

WEBES PROGRAMOZÁS

KÉSZÍTETTE: TÓTH ATTILA

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Szociális
Alap



BEFEKTETÉS A JÖVŐBE

A tananyag célja

- összefoglalni a kliens oldali programozást a JavaScript nyelven keresztül, bemutatva annak legfontosabb jellemzőit, eszközeit, használatát
- összefoglalni a szerver oldali programozást a PHP nyelven keresztül, bemutatva annak legfontosabb jellemzőit, eszközeit, használatát
- önálló gyakorlásra, tanulásra buzdítani feladatokkal
- felkelteni az érdeklődést a téma iránt

A tananyag nem fedi le a teljes PHP és JavaScript eszközkészletet, a további ismeretek megszerzése önálló feladat

ELŐFELTÉTELEK

A tananyag megértéséhez és elsajátításához szükséges előzetes ismeretek

- általános webes felhasználói ismeretek
- HTML nyelv ismerete
- CSS nyelv ismerete
- alapvető programozási ismeretek



TARTALOM

A tananyag három fő része

- I. Webes működés alapok
- II. Kliens oldali programozás, a JavaScript nyelv
- III. Szerver oldali programozás, a PHP nyelv



JavaScript

PHP



WEBES MŰKÖDÉS

SZÉCHENYI 



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Szociális
Alap



BEFEKTETÉS A JÖVŐBE

TARTALOM

- Webes szereplők
- Tartalomtípusok
- Kliens és szerver oldali programozás
- Kliens-szerver kommunikáció

SZEREPLŐK

A webes működésnek két alapvető szereplője van:

- Kliens
- Szerver

A kliensen jellemzően egy böngésző programot értünk, amelyet egy felhasználó irányít.

A szerveren egy web-szerver alkalmazást értünk, amely szolgáltatja a webes tartalmat. A szerverbe gyakran beleértük a szerveren található egyéb alkalmazásokat, például adatbázis-szervert

TARTALOMTÍPUSOK

Statikus tartalom

Statikus weboldalnak nevezzük az olyan oldalakat, amelyek tartalma állandó, a forráskódban le van írva.

Statikus weboldal készítéséhez nem szükséges programozás, elegendő két jelölő nyelv ismerete: a HTML és a CSS

(Valójában elég a HTML, de korszerű, igényes weboldal készítéséhez szükséges a CSS nyelv.)

TARTALOMTÍPUSOK

Dinamikus tartalom

Dinamikus weboldalnak olyan oldalakat nevezünk, amelyek tartalmát program állítja elő jellemzően egy változó forrásból kiolvastva, például adatbázis, fájl, stb.

A dinamikus weboldal felépítését végző programkódokat két csoportra osztjuk aszerint, hogy hol hajtódik végre:

- kliens oldali program
- szerver oldali program



KLIENS OLDALI PROGRAMOZÁS

A kliens oldali programkódot a böngésző hajtja végre a kliens oldal eszközén. Így csak olyan forrásokat tud kezelni, amelyek a kliens gépen megtalálhatóak. Ebből következik, hogy

- a kliens eszközt terheli,
- a forráskód a kliens oldalon elérhető,
- az alkalmazás szempontjából kevésbé biztonságos.

A legnépszerűbb kliens oldali programnyelv a *JavaScript*, illetve az erre épülő nyelvek

SZERVER OLDALI PROGRAMOZÁS

A szerver oldali programot a webszerver hajtja végre, illetve az azzal kapcsolatban lévő egyéb alkalmazások.

Ebből következik, hogy

- a szerver eszközt terheli,
- olyan forrásokat tud elérni, amelyek szerver oldalon találhatóak,
- a forráskód csak a szerveren érhető el, csak az eredménye jut le kliens oldalra.

Egyik népszerű szerver oldali programnyelv a *PHP*

WEBES KOMMUNIKÁCIÓ

Egy webes tartalom, egy weboldal eléréséhez a következő leegyszerűsített lépések vezetnek:

1. Kliens oldalon a böngészőt utasítják a tartalom megjelenítésére (általában egy felhasználó)
2. A kliens kérést küld a szerverhez a kívánt tartalom igényével
3. A szerver feldolgozza a kérést és előállítja a választ
4. A szerver visszaküldi a választ a kliensnek
5. A kliens fogadja és értelmezi a választ
6. A kliens megjeleníti a weboldalt

WEBES KOMMUNIKÁCIÓ

- A kliens-szerver kommunikáció a HTTP protokollon keresztül történik
- A kliens és a szerver között nincs folyamatos kapcsolat, csak egy üzenetküldés
- Kliens oldali programot a böngésző tud futtatni egyrészt a kérés elküldése előtt, illetve a válasz fogadása után
- A szerver oldali programkód a kérés értelmezése és a válasz előállításakor fut le

KLIENS OLDALI WEBES PROGRAMOZÁS A JAVASCRIPT NYELV

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Szociális
Alap



BEFEKTETÉS A JÖVŐBE

TARTALOM

- Javascript története, végrehajtása, változatok, általános szintaktikai szabályok
- Értékek, változók, literálok, adattípus konverzió, konstansok
- Kifejezések és operátorok
- Utasítások (blokk-, ciklus-, objektum manipulációs és kivételkezelő utasítások)
- Függvények (definiálás, meghívás, használat, függvények mint adatok, előre definiált függvények)
- Objektumok (tulajdonságok, létrehozás, indexelés, metódusok, prototípus és öröklés, referenciák, osztály és privát tulajdonságok, öröklés, törlés)

TARTALOM

- Szabályos kifejezések
- Javascript a böngészőben, egyszerű szkript-HTML együttműködés, a böngésző hozzáférhető objektumai)
- A DOM 0. és átmeneti szint, W3C DOM-ok
- DOM 0. szint, objektum hierarchia, elrendezés, hozzáférés a böngésző objektumokhoz,
- DOM 0. szint, események
- A böngésző objektumainak részletes ismertetése
- A W3C DOM szint
- JQuery

JAVASCRIPT BEVEZETŐ

Mi a JavaScript?

- A JavaScript egy HTML kódba ágyazott script nyelv
- Végrehajtásához értelmező (interpreter) szükséges
- A JavaScript értelmezőtől független része a Core JavaScript
- A HTML oldalba ágyazott JavaScriptet a böngésző értelmezi

The image shows the JavaScript logo, which consists of the letters 'JS' in a bold, black, sans-serif font. The letters are positioned on a bright yellow square background.

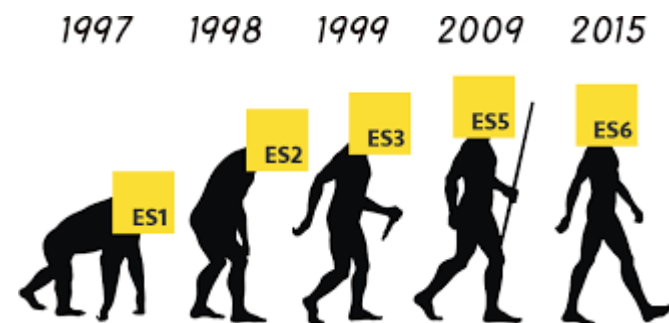
JAVASCRIPT BEVEZETŐ

Történet

- 1995-ben adják ki a Brendan Eich által fejlesztett Javascript nyelvet a Netscape böngészőhöz.
- 1997-ben az ECMA (European Computer Manufacturers Association) kiadja az ECMAScript nevű sztenderdet, a nyelv egységesítésért
- Azóta számos verziója jelent meg, jelenleg a 9, verziónál tartunk ECMAScript 2018 néven
- Az ECMAScript 5-től az összes modern, elterjedt böngésző ismeri
- A Javascript utolsó verziója az 1.8.5, ami megegyezik az ECMA 5-tel



NETSCAPE®



JAVASCRIPT BEVEZETŐ

Szintaktika

- Kis-nagybetű érzékeny
- Utasításokat pontosvesszővel ";" zárjuk
- Egy utasítás több sorba is kerülhet, egy sorban több utasítás is lehet
- Figyelman kívül hagyja a többszörös white-space karaktereket (szóköz, tabulátor, stb.)
- Megjegyzések
 - Egy soros: // megjegyzés
 - Több soros: /* megjegyzés */

JAVASCRIPT BEVEZETŐ

Szigorú mód

ECMAScript 5-től bevezették a szigorú módot, ami egy szigorúbb szabályrendszert ír el a JavaScript kódra. Pl. minden változót kötelező deklarálni.

Úgy adható meg, ha a kódba beírjuk

```
"use strict";
```

Ezután érvényes a szigorú szabályrendszer

Lehet csak lokálisan is megadni egy függvényen belül, akkor csak arra érvényes

'use strict';

JS

JAVASCRIPT BEVEZETŐ

Hova írható?

A HTML kódban bárhol szerepelhet, akár a `head`, akár a `body` részben

Egy oldalon belül több JavaScript rész is lehet

A `<script> ... </script>` tag-ek közé írható

Kerülhet külön állományba, ennek a kiterjesztése jellemzően `.js`

Külső állomány beillesztése a HTML kódba:

```
<script src="JavaScriptFile.js"></script>
```

JAVASCRIPT BEVEZETŐ

Hello.html

```
<html>
<head>
<script>
function Hello(){
    alert("Helló Világ!");
}
</script>
</head>
<body>
<script>
    Hello();
</script>
</body>
</html>
```

JAVASCRIPT BEVEZETŐ

JSFuggvenyeim.js

```
function Hello(){  
    alert("Helló Világ!");  
}
```

Hello.html

```
<html>  
<head>  
<script src="JSFuggvenyeim.js"></script>  
</head>  
<body>  
<script>  
    Hello();  
</script>  
</body>  
</html>
```

JAVASCRIPT BEVEZETŐ

Literálok

Egész szám	100, 0xFF, Infinity
Valós szám	3.14, 314e-2
Szöveg	"példa szöveg" vagy 'példa szöveg'
Logikai	true, false
Tömb	[1,2,3]
Objektum	{ev:2018, honap:"Január", nap:1}
null	"üres" érték (típusa object)
undefined	nem ismert érték

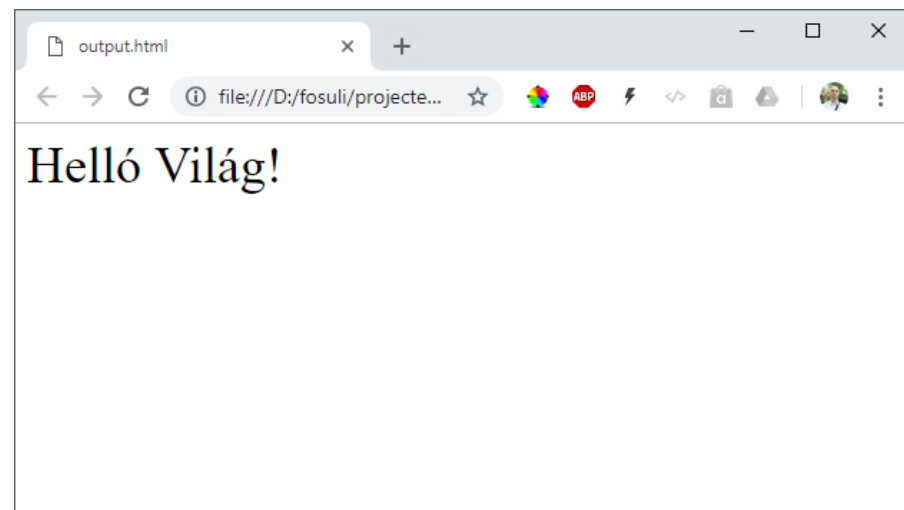
JAVASCRIPT BEVEZETŐ

Output

A HTML kódba írás

```
document.write("Helló Világ");
```

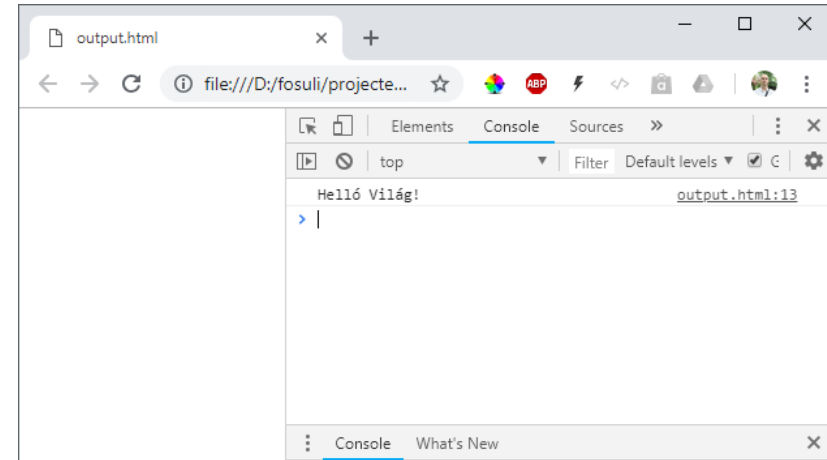
A `document.write()` csak az oldal betöltődése közben használható, ha a betöltődés után hajtódik végre (például esemény hatására), felülírja a teljes oldal tartalmát



