JavaScript és HTML ismeretek

JavaScript nyelv használata és a HTML nyelv ismétlése, újragondolt halhasználása.

Korcsák Gergely

Kövessi Erzsébet Baptista Technikum, Szakképző Iskola, Gimnázium, Szakiskola és Általános Iskola

2023. Február 21.

Tartalomjegyzék I

Tartalomjegyzék II

Tartalomjegyzék III

Bevezetés I

Mire jó a JavaScript?

A JavaScriptet jelenleg sok oldalon használjuk, beleértve kliens és szerveroldalon is. Például kliens oldalon fogunk tudni vele HTML oldalakat módosítani, hogy egy gombnyomásra, vagy eltelt idő után történjen valami érdekes.

JavaScript kód

```
1 document.getElementById("id").innerHTML =
  "Első JavaScript kódom.";
```

Eredmény

Első JavaScript kódom.

Bevezetés II

Története:

Eleinte a CSS (stílusok) támogatásáért jelent meg, azonban később sokaknak megtetszett. Egyszerű volt használni és nem volt alternatíva. *Újabban lehet hallani Pyscriptről is. (Python alapú webprogramozás)*

90-es években jelent meg.

95-ben átvette a Microsoft. (Ms Internet Explorer)

96-ban CSS támogatása/HTML kiegészítése.

97-99 között lett szabványosítva. (tehát elfogadott)

Felemelkedése:

A 2000-es évek elejére szinte 95%-os részesedése volt az Internet Explorernek, és ebben sztenderdé vált. Legelsők között talán a Google Mails használta, hogy ne kelljen újratölteni a levélláda frissítéséhez és ez sokaknak tetszett.

Bevezetés III

Mire használjuk?

Jelenleg nehéz olyan területet találni amiben nem kapott szerepet. Legelterjedtebb még mindig a weben, de számos szerepet kap még kisebb szerverek asztali alkalmazások és akár a robotika világában.

- Böngésző
- Webszerver
- NodeJS
- Asztali alkalmazások
- Telefonos alkalmazások
- Robotika

Szerkesztése

Milyen programot használjunk

A legtöbb HTML szerkesztő támogatja, de érdemesebb valami felhasználóbarát programot választani, mint pl. VSCode.

Hova kerülhet a kód?

A JavaScript kódot tehetjük <header>-be, <body>-ba, vagy külön fájlba.

```
1 <script>
2         JavaScript kód;
3 </script>
```

```
1 <script src="ELÉRÉSI.ÚTVONAL"></script>
```

Adatszerkezetek

Változók:

Adattípus	Tulajdonság	Mikor használjuk?
var	Globális változó	Globális értékek tárolására használjuk, amiket még el szeretnénk érni.
(let)	Blokon belüi változó	Csak abban a környezetben/részben létezik ahol létre lett hozva.
const	Kontans értékek	Értéke nem változtatható.

Típusok - Típustalanság? - Egyszerű típusok

A JavaScript nyelv gyengén típusos, ezért nem kell figyelnünk arra, hogy milyen változóba milyen értéket tárolunk le.
Megfordulhat a fejünkben, hogy jó e ez, és nem e lesz lassú, azonban rengeteg esetben nincs szükség valós idejű és ennél gyorsabb feldolgozásra.

Változók

Leírás

Változókat egy-egy szám, szöveg, vagy másképpen definiált adat tárolására használunk.

Példa:

```
1  var szam1 = 1
2  var szam2 = 2
3  var str1 = 'Hello '
4  var str2 = 'World!'
5
6  var osszeg = szam1 + szam2
7  var szoveg = str1 + str2
8
9  console.log(osszeg); console.log(szoveg)
10
```

Kimenet:

```
3
Hello World!
```

Tömbök

Leírás

Általában a tömböket több egyforma adat tárolására használunk. *Pl.: több szöveg, számok, vagy akár több objektum tárolása.*
JavaScript-ben a tömbök nem igénylik az egyforma adatokat, hanem listaként működnek, avagy bármit beletehetünk.

Példa:

```
1 var ures_tomb = []
2 var szamok = [1, 2, 3]
3 var tomb1 = [null, 'Hello', " world!"]
4 var tomb2 = [ures_tomb, szamok, tomb1]
5
6 console.log(tomb2)
7
```

Kimenet:

```
[ [], [ 1, 2, 3 ], [ null, 'Hello', ' world!' ] ]
```

Logikai változók

Leírás

Logikai változók alatt az olyan változókat értjük, amely csak igaz és hamis értéket tud felvenni.
Legelső esetben ezek megkülönböztetése annyi volt, hogy a változó 0, vagy bármi más értéket vett e fel.

Példa:

```
1 var logikai_feltetel1 = false
2 var logikai_feltetel2 = true
3
4 console.log(logikai_feltetel1, logikai_feltetel2)
5
```

Kimenet:

```
false true
```

Logikai műveletek I

Leírás

Sokszor fordul elő, hogy egy feledat elvégzéséhet 2, vagy akár több feltételt is össze kell hasonlítanunk, hogy eldöntsük melyik irányba kell tovább haladnunk.

