

JavaScript alapok

2021-2022

A JavaScript főbb tulajdonságai

- **Script nyelv** – azaz futás közben értelmezett, **interpretált nyelv**, a kódot a böngészőbe épített JavaScript *értelmező* (interpreter) értelmezi sorról sorra
 - nincs főprogram
 - a programban lévő hiba akkor derül ki, amikor az értelmező ráfut a hibát tartalmazó sorra → a *script* blokk futása leáll és a következő HTML kód rész töltődik be
 - A HTML-en belül `<script type="text/javascript"> ... </script>`
- Legfőbb alkalmazási területe: a HTML dokumentumok **dinamikussá, interaktívvá tétele**
- **Gyengén típusos nyelv**: egy változó típusa a benne tárolt értéktől függ azaz a típusok az értékekhez tartoznak, nem a változókhoz → rengeteg automatikus típuskonverzióval jár aminek vannak előnyei és hátrányai is.
- **Szintaxisa a C –szerű**

JavaScript és HTML

- Négyféleképpen adhatunk JavaScript kódot egy HTML dokumentumhoz:
 - Külső fájlban , amelyet illesztünk a HTML fejrészben

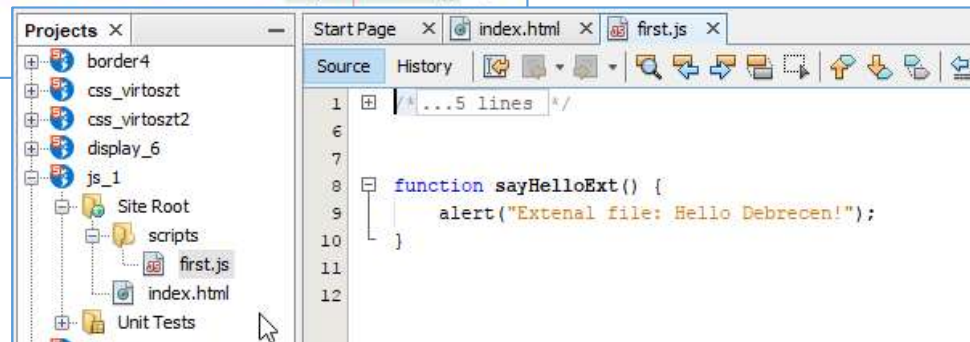
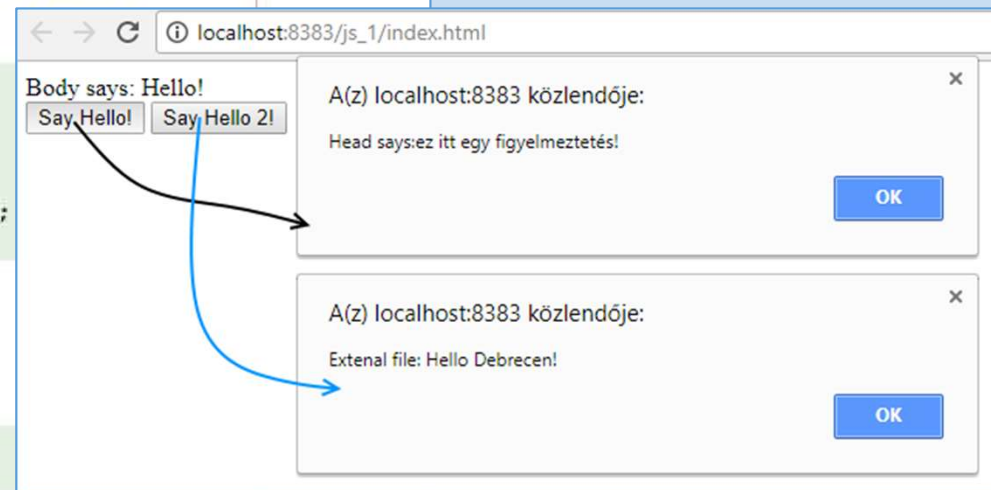
```
<!--külső javascript -->  
<script src="scripts/first.js" type="text/javascript"></script>
```

- A HTML fejrészébe írt scriptként
- A HTML törzsrészébe írt scriptként
- A HTML Fej és törzsrészébe is írt scriptként

```
<!--javascript a HEAD-ben és vagy BODY-ban -->  
<script type="text/javascript">  
  <!--  
  /*  
  |  
  | * Ez egy többsoros megjegyzés  
  | */  
  |  
  | function sayHello() {  
  |     alert("Head says:ez itt egy figyelmeztetés!");  
  | }  
  |  
  |//-->  
</script>
```