JAVASCRIPT BEVEZETŐ

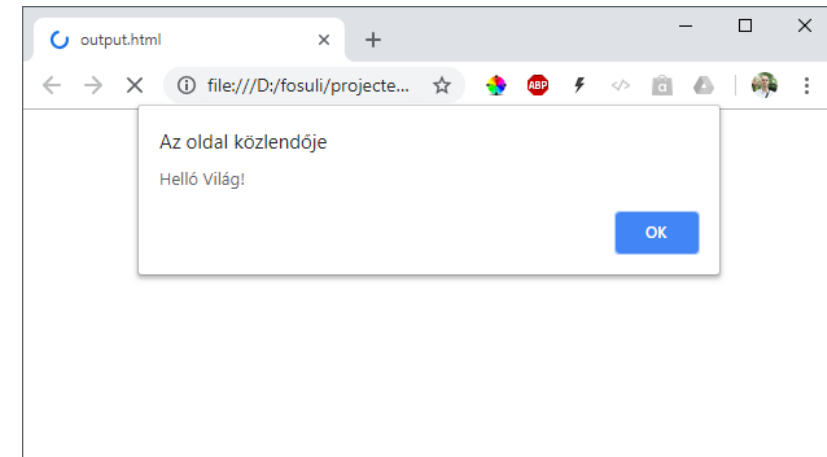
Konzolra írás

```
console.log("Helló Világ");
```



Felugró üzenet ablak

```
window.alert("Helló Világ");
```



VÁLTOZÓK

A változók elnevezésében szerepelhet betű, szám, alsóvonás és \$ jel, nem kezdődhet számmal (jellemzően betűvel kezdjük)

Változó deklarálás

```
var a = 10;
```

A JavaScript lazán típusos nyelv, deklaráláskor nem kell típust megadni, az értékből következik a típus, ami akár változhat is

```
var a = 10;  
a = "almafa";
```

VÁLTOZÓK

Nem kötelező a deklaráció, az első használat deklarál, de ebben az esetben globális változó lesz. (Szigorú módban kötelező a deklaráció!)

Az érték nélküli változó értéke `undefined`

```
var a; // értéke undefined
```

Egy változó újradeklarálásakor megőrzi az eredeti tartalmát

```
var a = 10;
```

```
var a; // értéke még mindig 10
```

KONSTANSOK

Konstans létrehozása

```
const start = "1900.01.01";
```

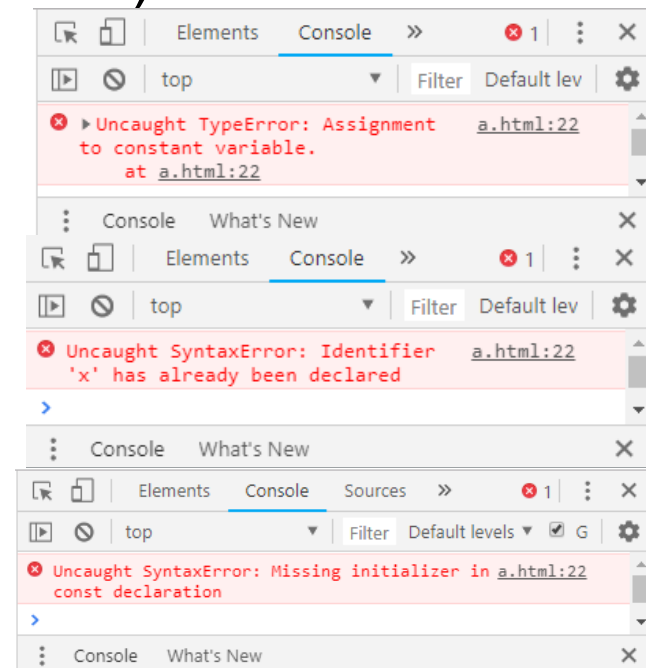
- Értéke nem módosítható
- Nem lehet újra deklarálni (konstansként se)
- Kötelező értéket adni a deklarációkor

```
const x = 10;
```

```
x = 20;           // nem lehet
```

```
var x = 20;       // nem lehet
```

```
const y;          // nem lehet
```



KONSTANSOK

Primitív típusú konstans értéke nem módosítható, de objektum típusú konstans tagjai módosíthatóak

```
const eb = { nev:"Bodri", kor:5 };
```

Módosul a konstans objektum egy értéke

```
eb.kor = 6; // lehet
```

Maga az objektum nem módosítható

```
eb = { nev:"Vakarcs", kor:1 } // nem lehet
```

KONSTANSOK

Hatóköre a létrehozás blokkjára vonatkozik
Más blokkban lehet a konstanst újra deklarálni

```
const x = 10;  
{  
    const x = 20;           // lehet  
}
```

Akár változóra is rá lehet deklarálni

```
var y = 10;  
{  
    const y = 20;           // lehet  
}
```

OPERÁTOROK

Értékadó operátorok

=	a = b;	
+=	a += b;	// a = a + b
-=	a -= b;	// a = a - b
*=	a *= b;	// a = a * b
/=	a /= b;	// a = a / b
%=	a %= b;	// a = a % b – maradékképzés
&=	a &= b;	// a = a & b – bitenként AND
=	a = b;	// a = a b – bitenként OR
^=	a ^= b;	// a = a ^ b – bitenként XOR

OPERÁTOROK

Számműveletek

+	$a + b$	összeadás
-	$a - b$	kivonás
*	$a * b$	szorzás
/	a / b	osztás
%	$a \% b$	maradékképzés
++	$a++$	növelés
--	$a--$	csökkentés

OPERÁTOROK

Összehasonlító operátorok

==	a == b	egyenlő
!=	a != b	nem egyenlő
<=	a <= b	kisebb egyenlő
>=	a >= b	nagyobb egyenlő
<	a < b	kisebb
>	a > b	nagyobb
===	a === b	azonosan egyenlő (érték és típus)
!==	a !== b	értékben vagy típusban nem egyenlő

OPERÁTOROK

Logikai operátorok

!	! a	tagadás - NOT
&&	a && b	és - AND
	a b	vagy - OR

Feltételes operátor

(feltétel) ? értéke1 : érték2

```
(a % 2 == 0) ? "páros" : "páratlan"
```

OPERÁTOROK

Bitműveletek

~	~ a	bitenkénti NOT
&	a & b	bitenkénti AND
	a b	bitenkénti OR
^	a ^ b	bitenkénti XOR
<<	a << b	balra léptetés <i>b</i> bittel
>>	a >> b	jobbra léptetés <i>b</i> bittel előjeltartással
>>>	a >>> b	jobbra léptetés <i>b</i> bittel előjeltartás nélkül

OPERÁTOROK

Bitművelet példák

15 & 9

... 00001111 & ...00001001 -> 1001 9

15 | 9

... 00001111 & ...00001001 -> 1111 15

15 ^ 9

... 00001111 & ...00001001 -> 0110 6

9 << 2

...00001001 << 2 -> ...00100100 36

9 >> 2

...00001001 >> 2 -> ...00000010 2

OPERÁTOROK

Szöveg operátor

+ "Helló " + "Világ" összefűzés (konkatenálás)
+= s += "helló" hozzáfűzés

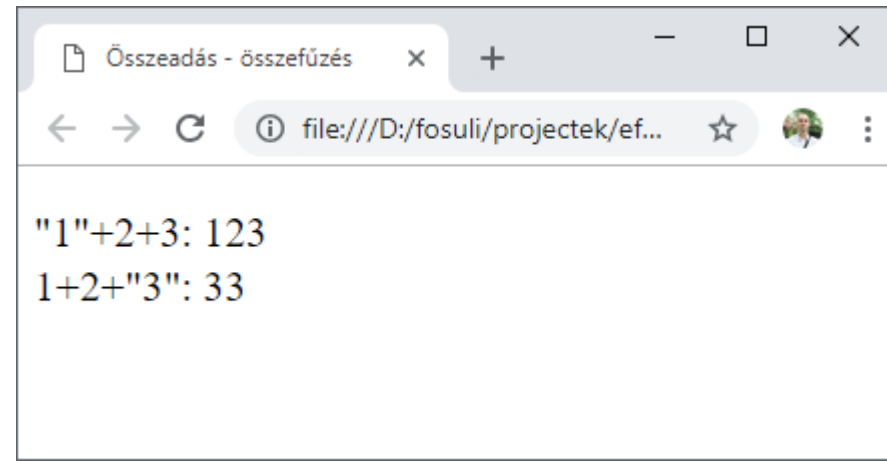
Szám és szöveg érték esetén a + jel összefűzést jelent

"5" + 5 // "55"

Az értelmezés balról jobbra történik

"1" + 2 + 3 // "123"

1 + 2 + "3" // "33"



OPERÁTOROK

Speciális operátorok

kif , *kif*

komma operátor

Balról jobbra kiértékeli a kifejezéseket és az utolsó értékét visszaadja

```
var x = 1;  
var y = 10;  
x++, y += x;  
// x értéke 2  
// y értéke 12
```

OPERÁTOROK

typeof

típusvizsgálat operátor

```
typeof "Helló" // String
```

in

tartalmazás operátor

```
1 in [1, 2, 3] // true
```

instanceof

példányvizsgáló operátor

```
[1,2,3] instanceof Array // true
```


OPERÁTOROK

Objektum operátorok

<code>new</code>	példányosító operátor
<code>this</code>	saját referencia operátor
<code>delete</code>	tag törlő operátor

```
function Kutya(n,k) {  
    this.nev = n;  
    this.kor = k;  
}  
  
var bodri = new Kutya("Bodri",2);  
delete bodri.kor;
```

BLOKK UTASÍTÁS

Utasítások csoportosítása

```
{  
    utasítás;  
    ...  
}
```

```
var x = 1;  
while (x<10) {  
    document.write(x);  
    x++;  
}
```

FELTÉTELES UTASÍTÁSOK

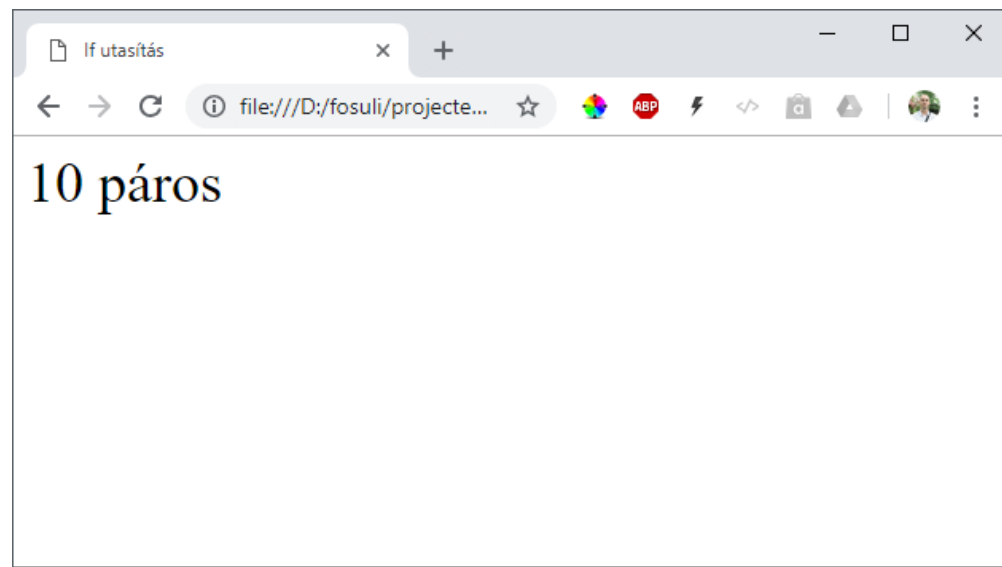
if utasítás

```
if (feltétel) { utasítás }  
    else { utasítás }
```

- Ha a feltétel igaz, az első utasításblokk hajtódik végre, különben a második.
- Az `else` ág elhagyható, ilyenkor hamis esetén a következő utasítással folytatja

FELTÉTELES UTASÍTÁSOK

```
var a = 10;  
if (a % 2 == 0) {  
    document.write(a+" páros");  
} else {  
    document.write(a+" páratlan");  
}
```



FELTÉTELES UTASÍTÁSOK

switch utasítás

```
switch (kifejezés) {  
    case érték1: utasítás; break;  
    ...  
    default: utasítás;  
}
```

- A kifejezést egyszer értékeli ki és sorban hasonlítja az értékekkel. Amelyikkel megegyezik, végrehajtja az oda tartozó utasítást.
- Ha egyik értékkel se egyezik meg, a `default` ág hajtódik végre. Ez az ág elhagyható.