Műveletek

ÉS: `bal_feltetel && jobb_feltetel` (AltGr + C)
VAGY: `bal_feltetel || jobb_feltetel` (AltGr + W)

Igazságtábla

ÉS

	Igaz	**Hamis**
Igaz	Igaz	Hamis
Hamis	Hamis	Hamis

VAGY

	Igaz	**Hamis**
Igaz	Igaz	Igaz
Hamis	Igaz	Hamis

Logikai műveletek II

Példa 1:

Legyen egy állításunk ami: *"Ha az idő napos és tiszta, akkor hétvége van."*

```
var az_ido_napos  = true
var az_ido_tiszta = true

var hetvege_van = az_ido_napos && az_ido_tiszta

console.log(hetvege_van)    //  => Kimenet: Igaz
```

Példa 2:

Legyen egy állításunk ami: *"Ha az idő felhős vagy éjszaka van, akkor sötét van."*

```
var az_ido_felhos = true
var ejszaka_van = false

var sotet_van = az_ido_felhos && ejszaka_van

console.log(sotet_van)  //  => Kimenet: Igaz
```

Elágazások I

Leírás

Az elágazások olyan programnyelvi elemek, amelyek egy feltétel vezérelt végrehajtást tesznek lehetővé.
Ez annyit tesz, hogy egy logikai változó, vagy feltétel alapján eldöntik, hogy az igaz vagy a hamis ágban folytassák a végrehajtást.

```
1 if (/* feltétel */) {    // Ha
2     /* Igaz ág */        // akkor
3 }
4 else {
5     /* Hamis ág */       // Különben
6 }
7
```

Elágazások II

Példa: Az logikai műveletek 1. példára visszatekintve tekintsük a következő megadást:

```
var az_ido_napos  = true
var az_ido_tiszta = false

if (az_ido_napos && az_ido_tiszta) {
    console.log('Hétvége van.')
}
else {
    console.log('Hétköznap van.')
}

```

Kimenet:

```
Hétköznap van.
```

Else if

Leírás

Ehhez a részhez érdemes tudni, hogy az **if** után nem feltétlenül szükséges **{}**-et rakni, azonban ekkor, csak az **if** utáni 1. utasítás fog lefutni.
Ugyanez igaz a hamis **else** ágra is.
Tehát az **else if** csak egy hamis ágba helyezett **if**, ami csak akkor fog lefutni, ha az első feltétele, hamis volt.

Példa:

```
1  if (/* hamis feltetel */) {
2      /* ... */
3  }
4  else if (/* feltetel */) {
5      /* ... */
6  }
7  else {
8      /* ... */
9  }
10
```

If vagy switch? I

Leírás

A switch egy olyan logikai választás, amely egy változó értékétől függően nem igaz és hamis ágra bontja az elágazást, hanem sok külön ágra, amelyeknek egyike, vagy akár több sorban fog lefutni.

```
1  var het_napja = 1
2  console.log('A het napja:', het_napja)
3  switch (het_napja) {
4      case 1: // Igaz lesz
5          console.log('Hétfõ van')
6          break   // Minden feltétel az elsõ breakig fog futni
7      case 2: console.log('Kedd van'); break
8      case 3: console.log('Szerda van')
9      case 4: console.log('Csütörtök van')
10     case 5: console.log('Péntek van')   /* ... */
11     default: console.log('Nincs ilyen nap!'); break
12 }
```

If vagy switch? II

Kimenetek:

```
A het napja: 1
Hétfő van

A het napja: 2
Kedd van

A het napja: 4
Csütörtök van
Péntek van
Nincs ilyen nap!

A het napja: 14
Nincs ilyen nap!
```

Ciklusok I

Leírás

Ciklusokat esetünkben tömbökön, vagy gyűjteményen való végigiterálásra használjuk. Ez lényegében ugyanazon utasítás végrehajtását jelenti egy adott feltétel eléréséig.

3 ciklust különböztetünk, meg:

```
1  while(/* logikai feltétel */) {   // elõl tesztelõ (while)
2      /* ciklusmag */
3  }
4
5  do {          // hátul tesztelõ (do-while)
6      /* ciklusmag */
7  } while(/* logikai feltétel */)
8
9  for (let i = 0; i < max; i++) { // számláló (for)
10     /* ciklusmag */
11 }
```

Ciklusok II

Oldjuk meg a következő problémát mindhárom ciklus használatával.

Probléma

Tültsünk fel egy tömb elemeit az adott elemszám négyzetével, a 10. elemig, úgy, hogy a 10. elem négyzete még szerepeljen a tömbben.

Megoldás elől tesztelő ciklussal:

```
1  var tomb = []
2  var i = 0
3
4  while (i <= 10) {
5      tomb.push(i * i)
6      i++
7  }
8
9  console.log(tomb)
10
```

Ciklusok III

Megoldás hátul tesztelő ciklussal:

Megjegyzés

Az elől tesztelő ciklussal ellentétben ebben az esetben az különbség, hogy a ciklusmag egyszer mindenképpen lefut a feltétel kiértékelése előtt.

```
1  var tomb = []
2  var i = 0
3
4  do {
5      tomb.push(i * i)
6      i++
7  } while (i <= 10)
8
9  console.log(tomb)
10
```

Ciklusok IV

Megoldás számláló ciklussal:

Megjegyzés

Míg az előző két ciklus mind egy-egy feltételt vár, hogy teljesüljön, a számláló ciklus mindig egy változó növelését fogja csak alkalmazni. *Tömbökön, vagy számsorokon való végigiterálás olyan gyakori, hogy saját ciklust kapott, hogy ne felejtsük le a növelendő változót.*

```
1 var tomb = []
2 var i = 0
3
4 for (let i = 0; i <= 10; i++) {
5     tomb.push(i * i)
6 }
7
8 console.log(tomb)
9
```

Ciklusok V

Mindhárom ciklus kimenete:

```
[
    0,  1,  4,  9, 16,
   25, 36, 49, 64, 81,
  100
]
```

Böngésző műveletek

Alert
document
getElementById
getElementsByClassName stb.
innerHTML, innerText

Fejlettebb műveletek?

Függvény, eljárás, metódus
Események
Objektumok

Egyéb felhasználás

JSON használata
Matematikai függvények
Kliens oldali ellenőrzés
Asztali felhasználás és JavaScript Keretrendszerek