekülj ki a szobából hanyatt-homlok attól rettegve, hogy ezen fogsz elhasalni. Hidd el nekem, hogy gond nélkül meg fogsz tudni birkózni a feladattal.

1.4. A szükséges üzemanyag-mennyiségek megállapítása

Nézzük tehát az első problémánkat: **a hazajutáshoz szükséges mateket.**

A következők adottak:

- Az űrhajó átlagos fogysztása, ami 100 kilométerenként X liter üzemanyag.
- És természetesen a Föld is adott távolságra van. Ez legyen egyelőre Y kilométer.

Megjegyzés: Most még nem tudod ezeknek a változóknak a pontos értékét – ezeket Ádám és Anna fogja megadni, amikor majd használják a számológépet. Ez ne zavarjon téged, változókkal pont ugyanolyan jól fel tudjuk írni a megfelelő egyenletet.

- Adott, hogy X liter üzemanyaggal 100 km távolságot tudtok megtenni.
- Az is adott, hogy a Föld jelenleg Y kilométerre van tőletek.
- És Z értékét szeretnénk kiszámolni, ami pedig a szükséges üzemanyag mennyisége.

Z értékét egy egyszerű arányszámítással kaphatjuk meg.

Ha: 100 km-hez X liter üzemanyag szükséges
Továbbá: Y km-hez Z liter üzemanyag szükséges
Akkor: $Z = Y * (X / 100)$

Tehát:

Szükséges üzemanyag mennyisége = Földtől lévő távolság * Átlagos fogyasztás

Meg is vagyunk a feladat első felével – már tudjuk, hogyan lehet kiszámolni az úthoz szükséges üzemanyag mennyiségét. Ennek ismeretében már nem lesz nehéz dolgunk kitalálni, hogy mennyi üzemanyagtól kell megszabadulnunk.

Tipp: Bátran írd le ezt a kis egyenletet egy darab papírra. A tervezési fázisban bevett szokás kézzel írt jegyzeteket készíteni a főbb gondolatokról és ötletekről, így biztos nem felejtész el semmit.

1.5. Annak kiszámítása, hogy mennyi üzemanyagot adhatunk le

Most már megvan a képletünk, amivel kiszámíthatjuk, hogy mennyi üzemanyagra van szükség a hazatéréshez.

Következő lépésként számoljuk ki, hogy mennyi üzemanyagtól szabadulhatunk meg ahhoz, hogy épp a megfelelő mennyiség maradjon az üzemanyagtartályban.

- Annától megtudhattuk, hogy az űrsikló üzemanyagtartálya jelenleg teljesen tele van.
- Tehát ennél több üzemanyag már nem is férne bele.

Tehát:

A lecsapolandó üzemanyag mennyisége = Üzemanyagtartály mérete - A hazatéréshez szükséges üzemanyag mennyisége

És ezzel meg is volnánk a matekkal!

Még mindig nem kell a pontos értékek miatt aggódnod. Azokat majd a többi űrhajós adja meg, amikor a fenti logika alapján felépített számológépet használják.

Apropó számológép: ideje leprogramozni azt is.

Tipp: Itt is nyugodtan ragadj papírt és ceruzát, és írd le magadnak az egyenletet. Jól fog jönni ez a jegyzet, amikor majd a programon dolgozol.

2. A munkakörnyezet felállítása

2.1. A fájlok létrehozása

Most már megvan a logikai váz, ideje hát, hogy ezt egy tényleges programmá változtassuk. minden projekt legelső lépése az, hogy kialakítjuk a szükséges munkakörnyezetünket.

Nyisd meg a kedvenc asztali kód szerkesztődet, és hozz létre egy **get-home** (hazajutás) nevű projektmappát.

Megjegyzés: Ne feledd, hogy a Marson nincs internet. Csak asztali kód szerkesztőben dolgozhatsz.

Két fájlra lesz szükséged:

- Egy **index.html** fájlra, hogy a böngészőben tudd kezelní a JavaScript programodat.
- Egy **app.js** fájlra, amibe a JavaScript programot írhatod.

A legelső teendőd: hozd létre ezt a két fájlt a **get-home** mappán belül.

Megjegyzés: Igazából bármilyen fájlnevet adhatnál nekik, de az **index.html** és az **app.js** elnevezések a legelterjedtebbek a programozás világában. Ez pedig egy remek alkalom arra, hogy begyakorold ezeket a szokásokat.

2.2. A JavaScript és HTML fájl összekapcsolása

Remek, most már megvannak a fájlaid. Következő lépésként kapcsold őket össze.

Miért? Mert a böngészők csak HTML fájlokat tudnak megjeleníteni. Egy csupasz JS fájlt nem küldhetsz el a felhasználóknak. Egy HTML fájlt kell elküldened, amely tartalmazza a JS-t, amit le akarsz futtatni.

Ehhez a következőket kell tenned:

1. Adj hozzá egy standard HTML boilerplate-et (sablont) az **index.html** fájlhoz, majd
2. linkeld be az **app.js** fájlt.

A HTML boilerplate hozzáadása a fájlhoz

Másold a következő kódrészletet az `index.html` fájlodba:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hazajutós számológép</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1">
  </head>
  <body>
    <h1>Ez az index.html fájl</h1>
  </body>
</html>
```

Mentsd el a fájlt.

Megjegyzés: A `h1` tag használata nem kötelező, de hasznos. Egyfajta beépített tesztként működik, amikor megnyitod a fájlt a böngészőben. Ha megjelenik a képernyőn, minden rendben. Ha nem jelenik meg, valami félrecksúszott.

A JavaScript fájl belinkelése

Ehhez a HTML kódban a `script` tagre lesz szükséged. A `script` a `link` taghez hasonlóan működik, de még utóbbival CSS stíluslapot tudunk hozzáadni a kódhoz, az előbbi arra való, hogy JavaScriptet adjunk hozzá a HTML fájlhoz.

Add hozzá a kódhoz a következő sort közvetlenül a `</head>` tag előtt:

```
<script src="app.js"></script>
```

És kész is vagyunk. Mentsd el a HTML-t, és menj tovább. A következő feladatban le fogjuk ellenőrizni, hogy sikerült-e összekapcsolni a HTML fájlt a JavaScripttel.

Megjegyzés: A `script` tagek legtöbbször a `head` elembe kerülnek, mert így biztos, hogy a JavaScript még azelőtt betölt, mielőtt a böngésző betölte a HTML többi részét. Van néhány kivétel ez alól a szabály alól, de ezek most nem fontosak. Lesz még szó róluk később.

2.3. A kapcsolat ellenőrzése

Lássuk, hogy sikerült-e megfelelően összekapcsolnod a HTML és a JavaScript fájlt.

Ehhez először is szükséged lesz valamire az `app.js` fájlban, amit tesztelni tudsz. Például egy `console.log()` parancsra. Ha az `index.html` fájl betöltésekor a böngésző megjelenít egy üzenetet a konzolban, tudni fogod, hogy az összekapcsolás sikeres.

Nyisd meg az `app.js` fájlt, és add hozzá ezt a sort:

```
console.log('A számológép használatra kész.');
```

Mentsd el a fájlt, és nyisd meg az `index.html`-t a böngészőben.

Megjegyzés: Ha van a szerkesztődben, használhatod a Live Preview funkciót is, de ne feledd, hogy ez csak azután fogja megjeleníteni a változtatásaidat, hogy elmentetted a `.js` fájlt, amelyen éppen dolgozol. Ne várj arra, hogy ugyanolyan gördülékenyen, élőben jelenítse meg őket, mint a HTML vagy a CSS fájlok esetében.

A böngészőben

Ha minden jó ment, csak az „Ez az index.html fájl” címsor fog megjelenni a képernyőn.

Nyisd meg a konzolt.

Megjegyzés: Ezt Chrome-ban a `Ctrl + Shift + J` billentyűparancssal teheted meg Windows és Linux esetében, illetve a `Cmd + Opt + J` kombinációval Macnél. Ha másmilyen böngészőt használsz, magadtól kell rájönnöd, hogy hogyan kell megnyitni a konzolt.

És itt is van: a vidám kis üzenetünk, amit az `app.js` fájlunk jelenített meg. Most már a kódot is tudod hova írni, és a változtatásokat is látni fogod.

3. Problémamegoldás lépésről lépésre

3.1. Gondolkodj, mielőtt kódolsz

Király, így már minden előkészítettünk a programozáshoz!

A problémát úgy fogjuk megközelíteni, mint minden rendes programozó:

1. Először is végiggondoljuk az előttünk álló feladatot.
2. Készítünk egy vázlatot a szerkesztőn belül.
3. Végül a vázlatba írt jegyzeteket felülírjuk tényleges kóddal.

Készen állsz? Akkor vágunk bele!

3.2. Az első megjegyzések

A végső célunk az, hogy kitaláljuk, mennyi üzemanyagtól kell megszabadulnunk ahhoz, hogy a súlylimit alá kerüljünk, de mégis elég üzemanyagunk legyen a hazaútra. Nevezzük ezt a változót `fuelToOffload`-nak, azaz leeresztendő üzemanyagnak.

Ahogy arról már volt szó, a `fuelToOffload` változót úgy kapjuk meg, hogy a maximális

üzemanyag-mennyiségből kivonjuk azt az üzemanyag-mennyiséget, amennyi feltétlenül szükséges a hazajutáshoz. Mindez a változónevekkel írható le:

```
fuelToOffload = tankCapacity - requiredFuel
```

Nyisd meg az `app.js` fájlt, és add hozzá a következő sort megjegyzésként, közvetlenül a `console.log()` utasítás alatt:

```
console.log('A számológép használatra kész.');// fuelToOffload = tankCapacity - requiredFuel
```

Megjegyzés: Ne feledd: a JavaScriptben az egysoros megjegyzéseket két törtvonallal (`//`) jelöljük a sor elején.

Szuper. Adjunk is hozzá gyorsan még pár megjegyzést, amelyek segíteni fognak felvázolni a logikát, amire a programunk épül majd:

```
console.log('A számológép használatra kész.');// Amiket ki kell számolnom:// fuelToOffload:// Az az üzemanyagmennyiség, amit leeresztve a súlylimit alákerülünk, de még haza tudunk jutni.// fuelToOffload = tankCapacity - requiredFuel// tankCapacity: Az ūrsikló üzemanyagtartályának ūrtartalma.// requiredFuel: Minimális üzemanyag-mennyiségek, amivelvisszajutunk a Földre.
```

Alakítsd ki azt a jó szokást, hogy a problémamegoldás első lépéseként egy vázlatot írsz. Hidd el, ez nagyon jól fog jönni, amikor elkezdesz a tényleges kódon dolgozni.

3.3. A jegyzetek véglegesítése

Most, hogy a megjegyzésekkel megvolnánk, térjünk vissza a konkrét feladatunkhoz.

Ahhoz, hogy kiszámoljuk, mennyi üzemanyagtól szabaduljunk meg, tudnunk kell, hogy mennyi üzemanyagra van legalább szükségünk ahhoz, hogy visszajussunk a Földre. Szerencsére a matematikai képletet már felírtuk ehhez:

```
Szükséges üzemanyag = Teljes táv * Átlagos fogyasztás
```

Ahol az átlagos fogyasztás X liter / 100 km.

Nem kell mást tenned, mint egy változóneveket használó egyenletté alakítanod ezt:

```
requiredFuel = totalDistanceToTravel * (averageLitersOverHundred /  
100)
```

Megjegyzés: Az egyenletben a `requiredFuel` a szükséges üzemanyag-mennyiséget jelöli, a `totalDistanceToTravel` a teljes távolságot, az `averageLitersOverHundred` pedig az átlagos fogyasztást 100 kilométerenként, literben megadva. Miért bontottuk elemeire az átlagos fogyasztást? Ez a paraméter egy kiszámolt szám: `X liter 100 km-`en. Ebből egyelőre az `X liter` ismeretlen számunkra. Tehát ez mindenkorban egy változó lesz. A `100 km` rész viszont biztosan nem fog változni, ez egy állandó érték. Persze ebből is csinálhatnánk egy külön változót, de ezzel több gond lenne, mint amennyit nyernénk vele. Szóval ez most azon ritka alkalmak egyike, amikor a konkrét szám használata a kódban (az úgynevezett hardcoding) az előnyösebb választás.

Másold be ezt az egyenletet az `app.js` fájlba a jegyzeteid után:

```
console.log('A számológép használatra kész.');// Amiket ki kell számolnom:  
// fuelToOffload:  
    // Az az üzemanyagmennyiség, amit leeresztve a súlylimit alá  
    // kerülünk, de még haza tudunk jutni.  
    // fuelToOffload = tankCapacity - requiredFuel  
// tankCapacity: Az ūrsikló üzemanyagtartályának ūrtartalma.  
// requiredFuel:  
    // Minimális üzemanyag-mennyiség, amivel visszajutunk a  
    // Földre.  
    // requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100)
```

Ha vetsz egy pillantást ezekre a megjegyzésekre, egyértelműen kirajzolódik egy minta:

- **Minden változót a programozási normáknak megfelelően nevezünk el.**
- **A kiszámítandó változókra vonatkozó egyenleteket kiírjuk.**

Ennek hamarosan egyértelmű haszna lesz.

Oké, most, hogy minden megvan, ami kell, változtassuk a jegyzeteinket egy programmá.

3.4. Az első lépés leprogramozása

Nyisd meg az `app.js` fájlt, és jegyezd fel e következő két könnyű lépést, amit a programnak végre kell hajtania:

```
// 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
// 2. lépés: Számolja ki a lecsapolandó üzemanyagot.
```

Ha valahogy be szeretnél csomagolni egy konkrét feladatot ellátó kódrészletet, a függvények használata tökéletes választás. Gyerünk, csomagold be a fenti két lépést egy függvénybe:

```
function calculateFuelToOffload() {  
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.  
}
```

Megjegyzés: A függvényt `calculateFuelToOffload`-nak neveztem el, mert ez a számológépünk fő funkciója (a `calculateFuelToOffload` annyit tesz magyarul, hogy „számold ki a leeresztendő üzemanyag-mennyiséget”). Ne feledd, hogy a változókat és a függvényeket mindenkor megfelelően érdemes elnevezni!

Akkor most végezzük el az első lépést. Már nagyrészt kész is vagy vele, a jegyzeteid 9. sorában le van írva. Csak be kell másolnod a megfelelő helyre:

```
function calculateFuelToOffload() {  
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
    requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100)  
  
    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.  
}
```

Ahhoz, hogy szintaktikailag helyes legyen, deklaráld a változót, és tegyél a végére pontosvesszőt:

```
function calculateFuelToOffload() {  
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
  
    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.  
}
```

Már szinte kész is vagyunk – csupán a `totalDistanceToTravel` („teljes távolság”) és az `averageLitersOverHundred` („átlagos fogyasztás 100 kilométerenként, literben megadva”) változókat kell még valahogy deklarálni. Ezeket a változókat még nem tudjuk, ezeket majd Ádámtól és Annától kapjuk. Az értéküket majd valahogyan meg kell adni a függvénynek.

A függvény paramétereit pedig pont erre valók:

```
function calculateFuelToOffload(totalDistanceToTravel,  
averageLitersOverHundred) {  
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
  
    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.  
}
```

Ezzel meg is volnánk, elvégeztük az első lépést. Most le kell ellenőrizned, hogy működik-e. Ehhez az alkalmazásnak meg kell jelenítenie a számítás eredményét. Erre a cérla használhatsz egy egyszerű `console.log()` utasítást:

```
function calculateFuelToOffload(totalDistanceToTravel,  
averageLitersOverHundred) {  
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +  
    ' L');  
  
    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.  
}
```

Oké, most feltételezzük, hogy a függvény meghívásakor a konzolban meg fog jelenni a keresett mennyiség. Nézzük meg, hogy igazunk van-e.

3.5. Az első lépés visszaellenőrzése

Amikor asztali szerkesztőben dolgozol, szükséged lesz egy pár plusz lépésre ahhoz, hogy megtekinthesd a munkád előnézetét. JavaScript programok esetében például a következőket kell tenned:

1. Linkeld be a JavaScript fájlodat egy HTML fájlba (ezt már meg is csináltuk korábban).
2. Nyisd meg a HTML fájlt a böngészőben.
3. Nyisd meg a fejlesztői eszközöket, és válts a konzolra.

Nyisd meg az index.html fájlt a böngészőben

Megjegyzés: És itt jön képbe a `h1` elem használata. Ennek segítségével ugyanis azonnal látni fogod, hogy betöltődött-e a HTML fájl, mert ha igen, akkor meg fog jelenni az „Ez az index.html fájl” felirat.

Nyisd meg a konzolt

Ha Chrome-ot használsz, akkor a fejlesztői eszközök megnyitásához nyomd meg a **Ctrl + Shift + I** billentyűparancsot (Windows és Linux esetén) vagy a **Cmd + Opt + I** billentyűkombinációt (Macen).

Kattints a konzol („Console”) fülre. Ha minden működik, és a JavaScript is megfelelően van belinkelve a fájlba, a konzolban rögtön meg kell jelennie a következő üzenetnek: "A számológép használatra kész."

Ellenőrizd le a függvényt, amit írtál

Eljött az igazság pillanata. Hívd meg a `calculateFuelToOffload` függvényt, és nézd meg, hogy megfelelően működik-e.

Ehhez szükséged lesz valamilyen fiktív adatra, amit megadhatsz a függvénynek.

Az egyszerűség kedvéért használunk olyan értékeket, amelyeket könnyű visszaellenőrizni. Mondjuk, hogy 600 km-t szeretnénk utazni, és a próbaúrhajónk átlagosan 5 liter üzemanyagot fogyaszt 100 kilométerenként. Hívd meg a függvényt a konzolban, és add meg a következő argumentumokat:

```
calculateFuelToOffload(600, 5);
```

Ha működik a kis számológépünk, akkor a következő üzenetet kell kapnod:

```
Szükséges üzemanyag-mennyiség: 30 L
```

Gratulálunk! Elkészültél a számológép első felével.

3.6. A második lépés megoldása

Az első lépéssel tehát készen vagyunk, ideje nekikezdeni a második lépésnek. Menj vissza az `app.js` fájlba, és fasd át, hogy mi szerepel eddig benne:

```
console.log('A számológép használatra kész.');

// Amiket ki kell számolnom:
// fuelToOffload:
    // Az az üzemanyagmennyiség, amit leeresztve a súlylimit alá
    // kerülünk, de még haza tudunk jutni.
    // fuelToOffload = tankCapacity - requiredFuel
// tankCapacity: Az ūrsikló üzemanyagtartályának ūrtartalma.
// requiredFuel:
    // Minimális üzemanyag-mennyiség, amivel visszajutunk a
    // Földre.
```

```
// requiredFuel = totalDistanceToTravel *
(averageLitersOverHundred / 100)

function calculateFuelToOffload(totalDistanceToTravel,
averageLitersOverHundred) {
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.
    var requiredFuel = totalDistanceToTravel *
(averageLitersOverHundred / 100);
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +
' L');

    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.
}
```

A jó hírünk az, hogy a 2. lépés nagyon egyszerű lesz, hiszen a munka oroszlánrészét már elvégezted a vázlatírás során. A folyamat pedig nagyon hasonló lesz az 1. lépéshöz.

Az egyenletet, amire szükséged lesz, egyszerűen másold ki a 6. sorból, és illeszd be a megjegyzésed alá:

```
function calculateFuelToOffload(totalDistanceToTravel,
averageLitersOverHundred) {
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.
    var requiredFuel = totalDistanceToTravel *
(averageLitersOverHundred / 100);
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +
' L');

    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.
    fuelToOffload = tankCapacity - requiredFuel
}
```

Tedd szintaktikailag helyessé a `var` és a pontosvessző hozzáadásával:

```
function calculateFuelToOffload(totalDistanceToTravel,
averageLitersOverHundred) {
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.
    var requiredFuel = totalDistanceToTravel *
(averageLitersOverHundred / 100);
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +
' L');

    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.
    var fuelToOffload = tankCapacity - requiredFuel;
```

{}

Akárcsak az előző lépésekben, itt is hozzá kell majd férned a `tankCapacity` (üzemanyagtartály ūrtartalma) értékéhez, így ezt mindenkiéppen add hozzá paraméterként. Az eredményt is meg kell majd valahogyan jelenítened, úgyhogy szükséged lesz még egy `console.log()` utasításra is:

```
function calculateFuelToOffload(totalDistanceToTravel,
averageLitersOverHundred, tankCapacity) {
    // 1. lépés: Számolja ki a szükséges üzemanyag-mennyiséget.
    var requiredFuel = totalDistanceToTravel *
(averageLitersOverHundred / 100);
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +
' L');

    // 2. lépés: Számolja ki a lecsapolandó üzemanyagot.
    var fuelToOffload = tankCapacity - requiredFuel;
    console.log('Leeresztendő üzemanyag: ' + fuelToOffload +
' L');
}
```

Mentsd el a változtatásaidat, és menj át a böngészőbe. Töltsd be újra az `index.html` fájlt (különben a változtatásaid nem fognak életbe lépni). Nyisd meg a konzolt, és teszteld a függvény legújabb verzióját. Jelen esetben az üzemanyagtartály 100 literes lesz, és továbbra is 600 kilométert szeretnénk megtenni, átlag 5 L/km fogyasztás mellett.

```
calculateFuelToOffload(600, 5, 100);
```

Ha minden jól megy, a következő üzenetnek kell megjelennie a képernyőn:

```
Szükséges üzemanyag-mennyiség: 30 L
```

```
Leeresztendő üzemanyag: 70 L
```

Gratulálunk, az alkalmazás logikai vázának végére értél. A következő leckében azt fogjuk megoldani, hogy az alkalmazás olyan, nem műszaki beállítottságú emberek számára is kényelmesen használható legyen, mint amilyen például Ádám és Anna is, hogy ne kelljen nekik a konzolban kotorászniuk.

4. Felhasználói felület

4.1. Felhasználói felületre márpedig szükség van

Ürhajós társaid nem programozók. Szükségük van egy grafikus kezelőfelületre, amin keresztül az újonnan épített számológépedet kezelni tudják.

Persze *megtehetnéd*, hogy felépítesz egy vadonatúj felhasználói felületet nulláról, de a jelen helyzetben nincs vesztegetni való időnk. Vagy oxigénünk... Szóval válasszuk inkább a leggyorsabb megoldást, és használjuk a JavaScript előre beépített metódusait a kommunikációra.

Az `alert()` metódust arra haszálhatjuk, hogy megjelenítsünk valamilyen információt a böngésző egy felugró ablakában. Ez tökéletes módja lenne annak, hogy kiírjuk az ūrhajósoknak a számítások eredményét.

Az `alert()` metódusnak van egy testvére is: a `prompt()` metódus. Ez is egy felugró ablakot használ, de nem valamilyen információ megjelenítésére, hanem annak begyűjtésére. Meg tud jeleníteni egy kérdést, és be tudja fogadni a felhasználó által a szövegmezőben megadott választ. Ez kiválóan megfelel arra, hogy begyűjtsd a kalkulációhoz szükséges számadatokat.

Most, hogy már van némi elképzelésed a megoldásról, térjünk vissza a szerkesztőhöz, és adjuk hozzá a felhasználói felületet.

4.2. További megjegyzések hozzáadása

Mivel most úgyis vissza kell térded kicsit a tervezőasztalhoz, javaslom, hogy ismét jegyezz le egy-két dolgot. A grafikus felhasználói felület hozzáadása mindenkihez meg fogja változtatni a jelenlegi kódmezőket. A legjobb, ha kicsit lelassítasz, és egyszerűen magyarul leírod, hogy mit akarsz elérni.

Szóval azt várod el az alkalmazástól, hogy tegyen fel pár kérdést a felhasználónak, végezzen el néhány számítást, majd jelenítse meg a számítások eredményét a képernyőn. Avagy egy kicsit részletesebben:

1. Kérdezzen rá a `totalDistanceToTravel` (az utazás teljes távolsága) értékére, és mentse el a választ egy változóba.
2. Kérdezze meg az `averageLitersOverHundred` (átlagos fogyasztás 100 km-en, literben megadva) értékére, és mentse el a választ egy változóba.
3. Kérdezzen rá a `tankCapacity` (üzemanyagtartály ürtartalma) értékére, és mentse el a választ egy változóba.
4. Végezze el a számítást (hívja meg a `calculateFuelToOffload` függvényt).
5. Jelenítse meg az eredményt a felhasználónak.

Add hozzá ezt a listát az `app.js` fájl tartalmához megjegyzések formájában. A végeredménynek valahogyan így kell kinéznie:

```
console.log('A számológép használatra kész.');

// 1) Kérdezz rá a „totalDistanceToTravel” értékére, és mentsd el
// a választ egy változóba.
```

```
// 2) Kérdezz rá az „averageLitersOverHundred” értékére, és mentsd el a választ egy változóba.  
// 3) Kérdezz rá a „tankCapacity” értékére, és mentsd el a választ egy változóba.  
// 4) Végezd el a számítást (hívд meg a „calculateFuelToOffload” függv nyt).  
  
function calculateFuelToOffload(totalDistanceToTravel,  
averageLitersOverHundred, tankCapacity) {  
    // Sz m tsd ki a sz ks ges  zemanyag-mennyis get  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    console.log('Sz ks ges  zemanyag-mennyis g: ' + requiredFuel +  
' L');  
  
    // Sz m tsd ki a leeresztend   zemanyag-mennyis get  
    var fuelToOffload = tankCapacity - requiredFuel;  
    console.log('Leeresztend   zemanyag: ' + fuelToOffload +  
' L');  
}  
  
// 5)  rd ki az eredm nyt a felhaszn l nak.
```

Megjegyz s: Figyeld meg, hogy közben kit r lt k az eredeti megjegyz sek. Mivel a logik nak ezen r sz t m r be p tt t k k dba, ezekre a megjegyz sekre m r nincs sz ks g nk. Az 1. l p s-re  s 2. l p s-re vonatkoz  jel l sek t is t r lt k a megjegyz sekb l a f ggv nyen bel l, neh gy  sszekeverj k  ket az  j megjegyz sekkel. A k d nagyon sokat tud v ltozni a programoz si folyamat sor n, sz val pr b lj meg min l nagyobb rendet tartani benne.

A követke o leck ben el fogod kezdeni megoldani a megjegyz sekben felsorolt feladatokat egyes vel.

4.3.  rt kek bek r se

Kezd k a legels vel: **k rdezz r  a „totalDistanceToTravel”  rt k re,  s ment d el a választ egy v ltoz ba.**

Ebben a helyzetben j l j het a **prompt()** met dus. Itt egy karakterl ancot (azaz stringet) tudsz megadni a z r jelek köz t t. Az itt megadott stringet azt n a b ng sz  egy felugr  ablakban jelen ti meg a felhaszn l  sz m ra, egy sz vegmez vel  s egy gombbal egy tt, amivel a felhaszn l  el tudja menteni a v laszt t.

Az els  prompttal azt fogjuk megk r dezni, hogy milyen t vols got akarnak az  haj val megt nni.  rd a k vetke o sort közvetlen l a megjegyz s al :

```
prompt('Hány kilométer távolságot kíván megtenni? Adjon meg egy számot!');
```

Megjegyzés: Fontos, hogy pontosan fogalmazzunk. Mivel a választ a `totalDistanceToTravel` változó értékeként akarjuk elmenteni, annak mindenkorban számnak kell lennie. Az is fontos, hogy a mértékegység km legyen, hiszen az egész egyenletünk erre épül. Ezért ennyire részletes a promptban megjelenő kérdés.

Ezzel még nem igazán végeztünk. Nem elég egy prompt metódust megjeleníteni, el is szeretnénk menteni az arra kapott választ a megfelelő változóba.

Egészítsd ki a sort a következőképpen:

```
var totalDistanceToTravel = prompt('Hány kilométer távolságot kíván megtenni? Adjon meg egy számot!');
```

Így kell hozzárendelni egy prompt értékét egy változóhoz. A deklaráció jobb oldala felveszi a felhasználó által megadott értéket, és ezt a bal oldalon megnevezett változó fogja eltárolni.

Most, hogy már tudod, hogyan lehet prompt metódussal értékeket begyűjteni, próbáld meg egyedül kóddá alakítani a második és harmadik megjegyzést.

- A második promphoz tartozó string legyen a következő: "Mi az ūrhajó átlagos fogyasztása literben? Adjon meg egy számot!"
- A harmadik pedig így szóljon: "Mi az ūrhajó üzemanyagtartályának ūrtartalma literben? Adjon meg egy számot!"

Írd meg a megfelelő kódot minden megjegyzés alatt. A következő leckében megmutatjuk a helyes megoldást, de először próbáld meg egyedül megoldani a feladatot.

4.4. Felesleges paraméterek eltávolítása

Biztosan sikerült egyedül kitalálnod, hogyan kérd be prompt metódussal a hiányzó változók értékét, de itt a helyes megoldás, hogy le tud ellenőrizni a kódodat:

```
// 1) Kérdezz rá a „totalDistanceToTravel” értékére, és mentsd el a választ egy változóba.  
var totalDistanceToTravel = prompt('Hány kilométer távolságot kíván megtenni? Adjon meg egy számot!');  
  
// 2) Kérdezz rá az „averageLitersOverHundred” értékére, és mentsd el a választ egy változóba.  
var averageLitersOverHundred = prompt('Mi az ūrhajó átlagos fogyasztása literben? Adjon meg egy számot!');
```

```
// 3) Kérdezz rá a „tankCapacity” értékére, és mentsd el a választ  
egy változóba.
```

```
var tankCapacity = prompt('Mi az ūrhajó üzemanyagtartályának  
űrtartalma literben? Adjon meg egy számot!');
```

Tipp: Ezen a ponton mindenkor prompttal kész vagyunk. Ha akarod, ki is próbálhatod őket. Mentsd el az `app.js` fájlodat, és frissítsd az `index.html` fájlt a böngészőben. Ha minden jól csináltál, a három promptnak sorban egymás után meg kell jelennie. A beírt válaszokkal még semmi sem történik, de emiatt ne aggódj. Egyelőre az volt a célunk, hogy hozzáadjuk a promptokat a kódhoz.

Mellesleg így már megint feleslegessé vált a kódunk egy része. Az előző iterációban a `calculateFuelToOffload` függvény paramétereit használtad arra, hogy meg tudd adni a szükséges változókat. Most, hogy már globálisan deklarált változókat használsz, ezeket a paramétereket nyugodtan törölheted.

```
// Változtasd ezt:  
function calculateFuelToOffload(totalDistanceToTravel,  
averageLitersOverHundred, tankCapacity) {...}  
  
// Ezzé:  
function calculateFuelToOffload() {...}
```

Mentsd el az `app.js` fájlt, és frissítsd a böngésző lapját. A következő leckében az `alert()` metódust fogjuk hozzáadni a kódhoz, hogy megjeleníthessük a számításaink eredményét.

4.5. Az eredmények megjelenítése

Már célegyenésben vagy. A promptok kész vannak, a számítások mögött meghúzódó logikát is lejegyeztük, már csak meg kell jeleníteni az eredményt a képernyőn.

Vessünk egy pillantást a `calculateFuelToOffload` függvény jelenlegi állapotára:

```
// 4) Végezd el a számítást (hívд meg a „calculateFuelToOffload”  
függv\'enyt).  
function calculateFuelToOffload() {  
    // Számítsd ki a szükséges üzemanyag-mennyiséget  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +  
' L');  
  
    // Számítsd ki a leeresztendő üzemanyag-mennyiséget  
    var fuelToOffload = tankCapacity - requiredFuel;  
    console.log('Leeresztendő üzemanyag: ' + fuelToOffload + '
```

```
L');  
}
```

Pillanatnyilag `console.log()` utasításokat használssz az eredmények konzolban való megjelenítéséhez. Az ellenőrzés kedvéért egyelőre próbálunk meg minél kevesebb változtatást végrehajtani, és egyszerűen adjunk hozzá egy `alert()` metódust a programhoz ugyanazzal a tartalommal minden `console.log()` utasítás alatt.