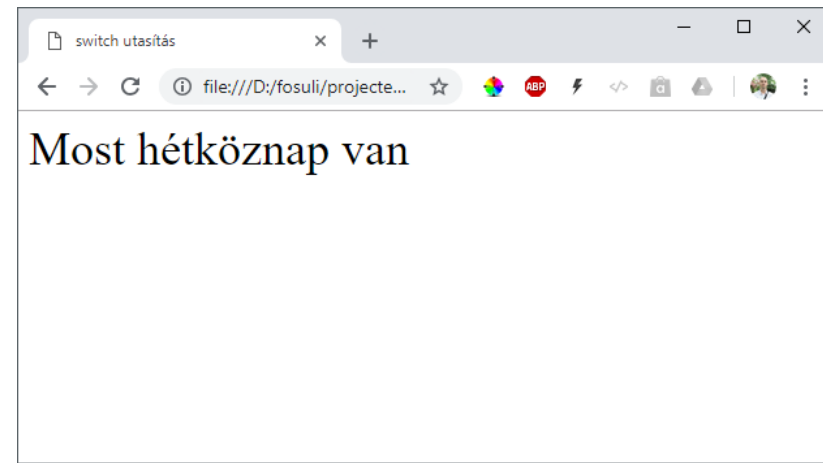
FELTÉTELES UTASÍTÁSOK

switch utasítás

- A kifejezésnek és az értékeknek típusban meg kell egyezni.
- Minden értéket összehasonlít a kifejezéssel. Ha ezt nem akarjuk, kell a `break` az ág végére, ekkor a `switch` utáni utasítással folytatja.
- A `default` ág nem kell, hogy a végén szerepeljen, de akkor ahhoz is kell a `break`
- A `break` nélkül lehetőség több ágakat egyben kezelni

FELTÉTELES UTASÍTÁSOK

```
var d = new Date();  
var t = "";  
switch (d.getDay()) {  
    case 0 :  
    case 6 : t = "hétvége"; break;  
    default : t = "hétköznapi";  
}  
document.write("Most "+t+" van");
```



FELADAT

1. Készítsen olyan kódot, amely kiírja a weboldalra egy adott számról, hogy pozitív, negatív, vagy nulla!
2. Készítsen olyan kódot, amely kiírja a weboldalra magyarul, hogy milyen nap van!

ITERÁCIÓK

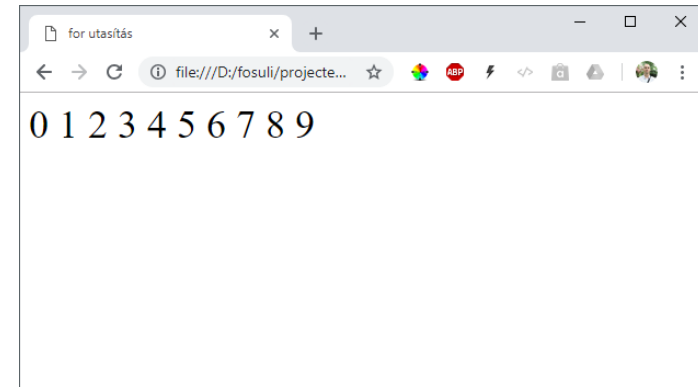
for ciklus

```
for (init; feltétel; lépés) {  
    utasítás;  
}
```

init	az iteráció elején hajtja végre egyszer
feltétel	a ciklusmag végrehajtásának feltétele
lépés	ciklusmag után hajtja végre minden körben

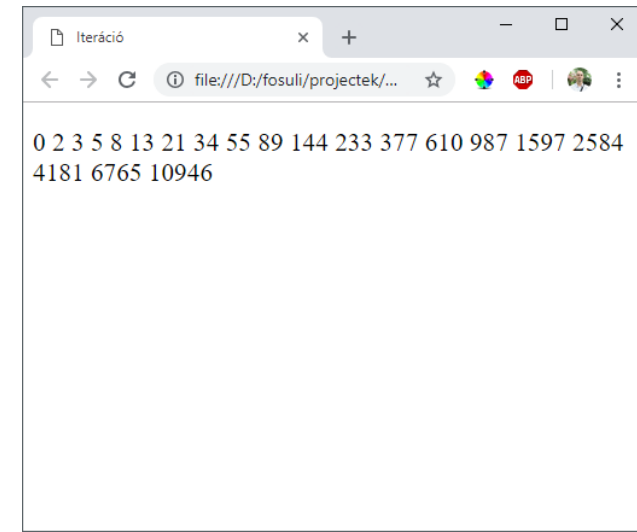
ITERÁCIÓK

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



```
for (a=1,b=1,f=0;f<1000;f=a+b,a=b,b=f) {  
    document.write(f);  
}
```

[mi az f értéke az iteráció végén?]



ITERÁCIÓK

while ciklus

```
while (feltétel) {  
    utasítás  
}
```

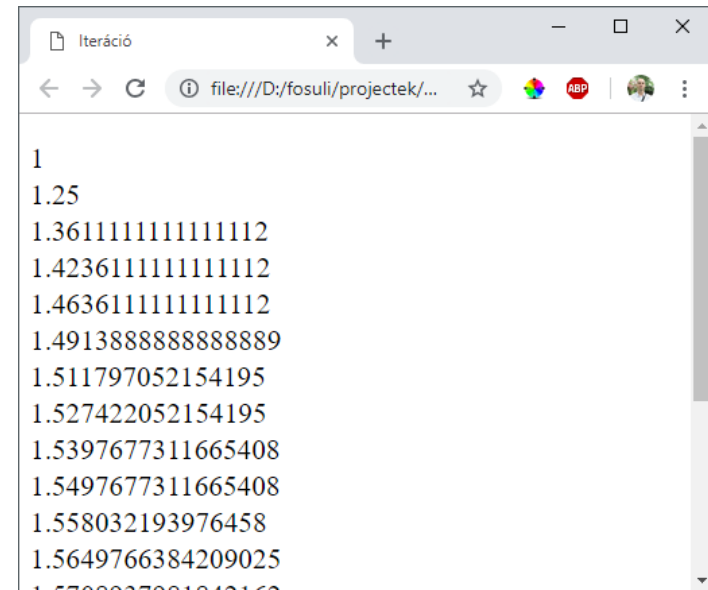
Amíg a feltétel igaz, végrehajtja a ciklusmagot

Elöl tesztelő ciklus

A ciklusmagot akár egyszer se hajtja végre

ITERÁCIÓK

```
var x = 0;
var i = 1;
while (x < 2) {
    x += 1/(i*i);
    document.write(x);
    i++;
}
document.write(x);
```



[mi az x és mi ennek az érdekessége?]

ITERÁCIÓK

do while ciklus

```
do {  
    utasítás  
} while (feltétel)
```

Először végrehajtja a ciklusmagot, majd ha a feltétel teljesül, visszatér a ciklusmaghoz

Hátul tesztelő ciklus

A ciklusmagot egyszer mindenképp végrehajtja

ITERÁCIÓK

```
var t = "";  
var a = 2;  
var h = 1;  
do {  
    t += h;  
    h *= a;  
} while (h<=1000);  
document.write(t);
```

[mi a *t* értéke az iteráció végén?]

ITERÁCIÓK

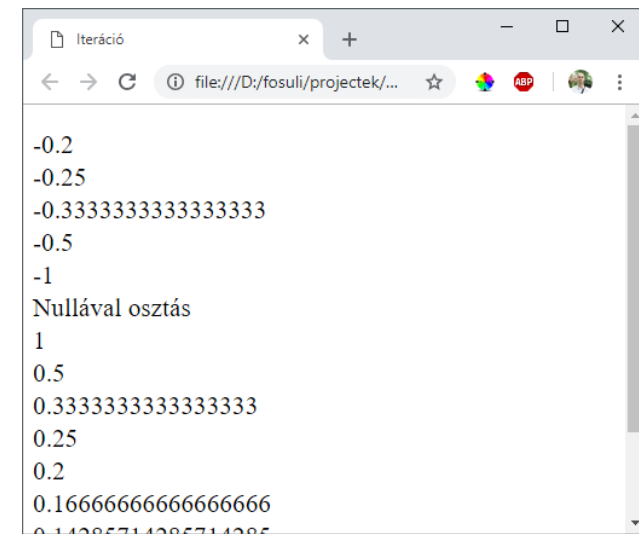
break

Megszakítja az iterációt és a következő utasítással folytatja

continue

Megszakítja az iteráció ciklusmagját és a következő iterációra lép

```
var a = -5, h = 1;
do {
    if (isNaN(a)) { break; }
    if (a == 0) {
        document.write("Nullával osztás");
        a++;
        continue;
    }
    document.write(h / a++);
} while (a < 10);
```



FELADAT

1. Egymáshoz közelítő sorozatok

Adott egy két számtani sorozat első eleme és a növekménye úgy, hogy az egyik sorozat nő, a másik csökken (negatív növekmény). Írjuk ki a sorozatok elemeit párban, amíg el nem érik egymást!

FELADAT

2. Kamatmentes hiteltörlesztés

Felveszünk egy hitelt. A hitel összegét minden hónapban növelik az aktuális összeg 5%-val.

Adott egy törlesztő részlet. Ha a részlet kisebb, mint az indulási összeg 5%-a, üzenettel álljon le!

Ha a következő hónapban kezdjük a törlesztést, akkor írjuk ki, hogy melyik év melyik hónapjában (szövegesen) mennyit törlesztettünk, mennyi van még addig, amíg ki nem fizettük az összeset!

KIVÉTELKEZELÉS

Hibakezelés

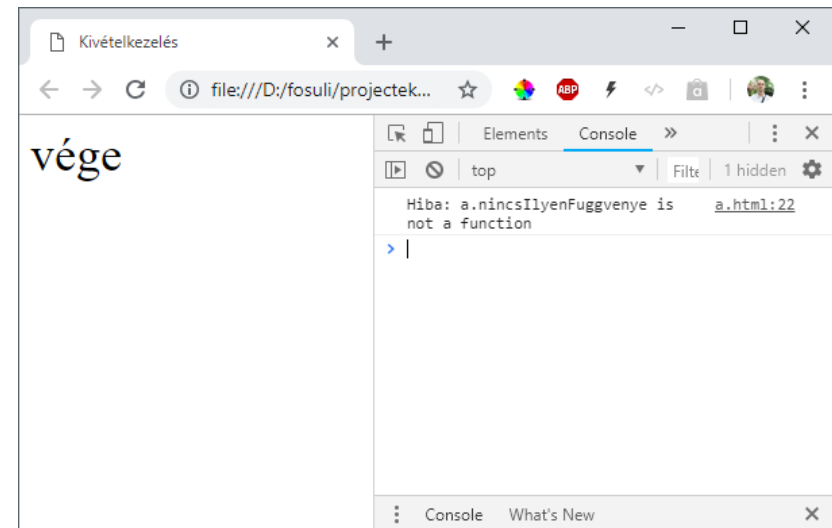
```
try {  
    utasítás  
}  
catch (hiba) { utasítás }  
finally { utasítás }
```

A `try` részben keletkező hibákat lehet elkapni a `catch` blokkokkal

A `finally` rész mindig lefut függetlenül a hibáktól

KIVÉTELKEZELÉS

```
a = {};  
try {  
    a.nincsIlyenFuggvenye();  
    document.write("Nem jut ide");  
}  
catch (err) { console.log("Hiba: "+err.message); }  
finally {  
    document.write("vége");  
}
```



KIVÉTELKEZELÉS

Hiba dobása

```
throw hiba
```

A hiba lehet szöveg, szám, vagy objektum

```
try {  
    if (x == 0) { throw "Nullával osztás"; }  
    y /= x;  
}  
catch (err) { console.log(err) }
```

KIVÉTELKEZELÉS

Beépített kivétel objektumok

A kivételobjektumoknak két tulajdonságuk van:

hiba neve (`name`)

üzenetszöveg (`message`)

Típusok

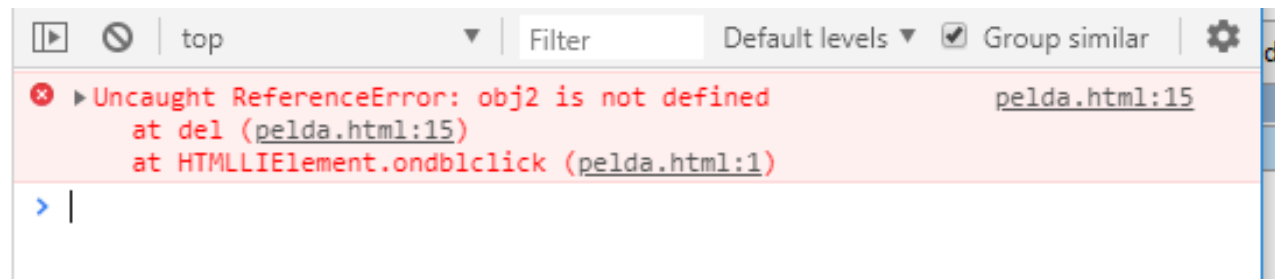
`RangeError`

`ReferenceError`

`SyntaxError`

`TypeError`

`URIError`



KIVÉTELKEZELÉS

```
try {  
    a.nincsIlyenFuggvenye();  
}  
catch (err) {  
    if (err.name == "ReferenceError") {  
        alert("Hibás hivatkozás");  
    }  
    else {  
        alert("Más hiba:" + err.message);  
    }  
};
```

FELADAT

Adott három érték: év, hó, nap. Döntse el, hogy helyes dátum-e a három érték együtt. Ha igen, írja ki, ha nem adjon megfelelő hibaüzenetet! A hibákat kezelje kivételekkel!