JavaScript és HTML

```
<!DOCTYPE html>
...5 lines
<html>
  <head>
    <title>javascript_megjegyzések_beágyazás </title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!--külső javascript -->
    <script src="scripts/first.js" type="text/javascript"></script>
    <!--javascript a HEAD-ben -->
    <script type="text/javascript">
      /*
       * Ez egy többsorosos megjegyzés
       */
      function sayHello() {
        alert("Head says:ez itt egy figyelmeztetés!");
      }
    </script>
  </head>
  <body>
    <!--javascript a BODY-ban -->
    <script type="text/javascript">
      //ez egy egysoros js megjegyzés
      document.write("Head says: Hello!");
    </script>
    <br>
    <input type="button" value="Say Hello!" name="btnHello" onclick="sayHello()" />
    <input type="button" value="Say Hello 2!" name="btnHello" onclick="sayHelloExt()" />
  </body>
</html>
```



Szintaxis, Megjegyzések:

- Az **unicode** karakterkészletet használja
- Az utasításokat pontosvessző (;) vagy újsor jel határolja.
- A { és } jelek közé zárva **utasítás-blokk**okat hozhatunk létre.
- Megjegyzések:

// egysoros megjegyzés

/* többsoros

megjegyzés */


- A JavaScript a kis- és nagybetűérzékeny, a betűk fogalmkörébe beletartoznak az "A" - "Z" (nagybetűs) és "a" - "z" (kisbetűs) karakterek.

Azonosítók, literálok:

- Az **azonosítók** betűvel, vagy aláhúzás jellel ("_") kezdődnek, tartalmazhatnak számjegyeket (0-9) is.
- **Literálok:**

- **Numerikus literálok**

- Az egész számokat decimális, hexadecimális és oktális formában lehet kifejezni. Ha egy szám első számjegye 0 az azt jelenti, hogy a szám oktális, ha az első számjegye 0x (0X) az azt jelenti, hogy hexadecimális.
- Alebegőpontos literálnak a következő részei vannak:
 - egy decimális egész, egy tizedespont (".")
 - egy törtrész (másik decimális szám)
 - egy exponens (e, E) és egy típus utótag. Az exponens rész az egy "e" vagy "E" és utána egy egész, ami lehet előjeles (+,-).



```
3.1415  
-3.1E12  
.1e12  
2E-12
```

- **Boolean literálok**

- A boolean típusnak két literálértéke lehet: igaz és hamis.

```
"blah"  
'blah'  
"1234"  
"one line \n another line"
```

- **Sztring literálok**

- nulla vagy több karakter dupla vagy egyszeres idézőjelek közé zárva
- egyszeres idézőjel közötti számot number, míg a kétszeres idézőjel közötti számot string típusúké
- Ha speciális karaktereket szeretnénk használni le kell védeni őket ('escape') a visszaper (backslash - \) jellel

```
\b - backspace (visszalépéses törlés)  
\f - lapdobás  
\r - kocsivissza  
\n - új sor  
\ - védett \  
\" - védett "  
\' - védett '
```

Adattípusok

- A JavaScript dinamikusan tipizált nyelv
 - nem kell deklaráláskor meghatározni egy változó adattípusát
 - az adattípusok automatikusan konvertálódnak, ahogy az a script futása során szükséges
- A JavaScript által megkülönböztetett adattípusok :

Adattípus neve	Informatív leírás
Number	Szám változó, ami egyaránt tartalmazhat egész vagy valós számokat is.
String	Karakterlánc változó. A konstansokat idézőjelek (") között adhatjuk meg, és használhatjuk a C-ben megszokott escape szekvenciákat is.
Boolean	Logikai változó. TRUE (igaz) vagy FALSE (hamis) értéket vehet fel. Ha számként használjuk, akkor a TRUE értéke 1, a FALSE értéke pedig 0 lesz.
Undefined	A változó létre lett hozva, de nem lett hozzárendelve érték. Arra vár, hogy értéket kapjon, és ekkor a típusa is megváltozik.
Null	Ez egy speciális adat, ami azt jelzi, hogy a változó nem tartalmaz semmit. Ha szöveggént használjuk automatikusan átalakítja "null" értéké, ha számként, akkor 0-ávé.
Function	Függvényváltozó. értéknek egy függvényt kaphat, amit akár meg is hívhatunk.
Object	Objektum változó, ami egy összetett adattípus.
Symbol (ECMAScript 2015)	Egyedi azonosítót jelöl, ami a Symbol() függvényt használva jön létre

Adattípusok

• Számok:

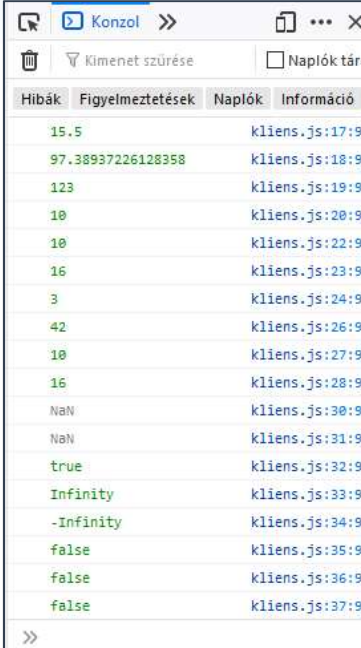
- A számok a JavaScript - ben a specifikáció szerint "dupla pontosságú 64-bites értékek.
- JavaScript - ben nincs olyasmihez hasonlítható, mint az integer, ami miatt óvatosan kell az aritmetikával bánni, ha a matematikát úgy alkalmazzuk mint a C -ben, vagy Java -ban.
- A parseInt() és a parseFloat() addig olvassák a stringet, amíg nem találnek egy olyan karaktert, ami a számrendszerben nem található, és csak az addig beolvasott számokat adják vissza értékül.
- Az unáris "+" viszont egyszerűen NaN értéket ad vissza, ha a string egy nem érvényes karaktert tartalmaz.

```
function Szamok(){
  var r = 15.5;
  var atmero = 2 * Math.PI * r;
  console.log(r);
  console.log(atmero);
  console.log(parseInt('123', 10)); // 123
  console.log(parseInt('010', 10)); // 10

  console.log(parseInt('010')); // 10( pedig 8 kellene, hogy legyen)
  console.log(parseInt('0x10')); // 16
  console.log(parseInt('11', 2)); // 3

  console.log(+ '42'); // 42
  console.log(+ '010'); // 10
  console.log(+ '0x10'); // 16

  console.log(parseInt('hello', 10)); // NaN
  console.log(NaN + 5); // NaN
  console.log(isNaN(NaN)); //true
  console.log(1 / 0); //Infinity
  console.log(-1 / 0); //-Infinity
  console.log(isFinite(1 / 0)); //false
  console.log(isFinite(-Infinity)); //false
  console.log(isFinite(NaN)); //false
}
```



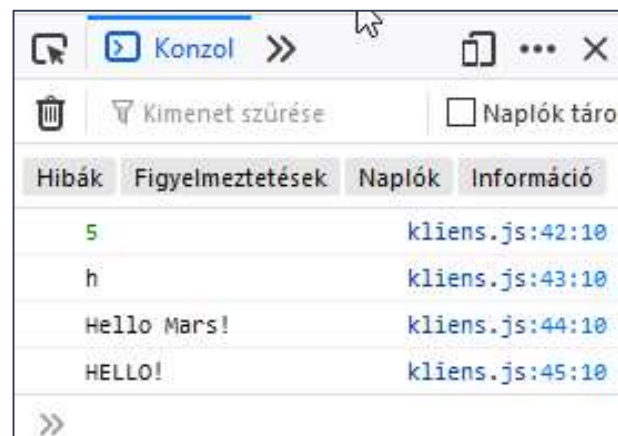
Hibák	Figyelmeztetések	Naplók	Információ
			15.5 kliens.js:17:9
			97.38937226128358 kliens.js:18:9
			123 kliens.js:19:9
			10 kliens.js:20:9
			10 kliens.js:22:9
			16 kliens.js:23:9
			3 kliens.js:24:9
			42 kliens.js:26:9
			10 kliens.js:27:9
			16 kliens.js:28:9
			NaN kliens.js:30:9
			NaN kliens.js:31:9
			true kliens.js:32:9
			Infinity kliens.js:33:9
			-Infinity kliens.js:34:9
			false kliens.js:35:9
			false kliens.js:36:9
			false kliens.js:37:9

Adattípusok

- **Karakterláncok:**

- A stringek a JavaScript -ben Unicode karakterek sorozata.
- Még pontosabban fogalmazva, a stringek UTF-16 kódjegységek sorozata; mindegy egyes egység egy 16-bites számmal van reprezentálva.
- Minden Unicode karakter egy, vagy két kódjegységből áll.

```
function Karakterlancok(){  
    console.log('hello'.length); //5  
    console.log('hello'.charAt(0)); //h  
    console.log('Hello World!'.replace('World','Mars')); //Hello Mars!  
    console.log('Hello!'.toUpperCase()); //HELLO!  
}
```



Változók, értékadás

- Deklarálás:

```
var a,b;  
var c=3; // deklarálás kezdőérték adásával  
var s="Szöveg";  
var z='másik szöveg';
```

- Függvény szintű változókat a *var* kulcsszó segítségével hozunk létre

- Változó létrehozásakor rögtön adhatunk a változónak kezdőértéket. Ha ezt nem tesszük meg, akkor a változó értéke *undefined* lesz.
- Nem kötelező deklarálni a változókat; az első értékadással mindenképpen létrejönnek.
- *Ha elhagyjuk a var kulcsszót, akkor minden esetben globális változó jön létre. Ennek nem várt mellékhatásai lehetnek, éppen ezért minden esetben ajánlott a var kulcsszó használata!!!*