```
// 4) Végezd el a számítást (hívd meg a „calculateFuelToOffload”  
függvényt).  
function calculateFuelToOffload() {  
    // Számítsd ki a szükséges üzemanyag-mennyiséget  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    console.log('Szükséges üzemanyag-mennyiség: ' + requiredFuel +  
' L');  
    alert('Szükséges üzemanyag-mennyiség: ' + requiredFuel + '  
L');
```



```
// Számítsd ki a leeresztendő üzemanyag-mennyiséget  
var fuelToOffload = tankCapacity - requiredFuel;  
console.log('Leeresztendő üzemanyag: ' + fuelToOffload + '  
L');
```



```
    alert('Leeresztendő üzemanyag: ' + fuelToOffload + ' L');
```

```
}
```

Ennek elméletileg meg kéne jelenítenie minden egyes lépés eredményét egy felugró alert ablakban. Mentsd el az `app.js` fájlt, menj át a böngészőbe, frissítsd a lapot, és próbáld ki. Teljes utazási távolságnak adj meg 600 km-t, a fogyasztás legyen 5 L / 100 km, és az üzemanyagtartály ürtartalma legyen most is összesen 100 liter.

Úgy tűnik, valami nem stimmel. A promptok megjelentek, de az alertek nem. De miért? Ha vetsz egy pillantást az `app.js` fájlra, könnyen felfedezheted a hibát. Bár deklaráltad a `calculateFuelToOffload` függvényt, elfelejtetted meghívni azt. **Ne feledd, a függvényeket meg kell hívni ahhoz, hogy lefussanak!**

Javítsuk ki ezt a kis hibát gyorsan. Hívd meg a függvényt úgy, hogy hozzáadod a következő sort az `app.js` fájl végéhez:

```
calculateFuelToOffload();
```

Most próbáljuk ki újra, hogy működik-e. Mentsd el a fájlt, és frissítsd a böngészőt.

Szuper, most már működik a program! Már csak egy-két apróság van hátra, és indulhat is az

űrhajó.

4.6. Végső simítások

A program most már teljesen működőképes, de még mindig nem túl elegáns. Sok felesleges megjegyzés és kódrészlet maradt a kódábrisban (pl. az alertek mellett már nincs szükség a `console.log()` utasításokra).

Mielőtt ünnepélyesen átadnán az alkalmazást Ádámnak és Annának, karcsúsítsunk még rajta egy kicsit.

A felesleges megjegyzések eltávolítása

A megjegyzések kifejezetten hasznosak a fejlesztés során, és valóban hasznosak lehetnek a kód karbantartása vagy másokkal való közös munka során is. **De soha nem szabad olyan dolgokat magyarázniuk, amelyek magából a kóból is egyértelműek.** Ebből a szempontból a programban lévő megjegyzések most már feleslegesek. Egy másik programozó gond nélkül megértené a kódábrisodat nélkülük is.

Törölj minden megjegyzést az `app.js` fájlból.

A `console.log()` utasítások törlése

Már nincs rájuk szükséged az alertek mellett. Ne felejtsd el törölni a legelsőt sem, amit még az elején arra használtunk, hogy ellenőrizzük, betöltött-e az `app.js`.

A változók áthelyezése a függvényen belülre

Jelen pillanatban a `totalDistanceToTravel`, az `averageLitersOverHundred` és a `tankCapacity` változóid a `calculateFuelToOffload` függvényen kívül szerepelnek, azaz globális változónak számítanak.

Bár ez nem számít konkrét hibának, nem is egy bevett megoldás. A globális változókat általában több függvényben is felhasználjuk, ezért vannak globálisan deklarálva.

Ezzel szemben az általad használt három változó logikailag a `calculateFuelToOffload`függvényhez tartozik. Így észszerű lenne őket a függvényen belülre helyezni, és lokális változóként deklarálni őket.

Egyszerűen fogd meg és helyezd őket a függvénytörzs elejére:

```
function calculateFuelToOffload() {
    var totalDistanceToTravel = prompt('Hány kilométer távolságot
    kíván megtenni? Adjon meg egy számot!');
    var averageLitersOverHundred = prompt('Mi az ūrhajó átlagos
    fogyasztása literben? Adjon meg egy számot!');
    var tankCapacity = prompt('Mi az ūrhajó üzemanyagtartályának
    ūrtartalma literben? Adjon meg egy számot!');
```

```
var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    alert('Szükséges üzemanyag-mennyiség: ' + requiredFuel + '  
L');
```

```
var fuelToOffload = tankCapacity - requiredFuel;  
    alert('Leeresztendő üzemanyag: ' + fuelToOffload + ' L');
```

```
}
```

Az alertek kicsinosítása

A jelenlegi `alert()` metódusokkal két probléma van: az üzenetük nem egészen világos, és kettő van belőlük. Jobb lenne, ha csak egy `alert` lenne meg a folyamat végén, egy egyértelműbb üzenettel.

Töröld az első `alert()` metódust a programból, a másodikat pedig változtasd meg a következőképp:

```
alert('Az utazáshoz szükséges üzemanyag-mennyiség: ' +  
requiredFuel + ' L\n' + 'Leadandó üzemanyag-mennyiség: ' +  
fuelToOffload + ' L');
```

Tessék! Már össze is vontuk a két üzenetet.

Megjegyzés: A sor közepén látható `\n` karakter az úgynevezett „newline character” (új sor karakter). Ez egy sortörést ad a stringhez, így az üzenetek két külön sorban fognak megjelenni.

A cím megváltoztatása

Végül, de nem utolsósorban változtassuk meg az `index.html` fájl betöltésekor megjelenő `h1` és `title` elemek tartalmát. Legyen minden két elemben „Üzemanyag-kalkulátor”.

És kész is volnánk! A véleges kódhoz hasonlóan így kell kinéznie:

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="utf-8">  
        <title>Üzemanyag-kalkulátor</title>  
        <meta name="viewport" content="width=device-width,  
initial-scale=1">  
        <script src="app.js"></script>  
    </head>  
    <body>  
        <h1>Üzemanyag-kalkulátor</h1>
```

```
</body>  
</html>
```

```
function calculateFuelToOffload() {  
    var totalDistanceToTravel = prompt('Hány kilométer távolságot  
kíván megtenni? Adjon meg egy számot!');  
    var averageLitersOverHundred = prompt('Mi az űrhajó átlagos  
fogyasztása literben? Adjon meg egy számot!');  
    var tankCapacity = prompt('Mi az űrhajó üzemanyagtartályának  
űrtartalma literben? Adjon meg egy számot!');  
  
    var requiredFuel = totalDistanceToTravel *  
(averageLitersOverHundred / 100);  
    var fuelToOffload = tankCapacity - requiredFuel;  
  
    alert('Az utazáshoz szükséges üzemanyag-mennyiség: ' +  
requiredFuel + ' L\n' + 'Leadandó üzemanyag-mennyiség: ' +  
fuelToOffload + ' L');  
}  
  
calculateFuelToOffload();
```

Csodálatos. Ideje hazatérni, nem gondolod?

5. Hazaút

5.1. A számológép használata élesben

Most, hogy elkészültél a számológéppel, ideje megkeresned Ádámot és Annát. Magadhoz veszed a laptopot, és felmászol az űrhajó folyosóin. Űrhajós társaid a pilótafülkében várnak rád.

Átnyújtod Annának a laptopot a használatra kész üzemanyag-kalkulátorral. Most már ki fogja tudni számolni, hogy Ádám mennyi üzemanyagot szivattyúzzon ki a tartályból.

Izgatottan figyeled, ahogy Anna betáplálja az adatokat. A Föld 54 600 000 km távolságra van a Marstól. Az űrhajótok nagyon hatékony, minden 0,8 liter fogyaszt 100 km-enként (az ūrben nincs súrlódás). Az üzemanyagtartályba összesen 550 000 liter üzemanyag fér.

Anna arca felderül, ahogy a program kiírja az eredményt. Ádámmal együtt válon veregetnek, aztán Ádám lesiet a motortérbe, hogy leereszze a megfelelő mennyiségű üzemanyagot.

Pár órával később az Olympus Mons talaja beleremeg, ahogy az űrhajó felszáll, és széles

ívet leírva elhagyja a vörös bolygó ritkás levegőjét.

A találékonyságodnak és munkádnak köszönhetően minden nyílt biztonságosan haza fogtok jutni. Szép munka, ūrhajós, igazán szép munka!

Ellenőrző kérdés

Milyen számokat olvashatott Anna a képernyőn?

- Szükséges üzemanyag: 223 000 L, Fölösleges üzemanyag: 24 000 L
- **Szükséges üzemanyag: 436 800 L, Fölösleges üzemanyag: 113 200 L**
- Szükséges üzemanyag: 550 000 L, Fölösleges üzemanyag: 0 L
- Szükséges üzemanyag: 100 000 L, Fölösleges üzemanyag: 450 000 L

5.2. Összefoglalás

Hát, ez egy igazán különleges utazás volt. Most, hogy visszatértünk a valóságba, nézzük át egy kicsit, hogy mit is tanultunk ebben a könyvben, és mindez milyen útravalóval szolgál neked mint ifjú programozónak.

A projekt során a fő célunk végig az volt, hogy megmutassuk, hogyan közelítenek meg egy összetett problémát a programozók, és hogy hogyan is néz ki egy fejlesztői ciklus.

A projekt során a következőket sajátítottad el:

1. Ha egy összetett problémával kell megbirkóznod, **próbáld meg felosztani** kisebb, könnyebben kezelhető problémákra. Ezeket a kisebb problémákat aztán sorban, egyesével oldd meg, hogy felépíts egy komplex megoldást.
2. **Először készíts egy vázlatot.** Méghozzá igen részleteset. Gondold végig, mit szerethnél, hogy a számítógép csináljon, és írd le hétköznapi kifejezésekkel. Sokkal könnyebb lesz a szintaxist hozzáadni később, ha már megvan a folyamat logikai váza.
3. **A programozás egy iteratív mesterség.** Ne próbálj elsőre tökéletes programot írni – elég, ha olyan kódot írsz, ami működik. Aztán javítgasd addig, amíg már nincs mit csiszolni rajta.
4. **Tartsd rendben a kódodat.** Ahogy haladsz a programozással, a kódmezők jelentős változásokon fog keresztülmenni. Ne sajnáld az időt, hogy néha kicsit karbantartsd, így a programod minden rendezett és olvasható marad.

Ha észben tartod ezeket az útmutatásokat, nemcsak jobb programozó lesz belőled, de sok bosszúságostól is megkíméled magad.

Reméljük, hogy tetszett ez a modul, és hasznosnak találtad. Ha készen állsz a következőre, még rengeteg kalandot tartogatunk a számodra!

Az alapok gyakorlása

1. Bevezető a gyakorlófeladatokhoz

1.1. Gyakorlás rajzolással

Ahhoz, hogy az eddig tanultakat jobban megértsd és elsajátítsd, sokat kell gyakorolnod. Ennek a modulnak a célja, hogy ebben segítséget nyújtson.

A rajzolás kitűnő eszköz erre a célra, hiszen azonnali vizuális visszajelzést kapsz arról, hogy a kódod mit csinál.

Ebben a modulban rajzolni fogsz a JavaScript használatával. Azokat az alapismereteket kell majd alkalmaznod, amelyeket már megtanultál a korábbiakban.

Megjegyzés: Mint általában, most sincs egyetlen jó megoldása a feladatoknak. Bár elvégezheted a leckéket csak a korábbi CodeBerry-modulokra támaszkodva, mégis bármikor megteheted (és erősen buzdítunk is arra), hogy használd az interneten fellelhető tudásanyagot.

A gyakorlatok során HTML-kódot fogunk JavaScripttel együtt használni. A bevezetésben megtanulod, hogyan alkalmazhatod rajzolásra a `<canvas>` HTML-elemet.

Ha valami nem sikerül elsőre, ne add fel. Próbálkozz tovább, apró lépésekkel módosítva a kódot. Végül sikerülni fog!

Megjegyzés: A modul feladatainak megoldásakor használd bátran az általad preferált kód szerkesztőt (pl. a JS Bint, a CodePent, egy asztali szerkesztőt, amelyik neked legjobban tetszik). Ne feledd, hogy ha asztali kód szerkesztőben dolgozol, akkor a `.js` fájlodat hozzá kell kapcsolnod a `.html` fájlhoz.

1.2. A canvas elem

A `<canvas>` elem egy HTML-elem, amelyre JavaScript segítségével rajzolhatsz.

HTML-kód

A szükséges HTML-kód a következő:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
```

```
<title>Title</title>
</head>
<body>
    <canvas id="myCanvas" width="450" height="300"
style="border: 1px solid black"></canvas>
</body>
</html>
```

Megjegyzés: Ahhoz, hogy egy érvényes HTML-dokumentumot hozz létre, a HTML-kódodnak tartalmaznia kell bizonyos kötelező részeket, úgymint a doctype-deklarációt (`<!DOCTYPE html>`), valamint a `<html>`, `<head>` és `<body>` elemeket.

A `<canvas>` elemet az alábbi sor hozza létre:

```
<canvas id="myCanvas" width="450" height="300" style="border: 1px
solid black"></canvas>
```

Az `id`-ra azért van szükségünk, hogy a JavaScript-kódban hivatkozni tudjunk erre az elemre.

A magasság és szélesség értéke tetszőlegesen módosítható.

A keret egyértelműbbé teszi, hogy hova tudsz rajzolni.

Tipp: A méreteket és a stílust persze írhatod külön CSS-fájlba is.

JavaScript-kód

A JavaScript-kódban a `<canvas>` HTML-elementet az `id`-ja (`myCanvas`) segítségével választjuk ki, majd pedig eltároljuk egy `canvas` nevű változóban:

```
var canvas = document.getElementById('myCanvas');
```

Ezen az elemen létrehozunk egy kétdimenziós rajzolási kontextust, és elmentjük azt egy `context` nevű változóba:

```
var context = canvas.getContext('2d');
```

Innen töl kezdve tudunk rajzolni erre a `context` változóra. Például rajzolhatunk egy négyzetet, csak hozzá kell adnunk az alábbi sort a JavaScript-kódhoz:

```
context.fillRect(20, 10, 250, 175);
```

1.3. fillRect()

Négyszögek

Máris tudsz négyszöget rajzolni.

```
context.fillRect(20,10,250,175);
```

A `fillRect()` függvény rajzol egy négyszöget, és kitölti az éppen aktuális színnel (alapértelmezetten feketével).

Négy paramétert fogad: `x`, `y`, `width`, `height`.

Az `x` és `y` koordináták határozzák meg a négyszög bal felső sarkának helyzetét. (A (0,0) koordináták a `canvas` bal felső sarkában találhatók.) A `width` (szélesség) és a `height` (magasság) pedig a négyszög méreteit adják meg, pixelben.

Feladat

Nyisd meg a kedvenc kódszerkesztő alkalmazásod, és hozz létre egy új projektet. Másold bele az alábbi HTML- és JavaScript-kódblokkokat. Ezek adják majd a közös kiindulási alapot a következő feladathoz.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title>Title</title>
  </head>
  <body>
    <canvas id="myCanvas" width="450" height="300"
style="border: 1px solid black"></canvas>
  </body>
</html>
```

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

A feladatod, hogy a `canvas` közepére rajzolj egy fekete négyszöget a JavaScript segítségével.

A programod a következő kritériumoknak feleljen meg:

- Csak egy darab `fillRect()` függvényt használj.

- A fekete négyzet legyen 100 egység széles és 100 egység magas.
- Legyen tökéletesen középre igazítva a 450-szer 300-as canvas-on.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

1.4. fillStyle

Színek

Színesen sokkal jobb rajzolni.

```
context.fillStyle = 'red';
context.fillRect(20,10,250,175);
```

A `fillStyle` tulajdonság értéke bármilyen színre megváltoztatható. Onnantól kezdve azzal a színnel lesz kitöltve minden, amit kitöltesz.

Megjegyzés: A `fillRect()` egy függvény, melynek argumentumait a zárójelek közé kell írnod a meghívásakor. A `fillStyle` ezzel szemben egy tulajdonság, melynek az egyenlőségjel használatával adhatsz új értéket.

A `fillStyle`-hoz úgy rendelhetsz hozzá egy színt, hogy az adott szín angol nevét stringként adod meg (pl. `'red'` a piros szín esetén). Az elérhető színnevek listáját [itt találod](#).

Megjegyzés: Más módokon is meg lehet adni a színt. Ebben a modulban használni fogjuk az `rgb()` és `rgba()` függvényeket is. Az `rgb()` függvénynek három paramétere van (`rgb(red, green, blue)`), míg az `rgba()` függvénynek négy (`rgba(red, green, blue, alpha)`). A `red` (piros), `green` (zöld) és `blue` (kék) az adott szín intenzitását határozza meg, az értékük pedig egy 0 és 255 közé eső egész szám (például az `rgb(255, 0, 0)` egyenlő a piros színnel). Az `alpha` az átlátszóságot határozza meg, és az értéke egy szám 0 (teljesen átlátszó) és 1 (teljesen átlátszatlan) között.

Feladat

Nyisd meg a kedvenc kódszerkesztő alkalmazásod, és hozz létre egy új projektet. Másold bele az alábbi HTML- és JavaScript-kódblokkokat. Ezek adják majd a közös kiindulási alapot a következő feladathoz.

```
<!DOCTYPE html>
<html>
  <head>
```

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,
initial-scale=1">
<title>Title</title>
</head>
<body>
<canvas id="myCanvas" width="450" height="300"
style="border: 1px solid black"></canvas>
</body>
</html>
```

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

A feladatod, hogy a `canvas` közepére rajzolj egy narancssárga négyzetet.

A programod a következő kritériumoknak feleljen meg:

- A négyzet legyen 200 egység széles és 200 egység magas.
- Legyen tökéletesen középre igazítva a 450-szer 300-as `canvas`-on.
- A négyzet legyen narancssárga színű.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

1.5. `moveTo()` és `lineTo()`

Vonalak

A `moveTo(x,y)` függvény segítségével egy koordinátára mozoghatsz ("felemelt tollal").

A `lineTo(x,y)` függvény pedig a kiindulási állapotodból a megadott koordinátáig húz egy egyenes vonalat.

A vonal tulajdonképpen egy láthatatlan "útvonal" (path) része. Hogy látszódjon is, használnod kell valamelyik "tinta" metódust (ink method), például a `stroke()` (körvonal) függvényt vagy a `fill()` (kitöltés) függvényt.

```
context.beginPath();
context.moveTo(55,70);
context.lineTo(150,100);
context.stroke();
```

A fenti kód tehát az (55;70) koordinátából húz egy vonalat a (150;100) koordinátáig, és feketével megrajzolja a körvonalát.

A `beginPath()` függvénynek most még látványos eredménye nincs. Később a különböző láthatatlan útvonalaid összeakadhatnak, ha nem indítod mindeneket tiszta lappal, ezért jó gyakorlat már a kezdetektől használni.

Ha szeretnél a `stroke()`-nak színt adni, a `strokeStyle` értékét kell módosítanod a `stroke()` függvény használata előtt:

```
context.strokeStyle = 'orange';
```

Feladat

Nyisd meg a kedvenc kódszerkesztő alkalmazásod, és hozz létre egy új projektet. Másold bele az alábbi HTML- és JavaScript-kódblokkokat. Ezek adják majd a közös kiindulási alapot a következő feladathoz.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title>Title</title>
  </head>
  <body>
    <canvas id="myCanvas" width="450" height="300"
style="border: 1px solid black"></canvas>
  </body>
</html>
```

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

A feladatod, hogy rajzold meg pirossal a `canvas` egyik átlóját.

A programod a következő kritériumoknak feleljen meg:

- Az átló a 450-szer 300-as canvas bal alsó sarkától a jobb felső sarkáig húzódjon.
- Az átló legyen piros színű.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott](#)

[mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

1.6. A canvasWidth és a canvasHeight

A canvas méretei

A `canvas` méreteit használhatod számolásokban, anélkül, hogy tudnád a pontos értékeket:

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");

var canvasWidth = canvas.width;
var canvasHeight = canvas.height;

context.fillRect(50,50,canvasWidth-100,canvasHeight-100);
```

A fenti kódban a `canvas` szélességét és magasságát eltároltuk változókban, majd ezeket a változókat használtuk egy négyszög méretének a kiszámításához.

Feladat

Nyisd meg a kedvenc kód szerkesztő alkalmazásod, és hozz létre egy új projektet. Másold bele az alábbi HTML- és JavaScript-kódblokkokat. Ezek adják majd a közös kiindulási alapot a következő feladathoz.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <title>Title</title>
  </head>
  <body>
    <canvas id="myCanvas" width="450" height="300"
style="border: 1px solid black"></canvas>
  </body>
</html>
```

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

A feladatod, hogy rajzolj egy fekete négyszöget, amely kitölți a `canvas` jobb alsó negyedét.

A programod a következő kritériumoknak feleljen meg:

- A fekete négyzet töltse ki a `canvas` teljes jobb alsó negyedét.
- A `canvas` szélességét és magasságát tárold el változókban.
- A négyzet pozíciójának és méretének kiszámításához használj ezeket a változókat a `fillRect()` függvény meghívásakor.

Ha kész vagy, változtasd meg a `canvas` méretét a HTML- vagy CSS-kódban, hogy lásd, mi történik. Ha minden jól megy, a négyzet továbbra is kitölti ugyanazt a negyedet.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2. Bemelegítő feladatok

2.1. Egy magányos négyzet

A feladatod, hogy rajzolj egy zöld négyzetet a canvas jobb alsó sarkába.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj egy négyzetet a `fillRect()` függvény használatával.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a négyzet pozíciójának kiszámításához és méretének megadásához (előtte deklarálj egy `size` és egy `padding` változót).

- A négyzet legyen 100 egység széles és 100 egység magas.
- A négyzet a canvas jobb alsó sarkában helyezkedjen el, 10 egységre a szélektől.
- Színezd zöldre a négyzetet a `fillStyle` tulajdonság használatával.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.2. Két átfedő négyzet

A feladatod, hogy rajzolj két félleg átlátszó, egymást részben átfedő négyzetet a canvas közepére.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj két négyzetet úgy, hogy a `fillRect()` függvényt kétszer használod a kódodban.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a négyzetek pozíciójának és méretének megadásához.
- A négyzetek legyenek 100 egység szélesek és 100 egység magasak.
- A két négyzet a canvas közepe körül helyezkedjen el.
- A két négyzet fedje le egymás negyedét.
- Az első négyzet színe legyen félleg átlátszó piros (`rgba(255, 0, 0, .5)`).
- A második négyzet színe legyen félleg átlátszó kék (`rgba(0, 0, 255, .5)`).

Tipp: A négyzetet kitöltő szín megváltoztatásához a `fillRect()` függvény meghívása előtt módosítsd a `fillStyle` tulajdonság értékét.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni az általunk alkotott [mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.3. Zászló

A feladatod, hogy rajzolj egy zászlót (piros, fehér, zöld) a canvas-ra.



A programod a következő kritériumoknak feleljén meg:

- Rajzolj három téglalapot úgy, hogy a `fillRect()` függvényt háromszor használd a kódodban.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a téglalapok pozíciójának kiszámításához és méretének megadásához.
- A téglalapok legyenek 250 egység szélesek és 50 egység magasak.
- A zászló a `canvas` közepén helyezkedjen el.
- Az első téglalap legyen piros színű.
- A második téglalap legyen fehér színű.
- A harmadik téglalap legyen zöld színű.

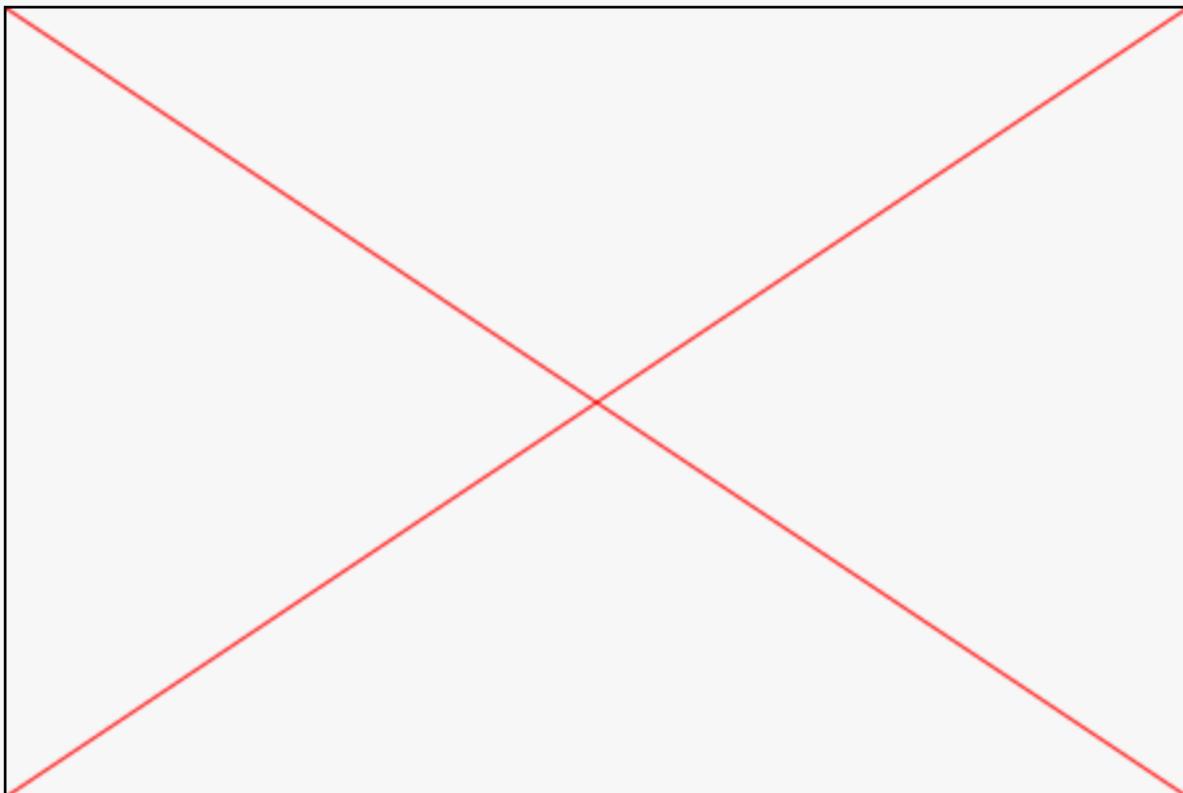
Tipp: A téglalapot kitöltő szín megváltoztatásához a `fillRect()` függvény meghívása előtt módosítsd a `fillStyle` tulajdonság értékét.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.4. Átlók

A feladatod, hogy rajzold meg pirossal a `canvas` két átlóját.



A programod a következő kritériumoknak felejen meg:

- Rajzold meg a `canvas` két átlóját.
- Amikor meghívod a `moveTo()` és `lineTo()` függvényeket, kizárolag változókat használj a koordináták kiszámításához.
- Az első átló a `canvas` bal felső sarkától a jobb alsó sarkáig húzdjon.
- A másik átló a `canvas` bal alsó sarkától a jobb felső sarkáig húzdjon.
- Az átlók legyenek piros színűek.

Tipp: A vonal színének megadásához a `stroke()` függvény meghívása előtt rendelj hozzá egy értéket a `strokeStyle` tulajdonsághoz.

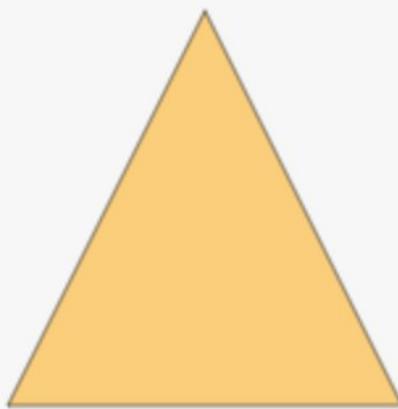
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden

rendben, akkor menj tovább a következő leckére.

2.5. Egy háromszög

A feladatod, hogy rajzolj egy háromszöget a `canvas`-ra.



A programod a következő kritériumoknak felejen meg:

- Rajzolj egy háromszöget a `canvas`-ra.
- Amikor meghívod a `moveTo()` és `lineTo()` függvényeket, használj változókat a koordináták kiszámításához.
- A háromszög alapja és magassága legyen 150 egységnyi (a háromszög legyen 150 egység széles és 150 egység magas).
- A háromszög bal alsó csúcsa 150 egységre helyezkedjen el a `canvas` bal szélétől és 225 egységre a felső szélétől.
- A vonalak színe legyen fél átlátszó szürke (`rgba(128, 128, 128, .5)`).
- A háromszöget kitöltő szín legyen fél átlátszó narancssárga (`rgba(255, 165, 0, .5)`).

Tipp: A vonal színének megadásához a `stroke()` függvény meghívása előtt rendelj hozzá egy értéket a `strokeStyle` tulajdonsághoz. A kitöltés színének meghatározásához a `fill()` függvény meghívása előtt rendelj hozzá egy értéket a `fillStyle` tulajdonsághoz.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni az [általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden

rendben, akkor menj tovább a következő leckére.

2.6. Felezővonalak

A feladatod, hogy rajzold meg a `canvas` vízszintes és függőleges felezővonalát.



A programod a következő kritériumoknak felejjen meg:

- Rajzold meg a `canvas` vízszintes és függőleges felezővonalát.
- Amikor meghívod a `moveTo()` és `lineTo()` függvényeket, használj változókat a koordináták kiszámításához.
- A vízszintes felezővonal a `canvas` bal szélénél közepétől a jobb szélénél közepéig húzdjön.
- A függőleges felezővonal a `canvas` felső szélénél közepétől az alsó szélénél közepéig húzdjön.
- A vízszintes felezővonal legyen piros színű.
- A függőleges felezővonal legyen zöld színű.

Tipp: Ne felejtsd el használni a `beginPath()` függvényt, hogy az útvonalak ne akadjanak össze.

Sok sikert!

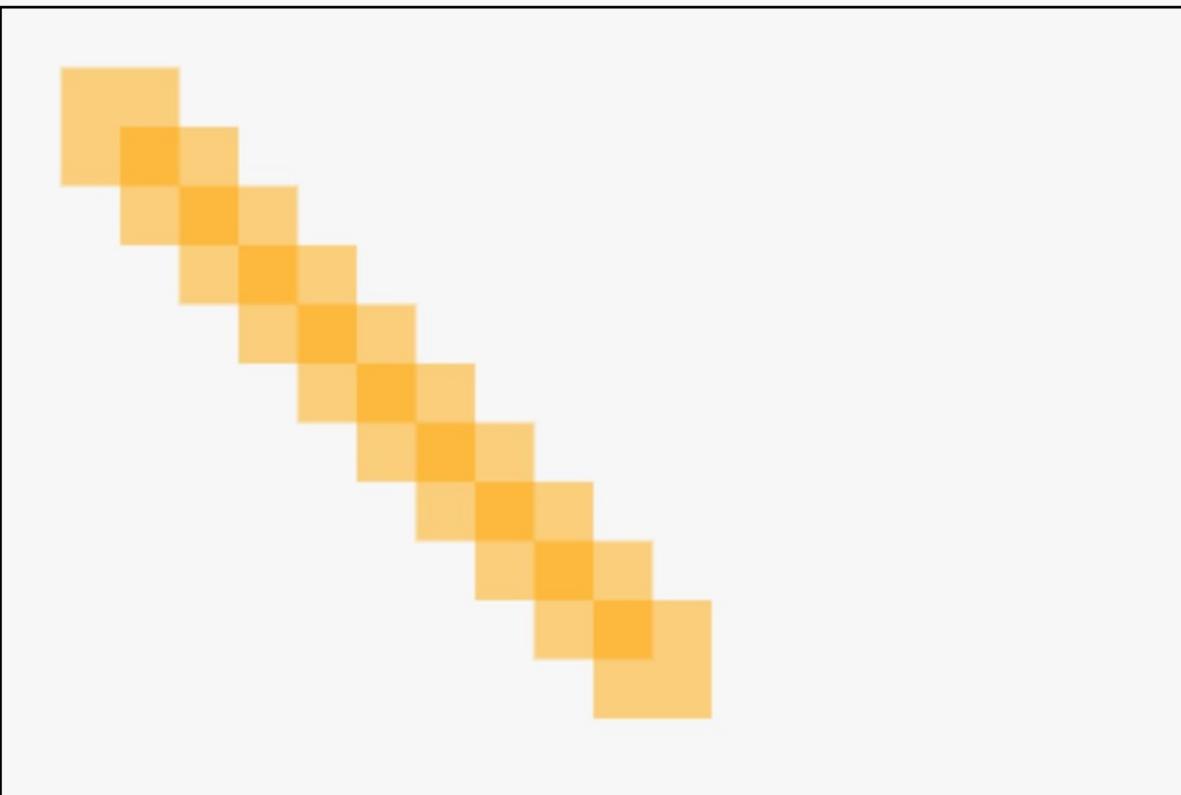
Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott](#)

[mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3. Ciklusok gyakorlása

3.1. DRY négyzetek

A feladatod, hogy rajzolj tíz egymást átfedő, narancssárga négyzetet a `canvas`-ra.



A programod a következő kritériumoknak felejen meg:

- Rajzolj tíz négyzetet úgy, hogy csak egyszer írod le a kódodban a `fillRect()` függvényt.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj argumentumként.
- minden négyzet legyen 45 egység széles és 45 egység magas.
- Az első négyzet a `canvas` bal felső sarkában helyezkedjen el, 20 egységre a szélektől.
- minden további négyzet az azt megelőző négyzet közepén kezdődjön.
- A négyzetek színe legyen félig átlátszó narancssárga (`rgba(255, 165, 0, .5)`).

Tipp: A pozicionáláshoz használt változót a cikluson kívül deklaráld, és a cikluson belül változtasd meg az értékét.

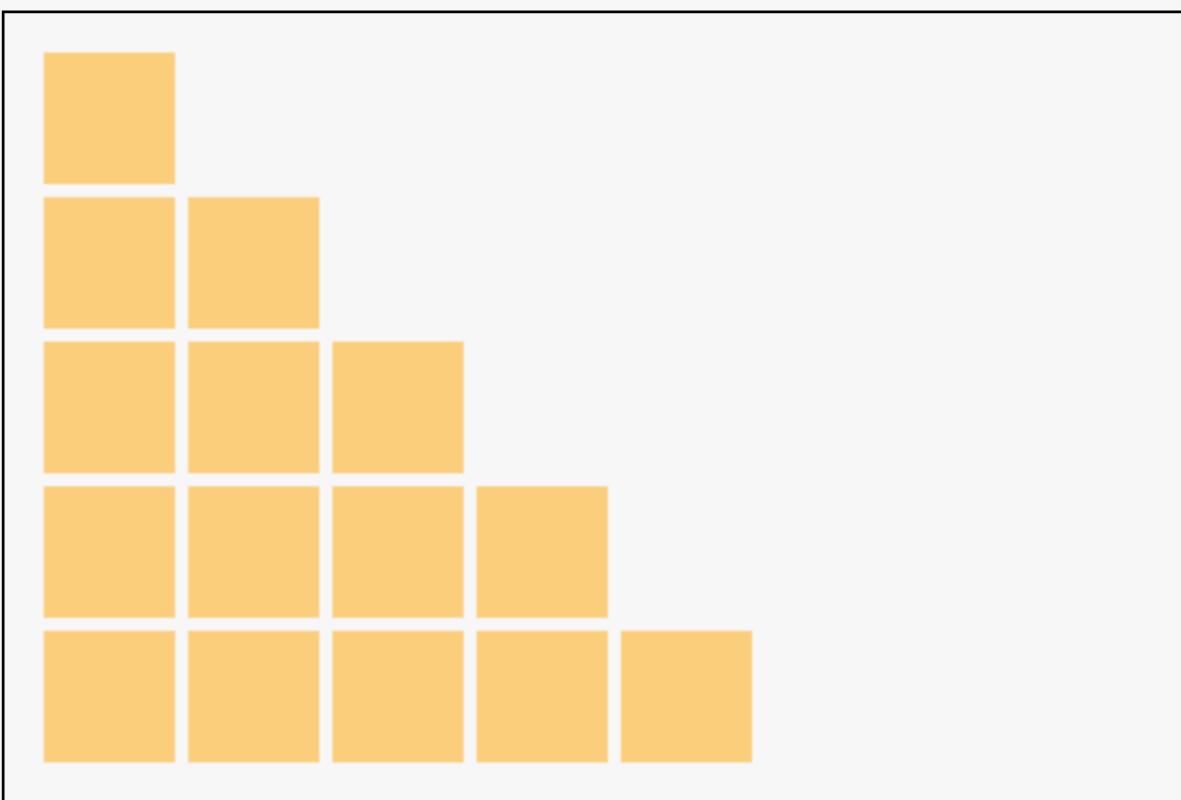
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott](#)

[mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.2. Sorok

A feladatod, hogy rajzolj 15 narancssárga négyzetet 5 sorba a `canvas`-on.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj 15 négyzetet 5 sorba úgy, hogy csak egyszer írod le a kódodban a `fillRect()` függvényt.
- minden sorban az aktuális sor számával egyenlő számú négyzet legyen (például a 3. sor 3 négyzetet tartalmazzon).
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a négyzetek pozíciójának kiszámításához és méretének megadásához.
- minden négyzet legyen **50** egység széles és **50** egység magas.
- Az első négyzet a `canvas` bal felső sarkában helyezkedjen el, **15** egységre a szélektől.
- A négyzetek között legyen **5** egységnyi térköz.
- A négyzetek színe legyen félig átlátszó narancssárga (`rgba(255, 165, 0, .5)`).