FÜGGVÉNYEK

Definíció

```
function függvénynév(paraméterek){ ... }
```

Visszatérési érték megadása

```
return érték
```

Hívása függvénynévvel és paraméterekkel

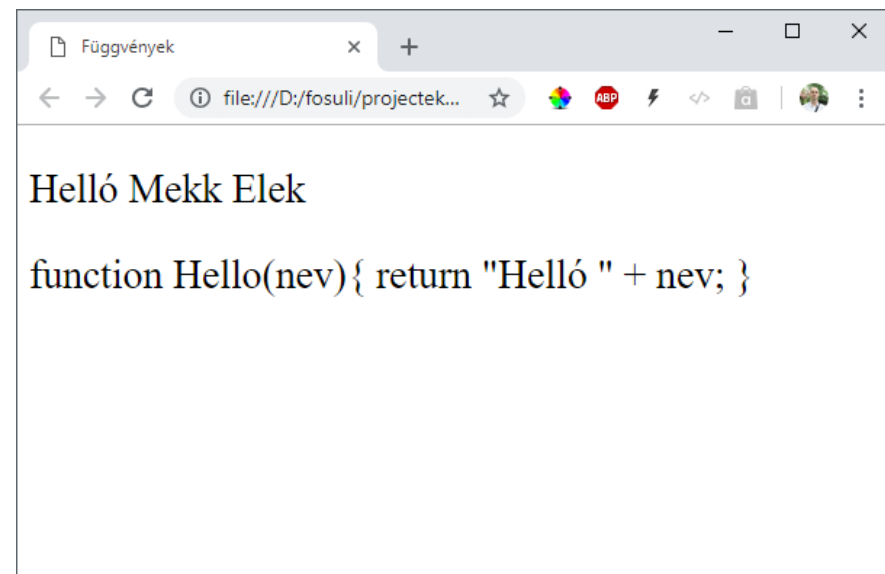
```
függvénynév(paraméterek)
```

Zárójel nélküli híváskor a függvény definícióját adja vissza

FÜGGVÉNYEK

```
function Hello(nev) {  
    return "Helló " + nev;  
}
```

```
// kiírja: Helló Mekk Elek  
document.write(Hello("Mekk Elek"));  
// kiírja a függvény leírását  
document.write(Hello);
```



FÜGGVÉNYEK

Visszatérés

Nem kell megadni a visszatérés típusát

Paraméterek

Érték szerinti paraméterátadás történik

Kivéve objektum átadásakor, azt referenciaként adja át

Ha a függvény módosítja az objektum valamelyik tagját, az az átadott paraméterre is érvényes

FÜGGVÉNYEK

```
function ujev(obj,vki) {  
    obj.ev++;  
    vki = "magamnak"  
    document.write("Boldog újévet "+vki);  
}  
  
var datum = { ev:2018, ho:1, nap:1 };  
var kinek = "Mirr Murr";  
ujev(datum);  
document.write(kinek);           // Mirr Murr  
document.write(datum.ev);       // 2019
```

FÜGGVÉNYEK

Paraméterek

Paraméterek elérhetők az `arguments[]` tömbből is

```
function osszead(a,b) {  
    return arguments[0] + arguments[1];  
}  
document.write(osszead(2,3));
```

FÜGGVÉNYEK

A függvények kezelhetők úgy, mint az adatok

```
function kob(x) {  
    return x*x*x;  
}
```

```
var a = kob(3);    // a értéke 27  
var b = kob;  
var c = b(4);      // c értéke 64
```

FÜGGVÉNYEK

Beépített függvények

`eval(kifejezes)`

Ha a paraméterében megadott szöveg kifejezésként értelmezhető, kiértékeli és az eredményt visszaadja

```
var a = eval("10+20+30");
```

Ha paraméterében megadott szöveg utasítás(ok)ként értelmezhető, akkor azt végrehajtja

```
eval("alert('Helló Világ')");
```

FÜGGVÉNYEK

Beépített függvények

`isNaN(s)`

Hamisat ad vissza, ha a paramétere szám, vagy azzá konvertálható

`isFinite(x)`

Hamisat ad vissza, ha a paramétere nem plusz vagy mínusz végtelen szám és nem NaN

FÜGGVÉNYEK

Beépített függvények

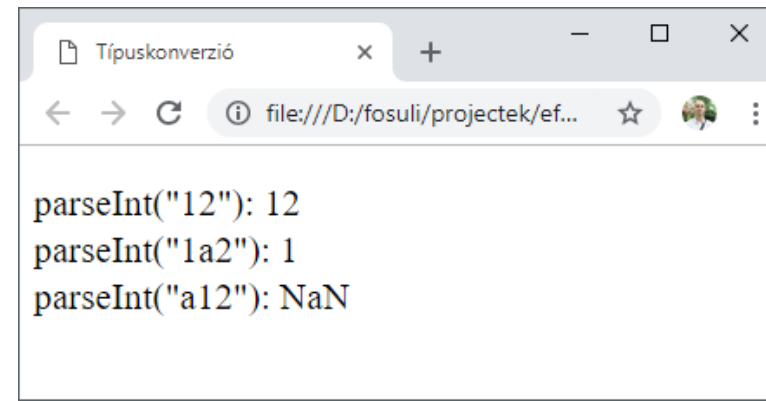
`parseInt(s)`

`parseFloat(s)`

A paraméterében megadott szöveget egész illetve valós számmá konvertálja és azt visszaadja

Balról vizsgálva az első nem szám karakterig konvertál, ha már az első karakter se szám, akkor NaN értékkel tér vissza

```
parseInt("12")      // 12
parseInt("1a2")     // 1
parseInt("a12")     // NaN
```



FÜGGVÉNYEK

Beépített függvények

`Number()`

A paraméterében megadott objektumot konvertálja számmá. Ha nem sikerül, NaN értékkel tér vissza.

```
Number("123"); // 123
```

```
Number("3.14"); // 3.14
```

```
Number(true); // 1
```

```
Numer("1a2"); // NaN
```

A `Number()` különbsége a `parseInt()` függvénytől

- az egész paraméterét egyben konvertálja
- egész és valós típust is előállít

FÜGGVÉNYEK

Beépített függvények

`String()`

A paraméterében megadott objektumot sztringgé konvertálja

```
String(123); // "123"
```

```
String(Boolean(0)) // "false"
```

```
String(new Date());
```

```
// "Sun Sep 23 2018 19:19:47 GMT+0200  
(közép-európai nyári idő)"
```

Valójában ugyanazt adja, mint az objektum `toString()` metódusa

FÜGGVÉNYEK

Beépített függvények

Speciális karakterek kódolása

```
encodeURIComponent ()
```

```
decodeURIComponent ()
```

Az URI-ben használt speciális karaktereken kívül minden más speciális karaktert kódol illetve visszakódol

```
encodeURIComponent("teszt példa.js?név=Ödön&jel=@")
```

eredmény

```
teszt%20p%C3%A9lda.js?n%C3%A9v=%C3%96d%C3%B6n&jel=@
```

FÜGGVÉNYEK

Beépített függvények

Speciális karakterek kódolása

`encodeURIComponent()`

`decodeURIComponent()`

Minden speciális karakter kódol és visszaalakít

```
encodeURIComponent("teszt példa.js?név=Ödön&jel=@")
```

eredmény

```
teszt%20p%C3%A9lda.js%3Fn%C3%A9v%3D%C3%96d%C3%B6n%26jel%  
3D%40
```

HATÓKÖR

Lokális

- ami a függvényben van deklarálva,
- kívülről nem elérhető,
- függvény futásaig él

Globális

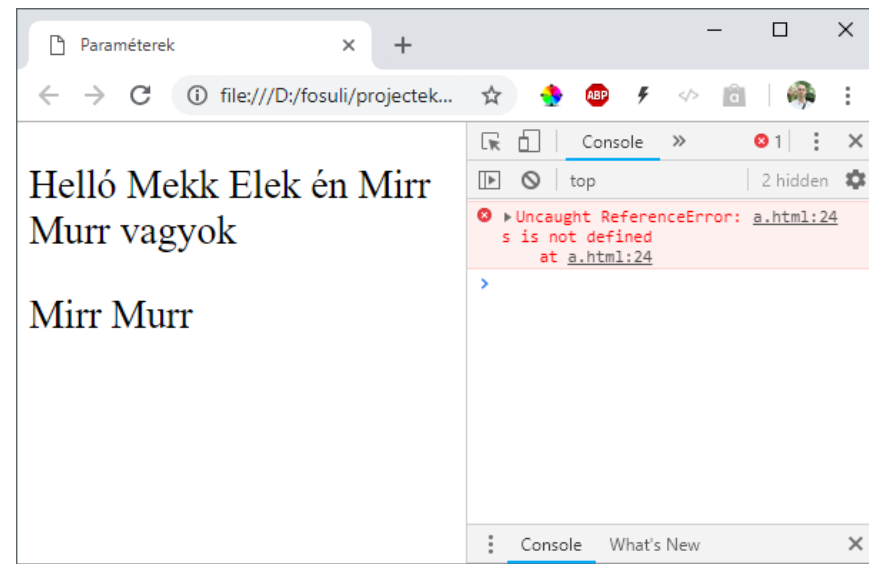
- függvényen kívül deklarált
- bárholnan elérhető
- függvénytől függetlenül él

Deklaráció nélküli változók (`var` nélkül) globális változók lesznek függetlenül a létrehozásuk helyétől

HATÓKÖR

```
function hello(nev) {  
    var s = "";           // lokális  
    en = "Mirr Murr";     // globális  
    s = "Helló "+nev+" én "+en+" vagyok";  
    return s;  
}
```

```
var vki = "Mekk Elek";    // globális  
document.write(hello(vki));  
document.write(en); // Mirr Murr  
document.write(s); // undefined
```



HATÓKÖR

Az ECMAScript 2015 a lokális és globális hatókör mellé bevezette a blokk hatókört

Blokk

Blokk hatókörű változó deklarálása

```
let a = 1;
```

Csak a blokk idejéig él, a blokkon kívül nem elérhető, a blokkon kívüli változókat nem írja felül

Az így létrehozott változó újra deklarálása nem megengedett

A konstans is blokk hatókörű, csak nem módosítható

```
const a = 1;
```

HATÓKÖR

```
function teszt() {  
    var x = 10;           // 10 - lokális  
    {  
        let x = 100;      // 100 - blokk  
    }  
    document.write(x);    // 10 - lokális  
}  
var x = 1;                // 1 - globális  
teszt();  
document.write(x)        // 1 - globális
```


FELADAT

1. Adott egy másodfokú egyenlet három együtthatója. Írjon függvényt, amely kiírja a megoldást!
2. Adott két egész szám. Írjon függvényt, amely megadja a két szám legnagyobb közös osztóját!