- Függvényen belül deklarált változó csak ott érhető el. Ha esetleg van azonos nevű globális változó, akkor azt *elfedi* a függvény futása idejére

- A változók **hatóköre** kétféle lehet

- Globális: Az egész javascript kódon belül elérhetők
- Lokális: Függvényen belül deklarált változó csak a függvényen belül érhető el. A függvény paraméterek mindig lokálisak a függvényre nézve

```
<script type="text/javascript">  
<!--  
var myVar = "global"; // Globális változó deklarálása  
function checkscope( ) {  
var myVar = "local"; // Lokális változó deklarálása  
document.write(myVar);  
}  
//-->  
</script>
```

Változók, értékadás - LET

• Deklarálás:

```
let a;  
let name = 'Simon';
```

- Blokkszintű változókat a *let* kulcsszó segítségével hozunk létre. Az így deklarált változó abból a kód blokkból érhető el, ahol azt definiálták.

```
function exampLet(){  
  //console.log('Előtte:' + myLetVariable);  
  // A myLetVariable itt *NEM* látható.  
  for (let myLetVariable = 0; myLetVariable < 5; myLetVariable++) {  
    // A myLetVariable csak itt elérhető  
    console.log(myLetVariable);  
  }  
  // A myLetVariable itt *SEM* látható.  
  console.log('Utána:' + myLetVariable);  
}
```

```
function exampVar(){  
  console.log('Előtte:' + myVarVariable);  
  // A myLetVariable itt *NEM* látható.  
  for (var myVarVariable = 0; myVarVariable < 5; myVarVariable++) {  
    // A myLetVariable csak itt elérhető  
    console.log(myVarVariable);  
  }  
  // A myLetVariable itt *NEM* látható.  
  console.log('Utána:' + myVarVariable);  
}
```

Kimenet szűrése		Naplók tára
Hibák	Figyelmeztetések	Naplók
0		kliens.js:55:13
1		kliens.js:55:13
2		kliens.js:55:13
3		kliens.js:55:13
4		kliens.js:55:13
!	ReferenceError: myLetVariable is not defined [További tudnivalók]	kliens.js:58:1

Kimenet szűrése		Naplók tára
Hibák	Figyelmeztetések	Naplók
	Előtte:undefined	kliens.js:62:9
0		kliens.js:66:13
1		kliens.js:66:13
2		kliens.js:66:13
3		kliens.js:66:13
4		kliens.js:66:13
	Utána:5	kliens.js:69:9

► Konstansok:

- A konstans változók értéke nem módosítható a szkript futása során.
- konstansok esetében a **const** kulcsszót kell használnunk, nevének pedig ugyanazon feltételeknek kell eleget tennie, mint a változóknak.
- Konstansokat deklarálhatunk alprogramon belül és kívül is, a kívül deklaráltak globálisan elérhetőek.
Pl.:

```
const a = '12';
```

FOGLALT SZAVAK, KONSTANSOK

► Foglalt szavak:

- Olyan szavak, amelyeket nem lehet változó, függvény, eljárás, ciklus címke vagy objektum nevének adni.

abstract	default	float	long	super	void
boolean	delete	for	native	switch	volatile
break	do	function	new	synchronized	while
byte	double	goto	null	this	with
case	else	if	package	throw	
catch	enum	implements	private	throws	
char	export	import	protected	transient	
class	extends	in	public	true	
const	false	Instanceof	return	try	
continue	final	int	short	typeof	
debugger	finally	interface	static	var	

Operátorok:

A JavaScript operátorai nagyrészt megfelelnek a C, illetve Java nyelvekben használatos operátoroknak, és meghatározott precedenciával rendelkeznek.

Az operátorok csoportosítása:

- **Értékadás**

értékadás \rightarrow =

értékadás bal-, illetve jobboldali operandusokkal. Pl.: $x*=y \leftarrow \rightarrow x=x*y$.

- **Összehasonlítás**

Az összehasonlítások logikai értékekkel térnek vissza.

Pl.: Egyenlőség vizsgálatnál lehet csak az értéket (==), valamint a típus egyenlőségét is (===) is vizsgálni.

- **Aritmetikai**

növelés (++), csökkentés (--), negáció (-), illetve a modulo (%)

- **Bitenkénti**

A bitenkénti operátorok használatakor az operandusokat először 32 bites számokká konvertáljuk, majd ezeken a biteken egymás után alkalmazzuk a műveletet. Például bitenkénti és esetében "15 & 9" eredménye 9 lesz.

- **Logikai**

A nyelv a logikai és, illetve vagy műveleteknél lusta kiértékelést használ, azaz ha már a kifejezés első tagja megadja az eredményt, a második tagot már nem is veszi figyelembe.