Tipp: Használj egymásba ágyazott ciklusokat, ügyelve arra, hogy ne változtasd meg az értékét olyasminek, amit nem akarsz módosítani.

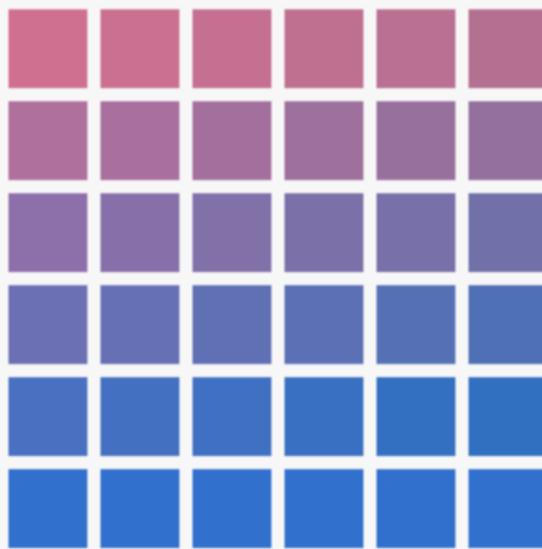
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott](#)

[mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.3. Színskála

A feladatod, hogy rajzolj egy 36 négyzetből álló rácsot a `canvas`-ra.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj egy 6x6-os négyzetrácsot úgy, hogy csak egyszer írod le a kódodban a `fillRect()` függvényt.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a négyzetek pozíciójának és méretének megadásához.
- minden négyzet legyen 30 egység széles és 30 egység magas.
- Az első négyzet 125 egységre helyezkedjen el a `canvas` bal szélétől és 50 egységre a felső szélétől.
- A négyzetek között legyen 5 egységnyi térköz.
- A legelső négyzet színe legyen `rgb(255, 79, 120)`.
- Négyzetenként csökkentsd az `rgb()` szín vörös komponensének értékét 7-tel, az előző négyzethez képest.
- Soronként növeld az `rgb()` szín kék komponensének értékét 15-tel, az előző sorhoz képest.

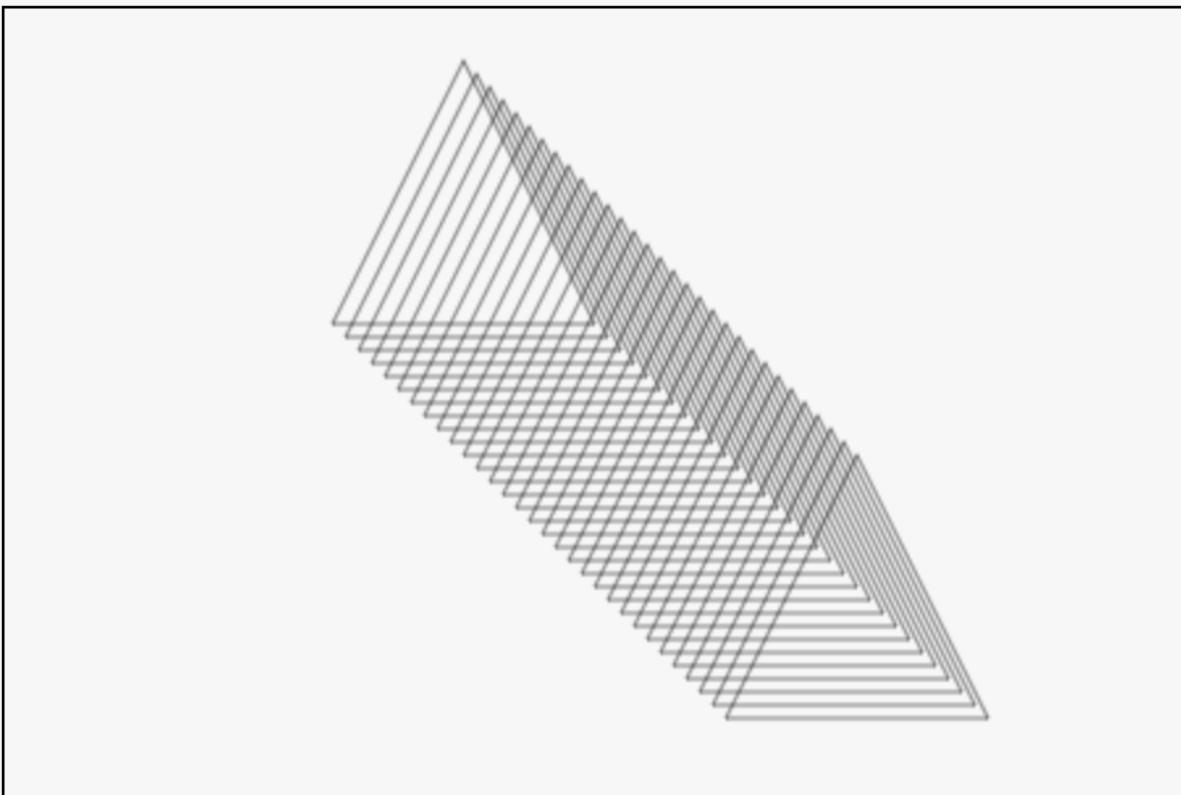
Tipp: Amikor hozzárendeled az `rgb()` színt a `fillStyle` tulajdonsághoz, használj összefűzést (angolul concatenation).

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.4. Háromszögalagút

A feladatod, hogy rajzolj 30 szürke háromszöget a **canvas**-ra.



A programod a következő kritériumoknak feleljen meg:

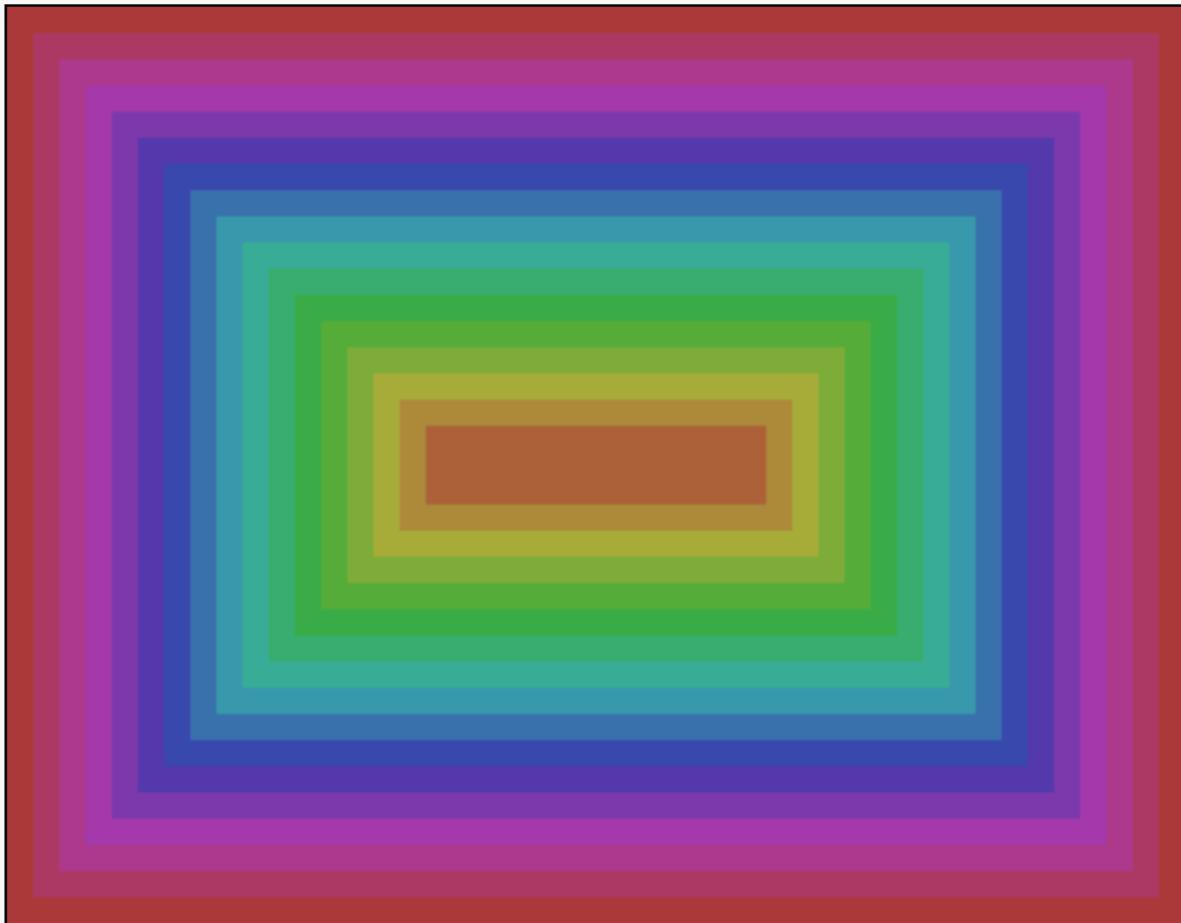
- Rajzolj harminc háromszöget úgy, hogy a `moveTo()` és `stroke()` függvényeket csak egyszer írod le a kódodban, a `lineTo()` függvényt pedig legfeljebb háromszor.
- Amikor meghívod a függvényeket, használj változókat a koordináták kiszámításához.
- A háromszögek alapja és magassága legyen **100** egységnyi (a háromszögek legyenek **100** egység szélesek és **100** egység magasak).
- Az első háromszög bal alsó csúcsa **120** egységre helyezkedjen el a szélektől.
- minden ezt követő háromszöget **5** egységgel tolj el minden két koordinátatengely mentén az előző háromszöghöz képest.
- A háromszögek körvonala legyenek szürkék.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

3.5. Szivárványszínű téglalapok

A feladatod, hogy rajzolj 17 színes téglalapot a `canvas`-ra.



A programod a következő kritériumoknak feleljön meg:

- Rajzolj 17 téglalapot úgy, hogy csak egyszer írod le a kódodban a `fillRect()` függvényt.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a téglalapok pozíciójának és méretének megadásához.
- Az első téglalap teljesen töltse ki a `canvas`-t.
- minden további téglalap összes csúcsa legyen az előző téglalap csúcsaihoz képest 10 egységgel beljebb pozicionálva a `canvas`-on.
- Az első téglalap színe legyen `hsl(360, 60%, 45%)`.
- Téglalaponként csökkentsd a `hsl()` szín `hue` értékét (az első paraméter értékét) 360/17-del, az előző téglalaphoz képest.

Megjegyzés: A `hsl()` függvény a szín megadásának egy további módja. Hárrom paraméterrel rendelkezik: `hsl(hue, saturation, lightness)`. A `hue` (színárnyalat) értéke egy 0 és 360 közé eső szám, és egy színkörön határoz meg egy fokot. A `saturation` (telítettség) és `lightness` (világosság) értékét százalékban kell meghatározni.

Tipp: Amikor hozzárendeled a `hsl()` színt a `fillStyle` tulajdonsághoz, használj

összefűzést (angolul concatenation).

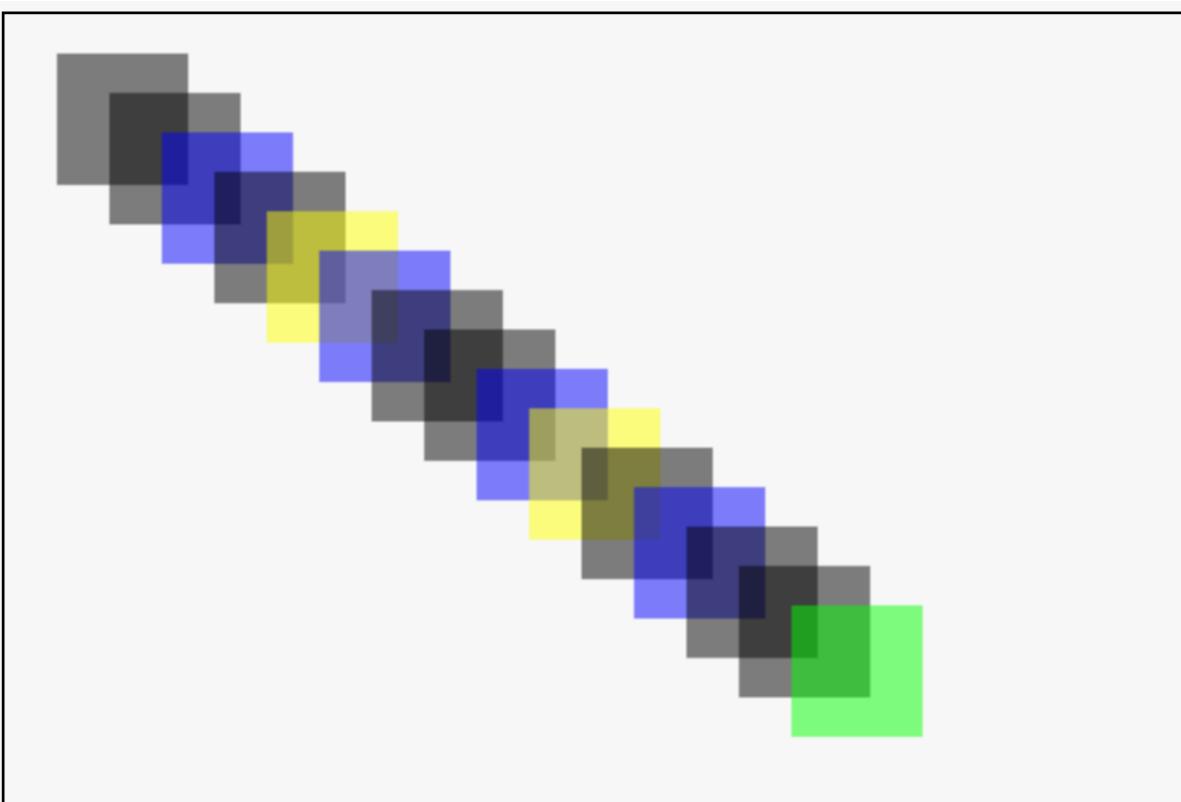
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4. Feltételek gyakorlása

4.1. Fizbuzz négyzetek

A feladatod, hogy rajzolj 15 egymást részben átfedő négyzetet a `canvas`-ra.



A programod a következő kritériumoknak feleljén meg:

- Rajzolj 15 négyzetet úgy, hogy csak egyszer írod le a kódodban a `fillRect()` függvényt.
- Amikor meghívod a `fillRect()` függvényt, kizárolag változókat használj a négyzetek pozíójának kiszámításához és méretének megadásához.
- minden négyzet legyen 50 egység széles és 50 egység magas.
- Az első négyzet a `canvas` bal felső sarkában helyezkedjen el, 20 egységre a `canvas` bal szélétől és 15 egységre a felső szélétől.
- minden ezt követő négyzetet 20 egységgel tolj el a vízszintes tengely mentén és 15 egységgel a függőleges tengely mentén, az előző négyzethez képest.
- Ha az aktuális négyzet sorszáma:
 - osztható 3-mal, akkor a négyzet színe legyen félig átlátszó kék

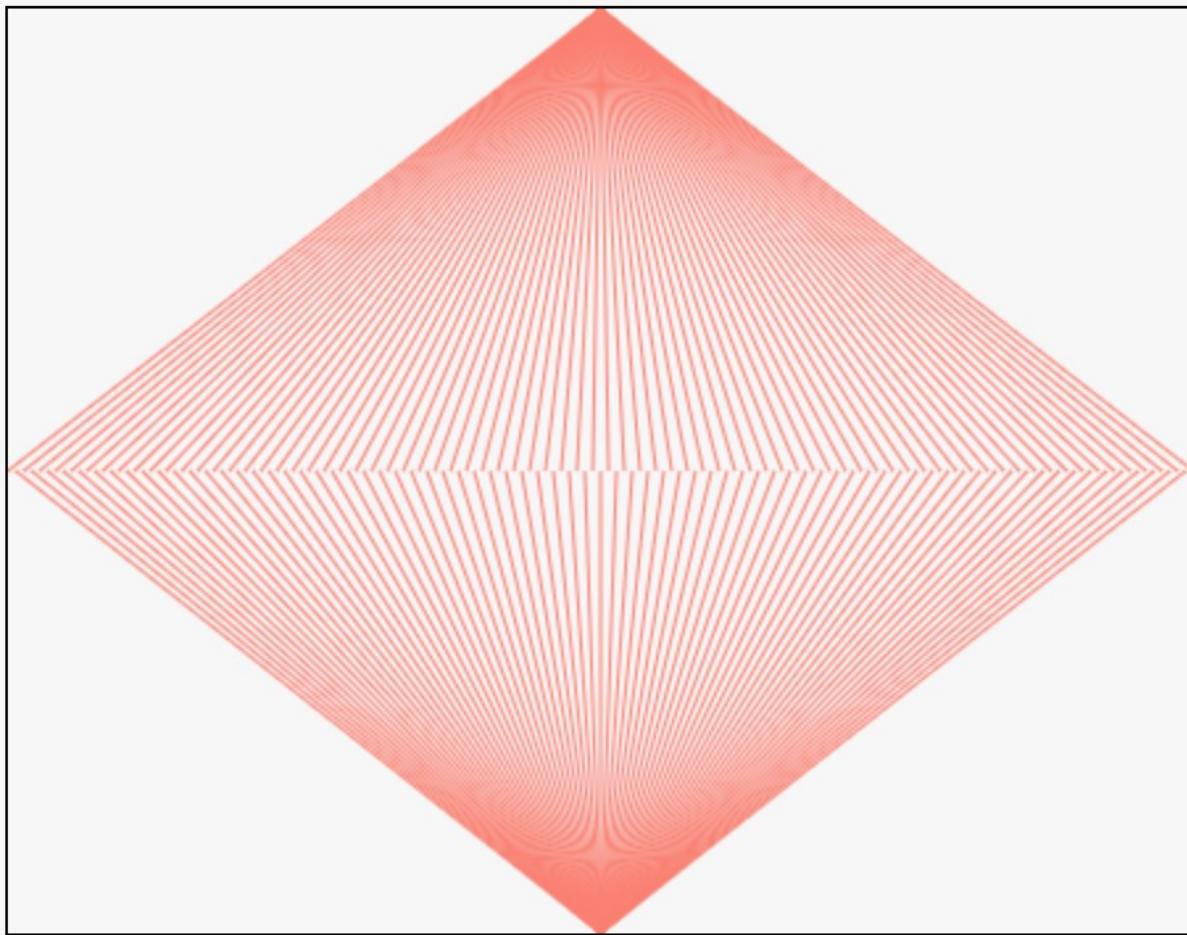
- (`rgba(0, 0, 255, .5)`) (például a 3. négyzet legyen kék).
- o osztható 5-tel, akkor a négyzet színe legyen fél átlátszó sárga (`rgba(255, 255, 0, .5)`) (például az 5. négyzet legyen sárga).
- o osztható 3-mal és 5-tel is, akkor a négyzet színe legyen fél átlátszó zöld (`rgba(0, 255, 0, .5)`) (azaz a 15. négyzet legyen zöld).
- o nem osztható sem 3-mal, sem 5-tel, akkor a négyzet színe legyen fél átlátszó fekete (`rgba(0, 0, 0, .5)`).

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4.2. Páratlan piramis

A feladatod, hogy a `canvas` vízszintes felezővonalától 3 egységenként húzz egy vonalat.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj vonalakat úgy, hogy a `moveTo()` és `stroke()` függvényeket csak egyszer írod le a kódodban, a `lineTo()` függvényt pedig legfeljebb kétszer.
- Amikor meghívod a függvényeket, használj változókat a koordináták kiszámításához.
- Az első vonal a `canvas` bal szélénél közepétől húzódjon.

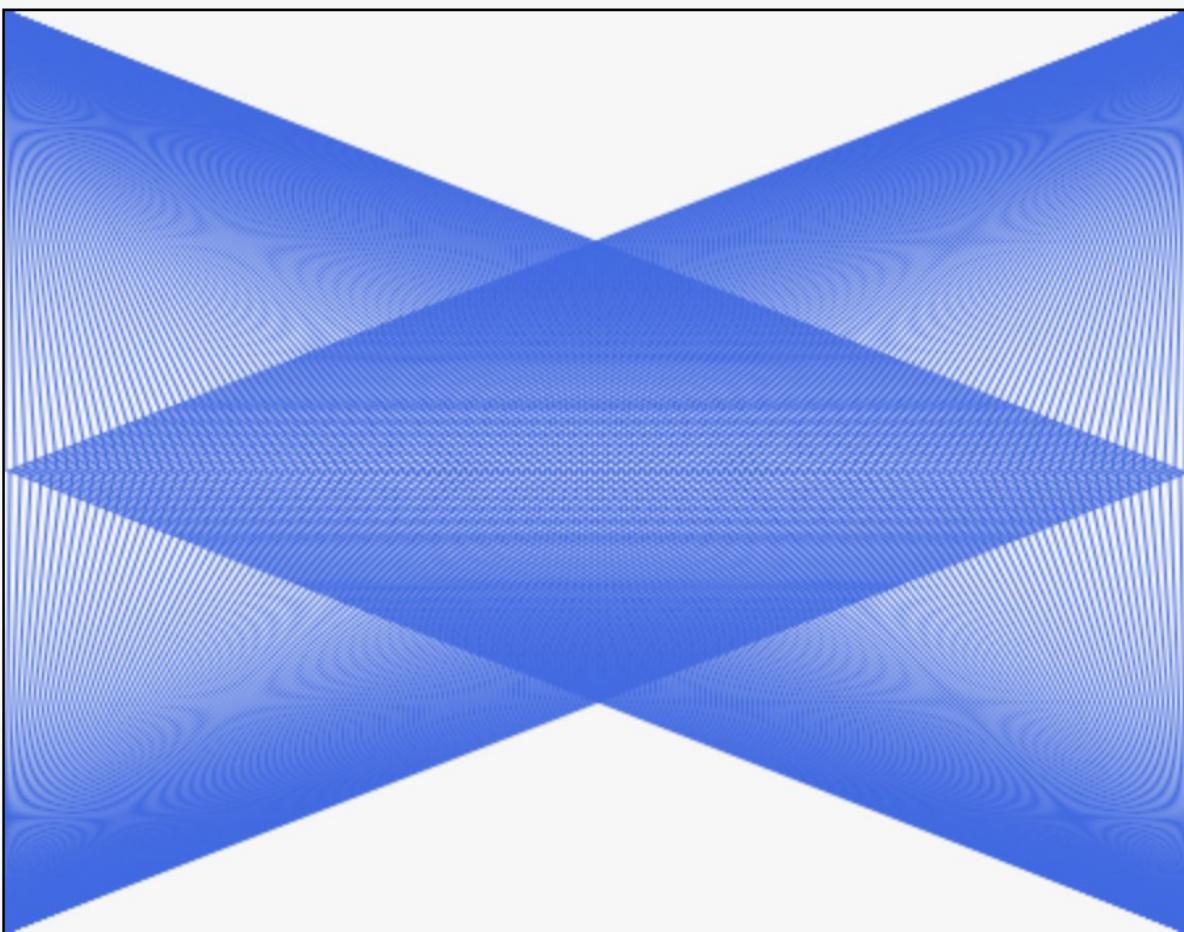
- minden ezt követő vonal kezdőpontját 3 egységgel told el az előző vonalhoz képest, a vízszintes felezővonal mentén.
- Az utolsó vonal a `canvas` jobb szélénél közepétől húzódjon.
- A vonal végpontja legyen:
 - a `canvas` felső szélénél közepé, ha a kezdőpont x koordinátája páros szám;
 - a `canvas` alsó szélénél közepé, ha a kezdőpont x koordinátája páratlan szám.
- A vonalak színe legyen félig átlátszó piros (`rgba(255, 0, 0, .5)`).

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

4.3. Mennyi háromszög

A feladatod, hogy a `canvas` vízszintes felezővonalától egységenként húzz egy vonalat.



A programod a következő kritériumoknak feleljen meg:

- Rajzolj vonalakat úgy, hogy a `moveTo()` és `stroke()` függvényeket csak egyszer írod le a kódodban, a `lineTo()` függvényt pedig legfeljebb négyeszer.
- Amikor meghívod a függvényeket, használj változókat a koordináták kiszámításához.

- Az első vonal a `canvas` bal szélénél közepétől húzódjon.
- minden ezt követő vonal kezdőpontját 1 egységgel told el az előző vonalhoz képest, a vízszintes felezővonal mentén.
- Az utolsó vonal a `canvas` jobb szélénél közepétől húzódjon.
- Ha a kezdőpont x koordinátája 4-gyel osztva:
 - 0-t ad maradék, akkor a vonal végpontja a `canvas` bal felső sarka legyen;
 - 1-et ad maradék, akkor a vonal végpontja a `canvas` jobb felső sarka legyen;
 - 2-t ad maradék, akkor a vonal végpontja a `canvas` bal alsó sarka legyen;
 - 3-at ad maradék, akkor a vonal végpontja a `canvas` jobb alsó sarka legyen.
- A vonalak színe legyen fél átlátszó kék (`rgba(0,0,255,.5)`).

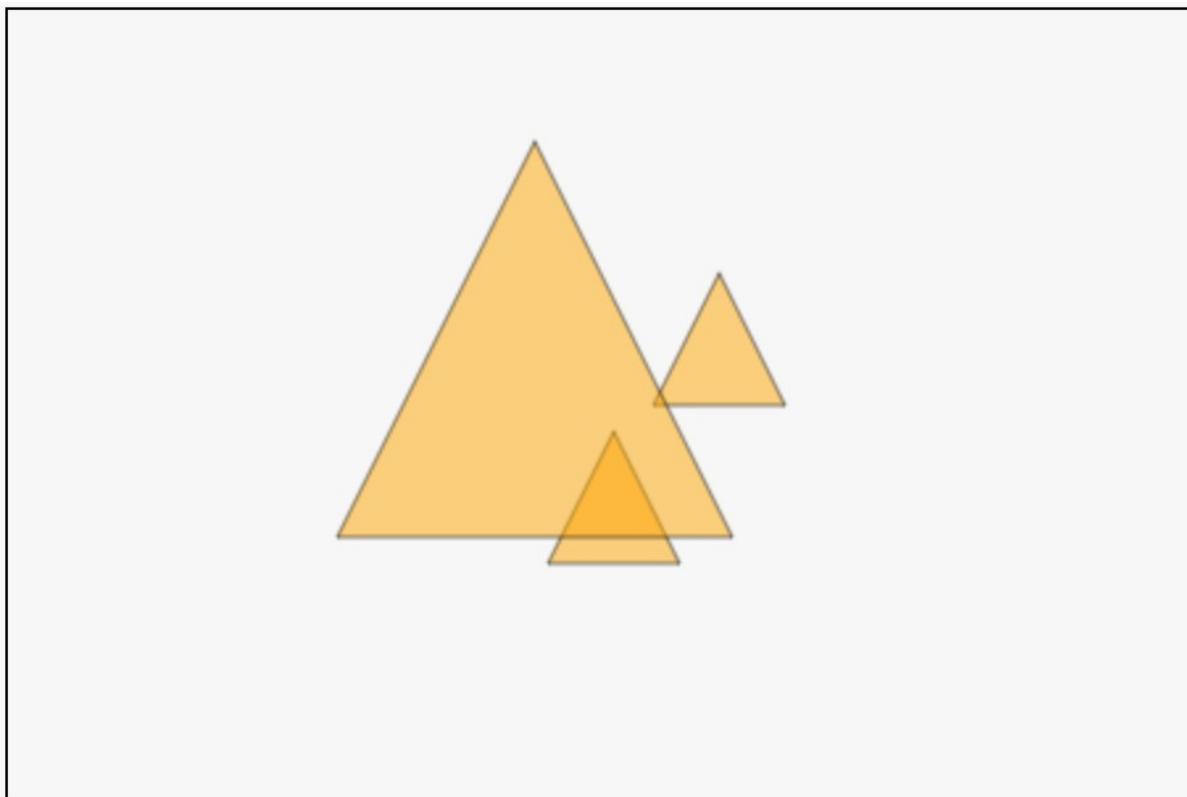
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

5. Függvények gyakorlása:

5.1. triangle()

A feladatod, hogy rajzolj három háromszöget a `canvas`-ra.



A programod a következő kritériumoknak feleljén meg:

- Hozz létre egy `triangle(positionX, positionY, size)` függvényt, amely a meghívásakor egy háromszöget rajzol.
- A függvény három paramétert fogad: a `positionX` és `positionY` a háromszög felső csúcsának a koordinátái, a `size` pedig a háromszög alapjának (szélességének) és magasságának a mérete.
- Használd ezeket a paramétereket a koordináták kiszámításához a `moveTo()` és `lineTo()` függvényekben.
- A vonalak színe legyen fél átlátszó fekete (`rgba(0,0,0,.5)`).
- A háromszöget kitöltő szín legyen fél átlátszó narancssárga (`rgba(255,165,0,.5)`).
- Rajzolj három háromszöget a függvényed háromszori meghívásával:

```
triangle(230, 160, 50);
triangle(270, 100, 50);
triangle(200, 50, 150);
```

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

5.2. star()

A feladatod, hogy rajzolj három csillagot a `canvas`-ra.



A programod a következő kritériumoknak feleljön meg:

- Hozz létre egy `star(positionX, positionY, size)` függvényt, amely a meghívásakor egy csillagot rajzol.
- A függvény három paramétert fogad: a `positionX` és `positionY` a csillag bal felső csúcsának a koordinátái, a `size` pedig a csillag szélessége.
- Használd ezeket a paramétereiket a koordináták kiszámításához a `moveTo()` és `lineTo()` függvényekben.
- A csillag legfelső csúcsa `size * 0.4` távolságra helyezkedik el a bal és jobb felső csúcsok felett, a bal és jobb alsó csúcsok pedig `size * 0.5` távolságra alattuk.
- A csillag legfelső csúcsa félúton helyezkedik el a bal és jobb felső csúcsok között, a bal és jobb alsó csúcsok pedig `size * 0.15` távolságra tőlük.
- A vonalak színe és a csillagot kitöltő szín legyen `rgb(233, 159, 184)`.
- Rajzolj három csillagot a függvényed háromszori meghívásával:

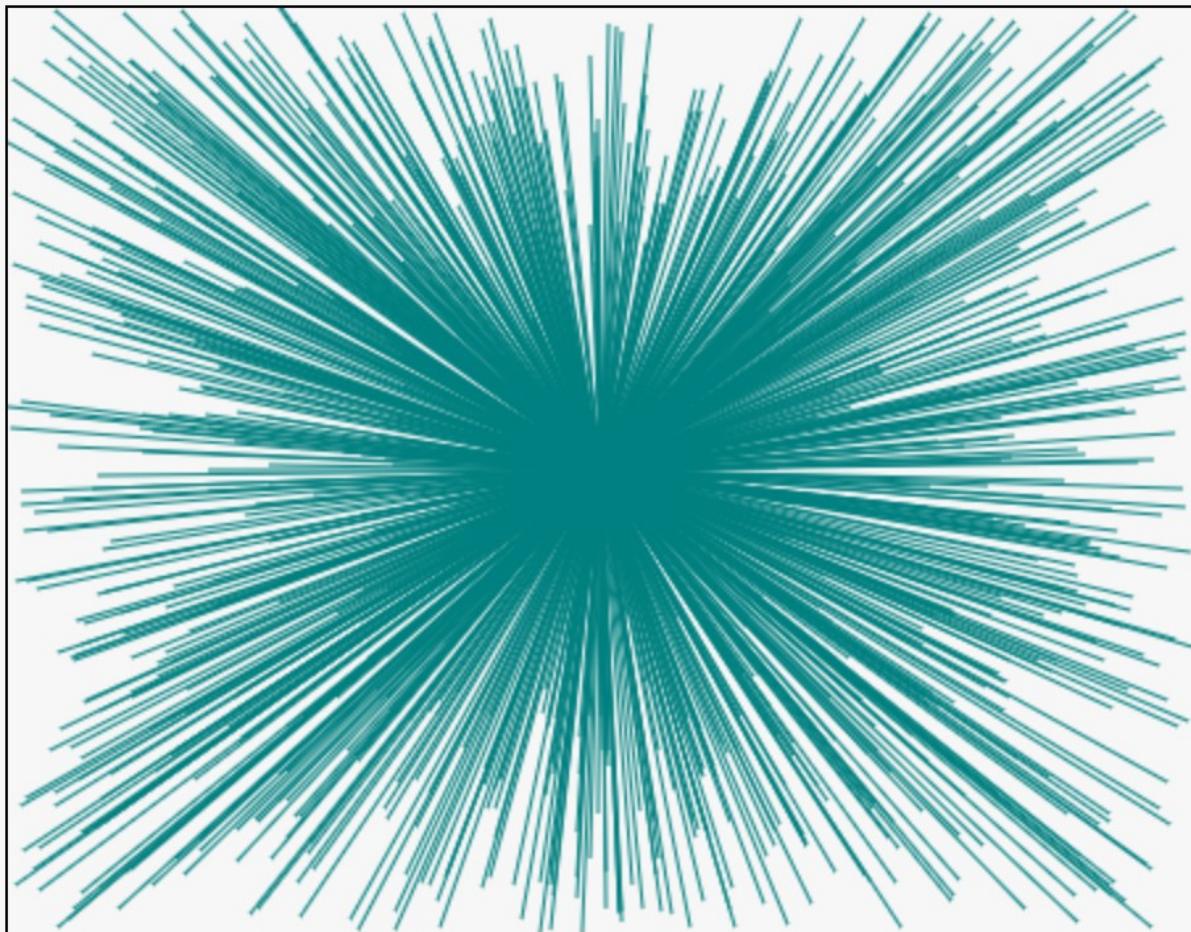
```
star(40, 50, 75);
star(130, 120, 100);
star(250, 220, 150);
```

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

5.3. `lineToCenter()`

A feladatod, hogy húzz 1000 vonalat a `canvas` középpontjába.



A programod a következő kritériumoknak feleljen meg:

- Hozz létre egy `lineToCenter(positionX, positionY, color)` függvényt, amely a meghívásakor húz egy vonalat a `canvas` középpontjába.
- A függvény három paramétert fogad: a `positionX` és `positionY` a vonal kezdőpontjának a koordinátái, a `color` pedig a vonal színe.
- Rajzolj 1000 vonalat úgy, hogy egy ciklus segítségével 1000-szer meghívod a függvényedet.
- A függvény minden meghívásakor változzon a vonal kezdőpontja a `canvas` egyik véletlenszerű pontjáról egy másikra.
- A véletlenszám-generáláshoz használd az alábbi függvényt:

```
function random(max, min) {  
    return Math.floor(Math.random() * (max - min + 1) + min);  
}
```

- Amikor meghívod a `random(max, min)` függvényt, használd a `canvas` méreteit argumentumként.