OBJEKTUMOK

Létrehozás

Objektum literállal

```
var sz = {  
    vezNev : "Mekk",  
    kerNev : "Elek",  
}
```

Csak ez az egy objektum jön létre ilyen szerkezettel és adattartalommal

OBJEKTUMOK

Metódus

A metódus az objektum egy tulajdonságára definiált függvény

```
var személy = {  
    vezNev : "Mekk",  
    kerNev : "Elek",  
    hello : function() {  
        return "Helló, "+this.vezNev+  
        " "+this.kerNev+" vagyok";  
    }  
}
```

OBJEKTUMOK

Létrehozás

Konstruktorral prototípus definiálható

Ekkor több ugyanolyan szerkezetű objektum hozható létre

```
function Szemely(vn,kn) {  
    this.vezNev = vn;  
    this.kerNev = kn;  
}  
  
var sz1 = new Szemely("Mekk","Elek");  
var sz2 = new Szemely("Oriza","Triznyák");
```

OBJEKTUMOK

Metódus

```
function Szemely(vn, kn) {  
    this.vezNev = vn;  
    this.kerNev = kn;  
    this.hello = function() {  
        return "Helló "+this.vezNev+  
            " "+this.kerNev+" vagyok";  
    }  
}
```

OBJEKTUMOK

Objektum használat

Tulajdonságok és metódusok elérése a névvel

```
objektum.tulajdonság
```

```
objektum["tulajdonság"]
```

Az adattagok kívülről is olvashatóak és írhatóak

```
var m = new Szemely("Mirr", "Murr");
```

```
var nev = m.vezNev+" "+m["kerNev"];
```

```
m.hello();
```

```
m.vezNev = "Oriza";
```

```
m.kerNev = "Triznyák";
```

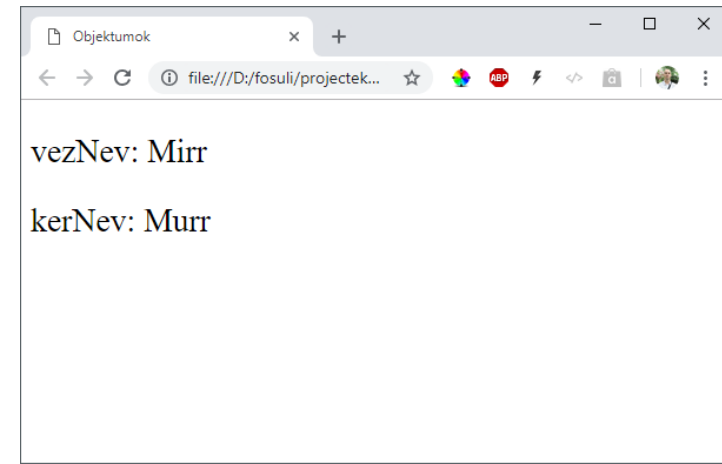
OBJEKTUMOK

Az objektum tulajdonságait végig lehet venni egy `for` ciklussal

```
for (változó in objektum) { ... }
```

A változó sorban a tulajdonságok nevét kapja

```
var m = new Szemely("Mirr", "Murr");  
for (x in m) {  
    document.write(x+": "+m["x"]);  
}
```



Objektumok módosítása

- JavaScript objektum modellje prototípus alapú, azaz az azonos prototípusból létrehozott objektumot eltérhetnek, lehetnek saját adattagjai és metódusai.
- Az objektum bővíthető adattaggal és metódussal úgy, hogy az nem vonatkozik ugyanabból a prototípusból létrehozott többi objektumra
- Bővíthető egy objektum prototípusa is, ilyenkor ez érvényes a több objektumra is, amelyek ebből a prototípusból lettek létrehozva

OBJEKTUMOK

Bővítés metódussal és tulajdonsággal

```
var sz1 = new Szemely("Mekk","Elek");  
var sz2 = new Szemely("Oriza","Triznyák");
```

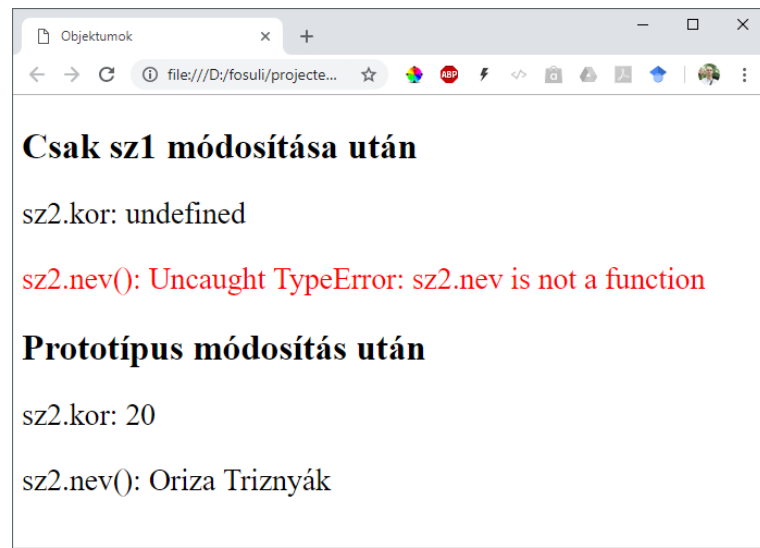
Csak az adott objektumot bővíti

```
sz1.kor = 20;           // új tulajdonság  
sz1.nev = function() { // új metódus  
    return this.vezNev+" "+this.kerNev;  
}  
sz2.kor           // undefined  
sz2.nev()         // undefined
```

OBJEKTUMOK

A prototípus változtatása

```
Szemely.prototype.kor = 20;  
Szemely.prototype.nev = function() {  
    return this.vezNev+" "+this.kerNev;  
}  
  
sz2.kor          // 20  
sz2.nev();       // Oriza Triznyák
```



FELADAT

1. Készítse el a téglalap prototípust! Hozzon létre két téglalapot és írja ki kerületüket és területüket metódusok segítségével!
2. Készítsen egy pont és egy vektor prototípust! A vektort határozza meg a két végpontja (adott sorrendben)! Lehesse a pontot és a vektor is eltolni egy vektorral, illetve a vektort elforgatni egy szöggel!

TÖMBÖK

Tömb létrehozása

A tömb objektum, kétféle módon hozható létre

Elemei megadásával

```
var tomb = [1, 2, 3];
```

Objektumként

```
var tomb = new Array(1,2,3);
```

Mindkettő ugyanazt csinálja, az első mód az ajánlott

Hossz megadásával is lehet

```
var tomb = new Array(10);
```

Ilyenkor az elemek `undefined` értékűek

TÖMBÖK

Elemek

Az elemek bármilyen típusúak lehetnek, akár eltérő típusúak is

```
var tomb = [1, "hétfő", true];
```

A tömb elemeit az indexük segítségével érjük el

Az index számozás 0-val kezdődik

<code>tomb[0]</code>	első elem
----------------------	-----------

Nincs asszociatív tömb, helyette használható az objektum

TÖMBÖK

A tömbhöz új elemet lehet adni az új elem indexére hivatkozva

```
var tomb = [0,1,2];  
tomb[3] = 3;           // [0,1,2,3]
```

Ha túlhivatkozott tömb elemnek adunk értéket, megnöveli a tömböt, de "lyukak" keletkeznek a tömbben és a köztes elemek `undefined` értéket kapnak

```
tomb[5] = 5;           // [0,1,2,3,undefined,4]
```

TÖMBÖK

Tulajdonság

`length` hossza (nincs zárójel utána)

Metódus (néhány fontosabb)

`join()` elemeit összefűzi szeparátorral

`push()` új elemet fűz a tömb végére

`pop()` elemet kivesz a tömb végéről és visszaadja

`shift()` új elemet fűz a tömb elejére

`unshift()` elemet kivesz a tömb elejéről és visszaadja

`concat()` összefűz két tömböt

`splice()` hozzáad és kivesz a tömbből elemet

`sort()` rendezi a tömböt ábécé sorrendbe

TÖMBÖK

```
var t = [1,2,3,4];  
var x = t.shift();  
t.unshift(t.pop());  
t.push(x);  
document.write(t.join("; "));
```

[mit ír ki?]

FELADAT

1. Készítsen két tömböt egész számokból! Fésülje össze úgy őket egy tömbbe, hogy először a párosak, majd a páratlanok szerepeljenek!
2. Készítsen egy $n \times m$ -s tömböt és töltse fel számokkal! Határozza meg a sorok maximumának a minimumát, illetve a sorok minimumának a maximumát! Készítsen minimum és maximum kereső függvényeket!

SZTRINGEK

A sztringek objektumok, kétféle módon hozhatók létre
Primitív típus-szerűen, az értékével

```
var s1 = "Helló Világ"
```

Objektumként

```
var s2 = new String("Helló Világ")
```

Mindkettő ugyanazt csinálja, az első mód az ajánlott

Az első esetben `String` típusú, a másodikban `Object`

```
s1 == s2      igaz
```

```
s1 === s2     hamis
```

SZTRINGEK

Escape karakter

Speciális karakterek használata escape karakter segítségével lehetséges: vissza-per \

Hibás

```
var s = "Azt mondta "helló" és ment"
```

Helyesen

```
var s = "Azt mondta \"helló\" és ment"
```

vagy

```
var s = 'Azt mondta "helló" és ment'
```

SZTRINGEK

Tulajdonság

`length` hossz (nincs zárójel utána)

Metódusok (néhány fontosabb)

`indexOf()`, `lastIndexOf()`, `search()`

 részsztring keresése

`slice()`, `substring()`, `substr()`

 sztring részének lekérése

`toUpperCase()`, `toLowerCase()`

 kis- és nagybetűre konvertálás

`charAt()` adott helyen lévő karakter

`trim()` white-space karakterek levágása

SZTRINGEK

```
var n = "nemecsek ernő";  
var i = n.indexOf(" ");  
n = n.charAt(0).toUpperCase() +  
    n.slice(1,i) + " " +  
    n.charAt(i+1).toUpperCase() +  
    n.slice(i+2);  
document.write(n);
```

[mit ír ki?]

FELADAT

1. Készítsen olyan függvényt, amely egy szövegben (pl fájlnevben) minden szóközt alsóvonásra (_), minden ékezetes karaktert pedig a megfelelő nem ékezetes karakterre cserél!
2. Készítsen olyan függvényt, amely megkeveri egy szövegben szereplő szavakat! Szó a szóköz, illetve írásjelek által határolt szövegrész.

OBJEKTUMOK ÖSSZEHASONLÍTÁSA

Az objektumok összehasonlítása referencia szerint történik,
nem tartalom szerint

```
var t1 = [1,2,3];  
var t2 = [1,2,3];  
t1 == t2      // false
```

```
var o1 = { nev : "Mekk Elek" }  
var o2 = { nev : "Mekk Elek" }  
o1 == o2      // false  
o3 = o1  
o3 == o1      // true
```

OBJEKTUMOK ÖSSZEHASONLÍTÁSA

Sztingek összehasonlítása

Attól függ, hogyan lett létrehozva, ha sztringként

```
var s1 = "példa szöveg";  
var s2 = "példa szöveg";  
s1 == s2      // true
```

De ha objektumként

```
var s3 = new String("példa szöveg");  
var s4 = new String("példa szöveg");  
s3 == s4      // false  
s1 == s3      // true  
s1 === s3     // false
```


BEÉPÍTETT OBJEKTUMOK

Dátum

```
d = new Date();
```

```
d = new Date(év,hó,nap,óra,perc,ezrmp)
```

Néhány metódus

<code>set/getFullYear()</code>	évszám 4 jegyen
<code>set/getMonth()</code>	hónap sorszáma
<code>set/getDate()</code>	nap sorszáma a hónapban
<code>getDay()</code>	nap sorszáma a héten (0-6)
<code>toString()</code>	sztringként a dátum

BEÉPÍTETT OBJEKTUMOK

Math

Beépített matematikai függvények

Nem kell példányosítani!

Néhány függvény

<code>Math.abs()</code>	abszolút érték
<code>Math.round()</code>	kerekítés
<code>Math.pow()</code>	hatványozás
<code>Math.sqrt()</code>	gyökvonás
<code>Math.random()</code>	véletlen szám 0..1 között
<code>Math.log()</code>	logaritmus
<code>Math.sin()</code>	szinus (radiánban a szög)

BEÉPÍTETT OBJEKTUMOK

Number

Globális metódusai

<code>Number()</code>	számmá konvertál
<code>parseInt()</code>	egész számmá konvertál
<code>parseFloat()</code>	valós számmá konvertál

Ha nem tudnak konvertálni NaN értéket adnak

Tulajdonságai

<code>MAX_VALUE</code>	a legnagyobb érték
<code>MIN_VALUE</code>	a legkisebb érték
<code>POSITIVE_INFINITY</code>	pozitív végtelen
<code>NEGATIVE_INFINITY</code>	negatív végtelen

REGULÁRIS KIFEJEZÉSEK

A reguláris kifejezés egy szövegminta, amely szövegekben keresés illetve cseréhez használható

/minta/módosító

Létrehozás

Literálisan

p = /minta/módosító;

Objektumként

p = new RegExp("minta","módosító");

REGULÁRIS KIFEJEZÉSEK

Módosítók

A minta az összehasonlításának módját határozza meg

- i nem kis-nagybetű érzékeny
- g végigkeresi a szövegben, nem csak az első találatig
- m több soros keresés

```
var p = /web/igm;
```

```
var p = new RegExp("web", "igm");
```

REGULÁRIS KIFEJEZÉSEK

Minta

A minta tartalmaz karaktereket és metakaraktereket

`/ [^ 0 - 9] /`

A karakterek magát a karaktert jelenti, a metakarakternek valamilyen eltérő jelentése van

A metakarakterek egyszerű karakterként való használatához escape karakter szükséges (vissza-per)

REGULÁRIS KIFEJEZÉSEK

Keresendő karakterek [] zárójellel

<code>[abc]</code>	bármelyik karakter a []-ben
<code>[0-9]</code>	bármelyik karakter a megadott tartományban
<code>[^abc]</code>	bármilyen, amint nincs a []-ben
<code>[^a-z]</code>	bármilyen, amint nincs a tartományban
<code>/[A-Z]/g</code>	// az összes nagybetű

Alternatívák () zárójellel

`(s1 | s2)` akár *s1* akár *s2*
`/(file|fájl)/i`

A *file* vagy a *fájl* szöveg akár kis- akár nagybetűvel

REGULÁRIS KIFEJEZÉSEK

Metakarakterek

.	Bármely karakter (kivéve újsor)
\w	nem írásjel szó-karakter (betű, szám, _)
\W	ami nem az előző
\d	számjegy
\D	nem számjegy
\s	white-space karakter
\S	nem white-space karakter