Operátorok:

- **Sztring**

A sztringekre az összehasonlító operátorokon kívül a konkatenáció operátort (+) és a rövidített értékadó operátort (+=) is lehet használni sztringek konkatenálására.

- **Különleges**

ide tartozik minden más operátor:

- **feltételes kiértékelés** (a ? b : c): a logikai értékétől függően b-t, vagy c-t adja vissza
- **vessző**: tömbelemek felsorolásához
- **delete**: objektumtörléshez
- **in**: objektum bejárásához
- **instanceof**: objektum típusának lekérdezése
- **new**: új objektum létrehozása
- **this**: saját objektum
- **typeof**: objektum típusának lekérdezése
- **void**: kifejezés kiértékelése visszatérési érték nélkül

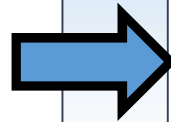
Operátorok:

• Operatorok precedenciája

Value	Operator	Description	Example	Value	Operator	Description	Example
19	()	Expression grouping	(3 + 4)	11	<	Less than	x < y
18	.	Member	person.name	11	<=	Less than or equal	x <= y
18	[]	Member	person["name"]	11	>	Greater than	x > y
17	()	Function call	myFunction()	11	>=	Greater than or equal	x >= y
17	new	Create	new Date()	10	==	Equal	x == y
16	++	Postfix Increment	i++	10	===	Strict equal	x === y
16	--	Postfix Decrement	i--	10	!=	Unequal	x != y
15	++	Prefix Increment	++i	10	!==	Strict unequal	x !== y
15	--	Prefix Decrement	--i	6	&&	Logical and	x && y
15	!	Logical not	!(x==y)	5		Logical or	x y
15	typeof	Type	typeof x	3	=	Assignment	x = y
14	*	Multiplication	10 * 5	3	+=	Assignment	x += y
14	/	Division	10/5	3	-=	Assignment	x -= y
14	%	Modulo division	10 % 5	3	*=	Assignment	x *= y
14	**	Exponentiation	10 ** 2	3	%=	Assignment	x %= y
13	+	Addition	10 + 5	3	<<=	Assignment	x <<= y
13	-	Subtraction	10 - 5	3	>>=	Assignment	x >>= y
12	<<	Shift left	x << 2	3	>>>=	Assignment	x >>>= y
12	>>	Shift right	x >> 2	3	&=	Assignment	x &= y
12	>>>	Shift right (unsigned)	x >>> 2	3	^=	Assignment	x ^= y
				3	=	Assignment	x = y

Operátorok:

```
function AritmeticalOps() {  
    var x = 100;  
    var y = 200;  
    var szo = "Teszt";  
    var sortores = "<br/>";  
  
    document.write("x = ", x, sortores);  
    document.write("y = ", y, sortores);  
    document.write("szo = ", szo, sortores);  
    document.write(sortores);  
  
    document.write("x + y =");  
    eredmeny = x + y;  
    document.write(eredmeny);  
    document.write(sortores);  
  
    document.write("x - y =");  
    eredmeny = x - y;  
    document.write(eredmeny);  
    document.write(sortores);  
}
```



Aritmetikai operátorok
Összehasonlító operátorok
Logikai operátorok
Bitenkénti operátorok
Értékadó operátorok
Feltételes operátorok
Típusa operátorok