A vonalak színe legyen '`teal`'.

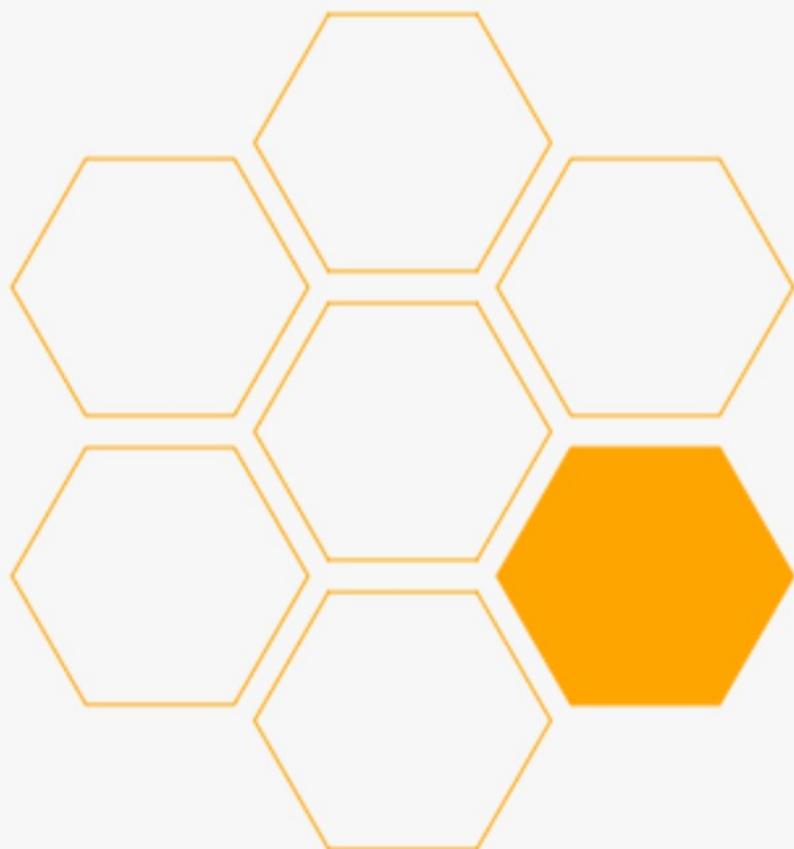
Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

6. Haladó gyakorlatok

6.1. A JavaScript telepesei

A feladatod, hogy rajzolj 7 hatszöget a `canvas`-ra.



A programod a következő kritériumoknak feleljen meg:

- Hozz létre egy `drawHexagon(positionX, positionY)` függvényt, amely a meghívásakor rajzol egy hatszöget a `canvas`-ra.
- A függvény két paramétert fogad: a `positionX` és `positionY` a hatszög legbaloldalibb csúcsának a koordinátái.
- A hatszög szélessége `113` egység, a magassága pedig `98` egység.
- A vonalak színe legyen '`orange`'.
- Rajzolj 7 hatszöget a `canvas`-ra a függvényed alábbi módon történő meghívásával:

```
drawHexagon(76, 120);
drawHexagon(76, 230);
drawHexagon(168.5, 65);
drawHexagon(168.5, 175);
drawHexagon(168.5, 285);
drawHexagon(261, 120);
drawHexagon(261, 230);
```

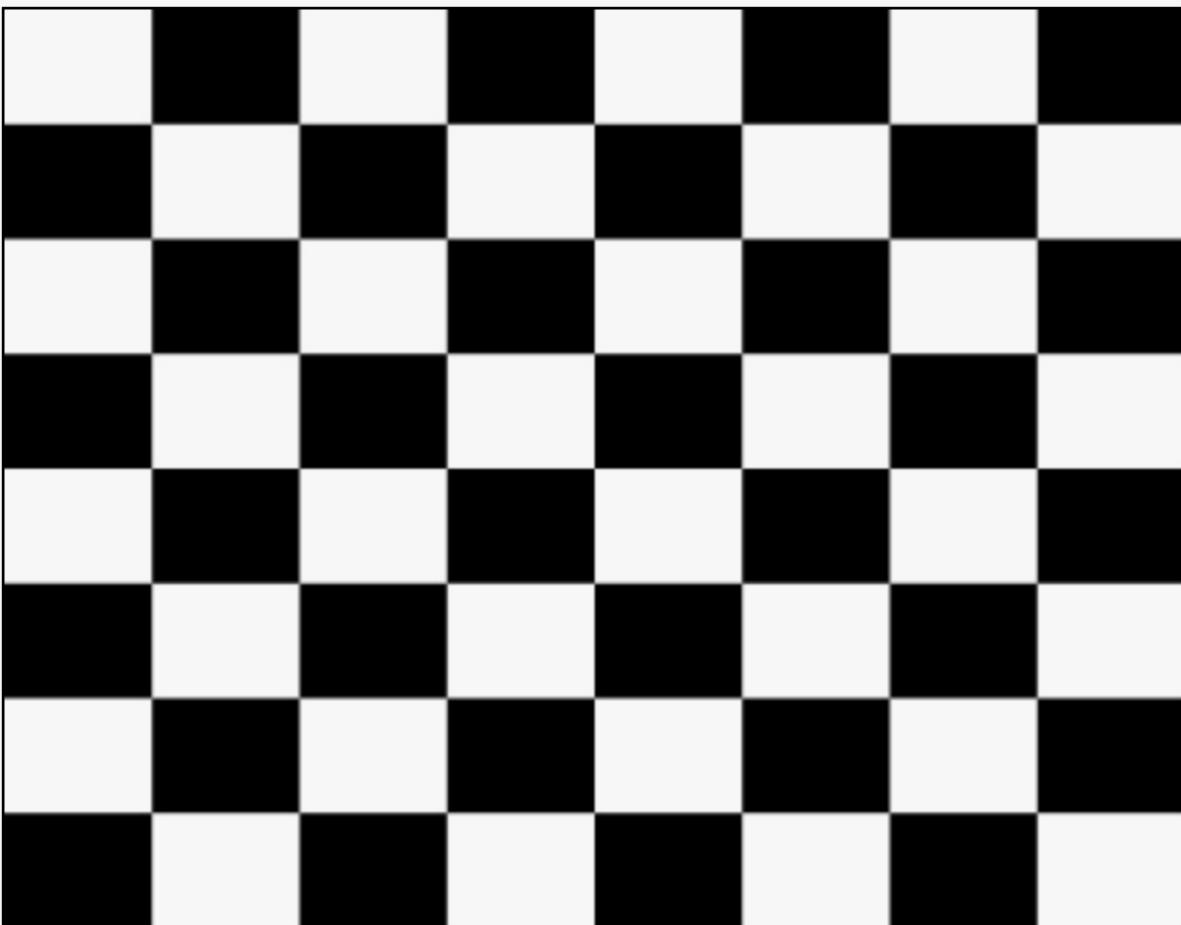
- Az utolsó hatszöget töltsd ki 'orange' színnel.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

6.2. drawCheckeredPattern()

A feladatod, hogy sakktáblamintát rajzolj a **canvas**-ra.



A programod a következő kritériumoknak feleljen meg:

- Hozz létre egy **drawCheckeredPattern(row, col)** függvényt, amely a

meghívásakor sakktáblamintát rajzol a `canvas`-ra.

- A függvény két paramétert fogad: a `row` a sorok száma, a `col` pedig az oszlopok száma.
- Rajzolj egy 8x8-as rácsot a `canvas`-ra a függvényed alábbi módon történő meghívásával:

```
drawCheckeredPattern(8,8);
```

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

6.3. Tetractys

A feladatod, hogy rajzolj 6 háromszöget 3 sorba a `canvas`-on.



A programod a következő kritériumoknak feleljen meg:

- Hozz létre egy `drawTriangle(positionX, positionY)` függvényt, amely a meghívásakor 6 háromszöget rajzol 3 sorba a `canvas`-on.
- A függvény két paramétert fogad: a `positionX` és `positionY` az első háromszög

- felső csúcsának a koordinátái.
- A háromszög alapja **100** egységnyi, a magassága pedig **86.6** egységnyi (a háromszög **100** egység széles és **86.6** egység magas).
 - A vonalak színe és a háromszöget kitöltő szín legyen:
 - **rgb(227, 98, 102)** az első sorban,
 - **rgb(38, 172, 73)** a második sorban,
 - **rgb(34, 128, 128)** a harmadik sorban.
 - Rajzolj 6 háromszöget 3 sorba a **canvas**-on, a függvényed alábbi módon történő meghívásával:

```
drawTriangle(225, 33);
```

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

JavaScript alapok VI. (tömbök)

1. Mi az a tömb?

1.1. Bevezetés

A hétköznapi életünkben gyakran használunk listákat, hogy megjegyezzük bizonyos információkat. Például ha vásárolni mész, listát írsz a dolgokról, amiket venni akarsz, nehogy elfelejts valamit: kenyér, tej, alma.

Az információk tárolásához JavaScriptben is listákra van szükséged.

Ebben a projektben a következőket fogod megtanulni:

- hogyan néz ki egy lista a JavaScriptben,
- hogyan kezelheted a listát elemek hozzáadásával, eltávolításával vagy módosításával, és
- hogyan könnyíthetik meg ezek a listák a ciklusokkal végzett munkád.

Készen állsz? Akkor vággunk bele.

1.2. Tömb írása

Tegyük fel, hogy vásárolni mész. Szeretnél venni egy kiló kenyeret, egy kis tejet és pár almát. Ezeket elég könnyű listába foglalni magyarul – egyszerűen leírod a szavakat egymás után, és vesszővel elválasztod őket:

kenyér, tej, alma

A JavaScriptben készített listák is nagyon hasonlóan néznek ki. Ugyanígy kell leírnod a felsorolást, csak szögletes zárójelet is írsz köré:

[kenyér, tej, alma]

Természetesen a stringek – ahogy azt már az előző JavaScript-leckékben említettük – félidezőjelbe (single quote) kerülnek:

['kenyér', 'tej', 'alma']

Megjegyzés: Ne feledd, hogy a szöveges értékek minden stringnek számítanak a JavaScriptben.

És ezzel már meg is írtad az első listádat JavaScriptben. Gratulálok!

Azt még fontos tudni, hogy JavaScriptben a listákat tömbnek (angolul array-nek) hívják, szóval igazából az első tömböt írtad meg.

Feladat

A Trónok harca egy nagyon jó sorozat, nem? Sorold fel a Hét Királyság nemesi házainak nevét egy tömbben.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek kilenc eleme van.
- Szintaktikailag helyes.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

1.3. Tömb elmentése

Többnyire azért írunk listákat, hogy később is elővehessük őket, így fontos, hogy el is tudjuk menteni. Hiába írsz bevásárlólistát, ha nem mented el a telefonodon, és nem tudod a boltban megnézni.

A tömbökkel ugyanez a helyzet JavaScriptben. Általában érdemes elmenteni a létrehozott tömböket.

Szerencsére azt már megtanultad, hogy ezt hogyan kell csinálni. Emlékszel a változókra?

Már volt róla szó, hogy a JavaScriptben használhatsz változókat különböző dolgok elnevezésére. Például számok, szövegek stb. esetében. És igen, a tömböknél is.

Ha egy tömböt elmentesz egy változóba, később az adott változó segítségével hivatkozhatsz rá.

Lássuk hogy működik ez a valóságban. Mentsük el a bevásárlólistánkat (`['kenyér', 'tej', 'alma']`) egy `shoppingList` nevű változóba.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['kenyér', 'tej', 'alma'];
```

A konzol a következőt adta vissza: `undefined`. Remek, ez azt jelenti, hogy sikeresen deklaráltad a változót. Mostantól a változó nevével hivatkozhatsz a tömbre.

Gépeld be ezt a konzolba, és nyomd meg az Entert:

```
shoppingList;
```

Ahogy várható volt, a tömböt adja vissza eredményként:

```
[ "kenyér", "tej", "alma" ]
```

És ezzel meg is volnánk. Most már azt is tudod, hogyan kell a tömböket változókba elmenteni.

Feladat

Ha már úgyis felsoroltad a Trónok harca nemesi házait, érdemes lenne el is menteni. Mentsd el a tömböt egy változóba.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek kilenc eleme van.
- A tömb egy `gameOfThronesHouses` nevű változóba van elmentve.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

1.4. Adatok tárolása tömbökben

Ahogy az bármilyen listától elvárható, a JavaScript-tömbökben is rengetegféle információt lehet tárolni.

Azt már láttad, hogyan tárolhatunk stringeket egy tömbben:

```
var shoppingList = [ 'kenyér', 'tej', 'alma' ];
```

De számokat is lehet tömbben tárolni:

```
var evenNumbers = [ 2, 4, 6 ];
```

Megjegyzés: A tömbök másfajta adatok tárolására is alkalmasak, legyenek azok más tömbök, változók vagy adatszerkezetek (más néven objektumok). Ezzel most még nem érdemes foglalkoznod, ha eljött az ideje, ezekkel is megismeredhetsz.

Feladat

A Fibonacci-sorozat egy olyan számsorozat, amely a természetben is rengetegszer előfordul. Sorold fel a sorozat első tíz számát egy tömbben, és mentsd el a tömböt egy változóba. Ha még nem hallottál a Fibonacci-számokról, keress rá a Wikipédián.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek tíz eleme van.
- A tömb egy `fibonacciSequence` nevű változóba van elmentve.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2. Elemek hozzáadása és törlése egy tömb elejéről vagy végéről

2.1. Új elem hozzáadása egy tömb végéhez

Tegyük fel, hogy úton a bolt felé eszedbe jut, hogy tojást is kéne venned. Egyelőre a bevásárlólistádon csak a kenyér, a tej és az alma szerepel. Úgyhogy megállsz, előveszed a listát, és felírod a tojást is.

A listák folyamatosan fejlődnek – menet közben is bármikor kihúzhatsz és hozzáadhatsz elemeket. Természetesen ez a JavaScript-tömbök esetében is igaz. Hozzáadhatsz új elemeket, és törölheted vagy módosíthatod a már meglévő elemeit.

Elsőként nézzük, hogyan adhatsz hozzá új elemeket. Először a tömb végéhez fogunk hozzáadni egy új elemet.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['kenyér', 'tej', 'alma'];
```

Ahhoz, hogy hozzáadd a tojást a tömb végéhez, gépeld be a következőt a konzolba, és nyomd meg az Entert:

```
shoppingList.push('tojás');
```

A konzol a következőt adta vissza eredményként: 4. De miért? Mert a tömb most már 4 elemből áll. A `.push()` metódus a tömbben található elemek számát adja vissza az új elem hozzáadását követően.

Anélkül, hogy túlságosan műszakilag fogalmaznánk, nézzük meg közelebbről a `.push()` metódust. Ezt a metódust bármilyen tömbnél alkalmazhatjuk, ha egy vagy több új elemet akarunk hozzáadni a tömb végéhez.

Az általános szintaxis a következő:

```
nameOfTheArray.push(newElement);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `newElement` az új elem, amit a tömb végéhez akarsz adni.

Ha le szeretnéd ellenőrizni, hogy a `.push()` metódus működött-e, csak írd be a tömb nevét a konzolba, és nyomd meg az Enter-t:

```
shoppingList;
```

Ez várható módon a tömb legfrissebb verzióját fogja visszaadni:

```
["kenyér", "tej", "alma", "tojás"]
```

Látod? A tojás tényleg megjelent a bevásárlólistád végén.

Azt már tudod, hogyan lehet hozzáadni egy új elemet a tömb végéhez. A következő feladatban azt is megmutatom neked, hogyan lehet egyszerre több új elemet hozzáadni a tömbhöz.

Feladat

Biztosan van néhány film, amelyet akárhányszor meg tudnál nézni, és mégsem unod meg. Sorold fel három kedvencedet egy tömbben! De mi a helyzet, ha egy negyedik filmet is hozzáadnál a listához? Mutasd meg, mit tanultál ebben a leckében, és adj egy negyedik elemet a tömb végéhez.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek három eleme van.
- A tömb egy `myFavoriteMovies` nevű változóba van elmentve.
- Adj hozzá egy új elemet a tömb végéhez.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.2. Több új elem hozzáadása egy tömb végéhez

Tegyük fel, hogy a lista végéhez még hozzá szeretnéd adni a rizst, a téstát és a babot. A valós életben ez nagyon egyszerű: csak leírod egyiket a másik után.

JavaScriptben hozzá tudsz adni új elemeket a listához egyesével, külön utasításokban. Ez azonban elég sok időt vesz igénybe. Szerencsére arra is van mód, hogy egyetlen sornyi kódban hozzáadhasd az összes új elemet a tömbhöz.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Enter-t:

```
var shoppingList = ['kenyér', 'tej', 'alma', 'tojás'];
```

Ahhoz, hogy hozzáadd a rizst, tésztát és babot a tömb végéhez, írd be a következőt a konzolba, és nyomd meg az Entert:

```
shoppingList.push('rizs', 'tészta', 'bab');
```

A `.push()` metódussal egyszerre több elemet is hozzáadhatsz a tömbhöz. Csupán annyit kell tenned, hogy a zárójelben lévő új elemeket elválasztod egy-egy vesszővel.

A `.push()` metódus használata után a konzol a 7-es számot adta vissza eredményül, ami – ahogyan azt már te is tudod – a frissített tömbben lévő elemek száma.

A tömb megtekintéséhez írd be a konzolba, hogy `shoppingList`;, és nyomd meg az Entert. Láthatod, hogy a bevásárlólistád végén megjelent a rizs, a tésztá és a bab is:

```
["kenyér", "tej", "alma", "tojás", "rizs", "tészta", "bab"]
```

Szuper! A `.push()` metódus általános szintaxisa, amivel több új elemet tudsz egyszerre hozzáadni a tömbhöz:

```
nameOfTheArray.push(newElement1, newElement2, newElement3);
```

Mostanra megtanultad, hogyan lehet egy vagy több új elemet hozzáadni egy tömb végéhez. A következő feladatban megvizsgáljuk a tömb másik oldalát is: megmutatjuk, hogyan tudsz hozzáadni új elemeket a tömb elejéhez.

Feladat

Ahogy azt a neves dán író és költő, Hans Christian Andersen mondta, „Utazni annyi mint élni”. Ezzel a gondolattal gyűjts össze egy tömbbe legalább három olyan országot, ahol már jártál (vagy ahova szívesen utaznál), és mentsd el a tömböt egy változóba. Ugyanakkor biztos vagyok benne, hogy van még jó pár olyan hely, ahova szívesen elutaznál. Adj hozzá három ilyen helyet a lista végéhez.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek három eleme van.
- A tömb egy `placesToVisit` nevű változóba van elmentve.
- Adj hozzá három új elemet a tömb végéhez.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.3. Új elemek hozzáadása egy tömb elejéhez

Tegyük fel, hogy rájöttél, hogy mégis a csokoládé a legfontosabb doleg, amit be kell szerezned. Ezért ezt a bevásárlólistád elejére szeretnéd tenni, hogy akkor is biztosan meg tudsz venni, ha esetleg nem maradna mindenre pénzed.

Most megtanulhatod, hogyan adhatsz hozzá új elemet a tömb elejéhez.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['kenyér', 'tej', 'alma', 'tojás', 'rizs',  
'tészta', 'bab'];
```

Ahhoz, hogy a csokoládét a tömb elejéhez adhasd hozzá, a következőt kell beínod a konzolba:

```
shoppingList.unshift('csoki');
```

A konzol a 8-as számot fogja visszaadni eredményként, mert az adott tömb most már 8 elemből áll.

Ha meg szeretnél róla bizonyosodni, hogy az új téTEL tényleg hozzáadódott a tömb elejéhez, írd be a konzolba, hogy `shoppingList`; , és nyomd meg az Entert. A konzol a frissített tömböt fogja visszaadni:

```
["csoki", "kenyér", "tej", "alma", "tojás", "rizs", "tészta", "bab"]
```

Remek! Sikeresen hozzáadtál egy új elemet a bevásárlólistád elejéhez.

Ehhez az `.unshift()` metódust használtad, ami nagyon hasonlít a `.push()` metódushoz. Az egyetlen különbség, hogy az `.unshift()` nem a tömb végéhez, hanem az elejéhez ad hozzá egy vagy több új elemet. Az `.unshift()` metódus általános szintaxisa:

```
nameOfTheArray.unshift(newElement);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `newElement` az új elem, amit a tömb elejéhez akarsz adni.

Ahogy a `.push()` metódusnál, úgy az `.unshift()` metódusnál is hozzáadhatsz egyszerre akár több elemet is a tömbhöz:

```
nameOfTheArray.unshift(newElement1, newElement2, newElement3);
```

Mostanra megtanultad, hogyan lehet egy vagy több új elemet hozzáadni egy tömb végéhez

vagy elejéhez. Gratulálunk! A következő feladatban megmutatjuk, hogyan tudsz egy elemet törölni a tömbből.

Feladat

Szereztél mostanában új barátokat? Sorold fel a nevüket egy tömbben, és mentsd el a tömböt egy változóba. Ha ez megvan, add hozzá a tömb elejéhez a legrégebbi barátaid nevét.

A programod a következő kritériumoknak feleljön meg:

- Egy tömbből áll, melynek legalább két eleme van.
- A tömb egy `myFriends` nevű változóba van elmentve.
- Adj hozzá három új elemet a tömb elejéhez.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.4. Az utolsó elem törlése

Már tudod, hogy ha új elemeket szeretnél hozzáadni egy tömbhöz, a következőket használhatod:

- a `.push()` metódust, ha a tömb végéhez szeretnél hozzáadni új elemet, és
- az `.unshift()` metódust, ha a tömb elejéhez szeretnél hozzáadni új elemet.

Az elemek törlése a tömb elejéről vagy végéről nagyon hasonlóan működik. A következőket kell hozzá használnod:

- a `.pop()` metódust, ha a tömb végéről szeretnél törölni egy elemet, vagy
- a `.shift()` metódust, ha a tömb elejéről szeretnél törölni egy elemet.

Példaként adjuk hozzá a bevásárlólistánk végéhez a teát a `.push()` metódussal, majd töröljük a `.pop()` metódussal.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['csoki', 'kenyér', 'tej', 'alma', 'tojás',  
'rizs', 'téeszta', 'bab'];
```

A tea hozzáadásához írd be a következőt a konzolba, majd nyomd meg az Entert:

```
shoppingList.push('tea');
```

A konzol a 9-es számot dobta vissza, ami a frissített tömbben szereplő tételek száma. Most

írd be a konzolba, hogy `shoppingList`; , és nyomd meg az Entert, hogy leellenőrizd a frissített tömböt:

```
[ "csoki", "kenyér", "tej", "alma", "tojás", "rizs", "tészta", "bab",  
"tea" ]
```

Ahhoz, hogy töröld a teát a tömb végéről, írd be a következőt a konzolba, és nyomd meg az Entert:

```
shoppingList.pop();
```

Biztosan feltűnt, hogy a szintaxis itt egy kicsit eltér a `.push()` és a `.unshift()` metódusban megszokottaktól: itt semmit sem kell írnod a zárójelekbe. De miért? Mert a `.pop()` metódussal csak egyetlen elemet lehet törölni a tömbből: a legutolsót.

Most a konzol azt adta vissza, hogy "`tea`" – azt a tételelt, amit töröltünk. A tömb utolsó elemének eltávolítása után a `.pop()` metódus a törölt elemet adja vissza. Ez azért hasznos, mert így a törölt tételelt elmenthetjük egy változóba, és később is használhatjuk, ha szükség van rá.

Megjegyzés: Ha szeretnéd elmenteni a törölt elemet egy változóba, ezt megteheted az elem eltávolításakor: `var deletedFromEnd = shoppingList.pop();`. Ez a kód egyszerre törli a tömb legvégén lévő elemét, és rendeli hozzá egy változóhoz.

Hogy megbizonyosodj arról, hogy a teát tényleg töröltük, írd be a következőt a konzolba, és nyomd meg az Entert:

```
shoppingList;
```

Ez a frissített tömböt fogja visszaadni:

```
[ "csoki", "kenyér", "tej", "alma", "tojás", "rizs", "tészta", "bab" ]
```

Hurrá, sikerült! A következő feladatban kipróbáljuk a `.shift()` metódust is.

Feladat

A barátaiddal vagy a családoddal társasjátékozni remek szórakozás. Sorold fel az általad ismert társasjátékokat egy tömbben, a legjobbtól a legrosszabbig haladva sorban, és mentsd el a tömböt egy változóba. Ha kész vagy, töröld az utolsó tételelt, azaz a legrosszabb társasjátékot, amelyet ismersz.

A programod a következő kritériumoknak feleljön meg:

- Egy tömbből áll, melynek legalább négy eleme van.
- A tömb egy `boardGames` nevű változóba van elmentve.
- Töröld a tömb utolsó elemét a `.pop()` metódussal.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.5. Az első elem törlése

A `.shift()` metódus ugyanúgy működik, mint a `.pop()`, annyi különbséggel, hogy nem a legutolsó, hanem a legelső elemet törli a tömbből. Példaképp most az `.unshift()` metódus segítségével adjuk hozzá a kávét a bevásárlólistánhoz, majd töröljük a `.shift()` metódussal.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['csoki', 'kenyér', 'tej', 'alma', 'tojás',  
'rizs', 'tészta', 'bab'];
```

Ahhoz, hogy a kávét hozzáadhassuk a lista elejéhez, írd be a következőt a konzolba, majd nyomd meg az Entert:

```
shoppingList.unshift('kávé');
```

A konzol a 9-es számot adta vissza, ami a frissített tömbben szereplő tételek száma. Most írd be a konzolba, hogy `shoppingList`;, és nyomd meg az Entert, hogy leellenőrizd a frissített tömböt:

```
["kávé", "csoki", "kenyér", "tej", "alma", "tojás", "rizs",  
"tészta", "bab"]
```

A kávé törléséhez a tömb elejéről írd be a következőt a konzolba, és nyomd meg az Entert:

```
shoppingList.shift();
```

A konzol a törölt elemet adta vissza eredményként: "kávé". A `.shift()` metódus eltávolítja a tömb legelső elemét, majd megjeleníti a törölt elemet.

Ugyanúgy, mint a `.pop()` metódusnál, a `.shift()` metódussal eltávolított elemet is elmentheted egy változóba, hátha később még szükséged lesz rá.

Megjegyzés: A tömb első elemét a következőképpen tudod egyszerre törölni és hozzárendelni egy változóhoz: `var deletedFromBeginning = shoppingList.shift();`

Ha most beírod a konzolba, hogy `shoppingList`; , és megnyomod az Entert, a következő tömböt fogod visszakapni:

```
[ "csoki", "kenyér", "tej", "alma", "tojás", "rizs", "tészta", "bab" ]
```

Éljen! Gratulálunk, most már nemcsak azt tudod, hogyan adhatsz hozzá új elemeket egy tömbhöz, de el is tudod távolítani őket.

Feladat

Feladatlisták készítésével hatékonyan kezelheted az elvégzendő feladataidat. Készíts egy feladatlistát egy tömbben, a legfontosabb dolguktól a kevésbé sürgős feladataid felé haladva. Miután elmentette a tömböt egy változóba, töröld az első elemét.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek legalább négy eleme van.
- A tömb egy `myToDos` nevű változóba van elmentve.
- Töröld a tömb első elemét a `.shift()` metódussal.

Sok sikert!

Megoldás: Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

2.6. Marcsi bakancslistája

Ebben a feladatban gyakorolhatod minden, amit eddig a JavaScript-tömbökről tanultál, és még Marcsinak is segíthetsz összeírni a bakancslistáját.

Most már önállóan fogsz dolgozni, nem segítünk közben. A feladat az, hogy a lenti felsorolás minden lépését hajtsd végre.

Ne aggódj, már mindennt tudsz, ami a feladathoz szükséges. Sőt, a következő oldalon vár rád egy példamegoldás is, de kérlek, próbáld meg egyedül megcsinálni a feladatot, mielőtt megnéznéd.

Ha esetleg még nem hallottál volna a bakancslistáról: ez azoknak a dolgoknak a gyűjteménye, amelyeket még mindenképpen ki akarsz próbálni vagy meg akarsz tenni az életedben.

Marcsi például szeretne írni egy regényt, megtanulni olaszul, elmenni Új-Zélandra és kipróbálni a búvárkodást. Ezekről a céljairól pedig szeretne egy listát írni, hogy elővehesse, amikor inspirációra van szüksége. Ebben fogsz neki most segíteni.

Készítsd el Marcsi bakancslistáját az alábbi lépésekkel követve:

- Hozz létre egy tömböt, ami a jelenlegi terveit tartalmazza (regényt írni, megtanulni

olaszul, elutazni Új-Zélandra, búvárkodni). Mentsd el ezt a tömböt egy `bucketList` nevű változóba.

- Adj hozzá három új elemet (megnézni a sarki fényt, megtanulni gitározni, ejtőernyőzni) a `bucketList` tömb végéhez.
- Adj hozzá két új elemet (vegetáriánussá válni, legalább 3 hónapra Franciaországba költözni) a `bucketList` tömb elejéhez.
- Töröld a `bucketList` tömb utolsó elemét (ejtőernyőzni). Marcsei időközben rájött, hogy tulajdonképpen eléggé tériszonyos.
- Töröld a `bucketList` tömb első elemét (vegetáriánussá válni). Marcsei arra a következetésre jutott, hogy igazából ő nagyon szereti a húst.
- Végül jelenítsd meg a `bucketList` tömböt a konzolban a `console.log()` segítségével, hogy megtekinthesd Marcsei bakancslistájának végső változatát.

Nyiss egy új bint a JS Binben, és építsd fel Marcsei bakancslistáját a fenti lépésekkel követve. A kód megírásához használ a JS Bin JavaScript-panelét.

2.7. Példamegoldás a bakancslistára

Itt egy példa arra, ahogy Marcsei bakancslistáját fel lehet építeni:

```
var bucketList = ['regényt írni', 'megtanulni olaszul', 'elutazni Új-Zélandra', 'búvárkodni']; // tömb létrehozása és mentése egy változóba
bucketList.push('megnézni a sarki fényt', 'megtanulni gitározni', 'ejtőernyőzni'); // három új elem hozzáadása a tömb végéhez
bucketList.unshift('vegetáriánussá válni', 'legalább 3 hónapra Franciaországba költözni'); // két új elem hozzáadása a tömb elejéhez
bucketList.pop(); // az utolsó elem törlése
bucketList.shift(); // az első elem törlése
console.log(bucketList); // tömb megjelenítése a konzolban
```

Már kész is! Ha szeretnéd, most már a saját bakancslistádat is elkészítheted JavaScriptben.

2.8. Összefoglalás

Ebben a leckében megtanultad, hogyan adhatsz hozzá és törölhetsz elemeket egy tömb elejéről vagy végéről. A következő metódusokról volt szó:

- `.push()` metódus, amivel a tömb végéhez lehet hozzáadni új elemet
 - egy új elem hozzáadásakor: `nameOfTheArray.push(newElement)`;
 - több új elem hozzáadásakor: `nameOfTheArray.push(newElement1, newElement2, newElement3)`;
- `.unshift()` metódus, amivel a tömb elejéhez lehet hozzáadni új elemet
 - egy új elem hozzáadásakor: `nameOfTheArray.unshift(newElement)`;
 - több új elem hozzáadásakor: `nameOfTheArray.unshift(newElement1, newElement2, newElement3)`;
- `.pop()` metódus, amivel a tömb végéről lehet törölni egy elemet:

- ```
nameOfTheArray.pop();
```
- `.shift()` metódus, amivel a tömb elejéről lehet törölni egy elemet:  
`nameOfTheArray.shift();`

### 3. Egy tömb egy elemének megváltoztatása

#### 3.1. Hozzáférés a tömb egy eleméhez

Az igazi bevásárlólistánál bármikor megváltoztathatod a tételeket. Például ha úgy döntesz, hogy csoki helyett cukorkát vennél, egyszerűen kikeresed a csokit a listában, és átírod cukorkára.

Ezt a JavaScript-tömbökkel is megteheted: a tömb bármelyik elemének megtekintheted, illetve megváltoztathatod az értékét.

Először nézzük, hogyan férhetsz hozzá egy elem értékéhez.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['csoki', 'kenyér', 'tej', 'alma', 'tojás',
'rizs', 'tészta', 'bab'];
```

A 'csoki' elem eléréséhez írd be ezt a konzolba, és nyomd meg az Entert:

```
shoppingList[0];
```

A konzol azt adta vissza eredményként, hogy "csoki". Remek! Sikeresen elérte a tömb első elemének értékét.

Amit ehhez használtál, az a tömb elemének úgynevezett indexe (mutatószáma). Az index egy adott elem tömbön belül elfoglalt helyét mutatja. De miért 0 és nem 1? Az a helyzet, hogy a fejlesztők 0-tól szokták kezdeni a számozást, nem pedig 1-től. Úgyhogy a tömbök első elemének indexe mindig 0.

Ugyanezt a logikát követve a `shoppingList` utolsó eleméhez ('bab') való eléréshez a `shoppingList[7]`; kódot kell beírnod, nem pedig azt, hogy `shoppingList[8]`; (hiába áll a tömb 8 elemből). Az utolsó elem a 7-es indexen áll, ezért a `shoppingList[7]`; kód eredménye a "bab" lesz. Ezzel szemben a `shoppingList[8]`; kód eredménye az lenne, hogy `undefined`, hiszen a tömbben jelenleg nincs elem a 8. Indexnél.

Most, hogy már tudod, mi az az index, lássuk az általános szintaxist, amivel egy tömb elemének értékét elérhetjük:

```
nameOfTheArray[i];
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` a tömb adott elemének az indexszáma (az index 0-ról indul).

Most már tudod, hogyan férhetsz hozzá a tömb egy adott elemének értékéhez. A következő feladatban azt is megtudhatod, hogyan változtasd meg azt.

### Feladat

Mivel érdekel a kódolás, biztosan sokféle programozási nyelvről hallottál már. Sorold fel ezeket egy tömbben, és mentsd el a tömböt egy változóba. Ezután jelenítsd meg a tömb '`JavaScript`' elemét az indexszáma segítségével.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek legalább két eleme van.
- A tömb egy `programmingLanguages` nevű változóba van elmentve.
- Érd el a konzolban a '`JavaScript`' elemet az indexszámának használatával.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 3.2. Egy tömb egy elemének megváltoztatása

Most, hogy már tudod, hogyan érheted el a tömb egy adott elemének értékét, a csokit már gyerekjáték lesz kicserélni cukorkára a bevásárlólistádban.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['csoki', 'kenyér', 'tej', 'alma', 'tojás',
'rizs', 'tészta', 'bab'];
```

Mivel a '`csoki`' a tömb legelső eleme, a `shoppingList[0]` kóddal férhetsz hozzá, majd hozzárendelhetsz egy új értéket. Ez a két lépés egyetlen kódsorral is elvégezhető.