REGULÁRIS KIFEJEZÉSEK

Metakarakterek

<code>\n</code>	új sor
<code>\t</code>	tabulátor
<code>\f</code>	lapdobás
<code>\r</code>	kocsi vissza
<code>\v</code>	függőleges tabulátor
<code>\0</code>	NUL karakter
<code>\unnnn</code>	Unicode karakter az <i>nnnn</i> hexa kóddal
<code>\xnn</code>	latin karakter <i>nn</i> hexa kóddal

REGULÁRIS KIFEJEZÉSEK

Ismétlések

$s\{n\}$	pontosan n -szer s
$s\{n, \}$	legalább n -szer s
$s\{n, m\}$	legalább n -szer, de legfeljebb m -szer s
$s?$	legfeljebb egyszer s , ua. $\{0, 1\}$
s^+	legalább egyszer s , ua. $\{1, \}$
s^*	akármennyiszer s , ua. $\{0, \}$

REGULÁRIS KIFEJEZÉSEK

Példák

```
/\d{2,3}/
```

két vagy háromjegyű szám

```
/\w{3}\d?/
```

három szókarakter és esetleg egy számjegy utána

```
/\s+JavaScript\s+/
```

A JavaScript önálló szó, azaz előtte utána white-space karakterek (de a szöveg elején és végén nem találja meg)

REGULÁRIS KIFEJEZÉSEK

Illeszkedés helye

$s\$$	szöveg végén s
s	szöveg elején s
$\backslash b$	szóhatár
$\backslash B$	nem szóhatár
$(?=s)$	amit az s követi
$(?!s)$	amit nem s követ

REGULÁRIS KIFEJEZÉSEK

Példák

```
/\s+JavaScript\s+/
```

A JavaScript önálló szó, de csak a szövegen belül

```
/\bJavaScript\b/
```

A JavaScript önálló szó, akár a szöveg elején és végén

```
/\B[Ss]cript\b/
```

Olyan szavak, amelyek vége script vagy Script, de nem ezzel kezdődik a szó, pl. JavaScript

REGULÁRIS KIFEJEZÉSEK

RegExp metódusok

`test()`

Igazat ad, ha megtalálja a mintát, hamisat különben

```
var s = "Helló Világ!";  
var minta = /világ/i;  
document.write(minta.test(s));    // true
```

REGULÁRIS KIFEJEZÉSEK

RegExp metódusok

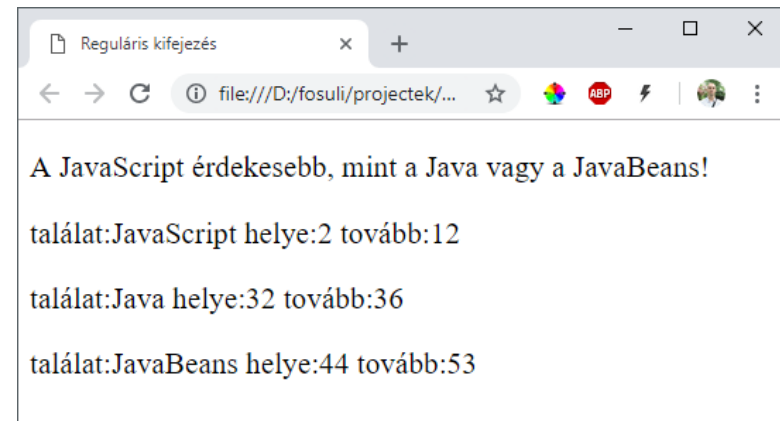
`exec()`

Ellenőrzi a mintát és visszaadja a találatot egy tömbben, ahol az első elem a találat, a második a találat helye és a harmadik az input. Ha nincs találat, null értéket ad vissza.

A *g* módosító esetén sztringnek a *RegExp* objektum *lastIndex* tulajdonságában tárolt pozícióban kezdi a keresést. Találat esetén azt visszaadja és utána állítja a *lastIndex* értékét. Ha nincs több találat, null értéket ad vissza és 0 értékre állítja a *lastIndex* tulajdonságot.

REGULÁRIS KIFEJEZÉSEK

```
var m = /\bJava\b*\b/g;  
var s = "A JavaScript érdekesebb, mint a  
        Java vagy a JavaBeans!";  
var res;  
while (res = m.exec(s)) {  
    document.write("találat:" + res[0]);  
    document.write("helye:" + res.index);  
    document.write("tovább:" + m.lastIndex);  
}
```



REGULÁRIS KIFEJEZÉSEK

String metódusok

`match()`

Megkeresi a mintát a szövegben és visszaadja az első találatot, *g* módosító esetén az összes találatot visszaadja tömbként. Ha nincs találat, *null* értéket ad vissza.

```
var s = "Helló Világ";  
document.write(s.match(/világ/i)); // true
```

REGULÁRIS KIFEJEZÉSEK

String metódusok

`search()`

Megkeresi a mintát és visszaadja az első találat helyét. Ha nincs találat, -1 értéket ad vissza

Figyelman kívül hagyja a *g* módosítót

```
var s = "Helló Világ!";  
var minta = /világ/i;  
document.write(s.search(m));           // 6
```

REGULÁRIS KIFEJEZÉSEK

String metódusok

`replace()`

Megkeresi a mintát a szövegben és kicseréli az adott szövegre, majd visszaadja az új szöveget.

A behelyettesítendő szöveg helyén függvény is lehet, ilyenkor a függvény eredményével cserél

```
var s = "A kedvenc színem a barna";  
var s2 = s.replace(/barna/gi,"kék");
```

REGULÁRIS KIFEJEZÉSEK

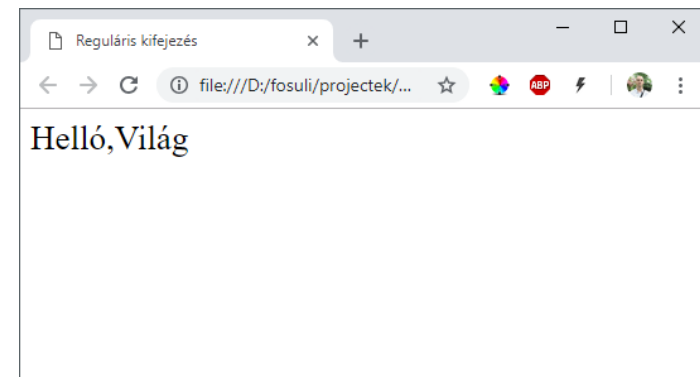
String metódusok

`split()`

A szöveget feldarabolja a megadott minta szerint és visszaadja a részeket egy tömbben.

Ha a mintát nem találja, az eredeti szöveget adja vissza.

```
var s = "Helló Világ";  
var s2 = s.split(/\s/);  
document.write(s2.join()); // Helló,Világ
```



FELADAT

1. Készítsen függvényt, amely összead egy szövegben található kétjegyű számokat és visszaadja az összeget!

FELADAT

2. Készítsen olyan függvényt, amely eldönti egy szövegről, hogy megfelelő e-mail cím-e! Azaz szerepel benne pontosan egy darab @ és a második részében legalább egy pont, nem azzal kezdődik és nem azzal végződik.

[valami@itt.ott](#)

helyes

valami

nem helyes

@itt.ott

nem helyes

valami@itt

nem helyes

valami@itt.

nem helyes

FELADAT

3. Készítsen függvényt, amely eldönti egy szövegről, hogy az megfelelő telefonszám formátum-e! Azaz csak számokat és elválasztó jeleket tartalmaz, valamint + jellel kezdődik, országkód, körzet és a telefonszám része van, közöttük / vagy – jel lehet. Az ország kód két jegyű, a körzet egy- vagy kétjegyű, a szám pedig hat- vagy hétjegyű.

+36/70/1234567

+36-1-123456

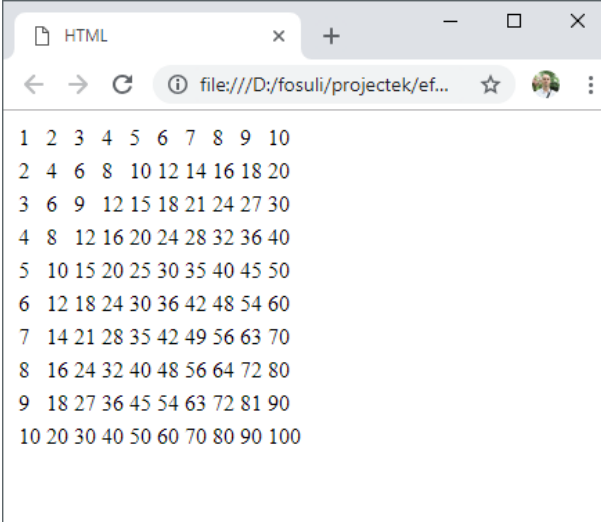
HTML ÉPÍTÉS

- JavaScript segítségével lehet a HTML tartalmat építeni
- A `document.write()` a HTML kódba írja a paraméterét, így lehet dinamikusan HTML kódot generálni szkriptből.
- A `document.write()` csak az oldal felépítésekor használható. A betöltődés után meghíváskor felülírja a dokumentum teljes tartalmát a paraméterében megadott szöveggel, ezáltal törölve az egész dokumentumot

HTML ÉPÍTÉS

Táblázat generálása

```
document.write("<table>");  
for (i=1;i<=10;i++){  
    document.write("<tr>");  
    for (j=1;j<=10;j++){  
  
        document.write("<td>" + (i*j) + "</td>");  
    }  
    document.write("<tr>");  
}  
document.write("</table>");
```

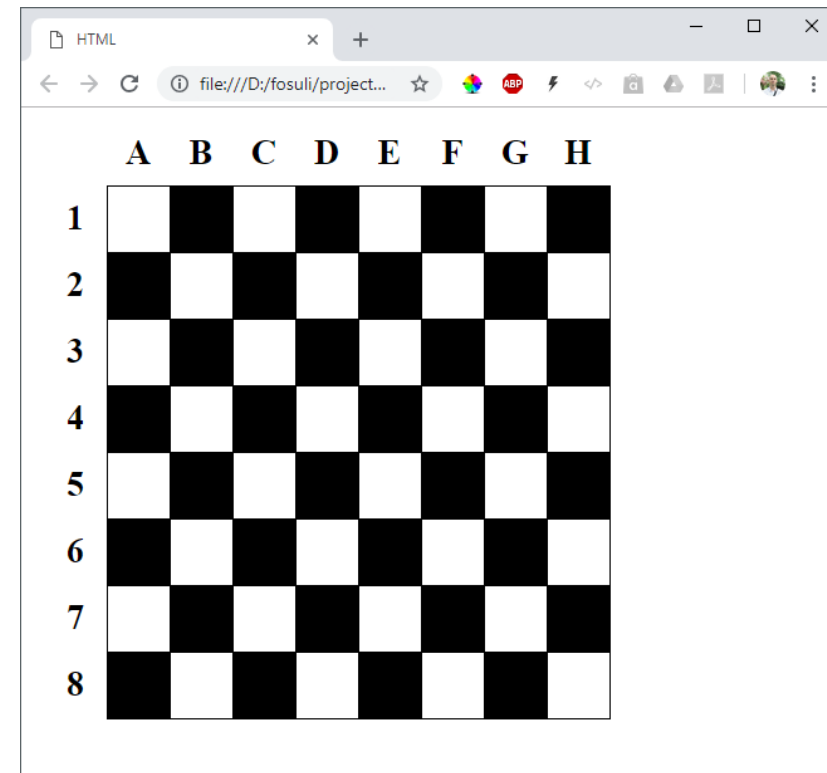


A screenshot of a web browser window showing the result of the JavaScript code. The browser's address bar shows a local file path. The page content is a 10x10 grid of numbers representing the product of row and column indices.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

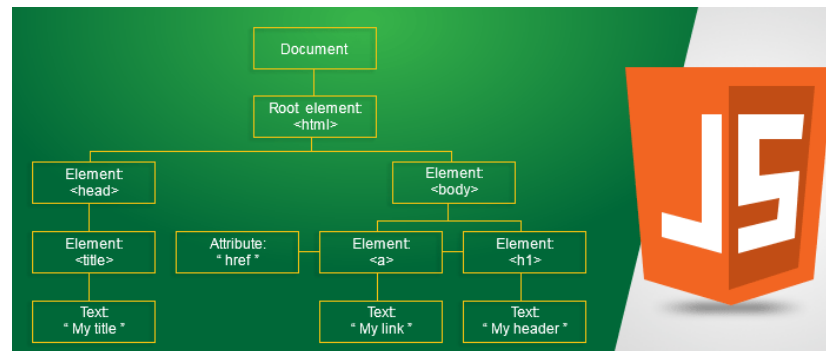
FELADAT

Generáljon egy sakktáblát! Az oszlopok fölött nagybetűk, a sorok mellett balról számok legyenek (keret nélkül). A bal alsó mező fekete.