x = 100
y = 200
szo = Teszt

x + y = 300
x - y = -100
x * y = 20000
x / y = 0.5
x % y = 100

előtte: x = 100
x++ = 100
után: x = 101

előtte: y = 200
--y = 199
után: y = 199

a = 2 ; b = 3

Value of a => (a = b) => 3

a = 3 ; b = 3

Value of a => (a += b) => 6

a = 6 ; b = 3

Value of a => (a -= b) => 3

a = 3 ; b = 3

Value of a => (a *= b) => 9

a = 9 ; b = 3

Value of a => (a /= b) => 3

a = 3 ; b = 3

Value of a => (a %= b) => 0

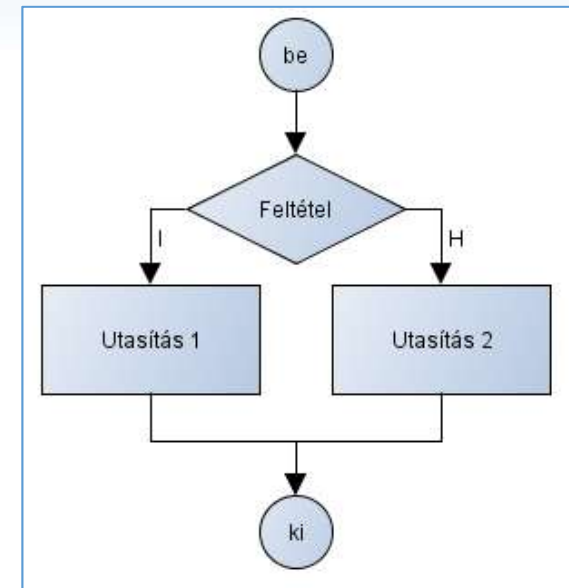
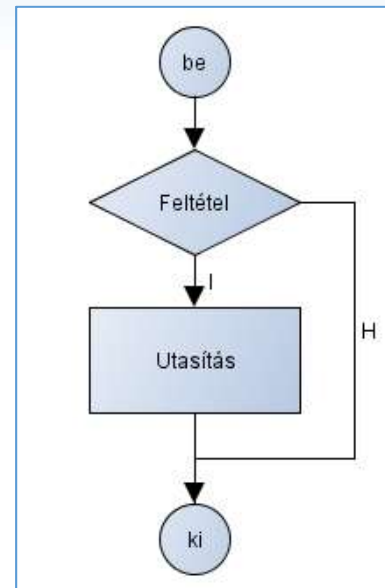
```
<html lang="hu">  
  <head>  
    <title>Operátorok!!!</title>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <link href="css/base.css" rel="stylesheet" type="text/css"/>  
    <script src="scripts/base.js" type="text/javascript"></script>  
  </head>  
  <body>  
    <input type="button" value="Aritmetikai operátorok" name="aritmetikai_peratorok" onclick="AritmeticalOps()" />  
    <input type="button" value="Összehasonlító operátorok" name="osszehasonlito_operatorok" onclick="CompareOps()" />  
    <input type="button" value="Logikai operátorok" name="logikai_operatorok" onclick="LogicalOps()" />  
    <input type="button" value="Bitenkénti operátorok" name="bitenkenti_operatorok" onclick="BitwiseOps()" />  
    <input type="button" value="Értékadó operátorok" name="ertekado_operatorok" onclick="AssignmentOps()" />  
    <input type="button" value="Feltételes operátorok" name="felteteles_operatorok" onclick="ConditionalOps()" />  
    <input type="button" value="Típusa operátorok" name="tipusa_operatorok" onclick="TypeofOps()" />  
  </body>  
</html>
```


Vezérlési szerkezetek - elágazások

- Kétirányú elágazás:

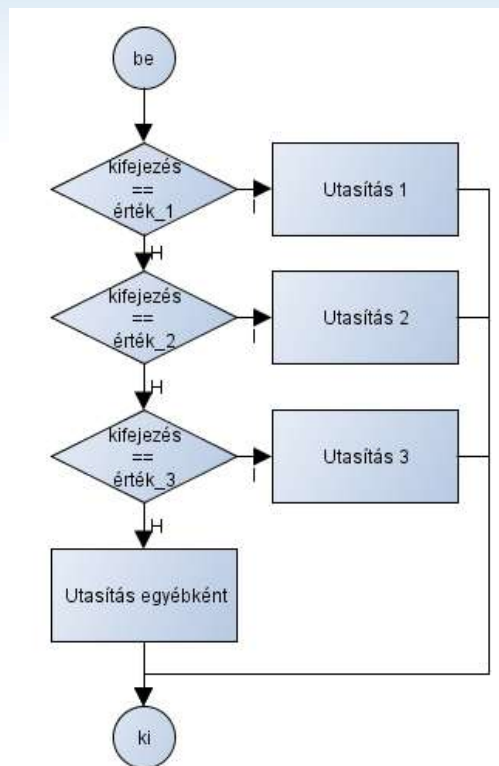
- **if-else** szerkezet. Az **if-et** a feltétel követi, amelynek ha logikai értéke *igaz* akkor az azt követő utasításblokkot hajtja végre, ha *hamis* akkor az **else-et** követő blokkot.
 - Az else ágat nem kötelező használni
 - Ha csak egy utasítást akarunk írni az ágakba, akkor elhagyhatjuk a blokkosítást
 - A kiértékelendő kifejezés tetszőlegesen összetett lehet, amit logikai és/vagy és operátorokkal köthetünk össze.

```
if(){  
    utasítások1;  
}  
else{  
    utasítások2;  
}
```