Gépeld be ezt a konzolba, és nyomd meg az Entert:

```
shoppingList[0] = 'cukorka';
```

A kód bal oldalán kiválasztottuk a megfelelő elemet a tömbből, a jobb oldalán pedig hozzárendeltünk egy új értéket. Ha minden jól csináltál, a konzol a következő eredményt adta vissza neked: "`cukorka`". Amikor megváltoztatsz egy elemet a tömbben, a konzol

megjeleníti az új értéket.

A frissített tömböt úgy ellenőrizheted le, hogy beírod a `shoppingList`; kódot a konzolba, és megnyomod az Enter-t. A következő fog megjelenni:

```
["cukorka", "kenyér", "tej", "alma", "tojás", "rizs", "tészta",
"bab"]
```

Látod, működött! Szép munka!

Egy tömbelem módosításának ez az általános szintaxisa:

```
nameOfTheArray[i] = newValue;
```

Ahol:

`nameOfTheArray` a tömb neve;

`i` a tömb adott elemének az indexszáma (a tömb elemeinek indexe 0-ról indul);

`newValue` az új érték, amit a tömb adott eleméhez rendeltél.

Most, hogy már tudod, hogyan változtathatod meg egy tömb bármelyik elemét, ideje a gyakorlatban is kipróbálni ezt a tudást. Találkozunk a következő feladatnál!

## Feladat

Van egy `colorsOfRainbow` nevű tömbünk, amely a szivárvány színeit tartalmazza:

`['vörös', 'narancssárga', 'sárga', 'zöld', 'kék', 'valamilyen szín', 'ibolya']`. De nézd csak meg jobban a hatodik elemet: '`valamilyen szín`'. Hmm, ez így nem tűnik jónak! Változtasd meg ezt az elemet a megfelelő színre.

A programod a következő kritériumoknak felejen meg:

- Egy tömbből áll, melynek hét eleme van.
- A tömb egy `colorsOfRainbow` nevű változóba van elmentve.
- Az indexszám használatával változtasd meg a 6. elemet a következő stringre: '`indigó`'.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 3.3. Jancsi vendélistája

Jancsi már jó ideje tervezgeti a 25. születésnapi buliját. A vendélistával már korábban elkészült: Tomi, Lili, Balázs, Zsuzsi, Misi, Kati, Janka és Pali szerepel rajta.

Mielőtt elküldte volna a meghívókat, gyorsan átfutotta a vendélistát, és észrevette, hogy

még mindig az előző barátnője (Kati) szerepel rajta az új barátnője (Marcsi) helyett. Hoppá! Természetesen szeretné orvosolni ezt a hibát, és ehhez kell a te segítséged.

A következőket kell tenned:

- Hozz létre egy tömböt, ami Jancsi eredeti vendéglistáját tartalmazza (Tomi, Lili, Balázs, Zsuzsi, Misi, Kati, Janka, Pali). Mentsd el a tömböt egy `guestList` nevű változóba.
- Cserél le Katit Marcsira a `guestList` tömbben.
- Végül jelenítsd meg a `guestList` tömböt a konzolban a `console.log()` segítségével, hogy megtekinthesd Jancsi végső vendéglistáját.

A következő oldalon vár egy példamegoldás. Ennek ellenére kérlek, próbáld meg előbb önállóan megoldani a feladatot, hogy hatékonyabban gyakorolhasd az imént tanultakat.

Nyiss egy új bint a JS Binben, és írd meg Jancsi vendéglistáját a fenti lépéseket követve. A kód megírásához használd a JS Bin JavaScript-panelét.

### 3.4. Példamegoldás a vendéglistára

Itt egy példa arra, ahogyan Jancsi vendéglistáját meg lehet írni:

```
var guestList = ['Tomi', 'Lili', 'Balázs', 'Zsuzsi', 'Misi',
'Kati', 'Janka', 'Pali']; // tömb létrehozása és mentése egy
változóba
guestList[5] = 'Marcsi'; // a tömb 6. elemének megváltoztatása
console.log(guestList); // tömb megjelenítése a konzolban
```

Fantastikus! Jancsi végre kiküldheti a szülinapi meghívókat!

## 4. Elemek hozzáadása és törlése bárhonnan egy tömbből

### 4.1. Elem törlése bárhonnan

Ebben a projektben már megtanultad, hogyan törölheted egy tömb első és utolsó elemét a `.shift()` és a `.pop()` metódusokkal.

Ez szuper, de mi van akkor, ha egy másik elemet szeretnél törölni a tömbből? Végtére is, az igazi listáknak bármelyik elemét ki lehet törölni. Ezt a JavaScriptnek is igazán meg kéne tudnia csinálni. A jó hír az, hogy meg is tudja.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['cukorka', 'kenyér', 'tej', 'alma', 'tojás',
'rizzs', 'téteszta', 'bab'];
```

Mondjuk, hogy mégsem szeretnél tojást venni, és ezért ezt a tételt ki szeretnéd húzni a listáról. Írd be ezt a konzolba, és nyomd meg az Entert:

```
shoppingList.splice(4, 1);
```

A konzol az imént törölt elemet adja vissza eredményül: `["tojás"]`.

A `.splice()` metódus bármilyen elemet törölni tud a tömbből, majd a törölt elemet egy új tömbként adja vissza.

Mindez szép és jó, de mik voltak azok a számok a zárójelben, amikor a `.splice()` metódust használtuk?

Az első szám a tömb azon elemének az indexe, amelyet törölni akarunk.

Ebben az esetben az 5. tételt akartuk eltávolítani, ezért a 4-es indexet használtuk – ne feledd, az index 0-tól kezdődik.

A második szám azt jelöli, hogy hány elemet szeretnénk törölni.

Most csak egyet töröltünk, ezért az 1-es számot használtuk.

Ha meg szeretnél róla bizonyosodni, hogy a `'tojás'` tényleg törlődött a tömbből, írd be a konzolba, hogy `shoppingList;`, és nyomd meg az Entert. A konzol a módosított tömböt fogja visszaadni:

```
["cukorka", "kenyér", "tej", "alma", "rizzs", "tészta", "bab"]
```

Láthatod, hogy a `'tojás'` valóban eltűnt a tömbből.

A `.splice()` metódus általános szintaxisa a következő:

```
nameOfTheArray.splice(i, n);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` az indexszám, ahonnan az elemet törölni szeretnéd a tömbből (az index 0-ról indul);
- `n` az a szám, amennyi elemet törölni szeretnél.

Ahogy azt már sejtheted, a `.splice()` metódussal több elemet is törölhetsz egyszerre. Hogy ez pontosan hogyan működik, az a következő feladatból kiderül.

## Feladat

Egy fiatal pár készített egy ajándéklistát az esküvőjük előtt, hogy a szeretteik tudják, milyen nászajándéknak örülnének a legjobban. Írtak egy `weddingGifts` nevű tömböt, amely a

következő elemeket tartalmazza: `['turmixgép', 'kenyépirító', 'kávégőző', 'pohárkészlet', 'étkészlet', 'ágynemű', 'törölközök']`. Szeretnél törölni a kávégőzöt a listából, mert megegyeztek, hogy kevesebb kávét fognak inni. Írd meg a kódot, amelyet ahhoz használnál, hogy töröld ezt az elemet.

A programod a következő kritériumoknak feleljén meg:

- Egy tömbből áll, melynek hét eleme van.
- A tömb egy `weddingGifts` nevű változóba van elmentve.
- Töröld a tömb 3. elemét a `.splice()` metódussal.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 4.2. Több elem törlése bárhonnan egy tömbből

Az előző feladatban megtanultad, hogyan használhatod a `.splice()` metódust ahhoz, hogy a tömb bármely tételét törölhesd. De mi van akkor, ha több elemet is szeretnél törölni? Például hogyan távolíthatnád el egyszerre a rizst, téstát és babot is a `shoppingList` nevű JavaScript-tömbből?

Ezt természetesen úgy is megteheted, hogy egyenként törlöd őket, de ez kicsit sok időt venne igénybe. Szerencsére a `.splice()` metódussal több elemet is eltávolíthatsz egyszerre. Próbáljuk is ki.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['cukorka', 'kenyér', 'tej', 'alma', 'rizs',
'tétesztta', 'bab'];
```

A rizs, a tétesztta és a bab egyidejű törléséhez írd be a konzolba a következőt, és nyomd meg az Entert:

```
shoppingList.splice(4, 3);
```

A konzol ezt fogja visszaadni: `["rizs", "tétesztta", "bab"]`.

És erre is számítottunk, hiszen ezeket az elemeket akartuk törölni. A `.splice()` metódus – ahogyan már volt is erről szó – a törölt elemeket egy új tömbként adja vissza.

Ha meg szeretnél róla bizonyosodni, hogy a rizs, tétesztta és bab tényleg törlődött a tömbből, írd be a konzolba, hogy `shoppingList;`, és nyomd meg az Entert. A konzol ezt fogja visszaadni:

```
["cukorka", "kenyér", "tej", "alma"]
```

Szép munka!

Vessünk még egy pillantást a zárójelben szereplő számokra a `.splice()` metódusnál.

Emlékeztetőül, a `.splice()` általános szintaxisa a következő:

```
nameOfTheArray.splice(i, n);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` az indexszám, ahonnan az elemet törölni szeretnéd a tömbből (a tömb elemeinek indexe 0-ról indul);
- `n` az szám, amennyi elemet törölni szeretnél.

A fenti példában az 5. tételelől kezdve akartunk törölni pár elemet, és mivel az index 0-val kezdődik, a megfelelő indexszám az adott esetben a 4 volt.

Tekintve, hogy 3 darab elemet akartunk törölni, a második szám a zárójelben a 3 lett.

**Megjegyzés:** A `.splice()` metódusban nem kötelező megadni a törlendő elemek darabszámát. Ennek hiányában azonban a megjelölt indextől kezdődően minden elem törlődik a tömbből. Például a rizs, tézsza és bab elemek törléséhez, amelyek a tömb utolsó 3 elemét képezték, egyszerűen használhattuk volna a következő kódot:  
`shoppingList.splice(4);`

Most már tudod, hogyan használhatod a `.splice()` metódust ahoz, hogy a tömbből bárhonnan törölj egy vagy több elemet. De a `.splice()` nemcsak arra jó, hogy törölj bizonyos elemeket, hanem új elemek hozzáadására is alkalmas. A következő feladatban meg is tanulhatod, hogyan.

## Feladat

A nászajándéklista átolvasása közben a jegyespárnak eszébe jutott, hogy már van kenyépirítójuk és pohárkészletük. Töröld ezeket a tételeket a `weddingGifts` tömbből:  
`['turmixgép', 'kenyépirító', 'pohárkészlet', 'étkészlet', 'ágynemű', 'törölközök']`. Írd meg a kódot, amellyel ezt a két elemet törölni lehet további elemek törlése nélkül.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, melynek hat eleme van.
- A tömb egy `weddingGifts` nevű változóba van elmentve.
- Töröld a tömb 2. és 3. elemét a `.splice()` metódussal.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

### 4.3. Új elemek hozzáadása bárhol egy tömbben

Már volt szó ebben a projektben arról, hogyan adhatsz hozzá új elemeket egy tömb elejéhez vagy végéhez az `.unshift()` és a `.push()` metódussal. De a valódi életben nem mindig egy lista elejéhez vagy végéhez akarunk hozzáadni új elemeket. Például lehet, hogy épp a bevásárlólistád közepébe szeretnél beszúrni még egy-két tételelt.

Természetesen JavaScriptben ezt is megtehetjük. A `.splice()` metódust arra is használhatjuk, hogy új elemeket szúrunk be bárhol a tömbön belül.

Hogy lássuk, hogyan is működik ez, adjuk hozzá a sajtot a listánkhoz, a kenyér és a tej közé.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var shoppingList = ['cukorka', 'kenyér', 'tej', 'alma'];
```

A 'sajt' beszúrásához a 'kenyér' és a 'tej' közé írd be ezt a konzolba, és nyomd meg az Entert:

```
shoppingList.splice(2, 0, 'sajt');
```

A konzol egy üres tömböt adott vissza:

```
[]
```

Azt már tudod, hogy a `.splice()` minden a törlött elemeket adja vissza új tömbként. Mivel most nem akartunk semmit sem törölni, a törlendő elemek száma 0 volt. Ezért kaptunk eredményül egy üres tömböt.

Indexszámnak 2-t adtunk meg, pedig nem töröltünk semmit. Miért? Mert az indexszám nemcsak azt határozza meg, hogy honnan akarsz elemeket törölni a tömbből a `.splice()` metódussal, hanem azt is, ahová az új elemet szeretnéd beszúrni.

Fontos, hogy ez a szám nem annak az elemnek az indexét jelöli, amely után be szeretnéd szúrni az új tételelt, hanem azt az indexet, ahol az új téTEL meg fog jelenni.

Mivel a '`sajt`'-ot a '`kenyér`' és a '`tej`' közé szeretnénk beszúrni, a sajt a harmadik elem lesz az új listában, tehát a 2-es indexszámot kapja.

Ha meg szeretnél róla bizonyosodni, hogy most már a '`sajt`' a lista harmadik tétele, írd be a konzolba, hogy `shoppingList`;, és nyomd meg az Entert. A konzol a frissített tömböt fogja visszaadni:

```
["cukorka", "kenyér", "sajt", "tej", "alma"]
```

Látod, a '`sajt`' a megfelelő helyen jelenik meg.

Foglaljuk össze, hogyan kell új elemeket hozzáadni a tömbhöz, és nézzük át a `.splice()` metódus általános szintaxisát:

```
nameOfTheArray.splice(i, n, newElement);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` az index, ahonnan az elemet törölni szeretnéd vagy új elemet szeretnél hozzáadni a tömbhöz (a tömb elemeinek indexe 0-ról indul);
- `n` az a szám, amennyi elemet törölni szeretnél;
- `newElement` az új elem, amit a tömbhöz szeretnél adni.

**Megjegyzés:** Emlékszel, hogy a `.push()` és az `.unshift()` metódussal több tételt is hozzá tudtál adni egy tömbhöz? Csupán annyit kellett tenned, hogy elválasztottad őket egy-egy vesszővel. Hát, ez nagyából ugyanígy működik a `.splice()` metódusnál is:  
`nameOfTheArray.splice(i, n, newElement1, newElement2, newElement3);`.

Megtanultad, hogyan tudsz törölni, illetve hozzáadni elemeket egy tömbhöz a `.splice()` metódussal. Szuper! A következő feladatban élesben is begyakorolhatod a `.splice()` metódus használatát.

### Feladat

A jegyespár tárgyi ajándékok mellett szeretne új élményekkel is gazdagodni. Ezért szeretnék törölni az étkészletet és az ágyneműt az ajándéklistáról, és egyúttal hozzáadni egy raftingtúrát és egy bungee jumpingot. Ezeket az új elemeket a `weddingGifts` tömbben a turmixgép és a törölközök közé kellene beilleszteni. Emlékeztetőül, az ajándéklista jelenleg így áll: `[ 'turmixgép', 'étkészlet', 'ágynemű', 'törölközök' ]`. Írd meg a kódot, amelyet a tömb frissítéséhez használnál.

A programod a következő kritériumoknak feleljén meg:

- Egy tömbből áll, melynek négy eleme van.
- A tömb egy `weddingGifts` nevű változóba van elmentve.
- Egyetlen metódust használva cseréld le a tömb 2. és 3. elemét az új elemekre.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden

rendben, akkor menj tovább a következő leckére.

#### 4.4. Kati könyvlistája

Kati nemrég diplomázott, úgyhogy tankönyvek helyett végre azt olvashat, amit csak akar.

Ezért úgy döntött, hogy ír egy listát azokról a könyvekről, amelyeket már régóta szeretett volna elolvasni:

- Svejk, a derék katona,
- Jane Eyre,
- A Gyűrűk Ura trilógia (A Gyűrű Szövetsége, A két torony, A király visszatér), és
- Virágot Algernonnak.

Ebben a feladatban Katinak fogsz segíteni, hogy elkészíthesse és frissíthesse ezt a listát.

Ezt önállóan fogod elvégezni, az alábbi utasítások alapján. A következő oldalon bemutatunk egy példamegoldást, de kérlek, ne nézd meg, amíg nem teljesítetted a feladatot. Ne aggódj, már minden szükségeset tudsz a feladat megoldásához.

Hajtsd végre a következő lépéseket:

- Hozz létre egy tömböt azokkal a könyvekkel, amelyeket Kati már eredetileg is el szeretett volna olvasni (Svejk, a derék katona, Jane Eyre, A Gyűrű Szövetsége, A két torony, A király visszatér, Virágot Algernonnak). Mentsd el ezt a tömböt egy `booksToRead` nevű változóba.
- Töröld A Gyűrűk Ura trilógiát (A Gyűrű Szövetsége, A két torony, A király visszatér) a `booksToRead` tömbből, mert Kati rájött, hogy igazából nem nagyon szereti a fantaszt.
- Adj hozzá két új könyvet (Szerelem a kolera idején, Háború és béke) a `booksToRead` tömbhöz, mert Kati barátai nagyon ajánlották őket. Rögtön a Svejk, a derék katona után már el is szeretné olvasni ezeket a könyveket, úgyhogy a tömb 2. és 3. tételeként írd be őket.
- Végül jelenítsd meg a `booksToRead` tömböt a konzolban a `console.log()` segítségével, hogy megtekinthesd Kati végső listáját az elolvasandó könyvekről.

Nyiss egy új bint a JS Binben, és építsd fel Kati könyvlistáját a fenti lépésekkel követve. A kód megírásához használd a JS Bin JavaScript-panelét.

#### 4.5. Példamegoldás a könyvlistára

Itt egy példa arra, ahogy Kati könyvlistáját fel lehet építeni:

```
var booksToRead = ['Svejk, a derék katona', 'Jane Eyre', 'A Gyűrű Szövetsége', 'A két torony', 'A király visszatér', 'Virágot Algernonnak']; // tömb létrehozása és mentése egy változóba
booksToRead.splice(2, 3); // három elem törlése a tömbből a 2. indextől kezdve
booksToRead.splice(1, 0, 'Szerelem a kolera idején', 'Háború és béke')
```

```
béke'); // két új elem hozzáadása az 1. indexnél
console.log(booksToRead); // tömb megjelenítése a konzolban
```

Katinak most már szuper könyvlistája van. Jó olvasást, Kati!

## 4.6. Összefoglalás

Ebben a leckében megtanultad, hogyan használhatod a `.splice()` metódust ahhoz, hogy elemeket adj hozzá vagy törölj bárhonnan egy tömbből. A `.splice()` általános szintaxisa a következő:

```
nameOfTheArray.splice(i, n, newElement1, newElement2, newElement3);
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` az index, ahonnan az elemet törölni szeretnéd vagy ahol új elemet szeretnél hozzáadni a tömbhöz (ennek megadása kötelező);
- `n` az szám, amennyi elemet törölni szeretnél
  - ezt a paramétert nem kötelező megadni: ha nincs megadva, az adott indexszámtól kezdődően minden elem törlődni fog;
  - ha 0, egy elem sem törlődik;
- `newElement1`, `newElement2`, és `newElement3` azok az új elemek, amiket hozzá szeretnél adni a tömbhöz (nem kötelező megadni).

## 5. Egy tömb összes elemének megváltoztatása

### 5.1. Egy tömb hossza

A tömbben lévő elemek számát a tömb hosszának nevezzük.

Ezzel a számmal már találkoztál, amikor a `.push()` és az `.unshift()` metódust használtad. Emlékszel, hogy a konzol eredményként egy számot adott vissza, amikor ezekkel a metódusokkal adtál hozzá a tömbhöz új elemeket? Ez a szám az adott tömb hossza volt.

Jöjjön egy kis emlékeztető. Nyiss egy új bint a JS Binben, gépeld be a következőt a konzolpanelbe, és nyomd meg az Entert:

```
var numbers = [1, 2, 4, 7];
```

Adj hozzá még egy tételet (a 11-es számot) a tömb végéhez. Írd be ezt a konzolba, és nyomd meg az Entert:

```
numbers.push(11);
```

A konzol az 5-ös számot adta vissza eredményként: a tömb most már 5 elemből áll, azaz a tömb hossza 5.

Egy tömb hosszát persze új elemek hozzáadása nélkül is meg lehet jeleníteni. Ehhez csak írd be a következőt a konzolba, és nyomd meg az Enter-t:

```
numbers.length;
```

A konzol most is az 5-ös számot adta vissza, ahogy az várható volt.

A hosszúság tulajdonság szintaxisa tehát `nameOfTheArray.length`, ahol a `nameOfTheArray` az adott tömb neve.

**Megjegyzés:** A tömb hossza sohasem egyezik meg az utolsó elem indexszámával. Mivel a tömb elemeinek indexe minden 0-tól kezdődik, nem pedig 1-től, a tömb hossza minden eggyel nagyobb, mint az utolsó elem pozíciója.

### Feladat

Adott egy elég hosszú, `longArray` nevű tömb, amely a következő számokat tartalmazza: [3, 43, 67, 32, 691, 72, 52, 38, 342, 87, 65, 7, 49, 9456, 77, 30]. Szeretnéd megjeleníteni a tömb hosszát, hogy ne kelljen megszámolni az elemeit. Írd meg ehhez a kódot.

A programod a következő kritériumoknak feleljen meg:

- Egy tömbből áll, a fent felsorolt elemekkel.
- A tömb egy `longArray` nevű változóba van elmentve.
- Írasd ki a konzolba a tömb jelenlegi hosszát a `length` tulajdonság segítségével.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 5.2. minden elem megváltoztatása egyesével

Ebben a projektben már megtanultad, hogyan kell megváltoztatni a tömb egy elemét. Az erre vonatkozó általános szintaxis a következő:

```
nameOfTheArray[i] = newValue;
```

Ahol:

- `nameOfTheArray` a tömb neve;
- `i` a tömb adott elemének az indexszáma (a tömb elemeinek indexe 0-ról indul);
- `newValue` az új érték, amit a tömb adott eleméhez rendeltél.

De mi van akkor, ha a tömb összes elemén szeretnéd elvégezni ugyanazt a változtatást? Ezt persze megteheted egyesével is. Például a tömb összes eleméhez hozzáadhatsz hámat külön-külön utasításokkal.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var numbers = [1, 2, 4, 7, 11];
```

Ahhoz, hogy minden elemhez hozzáadj hámát, öt külön utasításra lesz szükséged:

```
numbers[0] += 3;
numbers[1] += 3;
numbers[2] += 3;
numbers[3] += 3;
numbers[4] += 3;
```

Ha most beírod a konzolba, hogy `numbers`; , és megnyomod az Entert, látni fogod, hogy a tömb frissült:

```
[4, 5, 7, 10, 14]
```

Ez eddig csodás, de képzeld el ugyanezt egy olyan tömbbel, amelynek több száz, ezer vagy akár millió eleme van! Ha ennek már a gondolata is elrémiszt, van egy jó hírünk: van egy sokkal gyorsabb és kényelmesebb mód is a tömb összes elemének megváltoztatására. A következő feladatban ezt fogod megtanulni.

## Feladat

Szeretnénk besorolni Harry Potter-t, Hermione Grangert és Ron Weasley-t a Roxfort megfelelő házába. Ehhez létrehoztunk egy `sortingHat` nevű tömböt, amelyből csak a ház neve hiányzik: `['Harry Potter háza a ', 'Hermione Granger háza a ', 'Ron Weasley háza a ']`. A stringek kombinálásával (amit ugye stringösszefűzésnek hívunk) add hozzá a ház nevét (Griffendél) a tömb minden eleméhez. Három külön utasítással változtasd meg a tömb elemeit.

A programod a következő kritériumoknak feleljön meg:

- Egy tömbből áll, a fent felsorolt elemekkel.
- A tömb egy `sortingHat` nevű változóba van elmentve.
- A tömb egy elemének a kiválasztásához használd az indexszámot.
- minden elemhez stringösszefűzéssel add hozzá a ház nevét.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden

rendben, akkor menj tovább a következő leckére.

### 5.3. Az összes elem megváltoztatása for ciklussal

Az előző feladatban egyesével módosítottad egy tömb összes elemét, külön utasításokban végezted el rajtuk ugyanazt a változtatást.

Bárcsak lenne rá mód, hogy ugyanazt a kódot többször is lefuttassuk anélkül, hogy újra meg újra be kéne gépelni... Ha jobban belegondolsz, igazából már tudod, hogy mi lehet a tökéletes megoldás erre: a ciklusok!

Már csak az a kérdés, hogyan lehet a ciklusokat a tömbökre alkalmazni.

Lássunk egy példát arra, ahogy a `for` ciklussal beszorozhatjuk egy tömb minden elemét 2-vel.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var numbers = [4, 5, 7, 10, 14];
```

Ahhoz, hogy minden elemet megszorozzunk 2-vel, csupán egy praktikus kis `for` ciklust kell írnunk:

```
for (var i = 0; i < 5; i++) {
 numbers[i] *= 2;
}
```

Nyomd meg az Entert.

**Megjegyzés:** A konzol a 28-as számot adta vissza eredményként. Ezzel nem kell foglalkozni, ez csak a tömb utolsó elemén végrehajtott utasítás eredménye.

Valami megváltozott, de nézzük csak, hogy pontosan mi is. Írd be a konzolba, hogy `numbers;`, és nyomd meg az Entert. Ha minden jól megy, a konzol a frissített tömböt fogja visszaadni:

[8, 10, 14, 20, 28]

Láthatod, hogy tényleg minden téTEL meg lett szorozva kettővel. Hát nem nagyszerű?

A következő feladatban végigvesszük, hogy pontosan hogyan is történt minden.

### 5.4. Hogyan változtatta meg a for ciklus a tömböt?

Most, hogy már rendelkezésünkre áll egy ilyen praktikus `for` ciklus, nézzük meg

közelebbről. Még egyszer lefuttatjuk a `for` ciklust, de most magába a ciklusba írjuk be a `console.log()` utasítást. Ez minden iteráció után vissza fogja adni a frissített tömböt, így pontosan nyomon követhetjük, hogy mi történik a tömb tételeivel.

Nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var numbers = [8, 10, 14, 20, 28];
```

Most írd be ezt a konzolba, és nyomd meg az Entert:

```
for (var i = 0; i < 5; i++) {
 numbers[i] *= 2;
 console.log(numbers);
}
```

A tömb elemei ismét meg lettek szorozva 2-vel. Az eredményünk pedig így néz ki:

```
[16, 10, 14, 20, 28]
[16, 20, 14, 20, 28]
[16, 20, 28, 20, 28]
[16, 20, 28, 40, 28]
[16, 20, 28, 40, 56]
```

Minden sor egy iterációnak felel meg, és ha jobban megnézed, láthatod, hogy a számok balról jobbra haladva változnak meg sorban, iterációnként mindenkor csak egy. De miért?

A `numbers[i] *= 2;` utasításban az `i`-t mint indexszámot használtuk.

Ez az utasítás arra készti a programot, hogy találja meg a tömb `i` indexnél lévő elemét, és szorozza meg kettővel. Mivel az `i` kezdeti értéke 0, és minden iterációban eggyel növekszik, az utasítás minden iterációban „eggyel jobbra lép”, és végrehajtja a szorzást az aktuális elemen.

A fenti folyamat addig ismétlődik, amíg `i` értéke eléri az 5-öt, ami után az `i < 5` feltétel már nem igaz, és a `for` ciklus véget ér.

**Megjegyzés:** Miért használtuk az 5-öst kontrollszámnak? Mert a tömb 5 elemből áll, és az utolsó elem indexszáma ennél eggyel kisebb (4).

Az eredményt itt láthatod, néhány hasznos megjegyzéssel együtt:

```
[16, 10, 14, 20, 28] // i = 0 --> megszorozza a 0. indexen lévő
elemet 2-vel, és i értékét eggyel növeli
```

```
[16, 20, 14, 20, 28] // i = 1 --> megszorozza az 1. indexen lévő
elemet 2-vel, és i értékét eggyel növeli
[16, 20, 28, 20, 28] // i = 2 --> megszorozza a 2. indexen lévő
elemet 2-vel, és i értékét eggyel növeli
[16, 20, 28, 40, 28] // i = 3 --> megszorozza a 3. indexen lévő
elemet 2-vel, és i értékét eggyel növeli
[16, 20, 28, 40, 56] // i = 4 --> megszorozza a 4. indexen lévő
elemet 2-vel, és i értékét eggyel növeli
// i = 5 --> a for ciklus véget ér
```

Most már tudod, hogyan lehet a **for** ciklus segítségével egy tömb minden elemén végrehajtani ugyanazt a kódblokkot. A következő feladatban azt is megtudod, hogy mi történik, ha a tömb hossza megváltozik, de a **for** ciklus ugyanaz marad. Találkozunk a következő oldalon!

### Feladat

Szeretnénk besorolni Draco Malfoyt, Vincent Crakot és Gregory Monstrót a Roxfort megfelelő házába. Ehhez létrehoztunk egy **sortingHat** nevű tömböt, amelyből csak a ház neve hiányzik: `['Draco Malfoy háza a ', 'Vincent Crak háza a ', 'Gregory Monstro háza a ']`. A stringek kombinálásával add hozzá a ház nevét (Mardekár) a tömb minden eleméhez. A tömb elemeit egyetlen **for** ciklussal változtasd meg.

A programod a következő kritériumoknak feleljön meg:

- Egy tömbből áll, a fent felsorolt elemekkel.
- A tömb egy **sortingHat** nevű változóba van elmentve.
- A ház nevét egy **for** ciklussal add hozzá az elemekhez.
- Használj stringösszefűzést a **for** cikluson belül.
- A tömb egy elemének a kiválasztásához használd az indexszámot.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 5.5. A tömb hosszának használata a for ciklusban

Az előző feladatban egy **for** ciklus segítségével változtattad meg egy tömb minden elemét. Most lássuk, mi történik, ha hozzáadunk még egy elemet a tömbhöz, de magát a **for** ciklust változatlanul hagyjuk.

Először is nyiss egy új bint a JS Binben, a konzolpanelbe írd be a következőt, majd nyomd meg az Entert:

```
var numbers = [16, 20, 28, 40, 56];
```

Adjuk hozzá a tömb végéhez a 76-os számot. Írd be ezt a konzolba, és nyomd meg az Entert:

```
numbers.push(76);
```

A `.push()` metódus a 6-os számot adta vissza eredményként, mivel a tömbnek éppen ez a hossza.

Most lássuk, mi történik, ha ismét lefuttatjuk az előző `for` ciklusunkat, ami minden változtatás után megjeleníti a frissített tömböt. Írd be ezt a konzolba, és nyomd meg az Entert:

```
for (var i = 0; i < 5; i++) {
 numbers[i] *= 2;
 console.log(numbers);
}
```

A konzol megmutatja, hogy mi történt a tömbbel:

```
[32, 20, 28, 40, 56, 76] // i = 0 --> megszorozza a 0. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[32, 40, 28, 40, 56, 76] // i = 1 --> megszorozza az 1. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[32, 40, 56, 40, 56, 76] // i = 2 --> megszorozza a 2. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[32, 40, 56, 80, 56, 76] // i = 3 --> megszorozza a 3. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[32, 40, 56, 80, 112, 76] // i = 4 --> megszorozza a 4. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
// i = 5 --> a for ciklus véget ér
```

Az első öt elemet sikeresen megszoroztuk kettővel, de az utolsó elem (76) nem változott. Miért? Mert a `for` ciklusban még mindig az 5-ös számot használtuk kontrollsámnak, pedig az elemek száma most már 6. Így aztán a `for` ciklus az első 5 elem iterációja után leállt, a ciklusban szereplő utasítás pedig nem lett végrehajtva a hatodik elemen.

**Megjegyzés:** A programozók ezt hívják „hard coding”-nak (kemény kódnak) – amikor a programhoz konkrét, fix kontrollsámkat használunk. Ezzel a kódod nem lesz képes dinamikusan változni, ami problémát jelenthet.

Hogyan tudnánk megoldani tehát, hogy a `for` ciklus a tömb mind a hat elemén lefusson? Sőt, hogyan érthetnénk el, hogy a `for` ciklusunk minden elem iterációját elvégezze, attól függetlenül, hogy éppen milyen hosszú a tömb?

Itt fogjuk nagy hasznát venni a hosszúságot leíró tulajdonságnak.

Ahhoz, hogy a **for** ciklust a tömb hosszával tud kontrollálni, írd be ezt a konzolba, és nyomd meg az Entert:

```
for (var i = 0; i < numbers.length; i++) {
 numbers[i] *= 2;
 console.log(numbers);
}
```

Nézd meg, mi történt:

```
[64, 40, 56, 80, 112, 76] // i = 0 --> megszorozza a 0. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[64, 80, 56, 80, 112, 76] // i = 1 --> megszorozza az 1. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[64, 80, 112, 80, 112, 76] // i = 2 --> megszorozza a 2. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[64, 80, 112, 160, 112, 76] // i = 3 --> megszorozza a 3. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[64, 80, 112, 160, 224, 76] // i = 4 --> megszorozza a 4. indexen
lévő elemet 2-vel, és i értékét eggyel növeli
[64, 80, 112, 160, 224, 152] // i = 5 --> megszorozza az 5.
indexen lévő elemet 2-vel, és i értékét eggyel növeli
// i = numbers.length --> a for ciklus véget ér
```

Tökéletesen műköött! A tömb minden tételeit megszoroztuk 2-vel.

Vegyük át, mit tanultunk ebben a feladatban. Ha egy tömb összes elemének iterációját egyetlen **for** ciklussal akarjuk elvégezni, akkor a ciklust a hosszúságot leíró tulajdonsággal érdemes kontrollálni, nem pedig egy fix értékkel. Így biztosak lehetünk benne, hogy a ciklus a tömb minden elemén elvégzi az iterációt, a tömb hosszától függetlenül.

## Feladat

Szeretnénk besorolni Cho Changot, Luna Lovegoodot és Padma Patilt a Roxfort megfelelő házába. Ehhez létrehoztunk egy **sortingHat** nevű tömböt, amelyből csak a ház neve hiányzik: `['Cho Chang háza a ', 'Luna Lovegood háza a ', 'Padma Patil háza a ']`. A stringek kombinálásával add hozzá a ház nevét (Hollóhát) a tömb minden eleméhez. A tömb elemeit egy **for** ciklussal változtasd meg, amelyet a hosszúságot leíró tulajdonsággal kontrollálsz.

A programod a következő kritériumoknak feleljön meg:

- Egy tömbből áll, a fent felsorolt elemekkel.
- A tömb egy **sortingHat** nevű változóba van elmentve.