DOM

- A **DOM** (Document Object Model) egy objektummodell többek között a HTML formátumához az elemek kezelésére. Leírja a HTML elemek és tartalmak szerkezetét.
- A böngésző ez alapján építi fel a forráskód által megadott HTML oldal szerkezetét.
- Az elemeket egy fa-szerkezetbe építi, ahol a csomópontok az oldal elemei, amelyek között alá és fölérendeltségi viszont definiál (szülő és gyerek)



DOM

Három része

Core DOM

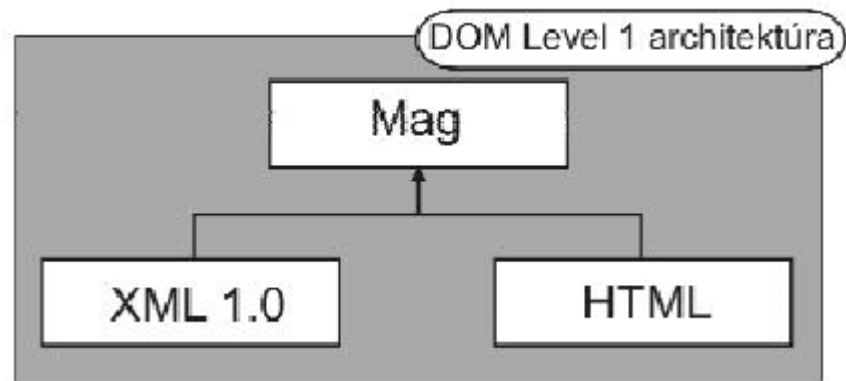
minden dokumentumtípushoz

XML DOM

XML dokumentumhoz

HTML DOM

HTML dokumentumhoz



DOM

DOM szintjei

DOM 0 (Legacy DOM)

- Korlátozott hozzáférés
- Nem szabványos
- Minden böngésző implementálja

Egyszerűbb feladatokra ma is használjuk

DOM átmeneti (Intermediate DOM)

- Nem szabványos
- Böngészők (Netscape, Explorer) eltérően implementálták
- Nem használjuk

DOM Szintjei

DOM 1

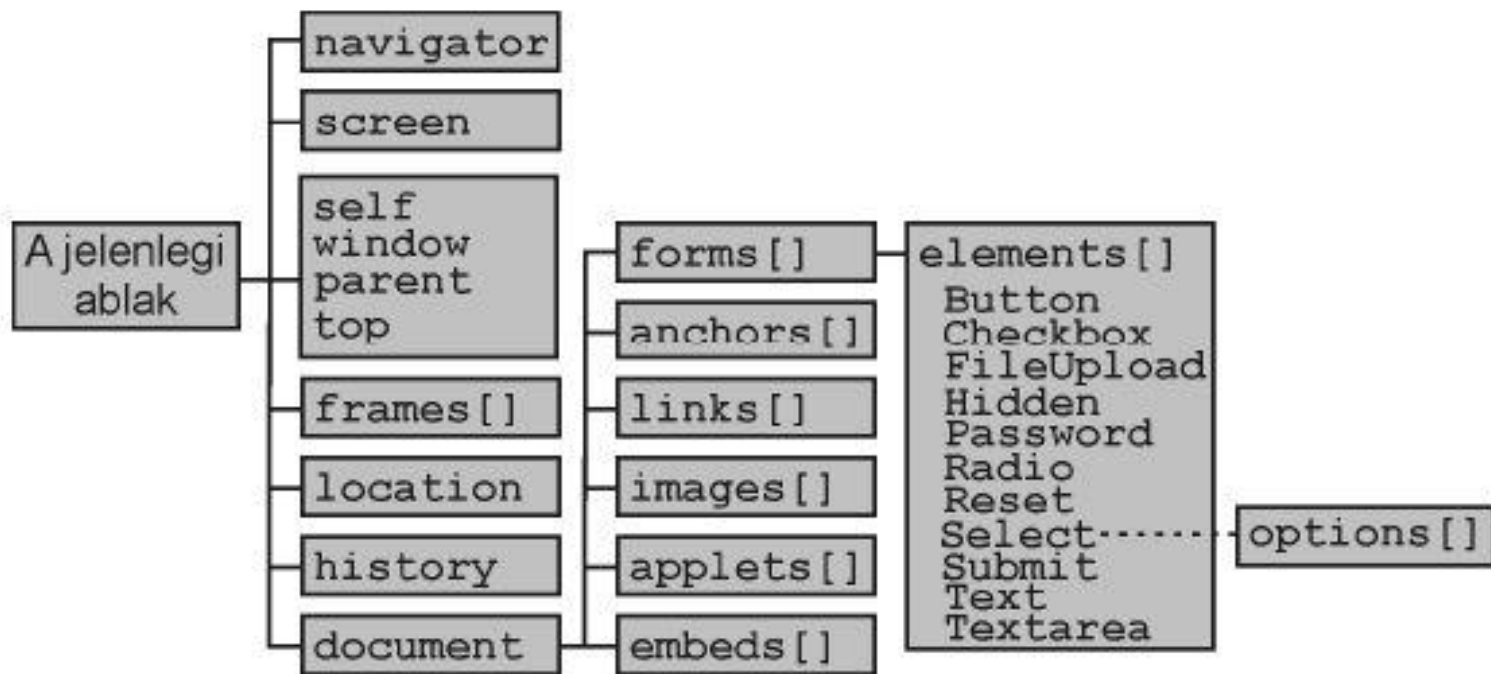
- Minden elemet hozzáférhetővé tesz
- W3C szabvány
- Jelenlegi legismertebb böngészők támogatják
- Jelenleg ezt használjuk

DOM 2, DOM 3

- DOM 1-t kibővítik
- W3C szabvány
- Jelenleg még korlátozott a támogatásuk

BÖNGÉSZŐ OBJEKTUMAI

A böngésző hozzáférhető objektumai



DOCUMENT OBJEKTUM

A `document` objektum felelős a weboldalért, ez a tulajdonosa az összes elemnek az oldalon.

Az elemek hozzáférhetőek

- közvetlenül
- azonosítójuk alapján
- nevük alapján
- tag típusuk alapján
- CSS osztályuk szerint
- CSS szelektor szerint

DOCUMENT OBJEKTUM

Közvetlenül elérhető elemek

- `document.anchors`
- `document.body`
- `document.documentElement`
- `document.embeds`
- `document.forms`
- `document.head`
- `document.images`
- `document.links`
- `document.scripts`
- `document.title`

DOCUMENT OBJEKTUM

Például

Az oldal címe (`title` tag tartalma)

```
document.title
```

Az oldalon található képek száma

```
document.images.length
```

Az oldalon található első űrlap `action` tulajdonságának értéke

```
document.forms[0].action
```

Az oldalon található első link hivatkozási címe

```
document.links[0].href
```

DOCUMENT ELEMEI

Elemek azonosító szerinti elérése

`document.getElementById(azonosító)`

Az elemek azonosítója az `id` tulajdonsággal adható meg
Egy oldalon az azonosítónak egyedinek kell lenni, így
legfeljebb egy elemet ad vissza

```
<h1 id="cim">Az oldal főcíme</h1>
```

```
document.getElementById("cim")
```

DOCUMENT ELEMEI

Elemek elérése név szerint

`document.getElementsByName(név)`

Az elemek neve a `name` tulajdonsággal adható meg

Egy oldalon több azonos nevű elem is lehet, így ezeket tömbként adja vissza

```
<input type="radio" name="re">Igen
```

```
<input type="radio" name="re">Nem
```

Ha ki van választva az első rádiógomb

```
var r = document.getElementsByName("re");  
if (r[0].checked) { ... }
```

DOCUMENT ELEMEI

Elemek elérése tag típus szerint

`document.getElementsByTagName (tagnév)`

Visszaadja az oldalon található összes adott típusú elemet egy tömbben

```

```

```

```

```

```

Az utolsó kép forrásállománya

```
var k = document.getElementsByTagName ( "img" );  
k[k.length-1].src
```

DOCUMENT ELEMEI

Elemek elérése CSS osztály szerint

`document.getElementsByClassName(osztálynév)`

Egy elem CSS osztályát a `class` tulajdonságával lehet megadni

Visszaadja az oldalon található összes olyan elemet egy tömbben, amelyhez hozzá van rendelve az CSS osztály

```
<span class="fontos">egyik hír</span>
```

```
<span class="fontos">másik hír</span>
```

Az első *fontos* osztályú elem tartalma

```
var f = document.getElementsByClassName("fontos");  
f[0].innerHTML
```

DOCUMENT ELEMEI

Elemek elérése CSS szelektor (kijelölő) szerint

`document.querySelectorAll(szelektor)`

Visszaadja az oldalon található összes olyan elemet egy tömbben, amelyet kijelöl az adott szelektor

<code><p class="bibl"></p></code>	<code>// kiválasztott</code>
<code><p></p></code>	<code>// nem kiválasztott</code>
<code><p class="bibl"></p></code>	<code>// kiválasztott</code>

A kiválasztott elemek száma

```
var b = document.querySelectorAll("p.bibl");  
b.length
```

DOCUMENT ELEMEI

Elemek manipulálása

Elem tartalmának módosítása

`innerHTML`

Az elem nyitó és záró tag-je közötti rész írható és olvasható ezzel

Csak olyan elemnél használható, amelynek van záró tag-je

```
<p id="hello"></p>
```

```
var p = document.getElementById("hello");  
p.innerHTML = "Helló Világ";
```


DOCUMENT ELEMEI

Elemek manipulálása

Egy elem valamilyen tulajdonságának módosítása

objektum.tulajdonság

```

```

```
var k = document.getElementById("kep");  
k.src = "pic.png";
```

DOCUMENT ELEMEI

Elemek manipulálása

Egy elem valamilyen stílustulajdonságának módosítása

objektum.style.tulajdonság

```
<p id="nev">Mekk Elek</p>
```

```
var n = document.getElementById("nev");  
n.style.color = "red";
```

FELADAT

Készítsen egy űrlapot! Készítsen egy függvényt, amely kiírja az űrlap elemek tartalmát: a nemüres szövegmezők tartalmát, a bekapcsolt kapcsolók értékét, a listából kiválasztott elemet, stb. Ezután ürítse ki az űrlapot: törölje a szövegmezőket, kapcsolja ki a kapcsolókat, stb. Ez nem reset, nem alaphelyzetre kell állítani, hanem törölni!

WINDOW OBJEKTUM

A BOM (Browser Object Model) modell lehetővé teszi a böngésző ablakkal való kommunikációt

A BOM definiálja a kommunikáció módját, bár nincs hivatalos szabvány rá

Az összes globális objektum a `window` objektum tagja

Ez az objektum az alapértelmezett, ezért nem kötelező kiírni

```
window.document.body
```