Vezérlési szerkezetek - elágazások

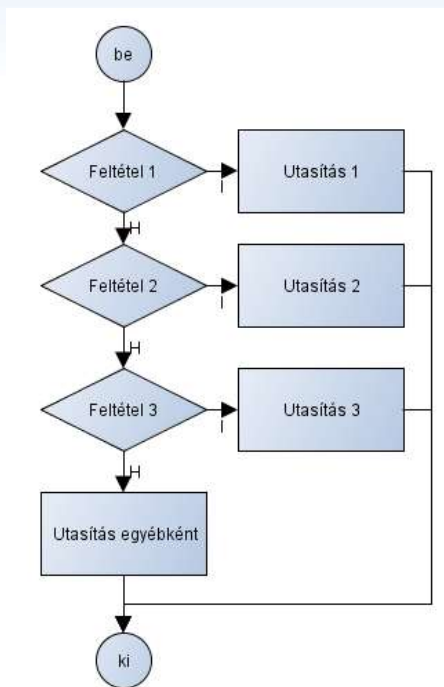
- Többirányú elágazás:
 - ha több ágon szeretnénk egy feltételt vizsgálni, akkor egy **switch szerkezettel** ezt megtehetjük.
 - A *switch* paramétereként egy kifejezést kaphat, amely nem csak logikai, hanem tetszőleges értéket vehet fel
 - A kifejezés a különböző felvett értékeknek megfelelően külön programblokkokat futtathatunk.
 - ☐ Az itt használatos programblokkban nem használunk "{" illetve "}" jeleket
 - ☐ a blokkok azonosítása a "case címke" paranccsal történik
 - ☐ amennyiben egyik érték (címke) sem felel meg a kifejezésnek, akkor írhatunk egy alapértelmezett (default) blokkot,
 - ☐ Ha nincs default ág, a végrehajtás a switch utáni következő utasításnál folytatódik.



```
switch (kifejezés) {  
    case címke_1: utasítások; break;  
    case címke_2: utasítások; break;  
    case címke_3: utasítások; break;  
    default: utasítások;
```

Vezérlési szerkezetek - elágazások

- Többirányú elágazás -2 :
 - Többágú elágazás létrehozására 2 lehetőségünk az, hogy **if else if szerkezettel** tetszőleges számú egymást kizáró feltételt vizsgálunk.
 - az **else if ágak** száma határozza meg az elágazások számát.
 - Az utolsó **else ág** léte vagy elhagyása határozza meg, hogy biztosan végrehajtódik-e valamelyik ág vagy a feltételek nem teljesülése esetén a program a következő utasításon folytatódik.



```
if (){  
    utasítások 1;  
}  
else if (logikai kifejezés 2){  
    utasítások 2;  
}  
else if (logikai kifejezés 3){  
    utasítások 3;  
}  
else{  
    utasítások egyébként;  
}
```

Vezérlési szerkezetek - elágazások

if-else

A :

B :

switch-case

Elért pontszám :

Érdemjegy:

```
<html lang="hu">
<head ...7 lines />
<body>
  <div>
    <p class="cimke"> if-else</p>
    <p>
      <label for="a"> A : </label>
      <input type="text" id="vltzA" name="a" placeholder=" Írd be A értékét!" />
    </p>
    <p>
      <label for="b"> B : </label>
      <input type="text" id="vltzB" name="b" placeholder=" Írd be B értékét!" />
    </p>
    <input type="button" value="Tesztel" name="tesztel" onclick="tesztel()"/>
  </div>
  <div>
    <p class="cimke"> switch-case </p>
    <p>
      <label for="pontszam"> Elért pontszám : </label>
      <input type="text" id="pontszam" name="pontszam" placeholder="Elért pontszám" />
    </p>
    <p>
      <label for="jegy"> Érdemjegy: </label>
      <input type="text" id="erdemjegy" name="jegy" placeholder="Érdemjegy" disabled />
    </p>
    <input type="button" value="Osztályoz" name="osztalyoz" onclick="osztalyoz()"/>
  </div>
</body>
</html>
```

```
function tesztel() {
  a = parseInt(document.getElementById("vltzA").value);
  b = parseInt(document.getElementById("vltzB").value);
  if (a == b) {
    document.writeln("A = B");
  } else if (a > b) {
    document.writeln("A > B");
  } else {
    document.writeln("B > A");
  }
}

function osztalyoz() {
  pont = parseInt(document.getElementById("pontszam").value) % 5 + 1;
  switch (pont) {
    case 1:
      document.getElementById("erdemjegy").value = "elégtelen";
      break;
    case 2:
      document.getElementById("erdemjegy").value = "elégséges";
      break;
    case 3:
      document.getElementById("erdemjegy").value = "közepes";
      break;
    case 4:
      document.getElementById("erdemjegy").value = "jó";
      break;
    default:
      document.getElementById("erdemjegy").value = "jeles";
      break;
  }
}
```