- A ház nevét egy **for** ciklussal add hozzá az elemekhez.
- A tömb hosszát használd a ciklus kontrollálásához.
- Használj stringösszefűzést a **for** cikluson belül.
- A tömb egy elemének a kiválasztásához használd az indexszámot.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 6. Összefoglalás

### 6.1. Összefoglalás

Gratulálunk, épp most lettél kész az első tömbökről szóló projekteddel. Rengeteg minden megtanultál a JavaScriptben használt listákról:

- létrehozni egy tömböt, és elmenteni azt egy változóba:  
`var nameOfTheArray = [element1, element2, element3];`
- új elemeket hozzáadni egy tömb elejéhez:  
`nameOfTheArray.unshift(newElement1, newElement2, newElement3);`
- új elemeket hozzáadni egy tömb végéhez:  
`nameOfTheArray.push(newElement1, newElement2, newElement3);`
- törölni egy tömb első elemét:  
`nameOfTheArray.shift();`
- törölni egy tömb utolsó elemét:  
`nameOfTheArray.pop();`
- elérni a tömb egyik elemét:  
`nameOfTheArray[i];`
- megváltoztatni a tömb egyik elemét:  
`nameOfTheArray[i] = newValue;`
- hozzáadni vagy eltávolítani az elemeket a tömbben bárhol:  
`nameOfTheArray.splice(i, n, newElement1, newElement2, newElement3);`
- megváltoztatni egy tömb elemeit a **for** ciklus segítségével:

```
for (var i = 0; i < numbers.length; i++) {
 // a kód, amit a tömb elemein végre kell hajtani
}
```

## 7. Teszt

### 7.1. Ellenőrizd a tudásod!

[13 kérdés, körülbelül 13 perc, kérdésenként 1 pont.](#)

**Tipp:** Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a „Küldés” (Submit) gombot a teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

**Megjegyzés:** A tesztet többször is kitöltheted.

Sok sikert!

# JavaScript alapok VII. (objektumok)

## 1. Mi az az objektum?

### 1.1. Bevezetés

Ha körbenézel ott, ahol ülsz, akkor sokféle tárgyat fogsz látni: asztalokat, székeket, polcokat, könyveket stb.

Ezeknek a tárgyaknak különféle tulajdonságaik vannak, mint a szín, anyag, márka és így tovább. Aztán minden egyik tulajdonságának különböző értékei lehetnek. Például a tárgy színe lehet kék vagy piros, az anyaga pedig műanyag vagy fa.

Végül pedig a tárgyak különböző funkciókat tölthetnek be: a széken állhatsz vagy ülhetsz, a polcra felenni és arról levenni is tudsz egy könyvet.

A JavaScriptben is léteznek tárgyak, ezeket objektumoknak hívjuk. Az igazi tárgyakhoz hasonlóan ezek is rendelkeznek tulajdonságokkal és funkciókkal.

Ahogy egyre összetettebbé válik a kódod, úgy kell egyre inkább egy strukturált formában tárolnod azt, hogy könnyen hozzáférhess a funkciókhoz, és ne torkoljon minden összevisszáságba. Ezt a célt szolgálják a JavaScript-objektumok.

Ebben a leckében a JavaScript-objektumokról fogsz tanulni:

- hogyan hozhatod létre és mentheted el őket,
- hogyan érheted el, változtathatod meg, adhatod hozzájuk vagy ellenőrizheted a tulajdonságaikat, és
- hogyan hozhatod létre és érheted el az objektum funkciót.

Kezdjük is el!

### 1.2. Objektum létrehozása

A JavaScript-objektumok jelképezhetnek valós tárgyat, például a kedvenc könyvemet: a címe *Ne bántsátok a feketerigót!*, a szerzője Harper Lee, a műfaja pedig fejlődésregény.

Tegyük fel, hogy le akarom jegyezni a könyv egy tulajdonságát. Először leírom a tulajdonság nevét (pl. `title`, ami angolul címet jelent), majd a tulajdonság értéke következik (`Ne bántsátok a feketerigót!`). Közéjük pedig kettőspontot teszek:

```
title: Ne bántsátok a feketerigót!
```

**Megjegyzés:** A JavaScript névadási szokásainak megfelelően a tulajdonságok neve angolul legyen, ahogyan a változók neve is.

Ezután, hogy egy részletesebb leírást adjunk a könyvről, egyszerűen leírom az összes tulajdonságát – a címet (angolul: **title**), a szerzőt (angolul: **author**), és a műfajt (angolul: **genre**) – egymás mellé, vesszővel elválasztva:

```
title: Ne bántsátok a feketerigót!, author: Harper Lee, genre:
fejlődésregény
```

Ez a JavaScriptben is hasonló, kivéve, hogy az objektumok kapcsos zárójelben vannak:

```
{title: Ne bántsátok a feketerigót!, author: Harper Lee, genre:
fejlődésregény}
```

Már majdnem készen vagyunk, csak még egy apróság. Mivel a stringeknek aposztrófok között kell állniuk, ezért adjuk hozzá ezeket is:

```
{'title': 'Ne bántsátok a feketerigót!', 'author': 'Harper Lee',
'genre': 'fejlődésregény'}
```

És kész is. Készítettünk egy teljesen érvényes JavaScript-objektumot három tulajdonsággal.

Hogy könnyebben olvasható legyen, érdemes minden tulajdonságot külön sorba írni:

```
{
 'title': 'Ne bántsátok a feketerigót!',
 'author': 'Harper Lee',
 'genre': 'fejlődésregény'
}
```

Jobban néz ki, ugye?

Tehát a JavaScript-objektumok általános szintaxisa a következő:

```
{
 propertyName1: PropertyValue1,
 propertyName2: PropertyValue2,
 propertyName3: PropertyValue3
}
```

Ahol:

- a **propertyName1**, **propertyName2** és **propertyName3** a tulajdonságok nevei, és
- a **PropertyValue1**, **PropertyValue2** és **PropertyValue3** a tulajdonságok értékei.

Jegyezd meg, hogy amikor objektumot hozunk létre, a vesszőt csak a tulajdonságok közé kell kitenni. Így amikor az objektum tulajdonságait külön sorokba írod, különösen figyelj oda, hogy az utolsó sor végére ne tegyél vesszőt.

Most, hogy már tudod, hogyan kell objektumot készíteni, jó lenne azt is tudni, hogy hogyan kell azt elmenteni. Erről lesz szó a következő oldalon.

### Feladat

Biztosan van olyan dal, amely nem megy ki a fejedből. Hozz létre egy objektumot, amely tárolja ennek a dalnak a címét és az előadóját.

A programod a következő kritériumoknak feleljén meg:

- Egy objektumból áll.
- Az objektum két tulajdonsággal rendelkezik.
- Szintaktikailag helyes.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

### 1.3. Objektum mentése

Amikor a való életben elkészítesz egy tárgyat, akkor azt általában meg is tartod. Ha mondjuk írás egy könyvet, remélhetőleg nem dobod ki egyből, amint befejezted, hanem elteszed későbbi használatra.

Természetesen nincs ez másként a JavaScriptben sem: az elkészült objektumot elmented egy változóba, ahogy a stringeket és tömböket is szoktuk. Így később tudsz az objektumra hivatkozni a változó használatával.

Példaként mentsük el a könyvobjektumot egy változóba. Nyiss egy új bint a JS Binen, írd a következőt a konzolpanelbe, majd nyomd meg az Entert:

```
var myBook = {
 'title': 'Ne bántsátok a feketerigót!',
 'author': 'Harper Lee',
 'genre': 'fejlődésregény'
};
```

A konzol azt írta ki, hogy `undefined`, ami azt jelenti, hogy a változót sikerült megadni, és a későbbiekben hozzáférhető. Ellenőrzésképp írd be a konzolba azt, hogy `myBook;`, majd nyomj Entert. Ezt kell látnod:

```
[object Object] {
 author: "Harper Lee",
 genre: "fejlődésregény",
 title: "Ne bántsátok a feketerigót!"
}
```

Noha az `[object Object]` elsőre furcsának tűnhet, ez csak a konzol módja annak jelzésére, hogy megtalálta a keresett objektumot (az `object` objektumot jelent angolul). Nagyszerű, hiszen ez azt jelenti, hogy sikeresen elmentette az objektumot egy változóba.

A konzol a tulajdonságok neveit és értékeit is kiírta neked, ABC-sorrendben. Ezkről a tulajdonságokról fogunk beszélni a következő oldalon.

### Feladat

Már létrehoztál egy objektumot, amely tárolja egy dal címét és előadóját. Most mentsd el ezt az objektumot egy változóba.

A programod a következő kritériumoknak feleljen meg:

- Egy objektumból áll.
- Az objektum egy `myFavoriteSong` nevű változóba van elmentve.
- Az objektum két tulajdonsággal rendelkezik.
- Szintaktikailag helyes.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 1.4. Tulajdonságok neve és tulajdonságok értékei

A könyveknek vannak közös tulajdonságai, mint cím, szerző, műfaj, főszereplők, kiadás éve, és így tovább. Az adott könyvtől függően minden tulajdonság különböző értékeket vehet fel.

Vessünk egy pillantást a könyvünk tulajdonságaira, ahogy a konzol kiírta őket:

```
[object Object] {
 author: "Harper Lee",
 genre: "fejlődésregény",
 title: "Ne bántsátok a feketerigót!"
}
```

Minden tulajdonságnak van neve (pl. `author`) és értéke (mint `"Harper Lee"`).

**Dióhéjban, a JavaScript-objektumok elnevezett értékekből állnak, amelyeket tulajdonságoknak hívunk.**

A fenti példában az összes érték string volt, de lehetnének számok vagy booleanértékek (`true` vagy `false`), tömbök vagy még más objektumok is.

Próbálunk is ki néhányféle értéktípust. Nyiss egy úgy bint a JS Binen, a konzolpanelbe írd be a következőket, és nyomj Entert:

```
var myBook = {
 'title': 'Ne bántsátok a feketerigót!',
 'author': 'Harper Lee',
 'genre': 'fejlődésregény',
 'year of publication': 2015,
 'has been read': true,
 'main characters': ['Jean Louise Finch', 'Jeremy Finch',
 'Atticus Finch']
};
```

Ha most beírod a konzolba, hogy `myBook;`, és Entert nyomsz, akkor láthatod, hogy a különböző típusú értékek különböző színekkel vannak jelölve:

```
[object Object] {
 author: "Harper Lee",
 genre: "fejlődésregény",
 has been read: true,
 main characters: ["Jean Louise Finch", "Jeremy Finch", "Atticus Finch"],
 title: "Ne bántsátok a feketerigót!",
 year of publication: 2015
}
```

Tehát a tulajdonságok értékei különféle típusú adatok lehetnek. De mi van a tulajdonságok neveivel?

Nos, a tulajdonságok nevei mindig stringként kerülnek elmentésre a JavaScriptben. Még akkor is, ha az egy szám, a JavaScript stringként fogja eltárolni.

Most, hogy már érted az objektumok tulajdonságait, ideje megtanulni, hogy milyen módon érhetjük el őket.

### Feladat

Valószínűleg sokszor szoktak érdeklődni a kedvenceid iránt. Miért ne tartanád őket egy strukturált formában? Hozz létre és ments el egy objektumot, amely tárolja a kedvenc

színedet (stringként), számodat (számkként) és ételeidet (tömbként).

A programod a következő kritériumoknak feleljen meg:

- Egy objektumból áll.
- Az objektum egy `myFavorites` nevű változóba van elmentve.
- Az objektum három tulajdonsággal rendelkezik.
- A tulajdonságok értéktípusai különbözők (string, szám és tömb).

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 2. Objektumtulajdonságok elérése

### 2.1. Pontjelölés

Ebben a leckében elkezdesz megépíteni egy könyvkatalógust, amelyben a saját könyvtáradat fogod nyilvántartani. Ennek a segítségével sokkal gyorsabban találsz majd információt a könyveidről.

Már láttad az én kedvenc könyvemet tároló objektumot:

```
var myBook = {
 'title': 'Ne bántsátok a feketerigót!',
 'author': 'Harper Lee',
 'genre': 'fejlődésregény',
 'year of publication': 2015,
 'has been read': true,
 'main characters': ['Jean Louise Finch', 'Jeremy Finch',
 'Atticus Finch']
};
```

Létrehoztam [egy bint angolul](#) a JS Binen, és beleírtam a JavaScript-panelbe a könyvemet.

Most te következel, hogy ugyanezt megteedd. Nyiss egy bint a JS Binen, és írj egy objektumot a JavaScript-panelbe, amelynek a neve `myBook`. Ez az objektum fogja tárolni a kedvenc könyved tulajdonságait. Győződj meg róla, hogy ugyanazokat a tulajdonságneveket használod, amelyeket én is: `title` a címnek, `author` a szerzőnek, `genre` a műfajnak, `year of publication` a kiadás évének, `has been read` annak megadására, hogy elolvastad-e már a könyvet (ennek a tulajdonságnak `true` booleanértéke lesz, hiszen a kedvenc könyvedet már biztosan olvastad), valamint `main characters` a főszereplőknek. Ezeknek a tulajdonságoknak a könyvedre vonatkozó értékei legyenek.

**Megjegyzés:** Eddig a konzolpanelben dolgoztál, de most az objektumot a JavaScript-panelbe kell írnod. Miért? Mert ezt a könyvet még használni fogjuk, ezért most el szeretnénk menteni. Tudod, ha becsukod a lapot, vagy a „Run with JS” gombra kattintasz a JavaScript-kód futtatásához, minden elvész, amit a konzolba írtál. Ugyanakkor a JS Bin elmenti azt a kódot, amit a JavaScript-panelbe írsz. Később a JavaScript-kód csak egy üres alkalmazást fog magába foglalni, és a tartalmat adatbázisokba fogod menteni, de momentán ez nem annyira fontos. Most még az alkalmazást és a tartalmat is a JavaScript-kódban mentjük el. A konzolt pedig az alkalmazással való interakcióhoz fogjuk használni a JavaScript-kód futtatása után.

Ha végeztél, akkor kattints a „Run with JS” gombra, hogy lefuttasd a JavaScript-kódodat. Ezután írd be a konzolpanelbe, hogy `myBook;`, és nyomj Enter-t. Ha minden rendben van, akkor a konzol kiírta a könyvedet.

Szuper, de amikor valamelyen konkrét információt keresel, nem feltétlenül akarod a könyved összes adatát látni, csak azt, ami téged érdekel. Például ha kizárolag a könyv szerzőjét szeretnéd ellenőrizni, írd be a következőt a konzolba, és nyomj Enter-t:

```
myBook.author;
```

Tessék, a konzol kiírta a könyved szerzőjét.

Ez hogyan történt? A pontjelölést használtad, hogy elérд az objektum egyik tulajdonságának az értékét. Pontjelölésnek hívják, mert pontot teszel az objektumnév és annak a tulajdonságnak a neve közé, amelyikhez hozzá szeretnél férfi.

Ez a pontjelölés általános szintaxisa:

```
objectName.propertyName;
```

Ahol:

- az `objectName` az objektum neve, és
- a `propertyName` a tulajdonság neve.

Mostanra a katalógusod már tartalmaz egy könyvet. Kicsit később még fogunk hozzáadni többet is.

Szereted a brownie-t? Ha igen, akkor itt vannak a hozzávalók ehhez a finom sütihez: `var brownie = {'butter': '1/2 bögre', 'sugar': '1 bögre', 'unsweetened cocoa powder': '1 bögre', 'eggs': '2 nagy darab', 'flour': '1/2 bögre'};` Érd el a tojások (`eggs`) szükséges számát a pontjelölést használva, és másold be a kódodat alább.

## 2.2. Zárójeljelölés

Most próbáljuk meg elérni a könyved egy másik tulajdonságát (`main characters`) a

pontjelöléssel. Nyisd meg a korábban elmentett binedet a JS Binen (ha szükséged van egy mintabinre, [itt az enyém](#) angolul). Kattints a „Run with JS” gombra, majd írd be a következőket a konzolpanelbe, és nyomj Enter-t:

```
myBook.main characters;
```

Ezt fogod látni:

```
"Unexpected identifier"
```

Ajjaj, ez hibának tűnik (azt jelenti, hogy „nem várt azonosító”).

Úgy tűnik, hogy a JavaScript egy úgynevezett azonosítót várt, és a `main characters` nem számít annak.

De mik azok az azonosítók? Nos, a JavaScriptben a változók elnevezésére használjuk őket. Emlékezz csak, hogy már tanultál a változók elnevezésének szabályairól: nem használhatsz szóközt, nem kezdődhet a név számmal, és nem használhatsz sem különleges karaktereket, sem [foglalt szavakat](#).

Lényegében a pontjelölés csak akkor működik, ha a tulajdonságnév egyben változónév is lehetne. Akkor változtassuk meg a tulajdonságnevünket `main characters`-ről `mainCharacters`-re? Ez is lehet egy megoldás, de nem az egyetlen.

Van egy másik módja is a tulajdonságok elérésének: úgy hívják, hogy zárójeljelölés. Szerencsére a zárójeljelést alkalmazhatjuk bármilyen tulajdonságnév esetében, még akkor is, ha abban szóköz van (pl. `first property name`), számmal kezdődik (pl. `1stPropertyName`), vagy speciális karaktert tartalmaz (pl. `propertyName#1`).

Na lássuk a zárójeljelést a gyakorlatban.

A `main characters`-hez való hozzáféréshez írd be a következőt a konzolpanelbe, és nyomj Enter-t:

```
myBook['main characters'];
```

Hurrá! A konzol végre kiírta a könyved főszereplőit.

Mindössze annyit kellett tenned, hogy először leírtad az objektum nevét, majd utána a tulajdonság nevét szögletes zárójelben.

Ahogy a neve is sugallja, amikor zárójeljeléssel akarsz egy tulajdonságot elérni, akkor a tulajdonságnevét szögletes zárójelbe kell tenned.

Van még egy másik fontos különbség a két jelölés között: a pontjelölésnél nem használtunk aposztrófokat a tulajdonságnév körül, de a zárójeljelésnél hozzáadtuk az aposztrófokat, mint ahogy általában tesszük a stringeknél.

Tehát a zárójeljelölés általános szintaxisa így néz ki:

```
objectName['property name'];
```

Ahol:

- az `objectName` az objektum neve, és
- a `property name` a tulajdonság neve.

Most, hogy már mestere vagy a pont- és zárójeljelölésnek, készen állsz, hogy megtanuld, miként tudod őket használni, hogy elérj egy objektumot egy másik objektumon belül.

Emlékszel a finom brownie összetevőire? Itt vannak: `var brownie = { 'butter': '1/2 bögre', 'sugar': '1 bögre', 'unsweetened cocoa powder': '1 bögre', 'eggs': '2 nagy darab', 'flour': '1/2 bögre'};`. Most érd el a kakaó (`unsweetened cocoa powder`) szükséges mennyiségét a zárójeljelöléssel, és másold be a kódodat alább.

### 2.3. Beágyazott objektumok

Amikor könyvet olvasok, szeretem megjelölni azokat a részeket, amelyek különösen inspirálóak, meghatóak vagy elgondolkodtatóak. Nagyon jó lenne, ha a katalógusunk is tudná tárolni ezeket az idézeteket, a hozzájuk tartozó oldalszámmal együtt.

A kérdés az, hogy hogyan tároljuk az összes idézetet egyetlenegy tulajdonságban, figyelembe véve, hogy minden idézetet a megfelelő oldalszámhoz kéne hozzárendelni. Úgy hangzik, mintha egy objektumon belüli objektumról lenne szó, ugye?

Ahogy korábban említtettem, egy objektum valamely tulajdonságának értéke lehet egy másik objektum is. Most van alkalmunk rá, hogy ezt ki is próbáljuk.

Nyisd meg az előzőleg elmentett binedet a JS Binen. ([Ez](#) az én jelenlegi binem angolul.) A JavaScript-panelben adj hozzá egy új tulajdonságot a könyvedhez. A neve `quotes` lesz (azaz „idézetek” angolul), az értéke pedig egy objektum. A `quotes` objektumban a tulajdonságnevek az oldalszámok lesznek (pl. [p116](#) a 116. oldal esetében), és a tulajdonságok értékei lesznek az idézetek. Az én könyvem például [így néz ki](#) az idézetekkel.

**Amit létrehoztunk, azt beágyazott objektumnak hívják, ami egy objektum egy másik objektumon belül.**

Menő, de mit csinálnánk, ha szeretnénk egy adott idézetet elolvasni a könyvből?

Elég egyszerű: úgy férhetsz hozzá, hogy összekötök a pont- és a zárójeljelölést. Először leírod a könyvobjektum nevét (`myBook`), majd a beágyazott objektum neve következik (`quotes`), végül az elérni kívánt tulajdonság neve (az oldalszám):

```
myBook.quotes.p116;
```

Vagy:

```
myBook['quotes']['p116'];
```

Vagy akár keverheted is a két jelölést:

```
myBook.quotes['p116'];
```

```
myBook['quotes'].p116;
```

Elég könnyű, ugye?

**Megjegyzés:** Mielőtt hozzáférsz egy tulajdonsághoz a konzolpanelben, ne felejtsd el lefuttatni a JavaScript-kódodat a „Run with JS” gombra való kattintással.

Ezzel már megtanultál minden, ami szükséges az objektum tulajdonságainak eléréséhez.

### Feladat

Hozz létre egy objektumot, amely tárolja egy focicsapat tagjait. Ha készen vagy, érd el az első számú kapus nevét.

A programod a következő kritériumoknak feleljen meg:

- Egy objektumból áll, amely az általad választott csapat után elnevezett változóba van elmentve.
- Az objektumban négy beágyazott objektum legyen: kapusok, védők, középpályások és csatárok.
- A beágyazott objektumokon belül a tulajdonságnevek a mezszámok lesznek, az értékek pedig a játékosok nevei, mindegyik hozzárendelve a megfelelő mezszámhoz.
- Érd el az első számú kapus nevét a pont- és zárójeljelölés összekötésével.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 2.4. Összefoglalás

Ebben a leckében megtanultuk, hogyan érjük el az objektumok tulajdonságait:

- pontjeljeléssel: `objectName.propertyName`;
- zárójeljeléssel: `objectName['propertyName']`;
- beágyazott objektumokon belül:
  - `objectName.nestedObjectName.propertyName`;

- vagy
  - `objectName['nestedObjectName']['propertyName']`;
  - vagy
    - `objectName.nestedObjectName['propertyName']`;
    - vagy
      - `objectName['nestedObjectName'].propertyName`;

Remek! Itt az ideje, hogy megtanuld, hogyan kell megváltoztatni, hozzáadni vagy ellenőrizni az objektum tulajdonságait.

### 3. Objektumok tulajdonságainak megváltoztatása, hozzáadása és ellenőrzése

#### 3.1. Objektum tulajdonságának megváltoztatása

Nézzük át a könyvkatalógusodat a JS Binben. (Ha szükséged van egy mintabinre, [itt az enyém](#) angolul.)

A katalógus már rendelkezik pár hasznos tulajdonsággal: el tudja tárolni a könyv címét, szerzőjét, kiadási évét, olvasottsági állapotát, főszereplőit és néhány idézetet oldalszámokkal.

Még egy információt nagyon jó lenne az alkalmazásban tárolni: hogy kölcsönadtad-e valakinek a könyvet. Elvégre nagyon idegesítő, amikor tudod, hogy egy könyved hiányzik a polcról, de nem emlékszel, hogy kinek adtad kölcsön.

Tegyük ezt a tulajdonságot a már korábban elmentett könyvedbe a JS Bin JavaScript-paneljében. A tulajdonság neve `borrowed by` lesz (ami azt jelenti, hogy „kölcsönvette”), és az értéke attól függ, hogy a könyvet kölcsönvette-e valaki vagy sem. Ha igen, akkor az érték az illető nevének megfelelő string lesz. Ha éppen nem emlékszel, hogy kinek adtad, akkor az érték `undefined` lesz. [Az én könyvem](#) nincsen kölcsönadva most senkinek, ezért a tulajdonság értékének `null`-t adtam meg.

**Megjegyzés:** Mind az `undefined`, mind a `null` JavaScript-adattípusok. Ha az érték még nem lett hozzárendelve egy tulajdonsághoz (vagy változóhoz), az érték `undefined`, a `null` pedig nem létező értéket jelképez, és szó szerint semmit jelent. Ha tudod, hogy a könyved kölcsön lett adva, csak nem rémlik, hogy kinek, a `borrowed by` tulajdonságnak akkor is kell hogy legyen egy értéke. Mivel azonban nem tudod azonnal hozzárendelni a tulajdonsághoz, az érték jelenleg `undefined`. Ha tudod, hogy a könyved nincs kölcsönadva senkinek, a `borrowed by` tulajdonságnak nem kell hogy legyen értéke, és pontosan ez az, amit a `null` kifejez: az érték akaratlagos hiányát.

Természetesen ennek a tulajdonságnak az értéke nem állandó, hanem idővel változhat. Most megtanulod, hogy miként lehet módosítani.

Ha kölcsönadtam Katának a könyvemet, és frissíteni akarom a katalógusom, akkor

rákattintok a „Run with JS” gombra, majd beírom a konzolpanelbe a következőt, és Entert nyomok:

```
myBook['borrowed by'] = 'Kata';
```

Próbáld ki te is! Ne aggódj, ez a JavaScript-kódodat nem fogja végérvényesen megváltoztatni. Ahogy már korábban említettük: minden, amit a konzolba írsz, elvész, amint becsukod a lapot, vagy ismét a "Run with JS" gombra kattintasz.

Ha megnyomod az Entert, a konzol kiírja, hogy "**Kata**". **Ha megváltoztatod egy tulajdonság értékét, az új értéket kapod vissza.**

A frissített könyved megtekintéséhez írd be a konzolba, hogy `myBook;`, és nyomj Entert. Azt fogod látni, hogy a `borrowed by` tulajdonság értéke megváltozott arra, hogy "**Kata**".

Most nézzük meg, hogyan néz ki az objektumtulajdonság módosításának szintaxisa. Ha emlékszel rá, hogyan kell egy tömbelemet megváltoztatni, akkor fel fogod ismerni a hasonlóságot: kiválasztod a tulajdonságot a bal oldalon, majd hozzárendelsz egy új értéket a jobb oldalon. Ahhoz, hogy hozzáférj az objektumtulajdonsághoz, használhatod akár a pont-, akár a zárójeljelölést.

Tehát az objektumtulajdonság megváltoztatásának általános szintaxisa a következő:

```
objectName.propertyName = newValue;
```

vagy

```
objectName['propertyName'] = newValue;
```

Ahol:

- az `objectName` az objektum neve,
- a `propertyName` a tulajdonság neve, és
- a `newValue` a tulajdonság új értéke.

Most, hogy tudod, hogyan kell megváltoztatni az objektum egy tulajdonságát, már egyszerűnek fog tűnni, hogy hozzáadj egy újat, mert az is hasonlóan működik. Ezt látod majd a következő oldalon.

## Feladat

Ismered a Naprendszer bolygóit és törpebolygóit? Itt van egy objektum, amely tárolja őket az osztályozásukkal együtt: `var solarSystem = { 'Mercury': 'bolygó', 'Venus': 'bolygó', 'Earth': 'bolygó', 'Mars': 'bolygó', 'Jupiter': 'bolygó', 'Saturn': 'bolygó', 'Uranus': 'bolygó', 'Neptune': 'bolygó', 'Pluto': 'bolygó', 'Ceres': 'törpebolygó', 'Makemake': 'törpebolygó', 'Haumea': 'törpebolygó', 'Eris': 'törpebolygó' };`.

De várj csak, a Pluto tényleg a 9. bolygó? Nem, pár éve átminősítették, és jelenleg törpebolyónak számít. Javítsd ki ezt a hibát a **Pluto** tulajdonság értékének a megváltoztatásával.

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- Változtasd meg a **Pluto** tulajdonság értékét a pont- vagy zárójeljelölés használatával.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

### 3.2. Új objektumtulajdonság hozzáadása

Mielőtt kölcsonadsz egy könyvet valakinek, valószínűleg véleményezed először, így a barátod tudni fogja, hogy érdemes-e elolvasnia. Előfordulhat azonban, hogy már nem emlékszel, milyennek is találtad, amikor olvastad, így ez megint egy olyan tulajdonság, amit jó lenne a katalógusban tárolni.

A *Ne bántsátor a feketerigót!* című könyvet egy 1-től 5-ig terjedő skálán 5-ösre értékelném. Hogy hozzáadjam a könyvhöz ezt az értékelést [a saját binemben](#) (ami angolul van), rákattintottam a „Run with JS” gombra, beírtam a következőt a konzolpanelbe, majd Entert nyomtam:

```
myBook.rating = 5;
```

Most, hogy hozzáadtuk az értékelést a könyvhöz, a konzol az 5-ös számot írja ki. **Miután hozzáadunk egy új tulajdonságot az objektumhoz, a konzol kiírja az új tulajdonság értékét.**

Ha vetsz egy pillantást a szintaxisra, akkor láthatod, hogy lényegében ugyanolyan, mint a tulajdonság értékének megváltoztatásakor:

```
objectName.propertyName = value;
```

vagy

```
objectName['propertyName'] = value;
```

Ahol:

- az **objectName** az objektum neve,
- a **propertyName** a tulajdonság neve, és

- a `value` a tulajdonság értéke.

Most próbáld ki ezt a saját könyved értékelésével. Nyisd meg a korábban JS Binen elmentett binedet, és kattints a „Run with JS” gombra. A konzolpanelben adj egy új tulajdonságot az objektumhoz. A tulajdonság neve `rating` lesz (ami angolul értékelést jelent), értéke pedig egy 1 és 5 közötti szám.

Ha kész vagy, írasd ki a konzollal az objektumot: írd be, hogy `myBook`; , majd nyomj Entert. Látni fogod, hogy az új tulajdonság sikeresen hozzá lett adva. Gratulálunk!

### Feladat

Az internet okkal a megszállottja a macskás videóknak. Néhány cica olyan furcsán néz ki, hogy nem is kell semmit sem csinálniuk ahhoz, hogy megnevettesseket. Itt van egy objektum néhány vicces macskafajtával és azok szokatlan jellemzőivel: `var cats = {'Selkirk Rex': 'göndör szőr', 'Scottish Fold': 'gyűrött fül', 'Exotic Shorthair': 'lapos orr'};`. Te melyik macskát találod a legkülönösebbnek? Add hozzá a `cats` objektumhoz a furcsa jellegzetességgel együtt.

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- Adj hozzá az objektumhoz még egy macskát a pont- vagy zárójeljelölés használatával.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

### 3.3. Objektum tulajdonságának ellenőrzése

Az előző feladatban egy új tulajdonságot adtál hozzá az objektumhoz a konzolban: a könyv értékelését. Ahhoz, hogy lásd, hogyan értékelted a könyvet, egyszerűen csak el kell érned a tulajdonságot, ugye?

Próbáljuk ki! Nyisd meg a korábban elmentett binedet (az enyém most `így` néz ki angolul). Kattints a "Run with JS" gombra ahhoz, hogy lefuttasd a JavaScript-kódodat, majd írd be a következőt a konzolpanelbe, és nyomj Entert:

`myBook.rating;`

Álljunk csak meg egy pillanatra! Az értékelés helyett a konzol azt írta ki, hogy `undefined`. Úgy látszik, a `rating` tulajdonság nem lett meghatározva. Akkor létezik egyáltalán ez a tulajdonság?

Az ellenőrzéshez írd be az alábbiakat a konzolba, és nyomj Entert:

```
'rating' in myBook;
```

A konzol a `false` booleanértéket írta ki, ami azt jelenti, hogy nem találta a `rating` tulajdonságot a `myBook` objektumban.

Hogy miért? Mert amikor hozzáadtad ezt az új tulajdonságot, a konzolpanelben dolgoztál, és – ahogy már korábban is említettük – minden, amit a konzolba írsz, elvész, amikor becsukod a lapot, vagy újrafuttatod a JavaScript-kódot.

**Hogy lásd, az objektum rendelkezik-e ezzel a tulajdonsággal, az `in` kulcsszót használtad, ami `true` értéket ír ki, ha létezik a tulajdonság, illetve `false` értéket, ha nem.**

Ez az általános szintaxisa annak, hogy miként ellenőrizzünk egy objektumot egy adott tulajdonságra nézve az `in` kulcsszóval:

```
'propertyName' in objectName;
```

Ahol:

- az `objectName` az objektum neve, és
- a `propertyName` a tulajdonság neve.

Próbáljuk ki egy olyan tulajdonságon, amelyik biztosan létezik. Írd be a következőt a konzolba, és nyomj Enter-t:

```
'title' in myBook;
```

A konzol a `true` booleanértéket írta ki, ami azt jelenti, hogy megtalálta a `title` tulajdonságot a `myBook` objektumban.

Most, hogy tudod, a könyv értékelése nem lett eltárolva, mentsd el rögtön úgy, hogy hozzáadod a tulajdonságot egy utasításban a JavaScript-panelben lévő kód végén. Valahányszor lefuttatod a JavaScript-kódot, ez a parancs végre lesz hajtva, így az osztályozás bekerül az objektumba. Az én könyvem például [így néz ki](#) most, a tulajdonság JavaScript-kódban való hozzáadása után.

Hogy biztosra menjünk, ellenőrizzük le újra, hogy létezik-e a tulajdonság. Kattints a "Run with JS" gombra a frissített JavaScript-kód futtatásához, majd írd be a konzolpanelbe az alábbiakat, és nyomj Enter-t:

```
'rating' in myBook;
```

A konzol azt írta ki, hogy `true`, mert ez alkalommal megtalálta a `rating` tulajdonságot a `myBook` objektumban.

**Megjegyzés:** Van egy másik módszer is arra, hogy lássuk, egy objektum rendelkezik-e egy adott tulajdonsággal: a `.hasOwnProperty()` metódus. Az általános szintaxisa a következő: `objectName.hasOwnProperty('propertyName');`. Tehát ha ellenőrizni akarjuk a `myBook`-ot a `rating` tulajdonságra, akkor ezt is beírhatjuk a konzolba: `myBook.hasOwnProperty('rating');`. Próbáld csak ki!

Most már tudod, hogyan kell megnézni, hogy egy objektum rendelkezik-e egy tulajdonsággal. Nagyon menő vagy!

### Feladat

Kata szeretne egy szobát bérelni. Van egy kutyája, úgyhogy olyan lakást keres, ahova lehet háziállatot vinni. Egy barátja ajánlott neki egy jó helyet az alábbi tulajdonságokkal: `var apartment = {'type of the building': 'téglá', 'floor': '4. emelet', 'number of rooms': '3 szoba', 'size': '62 négyzetméter', 'view': 'kert', 'pets allowed': ['kutyák', 'macskák'], 'shortest rental period': 'legalább egy év'};`. Segíts neki ellenőrizni, hogy lehet-e háziállatot tartani ebben a lakásban.

