Tartalom

Változó fogalma

A **változó,** egy mennyiség(adat) vagy egy objektum (több adat) szimbolikus jelölése, mint azt már a matematikában is használtuk. Programozásnál legtöbbször az érték tárolása a feladata, hogy azok a későbbiekben bármikor elő „vehetők” majd felhasználhatók lehessenek különböző feladatokra. Értéket többféle módon is kaphatnak, vagy még a kódban megadjuk nekik, vagy billentyűzetről olvassuk be, esetleg egyéb input felületet generálunk ki neki, random létrehozzuk, vagy csak simán egy fájlból/adatbázisból beolvassuk.

Változók típusai

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

Number:

Ebbe a típusba tartozik minden szám, a törteket is beleértve. A számok mindig idézőjel nélkül szerepelnek a kódban: 3, 6, 2019, 43.25. (valós vagy egész számokat ezen belül kezeljük)

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 aposztrófffal (') vagy idézőjellel (") van körülvéve. Pl. 'Ez egy szöveg.', "Ez is 1 szöveg.", '3s p3rsz3 3z 1s 4z!!!'

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. Ez úgy is nevezzük, hogy logikai változó típus. Valaminek az eldöntésére használjuk.

Null:

Ez egy úgynevezett „empty value” (más néven üres érték), amely a konkrét érték hiányát jelzi.

Undefined:

A nullhoz nagyon hasonló empty value. Működésük szituációtól függ.

Symbol:

A szimbólumok egyedi azonosítók, amelyek összetett kódolás során tudnak hasznosak lenni.

Object:

Az objektum több egymáshoz kapcsolódó adat gyűjteménye. Egy későbbi tananyagban bővebben is megismerkedsz majd velük.

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

Deklarálás és értékadás

Deklarálás

azt a folyamatot nevezzük ennek, amikor egy változót létrehozunk, de még konkrét célra nem használjuk, csak tudjuk, hogy szükségünk lesz egy ilyen névre majd a munkánk során, ilyenkor az előbb felsorolt típusok közül bármelyiket használhatjuk. Ekkor az adott változó ugye undefinied típusú lesz.

Értékadás

folyamata az, amikor a már létező változót felruházzuk valamilyen értékkel, ilyenkor a JavaScript mivel gyengén típusos, maga dönti el, milyen típusban fogja azt tárolni.

Viszont ügyelnünk kell arra, hogyha **const** (erről picit lentebb) tulajdonsággal **deklarálunk** egy üres változót később annak értékét nem lehet változtatni tehát fordítási **hibát fog okozni** a kódban, ha értéket adnánk neki.

Hatáskör és élettartam

Változó hatásköre:

A deklaráció helye befolyásolja az azonosító (változónév) **hatáskörét**. A hatáskör a **forráskódhoz köthető** fogalom, azt adja meg, hogy a név a forráskód mely soraiban használható fel (mely soraiban **látható**).

Változó élettartama:

A függvényekben definiált változók élettartama a változó definiálásától a függvény futásáig tart. Amennyiben nem függvényeket használunk, így azok JavaScipt esetén a típusuktól függenek melyek 3 értéket vehetnek fel Let, var és const.

- Statikus: Az egész programkódra kiterjed
- Dinamikus: A programkód csak egy adott részére terjed ki

Let, var, const, vagy „semmi”

A változóknak javascriptben 3 típusát megkülönböztetjük, de ez nem a bennük tárolandó értékek típusát határozza meg hanem a változók funkcióit.

let:

Élettartama dinamikus, újra deklarálást nem engedi, de az értékét bármikor módosíthatjuk.
Újra deklarálása (SyntaxError) ***[2015-ös verzió óta létezik]***

const:

Élettartama statikus, újra deklarálás nem engedi, az értékét a létrehozása után NEM módosíthatjuk.
Újra deklarálása (SyntaxError) új értékadás esetén (TypeError) ***[2015-ös verzió óta létezik]***

var:

Hatásköre statikus, újra deklarálni lehet, és az értékét is bármikor módosíthatjuk. *Nem okoz fordításkor hibákat, ellenben sok futtatáskori hibának okozója lehet!* ***[Régebbi böngészőkben való JavaScript futtatáshoz ezt KELL használnunk, a JavaScipt megjelenése óta létezik]***

Használata, ha nem ismerjük eléggé veszélyes lehet. Pl.: újra deklarálhatunk egy korábban megadott azonos nevű változót mert már nem emlékszünk rá, hogy létrehoztuk. Használata mindenképp tervezést igényel, bár inkább ne használjuk.