Vezérlési szerkezetek - Ciklusok

- **Elől tesztelős ciklus:**

- Először mindig feltétel ellenőrzés történik
 - → ha a feltétel teljesül, a ciklusmagban lévő utasítás (sorozat) végrehajtódik, majd a vezérlés ismét a feltétel ellenőrzéséhez kerül
 - → ha a feltétel nem teljesül akkor a vezérlés átadódik a ciklust követő utasításra, (azaz továbblépés , vagy kilépés a ciklusból)
- Mivel a ciklusmagot csak akkor hajtja végre, ha a feltétel igaz, az is lehetséges, hogy a mag utasításai egyszer sem futnak le
- a feltétel mindig igaz akkor végtelen ciklust kapunk

```
i = 0;
while ( i <= 10) {
    document.write(i + "<br/>");
    i = i + 1;
}
```

Vezérlési szerkezetek - Ciklusok

- **Hátul tesztelő ciklus:**

- először lefut egyszer a ciklusmag → majd ellenőriz egy ciklusfeltételt.
 - → ha a feltétel még mindig teljesül, akkor a lefut ismét a ciklusmag és a vezérlés ismét a feltétel ellenőrzéséhez kerül
 - → ha a feltétel nem teljesül akkor a vezérlés átadódik a ciklust követő utasításra, (azaz továbblépés , vagy kilépés a ciklusból)
- a mag egyszer mindenképpen lefut
- Ha a feltétel mindig igaz akkor végtelen ciklust kapunk

```
i = 0;  
do{  
    document.write(i + "<br/>");  
    i = i + 1;  
}  
while ( i <= 10)
```

Vezérlési szerkezetek - Ciklusok

- A **for** ciklus az előltesztelő ciklus átfogalmazása, ami addig ismétli a ciklusmagot, amíg a feltétel igaz.
- A működéséhez három paramétert(inicializálás, inkrementálás vagy dekrementálás, ciklusfeltétel) kell megadni .
 - Az inicializálásban megadhatunk egy, vagy több értékadást, amelyeket a későbbiekben ciklusváltozóként használunk.
 - Az inkrementálásban ezen változók változtatjuk, azaz nem kell egyenként növelnünk a változót.
 - A feltételben megadhatunk a ciklusváltozóra vonatkozó logikai feltételt.

inicializálás ciklusfeltétel {in|de}krementálás

```
for ( i = 0; i <= 10; i = i + 1){  
    document.write(i + "<br/>");  
}
```

Vezérlési szerkezetek - Ciklusok

- A **for ... in** *ciklust* arra használjuk, hogy végiglépjünk egy objektum tulajdonságain/attribútumain.
- az objektum minden attribútumát végigveszi, ezekre számszerűen hivatkozhatunk, mintha tömbelemekre hivatkoznánk.
- A feltételben meg kell adnunk az objektumot, illetve, hogy milyen változóval szeretnénk végigiterálni.

Például ha egy objektum változóinak nevét és értékét szeretnénk kiírni:

```
<script>
  var person = {
    fname: "John",
    lname: "Doe",
    age: 25
  };
  var text = "";
  var x;
  for (x in person) {
    text += x + " : " + person[x] + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
</script>
```

Példa A for ...in ciklusra

fname : John
lname : Doe
age : 25

Vezérlésátadó utasítások

- A **break** [*címke*] utasítás, használatkor futás kilép a ciklusmagból, és a vezérlést átadja a ciklust követő utasításnak. Ha egy *címke* nevét írjuk a **break** után, akkor a vezérlés ahhoz a címkéhez adódik át.
- A **continue** [*címke*] utasítást, a ciklusmagban megadva abbamarad a ciklusmag végrehajtása, és a feltétel-kiértékeléshez adódik át a vezérlés, azaz a következő ciklusban folytatódik a végrehajtás. Ha megadunk egy *címkét* is az utasításnak, ekkor az adott címkénél folytatódik a végrehajtás.

```
function testBreak() {  
    var i = 0;  
    var x = parseInt(document.getElementById("szam").value);  
    while (i < 6) {  
        document.getElementById("eredmeny").innerHTML += "i: " + i + "<br>";  
        if (i == 3) {  
            break;  
        }  
        i += 1;  
    }  
    document.getElementById("eredmeny").innerHTML += "Eredmény:" + i*x;  
}
```

Szám: 15

szamol

i: 0

i: 1

i: 2

i: 3

Eredmény:45

```
<script>  
    var i = 0;  
    var j = 8;  
  
    checkiandj: while (i < 4) {  
        console.log('i: ' + i);  
        i += 1;  
    }  
    checkj: while (j > 4) {  
        console.log('j: ' + j);  
        j -= 1;  
        if ((j % 2) == 0)  
            continue checkj;  
        console.log(j + ' is odd.');    }  
    console.log('i = ' + i);  
    console.log('j = ' + j);  
</script>
```

Vizsgáló Konzol H

Hálózat CSS JS Bi

i: 0
j: 8
7 is odd.
j: 7
j: 6
5 is odd.
j: 5
i = 1
j = 4
i: 1
i = 2
j = 4
i: 2
i = 3
j = 4
i: 3
i = 4
j = 4