A programod a következő kritériumoknak feleljén meg:

- Hozd létre a fenti objektumot.
- Ellenőrizd, hogy az objektumnak van-e `pets allowed` nevű tulajdonsága (amely azt jelenti, hogy „megengedett háziállatok”).

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 3.4. Összefoglalás

Ebben a leckében megtanultad, hogyan kell az objektum tulajdonságait megváltoztatni, hozzáadni és ellenőrizni. Remek munka!

Itt van az általános szintaxisuk:

- egy objektumtulajdonság értékének megváltoztatása:
  - `objectName.propertyName = newValue;`  
vagy
  - `objectName['propertyName'] = newValue;`
- egy új objektumtulajdonság hozzáadása:
  - `objectName.propertyName = value;`  
vagy
  - `objectName['propertyName'] = value;`
- egy objektumtulajdonság ellenőrzése:
  - `'propertyName' in objectName;`

- vagy
- `objectName.hasOwnProperty('propertyName');`

A következő leckében azt tanulod meg, hogyan lehet eltárolni ezeket az utasításokat az objektummodban függvényként.

## 4. Objektummetódusok

### 4.1. Objektummetódus létrehozása

Amikor egy könyvet olvasol, akkor természetesen megjelölök azt az oldalt, ahol abbahagytad, hogy később onnan tudsz folytatni. Hozzunk létre egy függvényt a könyvobjektumban erre a célra, amely azt is meg tudja mondani, hogy hány oldalt olvastál az utolsó alkalom óta.

Nyisd meg a könyved a JS Binen ([ez](#) az enyém jelenlegi állapota, angolul). A JavaScript-panelben illessz be egy tulajdonságot az objektumba, egyből a kölcsönzési állapot után. A neve `page marker` lesz (ami oldaljelölőt jelent), az értéke pedig egy függvény:

```
'page marker': function () {}
```

A fenti sorral meghatároztál egy olyan függvényt, amely egy objektumtulajdonság. **Azt a függvényt, ami egy objektum tulajdonsága, metódusnak nevezzük.**

**Megjegyzés:** A JavaScriptben sok beépített metódus van, amelyek közül jó párat már ismersz is, mint a `.prompt()`, az `.alert()`, a `.push()`, a `.pop()`, az `.unshift()`, a `.shift()`, a `.splice()`, és a `.hasOwnProperty()`.

Biztosan észrevettek már, hogy nem kellett megnevezned a függvényt, mint általában, mert a tulajdonság neve fogja meghívni a metódust.

Szóval mit is csináljon a metódusunk? Először is, kérdezze meg, hogy melyik oldalon tartasz. Ennek a legegyszerűbb módja a `.prompt()` beépített JavaScript-metódus használata, amely megjelenít egy üzenetet és egy, a válasznak szánt szövegmezőt:

```
'page marker': function () {
 prompt('Melyik oldalon tartasz?');
}
```

Éljön, megírtad az első objektummetódusod! (Az én objektumom [így](#) néz ki a page marker metódussal.)

Jó lenne megnézni, hogy működik-e. A következő oldalon ki fogjuk ezt próbálni, de előtte íme az objektummetódusok általános szintaxisa:

```
var objectName = {
 propertyName: function () {
 // végrehajtandó kód
 }
};
```

Ahol:

- az `objectName` az objektum neve, és
- a `propertyName` a tulajdonság neve.

Feladat

Nézzük meg újra annak a finom csokis brownie-nak az összetevőit: `var brownie = {'butter': '1/2 bögre', 'sugar': '1 bögre', 'unsweetened cocoa powder': '1 bögre', 'eggs': '2 nagy darab', 'flour': '1/2 bögre'};`. A brownie megsütéséhez először össze kell keverned mindenzt egy tálban. A `brownie` objektumon belül írj egy metódust, amely megjeleníti ezt az utasítást.

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- Használd a `console.log()`-ot az objektummetódusban az utasítás kiírásához.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 4.2. Objektum Metódus elérése

Akkor most próbáljuk ki a vadi új `page marker` metódusunkat!

Nyisd meg a könyvedet a JS Binen (ha szükséged van egy mintabinre, [itt az enyém](#) angolul), és kattints a „Run with JS” gombra a JavaScript-kód futtatásához.

Itt az idő, hogy meghívjuk a metódust. De hogyan is kell azt?

A metódusok objektumtulajdonságok, ezért úgy érheted el őket, ahogyan bármilyen másik tulajdonságát az objektumnak: pont- vagy zárójeljelöléssel. Tehát írd be a következőt a konzolpanelbe, és nyomj Enter-t:

```
myBook['page marker'];
```

Valami történt, de nem egészen az, amire számítottunk. Kiadta a függvény definícióját, de a metódus nem lett végrehajtva. Ha jobban belegondolsz, van benne ráció: amikor elérsz egy

tulajdonságot, az értékét kapod meg, és a metódus értéke a függvény definíciója.

A metódus tényleges meghívásához hozzá kell adnod a zárójelet, mint általában, amikor függvényt használsz. Úgyhogy írd be a konzolba a következőt, és nyomj Enter-t:

```
myBook['page marker']();
```

Hurrá! Felugrott egy kis ablak, amely megkérdezi, hogy melyik oldalon tartasz. Úgy tudod ezt megválaszolni, hogy beírsz egy számot a szövegmezőbe, és az OK-ra kattintasz.

A metódusok meghívásának általános szintaxisa az alábbi:

```
objectName['propertyName']();
```

vagy

```
objectName.propertyName();
```

Ahol:

- az **objectName** az objektum neve, és
- a **propertyName** a tulajdonság neve.

A metódusod most már meg tudja kérdezni az oldalszámot, ahol tartasz. Ez tök jó, de ahhoz, hogy ténylegesen megjelöljük az oldalt, el is kellene tárolnunk a választ. Ehhez a következő oldalon kibővítjük a metódust.

Lássunk hozzá a sütéshez! Hívd meg a **brownie** objektumon belül korábban létrehozott metódust. Írjon ki egy utasítást, miszerint össze kell keverned az összes összetevőt egy tálban.

#### 4.3. Oldalszám elmentése egy tulajdonságba

Nyisd meg a korábban elmentett könyvedet a JS Binen ([ez](#) az én binem angolul), és nézd meg a **page marker** metódust a JavaScript-panelen. Már meg tudja kérdezni, hogy melyik oldalon tartasz, de el is kellene mentenie a választ.

A kérdés az, hogy miben tároljuk azt. Persze egyszerűen hozzárendelhetnénk egy változóhoz, de jobb, ha strukturáltan tartjuk az adatokat az objektumban.

Mivel az oldalszám egy másik tulajdonsága a könyvnek, adjunk hozzá egy új tulajdonságot a **myBook** objektumhoz, és rendeljük hozzá a választ. A tulajdonság neve legyen **page** (ami azt jelenti, hogy „oldal”), az értéke pedig az a szám, amelyet a **.prompt()**-tal adsz meg:

```
'page marker': function () {
 myBook.page = prompt('Melyik oldalon tartasz?');
```

{}

A metódusunk most már meg tudja jelölni az oldalt. Éljen!

Ki tudod próbálni még egyszer, ha a "Run with JS" gombra kattintasz, és meghívod a metódust. Írd be ezt a konzolba, majd nyomj Entert:

```
myBook['page marker']();
```

Látszólag semmi nem változott, azonban az oldalszám most már mentésre került.

Azért jó lenne, ha kapnánk valamiféle visszaigazolást. Erre a célra a JavaScript egy másik beépített metódusát fogunk használni: az `.alert()`-et, amely megjelenít egy üzenetet egy kis felugró ablakban.

```
'page marker': function () {
 myBook.page = prompt('Melyik oldalon tartasz?');
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
az alábbi könyvedben: ' + myBook.title + '.');
}
```

Ezzel a metódus meg tud jeleníteni egy üzenetet, jelezve számodra, hogy hányadik oldalon tartasz. (Az én binem [így](#) néz ki most.)

Próbáld ki! Nyomd meg a „Run with JS” gombot a binedben, hogy lefuttasd a frissített JavaScript-kódöt, majd hívd meg újra a metódust. Írj egy számot a szövegmezőbe, és kattints az OK-ra. Ha minden rendben ment, akkor felugrott egy másik ablak, visszaigazolva az oldalszámot, amelynél épp tartasz. Tökéletes!

Még egy dolgot szeretnénk, ha a metódus tudna: mondja meg, hogy hány oldalt olvastál az utolsó alkalom óta. Ezzel fogunk foglalkozni a következő oldalon.

## Feladat

Szereted Leonardo DiCapriót, a nagyszerű amerikai színészt, aki végre nyert egy Oscar-t? Itt van egy objektum néhány információval róla: `var leonardoDiCaprio = {'born': 1974, 'birth name': 'Leonardo Wilhelm DiCaprio', 'most popular movies': ['Titanic', 'A téglá', 'Viharsziget', 'Eredet', 'Django elszabadul']};`. Hozz létre egy metódust ezen az objektumon belül, amely hozzáad egy új tulajdonságot. Az új tulajdonság fogja tárolni a film címét, amellyel Oscar-t nyert.

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- Az új tulajdonság értékét a `.prompt()` segítségével kérdezd meg.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

#### 4.4. Az elolvasott oldalak számának kiszámolása

Nyisd meg a könyvedet a JS Binen ([itt](#) az én binem angolul). A `page marker` metódus jelenleg meg tudja kérdezni, hogy melyik oldalon tartasz, elmenti a választ egy tulajdonságba, és visszaigazolja az oldalszámot, de még nem tudja megmondani, hány oldalt olvastál az utolsó alkalom óta.

Próbáljuk meg kiszámolni az olvasott oldalak számát úgy, hogy a korábbi oldalszámot kivonjuk a mostaniból. Ezt az információt beilleszthetjük az `.alert()`-tel kiírt üzenetbe:

```
'page marker': function () {
 myBook.page = prompt('Melyik oldalon tartasz?');
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page + ' az alábbi könyvedben: ' + myBook.title + '. Legutóbb az alábbi oldalon tartottál: ' + myBook.page + ', így ez alkalommal ' + (myBook.page - myBook.page) + ' oldalt olvastál.');
}
```

Hm, ez nem tűnik jónak. Úgy látszik, hogy a metódusban mindenhol ugyanazt az oldalszámot használjuk, kivonva saját magából.

Próbáld ki te is! A frissített JavaScript-kód futtatásához kattints a „Run with JS” gombra, és hívд meg a metódust. Nem számít, hogy hányszor hajtod végre és milyen oldalszámot adsz meg, mindig azt mondja, hogy ugyanazon az oldalon tartasz, mint legutóbb, és azóta 0 oldalt olvastál.

Miért? Mert a `myBook.page = prompt('What page are you on?');` sorral kicséréljük a `page` tulajdonság korábbi értékét egy új számra. Aztán a `myBook.page`-nek ez az új értéke kerül felhasználásra a metódusban mindenhol.

Ahhoz, hogy ki tudjuk számolni a különbséget a korábbi és az új oldalszám között, először el kell mentenünk a kezdeti oldalszámot valahova, a `page` tulajdonságon kívülre. Így a `myBook.page` értékének a `.prompt()`-tal való frissítése után is tudjuk majd azt használni.

A `myBook.page` eredeti értékének tárolásához hozzáadhatunk egy új tulajdonságot a `myBook`-hoz, és hozzárendelhetjük ahoz a kezdeti oldalszámot. Ugyanakkor erre az adatra nem lesz szükségünk többet, ezért nincs értelme egy új tulajdonságot létrehozni neki. Ehelyett inkább rendeljük hozzá a régi oldalszámot egy helyi változóhoz, amelyet nevezünk `pageInitial`-nek:

```
'page marker': function () {
 var pageInitial = myBook.page;
 myBook.page = prompt('Melyik oldalon tartasz?');
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
az alábbi könyvedben: ' + myBook.title + '. Legutóbb az alábbi
oldalon tartottál: ' + myBook.page + ', így ez alkalommal ' +
(myBook.page - myBook.page) + ' oldalt olvastál.');
}
```

Most már a `pageInitial` használatával meg tudjuk mutatni azt az oldalszámot, amelynél utoljára tartottál, valamint ki tudjuk számolni az azóta olvasott oldalak számát, kivonva a `pageInitial`-t a `myBook.page` frissített értékéből. Gyerünk, javítsuk ki az `.alert()`-tel megjelenített üzenetet:

```
'page marker': function () {
 var pageInitial = myBook.page;
 myBook.page = prompt('Melyik oldalon tartasz?');
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
az alábbi könyvedben: ' + myBook.title + '. Legutóbb az alábbi
oldalon tartottál: ' + pageInitial + ', így ez alkalommal ' +
(myBook.page - pageInitial) + ' oldalt olvastál.');
}
```

Jól néz ki (az enyém például [így](#)), de nézzük meg, hogy működik-e is!

Kattints a „Run with JS” gombra, és hívd meg a frissített metódust.

Felugrott egy ablak, amely megkérdezi, hogy melyik oldalon tartasz, pont úgy, ahogy vártuk. Írj be egy számot a szövegmezőbe, és kattints az OK-ra.

Megjelent egy másik ablak, amely visszaigazolja azt az oldalt, ahol most tartasz, valamint megmondja, hogy legutóbb az `undefined` oldalon voltál, így `NaN` oldalt olvastál ez alkalommal.

Húha, ez egy elég furcsa üzenet! De mit jelent? Elmagyarázzuk a következő oldalon.

### Feladat

Marczinak nagy tervei vannak Jancsi szülinapi ajándékát tekintve, egy jó ideje spórol már rá. Itt egy objektum, amely a legutóbbi bevételét (`income`) és kiadásait (`expenses`) tárolja: `var budget = {'income': 8000, 'expenses': 5952};`. Hozz létre egy metódust ezen az objektumon belül, amely kiszámolja Marcsi új megtakarításait azáltal, hogy kivonja a kiadásait a jövedelméből.

A programod a következő kritériumoknak feleljén meg:

- Hozd létre a fenti objektumot.
- A metódus írja ki a megspórolt összeget az `.alert()` segítségével.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

#### 4.5. A kiírt üzenet megértése

Ha megnyitod a `myBook` objektumot a JS Binben ([ez](#) az én binem jelenlegi állapotá angolul), futtatod a JavaScript-kódot a "Run with JS" gombra való kattintással, majd meghívod a `page marker` metódust, fura üzenetet fogsz kapni: legutoljára az `undefined` oldalon voltál, így ezúttal `NaN` oldalt olvastál.

Nézzük, mit is jelent ez.

Először is azt mondja, hogy a korábbi oldalszám – a `pageInitial` értéke – `undefined`. A `pageInitial` értéke egyenlő a `myBook.page` eredeti értékével. Tehát a valódi kérdés itt az, hogy mi volt a `myBook.page`, mielőtt értéket adtunk neki a `.prompt()`-tal.

Nos, a `page` tulajdonság nem is létezik addig, amíg értéket nem rendelünk hozzá a `.prompt()`-tal, ha pedig az objektumban egy nem létező tulajdonságot próbálsz elérni, az `undefined` érték fog megjelenni. Ennek eredményeként addig, amíg ténylegesen létre nem hozod a `page` tulajdonságot azáltal, hogy értéket adsz neki, az `undefined` lesz hozzárendelve a `pageInitial`-höz.

**Megjegyzés:** Valahányszor becsukod a bint, vagy újra lefuttatod a JavaScript-kódot a „Run with JS” gomb megnyomásával, az alkalmazással folytatott minden korábbi kommunikáció elvész, akár a konzolba, akár a `.prompt()`-tal megjelenített szövegmezőbe írtad azt. Így ha meg is adtál már egy oldalszámot, az el fog veszni, ha újra lefuttatod a kódot, mert jelenleg nem mentjük a tartalmat adatbázisokba.

Az üzenet azt is mondja, hogy ezúttal `NaN` oldalt olvastál. De mi ez a `NaN`?

Az `NaN` a **Not-A-Number**-t jelöli (amely annyit tesz magyarul, hogy "nem egy szám"), és akkor kerül kiírásra, ha egy matematikai műveletet nem lehet végrehajtani, mert az eredmény nem egy szám. Ebben az esetben megpróbáltuk kivonni az `undefined` értéket egy számból, és mivel az `undefined` nem egy szám, így az eredmény se volt szám.

A metódus első meghívása után már lesz egy szám hozzárendelve a `myBook.page`-hez. Ha még egyszer meghívod a metódust, akkor látni fogod, hogy a második meghívástól kezdve tökéletesen fog működni. Próbáld ki te is!

Amikor másodszor hívod meg a metódust, még egyszer meg fog kérdezni az oldalszámról.

Írd be a számot a mezőbe, és kattints az OK-ra.

És tessék, az alkalmazásod megmondja neked nemcsak azt az oldalt, amelyet éppen olvasol, hanem azt az oldalt is, ahol legutoljára tartottál, valamint az azóta elolvasott oldalak számát is. Ez azért elég jó, ugye?

Így már csak egy dolgot kell helyreraknunk: ha nincsen kezdeti oldalszám elmentve, akkor a metódusnak csak azt az oldalt kellene visszaigazolnia, ahol éppen tartasz, és nem kéne megjelenítenie sem a kezdeti oldalszámot, sem az olvasott oldalak számát.

Ezzel a következő leckében fogunk foglalkozni.

Az alábbi objektum áll rendelkezésre:

```
var odometer = {'initialValue': null, 'finalValue': 250, 'difference': function () {console.log('A különbség az ' + odometer.initialValue + ' és ' + odometer.finalValue + ' között ' + (odometer.finalValue - odometer.initialValue) + '.')}};
```

Hívд meg az `odometer.difference()` metódust, hogy lásd a különbséget az `odometer.initialValue` és az `odometer.finalValue` között. Mit írt ki a konzol? Mit gondolsz, mi magyarázza a kapott értéket? (A Stack Overflow egy kitűnő hely, ha választ keresel.)

## 4.6. A megfelelő üzenet megjelenítése

Nyisd meg a korábban elmentett `myBook` objektumot a JS Binen ([ez](#) az enyém angolul). A `page marker` metódus már majdnem kész, kivéve, hogy furán viselkedik, ha nincs elmentve kezdeti oldalszám. Hogy ezt kijavítsuk, ilyen esetben egy másfajta visszaigazoló üzenetre van szükségünk.

Ahhoz, hogy a megfelelő üzenetet jelenítsük meg, szükségünk lesz egy feltételes állításra. Rendben, de mi legyen a feltételünk?

Amíg el nem mentesz egy oldalszámot, a `pageInitial`-höz hozzárendelt érték `undefined`. Azaz ha a `pageInitial` értéke `undefined`, csak a jelenlegi oldalszámot kellene megmutatnia, míg más esetekben kiírhatjuk az elolvasott oldalak számát is.

Nézzük meg, hogyan tudjuk ellenőrizni, hogy a `pageInitial undefined`-e.

Kattints a "Run with JS" gombra, majd írd be a következőket a konzolba, és nyomj Entert:

```
typeof pageInitial;
```

Azt írta ki, hogy "`undefined`".

Az imént a `typeof`` műveletet használtad, amely megjelenít egy stringet az adat típusáról. Ha a változó értéke `undefined`, vagy a változó nem lett deklarálva, a kiírt string "`undefined`" lesz.

**Megjegyzés:** A `typeof` művelet stringek (pl. a `myBook.title`) esetén azt írja ki, hogy `"string"`, számoknál (pl. `myBook['year of publication']`) azt, hogy `"number"`, és booleanértékeknél (pl. `myBook['has been read']`) azt, hogy `"boolean"`. Azt fogja kiadni, hogy `"object"`, ha objektumokról van szó (pl. a `myBook` és a `myBook.quotes` esetében), és akkor is, ha tömbökről beszélünk (pl. `myBook['main characters']`), mivel a tömbök a JavaScriptben az objektumok különleges fajtáját képezik. Sőt, a `typeof` még a `null` esetén is azt fogja megjeleníteni, hogy `"object"`, ami igazából egy bug a JavaScriptben. Függvényeknél, beleértve az objektummetódusokat (pl. a `myBook['page marker']`-t) is, a `typeof` azt írja ki, hogy `"function"`.

Szóval, a feltételünk az alábbi lesz: `typeof pageInitial === 'undefined'`.

Szigorú összehasonlítást használunk (`==`), ami azt jelenti, hogy az állítás csak akkor lesz igaz, ha annak bal és jobb oldala ugyanolyan értékű és adattípusú.

**Megjegyzés:** Ha a `==` egyenlőségi művelettel összehasonlítod azt, hogy `3` és `"3"` (`3 == "3"`), akkor `true` fog megjelenni, ami azt jelenti, hogy egyenlők. Ha viszont a `==` szigorú egyenlőségi művelettel hasonlítod öket össze (`3 === "3"`), akkor `false` fog megjelenni. Miért? Mert a `==` egyenlőségi művelet átalakítja öket azonos típusúvá, míg a `==` szigorú egyenlőségi művelet ezt nem teszi meg. Mivel a `3` egy szám, a `"3"` pedig egy string, ha szigorúan hasonlítjuk öket össze, akkor nem lesznek egyenlők, mert más típusúak.

Ha a feltétel igaz, akkor csak a jelenlegi oldalszám lesz megjelenítve a `alert()`-tel, a többi esetben emellett megjelenik a korábbi oldal és az olvasott oldalak száma is. Tehát így néz ki a metódusunk a feltételes állítással:

```
'page marker': function () {
 var pageInitial = myBook.page;
 myBook.page = prompt('Melyik oldalon tartasz?');
 if (typeof pageInitial === 'undefined') {
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
 ' az alábbi könyvedben: ' + myBook.title + '.');
 } else {
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
 ' az alábbi könyvedben: ' + myBook.title + '. Legutóbb az alábbi
 oldalon tartottál: ' + pageInitial + ', így ez alkalommal ' +
 (myBook.page - pageInitial) + ' oldalt olvastál.');
 }
}
```

Azta, elég szuper metódust hoztál létre! Nyugodtan játszadozz még vele. Megdolgoztál érte! Ha szükséged van egy mintabinre, [itt](#) az enyém.

**Megjegyzés:** Még mindig van néhány fejleszthető dolog a metódusban. Például, amikor olyan oldalszámot adsz meg, amelyik kisebb a korábbinál, akkor az üzenetben most negatív szám jelenik meg az olvasott oldalak számaként. Ez elég viccesen hangzik, így talán jobb is

lenne, ha ilyenkor egy másik üzenet jelenne meg. Ha van hozzá kedved, oldd meg ezt a problémát.

Az előző feladatban az `NaN`-ről tanultál, amely értéket akkor kapod meg, ha egy matematikai műveletet nem lehet végrehajtani, mert a végeredmény nem szám. Mit gondolsz, milyen adattípusú az `NaN`? Nézd meg a `typeof` művelettel, és másold be alább a kapott stringet.

## 4.7. Összefoglalás

Ebben a leckében megismerkedtél az objektummetódusokkal. Már tudod, hogy hogyan kell:

- megírni egy objektummetódust:

```
var objectName = {
 propertyName: function () {
 // végrehajtandó kód
 }
};
```

- meghívni egy objektummetódust:
  - `objectName.propertyName()` ; vagy
  - `objectName[ 'propertyName' ]()` ;

Fantasztikus! A következő leckében még többet fogsz tanulni a metódusokról.

## 5. A `this` kulcsszó és a paraméterek használata metódusokban

### 5.1. Több könyv elmentése a katalógusba

Eljött az idő, hogy további könyveket mentünk el a katalógusba.

Először is, nyisd meg a korábban elmentett binedet a JS Binben ([ez](#) az enyém angolul).

A JavaScript-panelben nevezd át a `myBook`-ot `myBook1`-ra, mert ez lesz az első könyv a katalógusodban. Töröld ki az osztályozás hozzáadásához használt utasítást (pl. `myBook.rating = 5;`), és helyette írd a `rating` tulajdonságot közvetlenül az objektumba (pl. `'rating': 5`). Illeszd azt be a kölcsönzési státusz és a `page marker` metódus közé. Ne változtass meg semmi mást ebben az objektumban.

Ezután hozz létre két új objektumot `myBook2` és `myBook3` névvel, egy-egy könyvedet eltárolva bennük. Az új objektumoknak ugyanazok lesznek a tulajdonságnevei, mint a `myBook1`-nak, de ezek természetesen az adott könyvekhez tartozó értékeket fogják tárolni. Hagyd a `page marker` metódust változatlanul az objektumokban. (Jelenleg az én binem [így](#) néz ki.)

Ha kész vagy, kattints a "Run with JS" gombra a frissített JavaScript-kód futtatásához, majd hív meg a `page marker` metódust a második könyvben. Írd be a következőt a konzolba, és nyomj Enter-t:

```
myBook2['page marker']();
```

Jaj ne! Hibaüzenetet kaptál:

```
"myBook is not defined"
```

Az üzenet azt mondja, hogy a `myBook` nem lett meghatározva. Nos valóban, a `myBook` nem lett meghatározva, mert átneveztük `myBook1`-ra. De ennek mi köze van a `page marker` metódushoz?

Nézzük csak! A metódus most így néz ki:

```
'page marker': function () {
 var pageInitial = myBook.page;
 myBook.page = prompt('Melyik oldalon tartasz?');
 if (typeof pageInitial === 'undefined') {
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
 ' az alábbi könyvedben: ' + myBook.title + '.');
 } else {
 alert('Jelenleg az alábbi oldalon tartasz: ' + myBook.page +
 ' az alábbi könyvedben: ' + myBook.title + '. Legutóbb az alábbi
 oldalon tartottál: ' + pageInitial + ', így ez alkalommal ' +
 (myBook.page - pageInitial) + ' oldalt olvastál.');
 }
}
```

Mivel nem változtattuk meg ezt a metódust, ezért az még mindig egy nem létező `myBook` objektum tulajdonságaihoz próbál hozzáérni: a `myBook.page` és a `myBook.title` a `myBook page` és `title` tulajdonságaira hivatkozik, nem a `myBook2` tulajdonságaira.

Ezt a problémát megoldhatjuk úgy, hogy a `myBook`-ot kicseréljük `myBook1`-ra, `myBook2`-ra vagy `myBook3`-re az összes `page marker` metódusban, ahogyan azt én is megettettem [itt](#). minden egyes könyvben 7 cserét kell végrehajtanod. Elég sziszifuszi, ugye?

Jó lenne, ha létezne egy kényelmes megoldás arra, hogy ugyanazt a metódust használni tudnod több objektumban is. Na, mit gondolsz? Hát persze hogy létezik ilyen. A következő oldalon fogsz róla tanulni.

Hogy meggyőződj arról, hogy a `page marker` metódusok minden könyved esetében rendesen működnek, próbáld ki minden egyesével. (Mielőtt meghívánod a metódusokat, ne felejts el a "Run with JS" gombra kattintani, hogy a friss JavaScript-kódod fusson). Másold be alább az utasításokat, amelyeket a `page marker` metódusok meghívásához használtál.

## 5.2. A this kulcsszó használata

Szóval, hogyan tudod ugyanazt a metódust betenni több objektumba is, anélkül, hogy minden egyes jelölést át kelljen írnod a metódusban, hogy az adott objektum tulajdonságait érje el?

Nyisd meg a JavaScript-panelt a korábban elmentett binedben a JS Binen ([itt](#) az enyém angolul). A `page marker` metódusokban cseréld ki a `myBook1`, a `myBook2` és a `myBook3` minden a 7 előfordulását arra a szóra, hogy `this`, ahogy én is megtettem [itt](#):

```
'page marker': function () {
 var pageInitial = this.page;
 this.page = prompt('Melyik oldalon tartasz?');
 if (typeof pageInitial === 'undefined') {
 alert('Jelenleg az alábbi oldalon tartasz: ' + this.page +
az alábbi könyvedben: ' + this.title + '.');
 } else {
 alert('Jelenleg az alábbi oldalon tartasz: ' + this.page +
az alábbi könyvedben: ' + this.title + '. Legutóbb az alábbi
oldalon tartottál: ' + pageInitial + ', így ez alkalommal ' +
(this.page - pageInitial) + ' oldalt olvastál.');
 }
}
```

Elsőre furának tűnhet, de ez az egyszerű kis `this` szócska tökéletesen működik minden objektumban. Próbáld ki te is! Kattints a „Run with JS” gombra a frissített JavaScript-kód futtatásához, és hívd meg a `page marker` metódust bármelyik könyvedben.

Ugye? Pompásan működik.

**De mi ez a `this` varázsszó? Ez egy kulcsszó a JavaScriptben, amely bármilyen objektummetódusban használható, és mindig arra az adott objektumra hivatkozik, amelyikben a metódus szerepel.**

Hogy jobban megértsd, nyisd meg [ezt a bint](#) egy sample objektummal, és hívd meg a `print object` metódust. Kattints a „Run with JS” gombra, majd írd be a konzolba, hogy `sample['print object']()`; és nyomj Entert.

Az egész `sample` objektumot kiírta a `console.log(this)`; mert a `this` arra az objektumra hivatkozik, amelyik a "tulajdonosa" a `this`-t használó metódusnak.

**Megjegyzés:** Ne aggódj amiatt a különös `window.runnerWindow.proxyConsole.log` miatt, az csak egy bonyolultabb formája a `console.log`-nak.

Nézzünk egy másik példát! [Itt van](#) egy bin két objektummal. Mindkét objektumnak van egy-egy `print property` metódusa, amelyeknek a definíciója a `printProperty` változóhoz

hozzárendelt érték.

**Megjegyzés:** A korábbi JavaScript-tanulmányaid során megtanultad, hogyan kell függvényeket definiálni függvénydeklarációval: `function functionName() {}`, ahol a `functionName` a függvény neve. A fenti binben ehelyett függvénykifejezést használunk, kihagyva a függvény nevét: `function () {}`. Azért nem szükséges a függvény neve, mert a függvénykifejezés tárolható egy változóban, és a változó nevét használhatjuk a függvény meghívására. Jelen esetben a tulajdonságnevét fogjuk használni a metódus meghívására.

Most hívjuk meg a `sampleOne` objektum `print property` metódusát. Kattints a „Run with JS” gombra, írd be a konzolba, hogy `sampleOne['print property']()`; , és nyomj Enter-t. A `sampleOne.name` tulajdonság értékét kapod meg: 1. De ha azt írod be helyette, hogy `sampleTwo['print property']()`; , akkor a `sampleTwo.name` értéke kerül megjelenítésre: 2.

Hogyan? A `console.log(this.name)`; utasításban a `this` szó lényegében helyettesíti a nevét bármelyik objektumnak, amelyikben szerepel a meghívott metódus. Tehát a `this.name` az adott objektum `name` tulajdonságát éri el.

A `this` nagyon jól fog jönni az objektumokkal való munkád során. Nemcsak rengeteg időt és energiát tud neked megspórolni, de fenntarthatóbbá is teszi a kódodat.

## Feladat

Ebbe a kódba írj egy újabb metódust a `sample` objektumon belül, amely ki tudja írni a `name` tulajdonság értékét.

A programod a következő kritériumoknak feleljen meg:

- Használd a `console.log()`-ot az objektummetódusban a `name` tulajdonság értékének a kiírásához.
- A metódusban használd a `this` kulcsszót az objektum neve helyett.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 5.3. A page marker metódus frissítése

Nézzük meg a könyveidet ([ezek](#) az én könyveim angolul).

Jelenleg minden egyes könyvedben ugyanaz a `page marker` metódus szerepel. Ha meg akarsz valamit változtatni a függvénydefinícióban, akkor minden egyes helyen javítanod kell azt. Például ahhoz, hogy a könyv címe szerepeljen a `.prompt()`-tal feltett kérdésben, ezt a sort kell megváltoztatnod:

```
this.page = prompt('Melyik oldalon tartasz?');
```

a következőre:

```
this.page = prompt('Melyik oldalon tartasz az alábbi könyvben: ' +
this.title + '?');
```

minden egyes könyvben.

Csináld is meg! Frissítsd a **page marker** metódust minden egyes könyvedben a fenti sorral. Nem túl kényelmes, igaz?

Sokkal praktikusabb lenne, ha egyetlen **page marker** metódust le lehetne futtatni az összes könyvön, és közben minden rendezett és strukturált maradna.

Ezt megoldhatjuk olyan módon, hogy beágyazzuk az összes könyvet egy új objektumba. Ekkor a **page marker** metódust ebbe át tudjuk tenni, de a beágyazott objektumokon kívülre. Ez az új objektum úgy fog viselkedni, mint egy e-könyv-olvasó: több könyvet is tárol egyszerre, és van benne egy könyvjelző, amellyel bármelyik könyvben be tudsz jelölni egy oldalt.

A következő leckében elkészítjük ezt az új objektumot lépésről lépésre.

#### 5.4. A könyvkatalógusod építése

Akkor most hozzunk létre egy új objektumot, amely az összes könyvedet, valamint a **page marker** metódust is tudja tárolni.

Nyiss egy új bint a JS Binen, és készíts egy új objektumot a JavaScript-panelben, amelynek a neve **bookCatalog**:

```
var bookCatalog = {};
```

Amikor készen vagy, hozz létre egy beágyazott objektumot a **bookCatalog** objektumon belül, amit **myBook1**-nak hívunk. Ahogy ennek a projektnek az elején említettük, a tulajdonságnév (**myBook1**) és a tulajdonságérték (jelenleg egy üres objektum: **{}**) kettősponttal van elválasztva:

```
var bookCatalog = {
 myBook1: {}
};
```

Mivel a stringeket aposztrófok közé kell tenni, így adjuk azokat a **myBook1**-hoz:

```
var bookCatalog = {
 'myBook1': {}
};
```

Mielőtt beírod a `myBook2`-t is, kell egy vesszőt tenned a `myBook1` értéke után, mivel a tulajdonságokat vesszővel választjuk el:

```
var bookCatalog = {
 'myBook1': {},
 'myBook2': {}
};
```

Valamint szükséged lesz vesszöre a `myBook2` értéke után is, mivel a `myBook3` jön utána:

```
var bookCatalog = {
 'myBook1': {},
 'myBook2': {},
 'myBook3': {}
};
```

Vigyázz, hogy ne tegyél vesszőt a `myBook3` értéke után, mert a vesszők csak tulajdonságok között szükségesek!

Készen állsz, hogy beilleszd az egyes könyvek tulajdonságait a kapcsos zárójelek közé, ahogy én is tettem [itt](#) (az én binem angolul van).