nem írunk semmit: ilyenkor a típus megadásának műveletét elvégzi a böngésző

TypeOf

Lehetőségünk van használni a typeof() – parancsot, ahol a zárójelek közé a megfelelő változó nevét, vagy éppen egy imput értéket megadva, lekérdezhetjük, az milyen típusú.

próbáljuk ki az alábbi kódot, let var és const típusokkal, értékadással és anélkül

```
let a;
a=5;
document.write(typeof(a));
```

Ebben az esetben érdekes dolgok történnek… vagy épp nem történnek

```
const a;
a=5;
document.write(typeof(a));
```

Változók típusának módosítása (Convertálás/Castolás)

Előfordulhat a JavaScriptben, mivel ugye tanultuk, hogy gyengén típusos nyelv, hogy nem olyan típusra állította be a rendszer a változónkat, amire mi azt szerettük volna, az értékadáskor **(Mivel a JavaScript a változó típusának beállítását automatikusan értékadáskor elvégzi)**, vagy éppen csak **módosítani szeretné**k egy korábban szövegként „beolvasott" értéket számmá. Ilyenkor használjuk a különböző konvertálási folyamatokat.

Melyek primitív típusokká konvertálják az aktuális változót: ***számmá, szöveggé vagy logikai változóvá***

Konvertálás számmá – Number() , parseFloat(), parseInt()

number() – Szinte mindent számmá alakíthatunk vele 😊 (logikai érték, egész/valós szám, dátum)

```
Number("2")   // az értéke 2 lesz
Number(true)   // az értéke 1 lesz (False esetén 0)
Number("3.14")   // az értéke 3.14 lesz (De nem pontosan PI!!! Az a Math.PI)
Number(" ")      // az értéke 0 lesz
```

parseFloat() – esetén lebegőpontos számmá alakíthatjuk azt

parseInt() – esetén pedig egész számmá

```
parseInt("2")   // az értéke 2 lesz
parseFloat("3.14")   // az értéke 3.14 lesz (De nem pontosan PI!!!)
```

Lehetőségünk van dátumot is számmá konvertálni ilyenkor vagy a Number() függvényt használjuk a korábban látott módon, csak a változó esetén a dátum típusú elemet kapjuk meg, vagy a getTime() függvénnyel az alábi módon

```
valtozoNev.getTime();
```

Ilyenkor mindkét esetben egy elég hosszú 13 jegyű számot kapunk

Ügyeljünk arra, hogyha valós számot tartalmazó stringet tartalmazó számot konvertálunk egész számmá, csak az egész része marad meg!

Konvertálás szöveggé – String(), toString()

Amennyiben úgy döntünk a változónkat mégis szeretnénk szövegként tárolni erre is van lehetőség, a az alábbi formákban.

```
String("2")   // az értéke 2 lesz
String(true)   // az értéke „true" lesz (False esetén „false")
String(3.14)   // az értéke „3.14"
String(0)      // az értéke 0 lesz
String(Date())      // az aktuális dátum minden adatát megjeleníti
```

A toString() függvény használata esetén meg kell adni a változót és pont karakterrel hozzáfűzni a függvényt, hasonlóan mint a getTime() estén.

```
valtozoNev.toString();
```

Dátum adat kinyerő függvények

Előfordulhat, hogy az aktuális dátumot szeretnénk lekérdezni, de abból csak egy darabkát szeretnénk felhasználni, ilyenkor használjuk a dátum kezelő függvényeket.

getDate() – Az aktuális nap számával tér vissza(1-31)

getDay() – Az aktuális nap sorszámával tér vissza(0-6)

getFullYear() – Az akutális év számával tér vissza(4 számjegy)

getHours() – Az aktuális óra számával tér vissza(0-23)

getMilliseconds() – A dátum változó lekérdezésének aktuális millisecundum idejét adja vissza (0-999)

getMinutes() – Az akutális dátum perc értékével tér vissza (0-59)

getMonth() – Az aktuális dátum hónap értékével tér vissza (0-11)

getSecond() – Az aktuális dátum másodperc értékével tér vissza (0-59)

getTime() – Az 1970 január 1. óta eltelt milliszekundumok számával tér vissza

További konvertálással kapcsolatos anyagokat találsz itt: **w3schools**

Névadási szabályok

A JavaScript nyelvben az alábbi dolgokra kell ügyelnünk:

- A **változó nevében lehet betű akár kicsi akár nagy**, ugyanakkor bár lehet **ékezetes** betűt használni azokat mégis próbáljuk meg kerülni.
- A változó nevében **NEM lehet space** karakter.
- A változó **NEM kezdődhet számmal**!
- A változó **kezdődhet „_”***(alulvonás)* és **„$”***(dollár jel)* karakterekkel bár ez utóbbi ritkább.
- A betűk mellett a „_” és „$” karakterek is bárhol lehetnek a változó nevében.
- Törekedjünk arra, hogy a **név utaljon a** benne eltárolt **változó feladatára**.
- A nyelv **case-sensivitve**, ami azt jelenti, hogy a kis és nagybetűk között különbséget tesz! tehát „alma” nem egyenlő az „Alma” változó névvel
- A legtöbbször a **camelCase** változó neveket használjuk, a változót több szóra bontjuk, az első szó kisbetűvel a többi szó nagybetűvel kezdődik és az összes szót összefűzzük.
  pl.: camelCaseValtozonev, ezIsEgyValtozo, ittEgyFontosValtozo stb…

Esetleg használhatjuk az alulvonás karaktert is mint a szavakat elválaszó elemet, de ez is ritkább:
pl.: camel_case_valtozonev, ez_is_egy_valtozo, itt_egy_fontos_valtozo stb…

Érdekesség: Egyes programozók a const-t típusú változók neveit csupa nagybetűvel írják, a könnyebb olvashatóság érdekében, de ez egyéni megszokás kérdése.

Változó és string összefűzése (Megjelenítésük)

A változók és szöveg egyszerre való kiíratásának többféle módja is lehet, ezeket vesszük most sorra, és hogy miért van rá szükségünk, talán azért, mert előfordulhat velük, hogy szeretnénk megmagyarázni mi az az érték, amit kiíratunk. „Mit is jelenítünk meg”

Összefűzéses módszer

Az összefűzéses módszer igen egyszerű, leírjuk a szöveget, majd egy karakter segítségével, hozzáfűzzük a változót, aztán megint jöhet némi szöveg és még változó a végtelenségig.

Annyit kell megjegyezni, hogy a szöveges részt idézőjelek között "szöveg" szerepeltessük, a változókat, pedig csak simán megadjuk a nevüket a kiíratás sorában, az összefűző karakter pedig a „+” jel, mint az a példában is látható.

```
document.write("Az a változó értéke:"+a));
```

Ez a módszer a legtöbb nyelvben szerepel, csak az összefőzés karaktere más pl.: php-ban a (pont **„.”**).

Hivatkozásos módszer

Hivatkozások esetén több feltételnek is meg kell felelni, az első és legfontosabb, a megjelenített változókat kapcsoszárójelek közé kell tenni **„{}”** valamint eléjük **„$”** jelet kell helyezni, ezzel hivatkozunk arra a rendszerben, hogy mely változó az, amit itt meg szeretnénk jeleníteni. Valamint fontos a szöveget tartalmazó jel is melyet az **[alt gr] + 7 („backtick”)** kombináció egyszerre történő lenyomásakor érünk el.

```
document.write(`A változó értéke: ${a}`);
```

Változó megadásának módjai

Billentyűzetről – a prompt() parancs

A prompt() segítségével az egyik legegyszerűbb megadni a változónkat, amikor a kódunkban kiadjuk és a program elér a parancshoz, egy felugró ablakot fogunk látni, mely kér egy bemeneti inputot, ha a parancs előtt deklaráltunk egy változót, akkor azt az értéket a későbbiekben használni tudjuk.

```
let a=prompt();

document.write(a);
```

Változók kiolvasása űrlapról – InnerHTML value

Lehetőségünk van nem csak felugró ablakban bekérni adatot, hanem akár a weboldalon lévő input mezők adatait is feldolgozni, ilyenkor nem kell mást tennünk, mint annak a mezőnek, id értéket adni amivel szeretnénk dolgozni. Pl id=”adatMezo” és utána máris kiolvashatjuk annak értékét ha az alábbi módszerrel hivatkozunk rá.

```
let kiolvasottAdat=document.getElementById('adatMezo').value;
```

Random szám generálása

Random számot is generálhatunk a Math.random függvénnyel, ilyenkor egy 0 és 1 közötti valós szám generálódik ki, ha nagyobb értéket szeretnénk akkor szükségünk van a generál érték szorzására a generálás után. Illetve ha egész számot szerenénk akkor használjuk a Math.round függvényt is a következő példákban látható módon!

```
let randomSzam = Math.random();
```

Generál egy 0 és egy közötti értéket (valós szám)

```
let randomSzam = Math.round(Math.random());
```

Generál egy 0 vagy egy értéket (egész szám)

```
let randomSzam = Math.round(Math.random()*100;
```

Generál egy 0-100 közötti értéket (egész szám)

```
let randomSzam = Math.round(Math.random() * (felsoHatar-alsoHatar))+ alsoHatar);
```

Generál egy általunk megadott határok közötti értéket