Ha befejezted a katalógusod könyvekkel való feltöltését, akkor helyezd át a `page marker` metódust a `bookCatalog` objektumba, a beágyazott objektumokon kívülre. Most már kitörölheted a metódust a könyvekből, és ne felejtsd el eltávolítani a vesszőt az utolsó tulajdonság után minden egyes könyvben. [Itt](#) a pillanathnyi állapota a binemnek.

Nagyon jól néz ki, de ha rákattintasz a „Run with JS” gombra, és meghívod a metódust a `bookCatalog['page marker']()`; konzolba való beírásával és egy Enterrel, akkor látni fogod, hogy a könyv címe helyett az van megjelenítve, hogy `undefined`:

Melyik oldalon tartasz az alábbi könyvben: `undefined`?

Miért `undefined` a cím? A metódus jelenlegi tulajdonosa a `bookCatalog` objektum, így a `this.title` a `bookCatalog` egy nem létező `title` tulajdonságára hivatkozik. Ahogy korábban elmagyaráztuk, ha egy olyan tulajdonságot próbálsz elérni, amelyik nem létezik, akkor `undefined` lesz a válasz.

Ha beírsz egy számot a szövegmezőbe, az oldalszám visszaigazolásra kerül. Ugyanakkor jelenleg a `page` tulajdonság a `bookCatalog`-hoz kerül hozzáadásra egy könyv helyett, és

mi nem ezt szeretnénk.

Ezeket a problémákat fogjuk megoldani a következő oldalon.

## 5.5. Paraméterek használata objektummetódusokban

Nyisd meg a bint, ahova elmentetted a `bookCatalog`-ot ([ez](#) az enyém angolul), és menj a JS Bin JavaScript-paneléhez.

Most már csak egy `page marker` metódus van a katalógusban:

```
'page marker': function () {
 var pageInitial = this.page;
 this.page = prompt('Melyik oldalon tartasz az alábbi könyvben: '
+ this.title + '?');
 if (typeof pageInitial === 'undefined') {
 alert('Jelenleg az alábbi oldalon tartasz: ' + this.page +
az alábbi könyvedben: ' + this.title + '.');
 } else {
 alert('Jelenleg az alábbi oldalon tartasz: ' + this.page +
az alábbi könyvedben: ' + this.title + '. Legutóbb az alábbi
oldalon tartottál: ' + pageInitial + ', így ez alkalommal ' +
(this.page - pageInitial) + ' oldalt olvastál.');
 }
}
```

Azonban ez a metódus nem működik megfelelően: a `this.page` és a `this.title` a `bookCatalog.title`-ra és a `bookCatalog.page`-re hivatkozik, egy katalóguson belüli könyv `title` és a `page` tulajdonságai helyett.

Mivel helyettesítsük a `this.page`-et és a `this.title`-t, hogy hozzáférjünk egy beágyazott könyv tulajdonságaihoz?

Korábban ebben a projektben megtanultad, hogy ha egy beágyazott objektum tulajdonságát akarod elérni, akkor össze kell kötnöd a pont- és a zárójeljelölést:

`objectName[ 'nestedObjectName' ].propertyName`

Ahol:

- az `objectName` az objektum neve,
- a `nestedObjectName` a beágyazott objektum neve, és
- a `propertyName` a tulajdonság neve.

Jelen esetben az elejére a `this` kulcsszót írjuk, és ezzel hivatkozunk a `bookCatalog` objektumra, amelyen belül a `page marker` metódus jelenleg van:

```
this['nestedObjectName'].propertyName
```

A jelölés végére írjuk a `page`-et és a `title`-t, mivel ez az a két tulajdonság, amelyeket el akarunk érni:

```
this['nestedObjectName'].page
```

és

```
this['nestedObjectName'].title
```

Oké, de mi jöjjön közéjük? Ez a furfangosabb rész, mert a beágyazott objektum nevének változnia kell, attól függően, hogy melyik könyvben akarsz oldalt megjelölni.

Ezért meg kell találnunk a módját annak, hogy a `page marker` metódussal tudassuk, melyik könyv tulajdonságait akarjuk elérni. Hogyan tudunk ilyen információt adni egy metódusnak?

Nos, erre szolgálnak a függvényparaméterek. Ha beírsz egy paramétert a zárójelek közé a függvény meghatározásakor (`'page marker': function (parameter) {...}`, ahol a `parameter` a paraméter), akkor majd tudsz argumentumot adni a függvénynek, amikor meghívod azt (`bookCatalog['page marker'](argument)` ; , ahol az `argument` az argumentum). A paraméter lényegében egy változó, az argumentum pedig ennek a változónak az aktuális értéke.

A `page marker` metódusban a paramétert `book`-nak fogjuk hívni (ami könyvet jelent), mert az argumentum egy beágyazott könyvobjektum tulajdonságneve (pl. `'myBook1'`) lesz:

```
'page marker': function (book) {...}
```

Ezt a `book` paramétert fogjuk használni a jelölésben:

```
this[book].page
```

és

```
this[book].title
```

Talán észrevettek, hogy nem használtunk aposztrófokat a `book` körül a szöglletes zárójelben. Miért nem? Mert szeretnénk, ha a JavaScript értené, hogy a `book` nem egy tulajdonságnév, hanem egy paraméter. A tényleges tulajdonságnév az argumentum lesz (pl. `'myBook1'`), amit megadunk a metódus meghívásakor.

**Megjegyzés:** Ahogy látod, a zárójeljelölés használható paraméterekkel, de jó észben tartani, hogy a pontjeljelölés nem. Ha megpróbálnánk itt az utóbbit használni

(`this.book.page`), egy hibát jelezne a JS Bin ("Cannot read property 'page' of undefined"), mert megpróbálná elérni a `page` tulajdonságát egy olyan beágyazott objektumnak, amelynek a tulajdonságneve `book`, ilyen pedig nem létezik.

Tehát a `page marker` metódus jelenlegi verziója így néz ki:

```
'page marker': function (book) {
 var pageInitial = this[book].page;
 this[book].page = prompt('Melyik oldalon tartasz az alábbi
könyvben: ' + this[book].title + '?');
 if (typeof pageInitial === 'undefined') {
 alert('Jelenleg az alábbi oldalon tartasz: ' +
this[book].page + ' az alábbi könyvedben: ' + this[book].title +
'.');
 } else {
 alert('Jelenleg az alábbi oldalon tartasz: ' +
this[book].page + ' az alábbi könyvedben: ' + this[book].title +
'. Legutóbb az alábbi oldalon tartottál: ' + pageInitial + ', így
ez alkalommal ' + (this[book].page - pageInitial) + ' oldalt
olvastál.');
 }
}
```

Ha szükséged van egy mintabinre a könyvkatalógus jelenlegi állapotával, akkor [itt](#) az enyém.

Kattints a "Run with JS" gombra, és hívd meg a frissített oldaljelölő metódust a konzolban.

Ahhoz, hogy megjelölj egy oldalt egy adott könyvben, meg kell adnod a könyv tulajdonságnevét a metódusnak, amikor meghívod azt. Úgy tudsz tulajdonságnevét megadni egy metódusnak, hogy beírod azt a zárójelbe aposztrófok közé, mint az alábbiakban:

```
bookCatalog['page marker']('myBook3');
```

Működik! Próbáld ki te is!

Most már annyi könyvet adhatsz a katalógushoz, amennyit csak akarsz, anélkül, hogy bármilyen problémát adódna a `page marker` metódussal.

## Feladat

A mozhősök gyakran igazi nagymenők. Itt egy objektum három ilyen fiktív karakter kereszt- és vezetéknévvel: `var characters = {'character1': {'first name': 'James', 'last name': 'Bond'}, 'character2': {'first name': 'Luke', 'last name': 'Skywalker'}, 'character3': {'first name': 'Jason', 'last name': 'Bourne'}};`. Írj egy metódust az objektumon belül, amely képes köszönteni valamelyik karaktert a teljes nevén (pl. `Hello James Bond!`).

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- Írj egy metódust, amely egy paramétert fogad, és amikor meghívod, a megadott argumentum az egyik beágyazott objektum tulajdonságneve legyen (pl. `'character1'`).
- Használd az `alert()`-et az objektummetódusban a karakterek egyikének a köszöntéséhez.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

## 5.6. Összefoglalás

Ebben a leckében elmélyítettek és kibővítettek az objektummetódusokkal kapcsolatos tudásod.

Tanultál a `this` kulcsszóról, amellyel metódusokban lehet hivatkozni arra az objektumra, amelyen belül épp használják:

```
var objectName = {
 propertyName: function () {
 console.log(this);
 }
};
```

Emellett azt is megtanultad, hogyan kell paramétert használni egy metódusban, hogy tudasd vele, melyik objektum tulajdonságot akarod elérni, amikor meghívod:

```
var objectName = {
 propertyName1: PropertyValue1,
 propertyName2: function (parameter) {
 console.log(this[parameter]);
 }
};
objectName.propertyName2(argument);
```

Szép munka!

## 6. Végigiterálás egy objektumon a for...in használatával

### 6.1. Végigiterálás a katalógusodon a for...in használatával

Ha a könyvespolcod roskadozik a könyvektől, akkor nehéz észben tartani, hogy melyikeket olvastad már és melyikeket nem. Szóval miért ne hozzunk létre egy másik metódust a katalógusban, ennek a problémának a megoldására?

A metódus hozzáadhatná a még nem olvasott könyveket egy tömbhöz, majd megjeleníthetné ezt a tömböt.

Kezdjünk is hozzá! Nyisd meg a binedet a JS Binen ([ez](#) a jelenlegi állapota az enyémnek angolul), és menj a `bookCatalog` objektumhoz a JavaScript-panelben.

Először is, hozz létre egy új metódust a `bookCatalog`-on belül, a `page marker` metódus után, az összes beágyazott objektumon kívül. Hívjuk `unread`-nek (ami olvasatlant jelent), hiszen ezek azok a könyvek, amelyeket még nem olvastál:

```
'unread': function () {}
```

Rendben. Mi a következő lépés?

A metódus hozzá fogja adni az olvasatlan könyveket egy tömbhöz. Ezért létre kell hoznunk és el kell mentenünk egy üres tömböt, amelyhez később hozzáadjuk a könyvcímeket. Ezt a tömböt `booksToRead`-nek (elolvasható könyveknek) fogjuk hívni:

```
'unread': function () {
 var booksToRead = [];
}
```

Most jön a dolog nehezebb része. A metódusnak át kell néznie az összes könyvet az objektumban, és mindegyiken ugyanazt a kódot kell végrehajtania, hozzáadva minden egyes olvasatlan könyv címét a tömbhöz.

Úgy tűnik egy ciklusra lesz szükségünk, nemde?

Akkor nézzük meg, hogyan iterálunk végig egy objektum tulajdonságain:

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {}
}
```

Ezt a sort közelebbről is nézzük meg:

```
for (var book in this) {}
```

Ugyebár `for`-ral kezdődik, de mégsem néz ki úgy, mint egy `for` ciklus, mert valami fura dolgot írtunk a zárójelbe.

Először deklaráltunk egy `book` nevű változót, de nem rendeltünk hozzá értéket. Miért nem? Mert az értéke minden egyes iterációnál változni fog, az objektum egyik tulajdonságáról a másikra.

Az `in` szót követően meg kell adnunk azt az objektumot, amelynek a tulajdonságain végig akarunk iterálni. Ebben az esetben a `bookCatalog` objektumra hivatkoztunk a `this` kulcsszóval, ahogy azt már tudod.

Ezt a típusú ciklust `for...in` utasításnak hívjuk, és az általános szintaxisa a következő:

```
for (var variableName in objectName) {
 // az objektum összes tulajdonságán végrehajtandó kód
}
```

Ahol:

- a `variableName` a változó neve, és
- az `objectName` az objektum neve.

Most már van egy `for...in` ciklusunk, amely végig tud iterálni a `bookCatalog` objektum tulajdonságain. (Így néz ki éppen az én katalógusom.) A következő lépést pedig a következő oldalon tesszük meg.

## 6.2. Változó használata tulajdonság eléréséhez

Nyisd meg a könyvkatalógusodat a JS Binen ([ez](#) az enyém angolul), és keresd meg az `unread` metódust a JavaScript-panelben:

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {}
}
```

Először létrehoz egy üres tömböt, majd jön a `for...in` ciklus, amelynek a segítségével végig tudunk iterálni a katalógus tulajdonságain. Mire van még szükségünk?

Ahhoz, hogy eldöntsük, egy könyv olvasatlan-e és ezért hozzá kell-e adni a tömbhöz, szükségünk lesz egy feltételes állításra. Most akkor írjuk meg a feltételt.

Azt kell ellenőriznünk, hogy az adott könyv `has been read` tulajdonságának a booleanértéke `false`-e.

Ehhez először el kell érnünk minden egyes könyvben ennek a tulajdonságnak az értékét. Az előző leckéből tudod, hogy ezt hogyan kell megtenni:

```
this[book]['has been read']
```

Ugyanúgy, mint amikor a `book` paraméter volt, most sem kellett aposztróffal körbevennünk a szöglletes zárójelen belül. Az ok hasonló: azt akarjuk, hogy a JavaScript tudja, hogy a `book` nem egy tulajdonságnév, hanem egy változónév. A tényleges tulajdonságnév a `book` változó aktuális értéke lesz, amely minden egyes iterációnál változni fog, pl. '`myBook1`'-ról '`myBook2`'-ra.

A feltételben ellenőriznünk kell, hogy az aktuális könyv `has been read` tulajdonságának `false` booleanértéke van-e.

```
this[book]['has been read'] === false
```

**Megjegyzés:** A `for...in` ciklus végig fog iterálni a `bookCatalog` objektum minden tulajdonságán, beleértve a metódusokat. Mivel a metódusainknak nincs `has been read` tulajdonsága, a `this[book]['has been read']` náluk `undefined` értéket fog visszaadni. Végeredményben a feltétel nem lesz igaz a metódusokra.

Ha a feltétel igaz, a könyv címét hozzá kell adni a `booksToRead` tömbhöz:

```
if (this[book]['has been read'] === false) {
 // adja hozzá a könyv címét a booksToRead tömbhöz
}
```

A `title` tulajdonságot ugyanúgy fogjuk elérni, ahogy a `has been read` tulajdonságot értük el:

```
this[book].title
```

A címnek a tömbhöz való hozzáadásához a `.push()` metódust fogjuk használni:

```
booksToRead.push(this[book].title);
```

Tehát, így fog kinézni a feltételes állítás:

```
if (this[book]['has been read'] === false) {
 booksToRead.push(this[book].title);
}
```

Illesszük ezt be a `for...in` ciklusba:

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {
 if (this[book]['has been read'] === false) {
 booksToRead.push(this[book].title);
 }
 }
}
```

Nagyszerű!

Pillanatnyilag a katalógusom [így](#) néz ki.

Már majdnem készen vagyunk, de ahhoz, hogy ténylegesen lássuk is az olvasatlan könyveket, meg is kell jeleníttetnünk a tömböt. Ezt fogjuk megtenni a következő feladatban.

### Feladat

Marcsi szeret a szülővárosában biciklizni. Itt egy objektum, amely körülírja a bringáját: `var bicycle = {'type': 'városi bringa', 'color': 'mentazöld', 'frame material': 'acél'};`. Hozz létre egy metódust az objektumon belül, amely kiírja a bicikli tulajdonságait.

A programod a következő kritériumoknak feleljen meg:

- Hozd létre a fenti objektumot.
- A metódus iteráljon végig a tulajdonságokon.
- A metódus a `typeof` operátorral ellenőrizze, hogy az aktuális tulajdonság értéke nem függvény-e.
- Ha nem az, akkor a tulajdonság értékét adja hozzá egy tömbhöz.
- Végül írja ki a tömböt a `console.log()` segítségével.

Sok sikert!

**Megoldás:** Amikor elvégezted a feladatot, itt meg tudod nézni [az általunk alkotott mintamegoldást](#). Hasonlítsd össze a kettőt, és ellenőrizd a saját verziódat. Ha minden rendben, akkor menj tovább a következő leckére.

### 6.3. Olvasatlan könyveid kiíratása

Nyisd meg a binedet a könyvkatalógusoddal, hogy lássuk az `unread` metódus jelenlegi állapotát ([itt](#) az enyém angolul).

Már hozzá tudja adni az olvasatlan könyvek címét a `booksToRead` tömbhöz, de még nem jeleníti meg azt. Essünk neki!

Erre a célra a `console.log()` metódust fogjuk használni:

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {
 if (this[book]['has been read'] === false) {
 booksToRead.push(this[book].title);
 }
 }
 console.log('Nem olvastad még el az alábbi könyveket: ' +
booksToRead);
}
```

Próbáld ki! Kattints a „Run with JS” gombra, és hívd meg a metódust a konzolban. Ha van olvasatlan könyved, akkor kilistázza azokat, és megmondja, hogy még nem olvastad el őket. Nagyon vagány!

Ha viszont nincs olvasatlan könyved, nem lesz könyvcím megjelenítve, csak ez:

"Nem olvastad még el az alábbi könyveket: "

Ez egy kicsit furának tűnik, ugye? Jobb lenne, ha minden könyvedet elolvastad. Más esetben pedig listázná az olvasatlan könyveidet, mint normálisan.

Tehát szükségünk lesz egy másik feltételes állításra. Kezdjük a feltétel megírásával!

Ha nincsen olvasatlan könyved, a `booksToRead` tömb üres marad. Tehát tudnánk használni az alábbi feltételt?

`booksToRead === []`

Nézzük csak meg! Ha ez a feltételünk, akkor a metódusunk az új feltételes állítással az alábbi módon néz ki:

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {
 if (this[book]['has been read'] === false) {
 booksToRead.push(this[book].title);
 }
 }
 if (booksToRead === []) {
 console.log('Elolvastad az összes könyvedet.');
 } else {
 console.log('Nem olvastad még el az alábbi könyveket: ' +
```

```
booksToRead) ;
 }
}
```

Próbáljuk ki (ha szükséged van egy binre a munka jelenlegi állapotával, [itt](#) az enyém). Kattints a "Run with JS" gombra a frissített JavaScript-kód futtatásához, majd hív meg az [unread](#) metódust!

Ha van olvasatlan könyved, akkor remekül fog működni, viszont ha az összes könyvedet olvastad már, akkor még minden az alábbiakat fogja kiírni:

"Nem olvastad még el az alábbi könyveket: "

Mi a probléma a `booksToRead === []` feltétellel? A következő oldalon fogjuk ezt megbeszélni.

Itt van két üres tömb: `var emptyArrayOne = [];` és `var emptyArrayTwo = [];`. Melyik booleanértéket kapod válaszul, ha az `emptyArrayOne` és az `emptyArrayTwo` kerül összehasonlításra egy egyenlőségi művelettel (`==` vagy `===`)?

#### 6.4. Az objektumok referenciatípusok

Ahogy az előző leckében láttuk, valamilyen oknál fogva az `unread` metódus nem működik megfelelően (ha szükséged van egy mintabinre, [itt](#) az enyém angolul).

Ha nincs olvasatlan könyved, azt kellene megjelenítenie, hogy "`Elolvastad az összes könyvedet.`". De ehelyett ezt írja ki:

"Nem olvastad még el az alábbi könyveket: "

Úgy látszik, hogy valami gond van a `booksToRead === []` feltétellel az alábbi feltételes állításban:

```
if (booksToRead === []) {
 console.log('Elolvastad az összes könyvedet.');//
} else {
 console.log('Nem olvastad még el az alábbi könyveket: ' +
booksToRead);
}
```

De mi a gond? Ha nincsen olvasatlan könyv, a `booksToRead` tömbnek üresnek kell lennie, igaz?

Nos, igen, valóban üres, de nem ez az, amit a `booksToRead === []` feltétel ellenőriz.

Ahogy korábban említettük, a tömbök objektumok a JavaScriptben. **A JavaScriptben az**

**objektum egy referenciatípus, nem egy értéktípus.** Mit jelent ez? Ha két objektumot összehasonlítasz egy egyenlőségi művelettel (`==` vagy `===`), akkor nem az értéküket veted össze, hanem a referenciájukat, azt kérdezve, hogy ugyanarról az objektumról van-e szó. Következésképpen, mivel egy objektum kizárolag saját magával egyenlő, így csak akkor kapsz `true` booleanértéket, ha saját magával hasonlítod össze az objektumot. Egyébként `false` lesz a válasz, még akkor is, ha a két objektum tartalma teljesen megegyezik.

Jelen esetben a `booksToRead` csak a `booksToRead`-del egyenlő. Amikor a `booksToRead`-et egy üres tömbbel hasonlítod össze, akkor valójában két különböző tömb referenciáját veted össze, így még ha a két tömb tartalma meg is egyezik, nem lesznek egyenlők.

**Megjegyzés:** Az objektumon kívüli JavaScript-adattípusokat – úgy mint stringeket, számokat, booleanértékeket, az `undefined`-ot és a `null`-t – primitív adattípusoknak hívjuk. Ezeknek az értékét adjuk át, nem a referenciáját, így ha például két olyan stringet hasonlítasz össze, amelyeknek a tartalma megegyezik, akkor egyenlőséget fogsz találni.

Rendben, de akkor hogyan tudjuk leellenőrizni, hogy a `booksToRead` tömb üres-e?

Nos, használhatjuk a tömb hosszát. Emlékeztetőül: a tömb hossza az elemek száma a tömbben. Ha egy tömb üres, akkor a hossza 0.

Szóval a feltételben le kell ellenőriznünk, hogy a `booksToRead` hossza 0-e.

```
booksToRead.length === 0
```

Javítsuk ki az `unread` metódust a fenti feltétellel (a frissített katalógusomat megtalálod [itt](#)):

```
'unread': function () {
 var booksToRead = [];
 for (var book in this) {
 if (this[book]['has been read'] === false) {
 booksToRead.push(this[book].title);
 }
 }
 if (booksToRead.length === 0) {
 console.log('Elolvastad az összes könyvedet.');
 } else {
 console.log('Nem olvastad még el az alábbi könyveket: ' +
booksToRead);
 }
}
```

Nézzük meg, hogy működik-e! Kattints a „Run with JS” gombra a frissített JavaScript-kód futtatásához, és hívda meg az `unread` metódust.

Hurrá, tökéletes! Még akkor is a megfelelő üzenetet írja ki, ha az összes könyv elolvasásra került.

Most, hogy már sokat tanultál az objektumokról, készen állsz az önálló munkára. A következő leckében az új képességeidet fogod gyakorolni. Jó munkát!

Kata exe (Jancsi) és az új barátja (Tomi) egészen hasonlítanak egymásra, ahogy azt az alábbi, őket körülíró JavaScript-objektumok is jól mutatják: `var jancsi = {'hair color': 'barna haj', 'eye color': 'barna szem', 'face shape': 'ovális arc', 'build': 'vékony', 'height': 'magas'};` és `var tomi = {'hair color': 'barna haj', 'eye color': 'barna szem', 'face shape': 'ovális arc', 'build': 'vékony', 'height': 'magas'};`. A két objektum tartalma teljesen megegyezik. Ez azt jelenti, hogy egyenlők? Vizsgáld meg egy egyenlőségi műveettel.

## 7. Újonnan szerzett tudásod gyakorlása

### 7.1. Tulajdonságot megváltoztató metódus létrehozása

Ebben a feladatban egyedül fogsz dolgozni, hogy gyakorold minden, amit az objektumokról tanultál. Egy új metódust fogsz írni a Könyvkatalógusodba ([itt van](#) az enyém angolul), az alábbi információk alapján.

Itt van a metódus leírása:

- A `bookCatalog` objektumon belül van, a beágyazott objektumokon kívül.
- A neve `change_property` (ami azt jelenti, hogy "változtasd meg a tulajdonságot"), hiszen arra fogod használni, hogy megváltoztasd egy tulajdonság értékét valamelyik könyvobjektumon belül.
- Kettő paramétere van: egy a könyvobjektum nevének és egy a megváltoztatni kívánt tulajdonság nevének.
- A tulajdonság új értékét a `.prompt()` kérdezi meg és az `.alert()` igazolja vissza.

A következő oldalon találni fogsz egy mintamegoldást, de kérlek, ne nézd meg a feladat befejezése előtt!

A JS Binben korábban elmentett bined JavaScript-paneljében hozz létre egy új metódust a `bookCatalog` objektumon belül, a fenti információk ismeretében. Ha elkészültél, másold be a bined linkjét ide.

### 7.2. A metódus példamegoldása

Itt van egy mintamegoldás a `change_property` metódusra:

```
'change property': function (book, property) { // hozz létre egy metódust két paraméterrel
```

```
 this[book][property] = prompt('Mi az új értéke a ' + property +
 ' tulajdonságnak az alábbi könyvben: ' + this[book].title +
 '?'); // kérdezd meg a tulajdonság új értékét
 alert('Az új értéke a ' + property + ' tulajdonságnak az alábbi
 könyvben: ' + this[book].title + ' a következő: ' + this[book]
 [property] + '.'); // igazold vissza a tulajdonság új értékét
}
```

És [így](#) néz ki most az én binem (angolul).

Most próbáld ki a szuper új metódusodat, annak meghívásával. Ne felejtsd el, hogy most két argumentumot kell megadnod a metódushoz: a könyvobjektum nevét és a megváltoztatandó tulajdonság nevét.

Például ahhoz, hogy megváltoztassam a *Galaxis útikalauz stopposoknak* értékelését, rákattintok a „Run with JS” gombra, beírom a következőt a konzolba, és Entert nyomok:

```
bookCatalog['change property']('myBook3', 'rating');
```

Te is próbáld ki! Használhatod a metódust a könyv értékelésének frissítésére, csakúgy, mint az olvasottsági és kölcsönzési státusz megváltoztatására.

### 7.3. Idézetet hozzáadó metódus létrehozása

Lehet, hogy könyvolvasás közben szeretnél érdekes idézeteket hozzáadni a katalógusodhoz. Ebben a feladatban egy új metódust fogsz erre a célra elkészíteni. (Ha szükséged van egy binre a katalógus jelenlegi állapotával, [itt](#) az enyém angolul.)

Ezt a metódust most saját magad fogod megírni, gyakorolva minden, amit már tudsz az objektumokról.

Kérlek, kövesd az alábbi iránymutatást:

- A metódus a `bookCatalog` objektumban található, a beágyazott objektumokon kívül.
- A neve `add quote` (ami azt jelenti, hogy "adj hozzá idézetet"), mert arra fogod használni, hogy idézetet adj hozzá egy könyvhöz.
- Két paramétere van: egy a könyvobjektum nevének, egy pedig az oldalszámnak, ahol az idézet található.
- Az idézetet a `.prompt()` kérdezi meg és az `.alert()` igazolja vissza.

Ha készen vagy, nézd meg a mintamegoldást a következő oldalon!

A JS Binen korábban elmentett bined JavaScript-paneljében hozz létre egy új metódust a `bookCatalog` objektumon belül, a fenti információk ismeretében.

## 7.4. A metódus példamegoldása

Ez egy mintamegoldás az `add quote` metódushoz:

```
'add quote': function (book, page) { // hozz létre egy metódust
két paraméterrel
 this[book].quotes[page] = prompt('Mi az idézet, amit hozzá
szeretnél adni az idézetekhez az alábbi könyvben: ' +
this[book].title + '?'); // kérdezd meg az idézetet
 alert('Az alábbi idézetet hozzáadtad az idézetekhez a könyvben,
melynek címe ' + this[book].title + ': ' +
this[book].quotes[page]); // igazold vissza az idézetet
}
```

Ha szeretnéd ellenőrizni a metódust az én könyvkatalógusomban, akkor [itt](#) megtaláld azt (angolul).

**Megjegyzés:** Jelenleg az a helyzet, hogy ha egy újabb idézetet rendelsz hozzá egy olyan oldalhoz, amely már tárol egy idézetet ugyanabban a könyvben, akkor az új idézet le fogja cserélni a régit. Ezt úgy tudod elkerülni, hogy az ugyanazon az oldalon szereplő összes idézetet egyetlen tömbbe mented el, majd a `.push()` metódussal adsz hozzá új idézeteket. Ha gondolod, változtasd meg ennek alapján a kódod.

## 7.5. A kölcsönvett könyveket listázó metódus létrehozása

Ha van egy rés a könyvek között a polcodon, mert kölcsönadtál egyet, akkor nem minden egyszerű visszaemlékezni, melyik könyv is hiányzik.

Ennek a problémának a megoldásához egy metódust fogsz létrehozni a katalógusban ([itt](#) az enyém jelenlegi állapota angolul), amely kilistázza a könyveket, amelyek épp kölcsön vannak adva.

Ezt önállóan fogod megtenni az alábbi lépésekkel követve:

- A metódus a `bookCatalog` objektumban található, a beágyazott objektumokon kívül.
- A neve `borrowed` (ami azt jelenti, hogy "kölcsönvett"), mert a kölcsönadott könyvek listázására fogod használni.
- A `for...in` utasítást használja a `bookCatalog` tulajdonságain való végigiteráláshoz.
- A `typeof` művelettel ellenőrzi, hogy az aktuális tulajdonság értéke nem függvénye.
- Ha a könyvet valaki kölcsönvette, a címe hozzá lesz adva egy tömbhöz.
- A tömböt kiírja a `console.log()`-gal.

Ennek a metódusnak a létrehozása kicsit nehezebb lesz, mint a többié, de ne aggódj, minden hozzá szükséges tudás birtokában vagy.

Ha befejezted, összehasonlíthatod a megoldásodat a következő oldalon levővel.  
Ugyanakkor kérlek, ne nézd meg azt előbb, hiszen a cél az, hogy gyakorolj!

A JS Binen korábban elmentett bined JavaScript-paneljében hozz létre egy új metódust a `bookCatalog` objektumon belül, a fenti információk ismeretében. Ha elkészültél, másold be a bined linkjét ide.

## 7.6. A metódus példamegoldása

Ez egy lehetséges megoldás a `borrowed` metódushoz:

```
'borrowed': function () { // hozz létre egy metódust
 var borrowedBooks = []; // hozz létre egy üres tömböt
 for (var book in this) { // iterálj végig a bookCatalog
 tulajdonságain
 if (typeof this[book] !== 'function') { // ha a bookCatalog
 aktuális tulajdonsága nem egy függvény
 if (this[book]['borrowed by'] !== null) { // ha a borrowed by
 tulajdonság értéke nem null
 borrowedBooks.push(this[book].title); // add hozzá a könyv
 címét a tömbhöz
 }
 }
 }
 if (borrowedBooks.length === 0) { // ha a tömb üres
 console.log('Egyik könyved sincs kölcsönadva épp.');// jeleníts meg egy üzenetet
 } else { // ha a tömb nem üres
 console.log('Az alábbi könyvek vannak épp kölcsönadva: ' +
 borrowedBooks); // jeleníts meg egy másik üzenetet
 }
 }
}
```

Az én könyvkatalógusom [így](#) néz ki a `borrowed` metódussal (angolul van).

Egy kis magyarázat a `typeof this[book] !== 'function'` feltételhez: ezzel azt ellenőrizzük, hogy a `bookCatalog` aktuális tulajdonsága nem egy függvény-e. Hogy miért?

Nos, a `for...in` ciklus a `bookCatalog` összes tulajdonságán végigiterál, beleértve a metódusokat is.

Mivel a metódusoknak nincs `borrowed by` tulajdonsága, a `this[book]['borrowed by']` mindegyikükönél `undefined` értéket ad vissza, mert – ahogy azt megtanultuk – amikor olyan tulajdonságot próbálsz elérni, amely nem létezik, a válasz `undefined`.

Ami azt jelenti, hogy a `this[book]['borrowed by'] !== null` feltétel igaz lenne minden metódusra. Következésképp minden egyes metódusnak egy nem létező `title` tulajdonsága hozzáadódna a `borrowedBooks` tömbhöz.

Ezért van szükségünk a `typeof this[book] !== 'function'` feltételre: ez kiszűri az összes metódust.

Természetesen lehetnek más megoldások is erre a problémára, így előfordulhat, hogy a te metódusod különbözik, és mégis tökéletesen működik.

## 8. Összefoglalás

### 8.1. Összefoglalás

Gratulálok, nagyon sokat tanultál a JavaScript-objektumokról! Most már tudod, hogy hogyan kell:

- létrehozni egy objektumot és elmenteni azt egy változóba:

```
var objectName = {
 propertyName1: PropertyValue1,
 propertyName2: PropertyValue2,
 propertyName3: PropertyValue3
};
```

- elérni egy objektumtulajdonságot a pontjelöléssel:  
`objectName.propertyName;`
- elérni egy objektumtulajdonságot a zárójeljelöléssel:  
`objectName['property name'];`
- elérni egy objektumtulajdonságot egy beágyazott objektumban:  
`objectName.nestedObjectName.propertyName;`  
`objectName['nestedObjectName']['propertyName'];`  
`objectName.nestedObjectName['propertyName'];`  
`objectName['nestedObjectName'].propertyName;`
- megváltoztatni egy objektumtulajdonság értékét:  
`objectName.propertyName = newValue;`  
`objectName['propertyName'] = newValue;`
- hozzáadni egy új objektumtulajdonságot:  
`objectName.propertyName = value;`  
`objectName['propertyName'] = value;`
- ellenőrizni egy objektumot adott tulajdonságra:  
`'propertyName' in objectName;`  
`objectName.hasOwnProperty('propertyName');`
- megírni egy objektummetódust:

```
var objectName = {
 propertyName: function () {
 // végrehajtandó kód
 }
};
```

- meghívni egy objektummetódust:  
`objectName.propertyName();`  
`objectName[ 'propertyName' ]();`
- használni a this kulcsszót:

```
var objectName = {
 propertyName: function () {
 console.log(this);
 }
};
```

- paramétereket használni egy objektummetódusban:

```
var objectName = {
 propertyName1: PropertyValue1,
 propertyName2: function (parameter) {
 console.log(this[parameter]);
 }
};
objectName.propertyName2(argument);
```

- végigiterálni egy objektum tulajdonságain a `for...in` használatával:

```
for (var variableName in objectName) {
 // az objektum összes tulajdonságán végrehajtandó kód
}
```

Wow, ez egy elég hosszú lista az új képességeidről. Bravó!

## 9. Teszt

### 9.1. Ellenőrizd a tudásod!

[15 kérdés, körülbelül 15 perc, kérdésenként 1 pont.](#)

**Tipp:** Hogy a lehető legtöbbet tanulj ebből a tesztből, azt javasoljuk, hogy egyedül, segédanyagok használata nélkül töltsd ki.

Ha végeztél, nyomd meg a „Küldés” (Submit) gombot a teszt alján, hogy megkapd az eredményed. Itt látni fogod a helyes válaszokat is.

**Megjegyzés:** A tesztet többször is kitöltheted.

Sok sikert!

*Gratulálunk!*

*A tananyag végére értél, igazán büszke lehetsz magadra!  
További sok sikert kívánunk neked.*

*CodeBerry*