Ugyanaz, mint

```
document.body
```

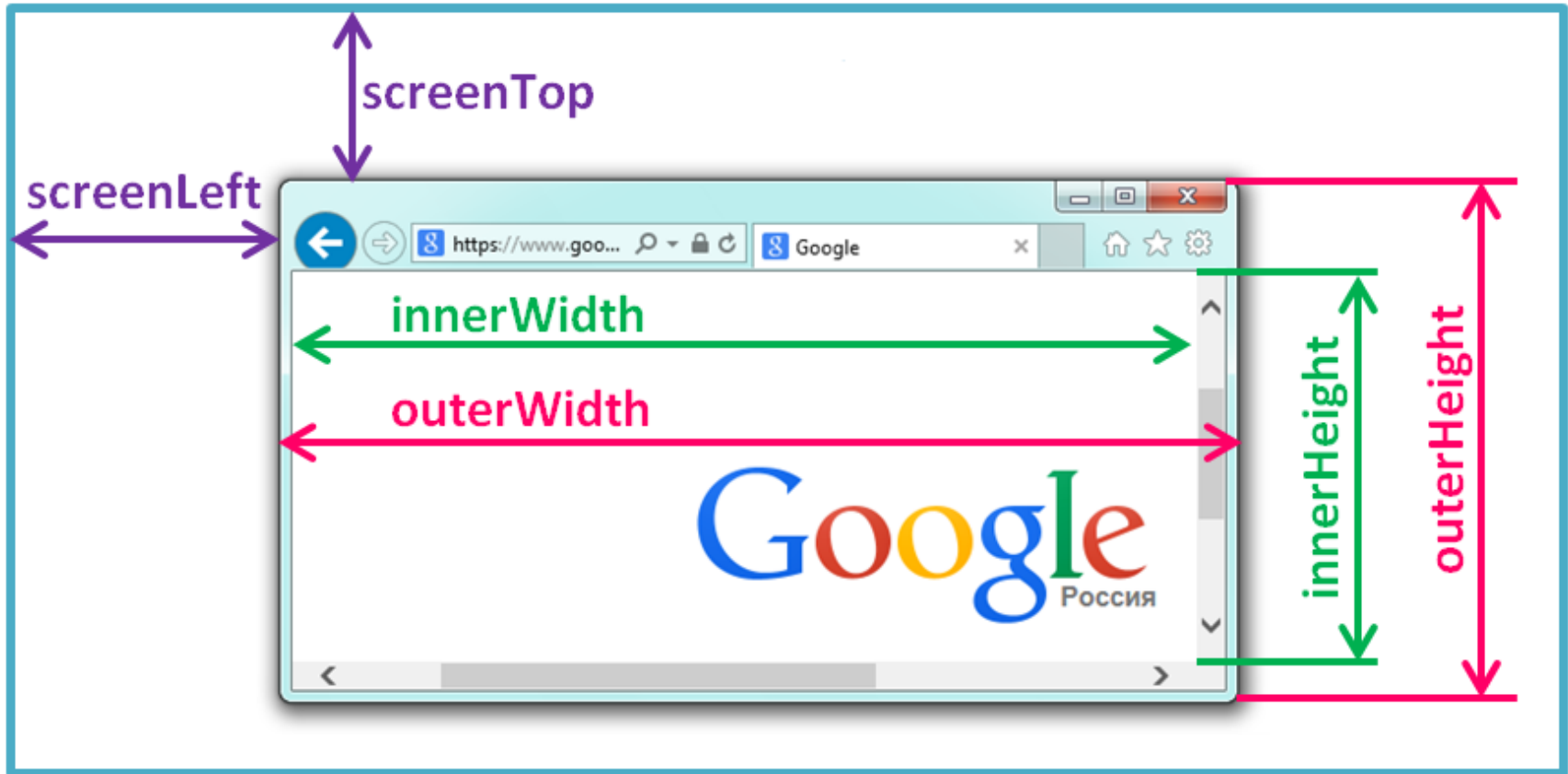
WINDOW OBJEKTUM

window néhány tulajdonsága

<code>opener</code>	a nyitó ablak referenciája
<code>status</code>	a státuszsor szövege
<code>outerWidth</code>	az ablak külső szélessége
<code>outerHeight</code>	az ablak külső magassága
<code>innerWidth</code>	az ablak belső szélessége
<code>innerHeight</code>	az ablak belső magassága

Pixelben mérve, a külső méretben benne van az eszköztár és a görgetősáv, a belső méretben nincs

WINDOW OBJEKTUM



WINDOW OBJEKTUM

window néhány metódusa

<code>open()</code>	új ablakot nyit
<code>close()</code>	bezárja az ablakot
<code>focus()</code>	fókuszot ad az ablaknak
<code>print()</code>	kinyomtatja az ablak tartalmát
<code>moveTo()</code>	mozgatja az ablakot
<code>resizeTo()</code>	méretezi az ablakot
<code>scrollBy()</code>	görgeti az ablakot valamennyivel
<code>scrollTo()</code>	görgeti az ablakot valamelyik pontra

SCREEN OBJEKTUM

A `window.screen` objektum a kliens képernyőjére vonatkozik

Néhány tulajdonsága

<code>width</code>	szélesség pixelben
<code>height</code>	magasság pixelben
<code>colorDepth</code>	színmélység bitben

LOCATION OBJEKTUM

A `window.location` objektum az aktuális oldal címét kezeli

Tulajdonságai

<code>href</code>	aktuális oldal címe
<code>hostname</code>	szerver címe (URL)
<code>pathname</code>	weboldal elérési útja és fájlneve
<code>protocol</code>	protokoll
<code>port</code>	port (ha a default, 0-t vagy semmit nem ad)

LOCATION OBJEKTUM

Property	Description
<code>location.href</code>	<code>'http://www.example.com:8080/tools/display.php?section=435#list'</code>
<code>location.protocol</code>	<code>'http:'</code>
<code>location.host</code>	<code>'www.example.com:8080'</code>
<code>location.hostname</code>	<code>'www.example.com'</code>
<code>location.port</code>	<code>'8080'</code>
<code>location.pathname</code>	<code>'/tools/display.php'</code>
<code>location.search</code>	<code>'?section=435'</code>
<code>location.hash</code>	<code>'#list'</code>

LOCATION OBJEKTUM

Metódusai

<code>reload()</code>	újrátölti az oldalt
<code>assign()</code>	a megadott címre irányít
<code>replace()</code>	az aktuális dokumentum helyére betölti a megadott dokumentumot

A különbség az `assign` és a `replace` között, hogy a `replace` kitörli az aktuális címet a `history`-ből, így a `back` nem tud visszatérni rá

HISTORY OBJEKTUM

A `window.history` objektum felelős a böngészési előzmények kezeléséért

Metódusai

`back()`

betölti az előzőleg látogatott oldalt

`forward()`

betölti az következőleg látogatott oldalt

POPUP ABLAKOK

A `window` objektumhoz három fajta felugró párbeszéd ablak tartozik

Figyelmeztető ablak

```
alert(szöveg)
```

A megadott szöveget kiírja

A felhasználó csak elfogadhatja

```
alert("Helló világ")
```

Az oldal egyik beágyazott oldalának közlendője

Helló Világ

OK

POPUP ABLAKOK

Beleegyezést kérő ablak

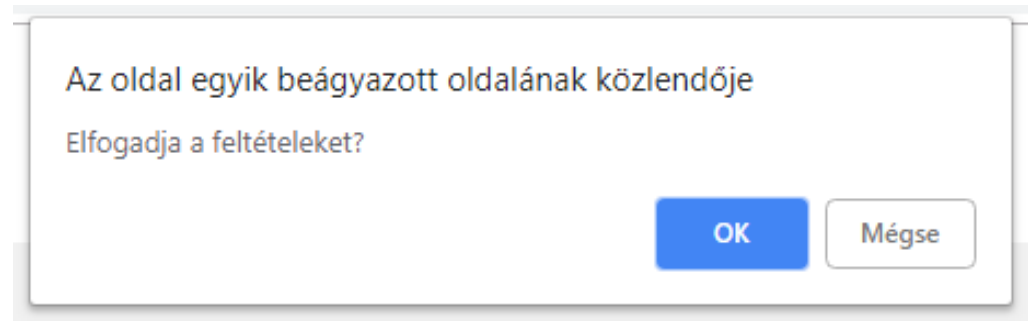
```
confirm(szöveg)
```

Kiírja a megadott szöveget

A felhasználó két lehetőség közül választhat

Igazat ad vissza az OK gomb vagy Enter billentyűre,
hamisat a Cancel gombra vagy Esc billentyűre

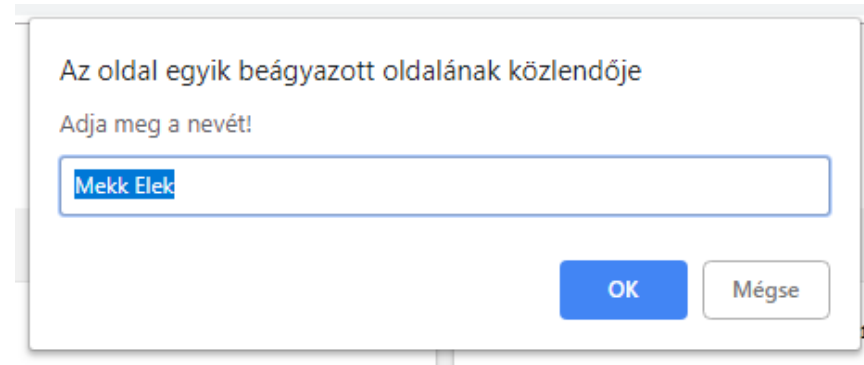
```
confirm("Elfogadja a feltételeket?")
```



POPUP ABLAKOK

Adatbekérő ablak

`prompt (szöveg)`



Kiírja a megadott szöveget

Tartalmaz egy szövegmezőt, amibe a felhasználó beírhat

Visszaadja a felhasználó által megadott értéket az OK gomb vagy Enter hatására, Mégse gomb vagy Esc hatására `null`-t ad vissza

Második paraméterben megadható a beviteli mező alapértéke

```
prompt("Adja meg a nevét", "Mekk Elek")
```

POPUP ABLAKOK

```
var nev = "";
do {
    nev = prompt("Adja meg a nevét!");
    if (nev == null) {
        if (!confirm("Megadja a nevét?")) {
            break;
        }
    }
} while (nev == "" || nev == null);
if (nev != null) {
    alert("Helló "+nev);
}
```


ÚJ ABLAK

Új böngésző ablakot lehet nyitni a `window.open()` metódussal

`window.open(url, név, tulajdonságok)`

url az ablakba nyitandó dokumentumot

név az ablak neve

tulajdonságok néhány tulajdonság

- `height, width` méret
- `fullscreen` teljes méretű-e
- `resizable` átméretezhető-e
- `top, left` elhelyezkedése
- `menubar` legyen-e menüsora

ÚJ ABLAK

Egy ablakoz lehet egy output streamet nyitni, amelybe a `write` metódussal írni lehet.

```
document.open()
```

Az output után a streamet le kell zárni

```
document.close()
```

Ha már létezett az ablakban output, akkor azt felülírja!

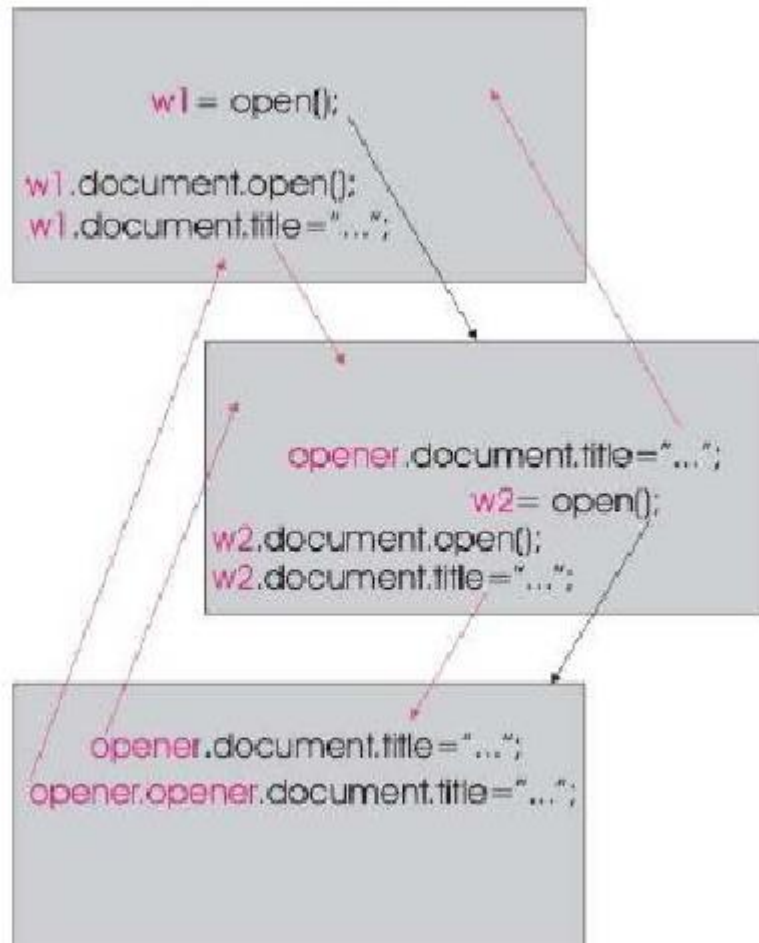
ÚJ ABLAK

Új ablak nyitása és abba írás

```
var ujAblak = window.open();  
ujAblak.document.open();  
var tartalom = "<p>Helló Világ</p>";  
ujAblak.document.write(tartalom);  
ujAblak.document.close();
```

ÚJ ABLAK

Hivatkozás az ablakok között



ÚJ ABLAK

egyik.html

```
var ujAblak =  
window.open("masik.html", "ujablak",  
            "width=200,height=200,menubar=no");
```

masik.html

```
var szuloAblak = window.opener;  
document.write(szuloAblak.location.href);
```

FELADAT

Készítsen függvényt, amely az oldalon található összes linket megnyit egy új ablakban!

IDŐZÍTÉS

A `window` objektum metódusaival két módon lehet időzítve végrehajtani utasítást

Adott idő eltelte után egyszer

```
setTimeout(függvény, idő)
```

Az függvényhívást végrehajtja a megadott idő letelte után, az idő ezredmásodpercben van megadva

Visszaadja az időzítő objektumot

Az idő eltelte előtt megszakítható az időzítés

```
clearTimeout(időzítő)
```

A paramétere az időzítő objektum, amit a `setTimeout()` adott vissza indításkor

IDŐZÍTÉS

```
function hello() {  
    alert('Helló');  
}
```

```
<button onclick="t=setTimeout(hello, 10000)">  
    Üdvözöl
```

```
</button>
```

```
<button onclick="clearTimeout(t)">  
    Inkább mégse
```

```
</button>
```


IDŐZÍTÉS

Ismétlődve adott időközönként

`setInterval(függvény, időköz)`

Az adott időközönként meghívja a megadott függvényt

Visszaadja az időzítő objektumot

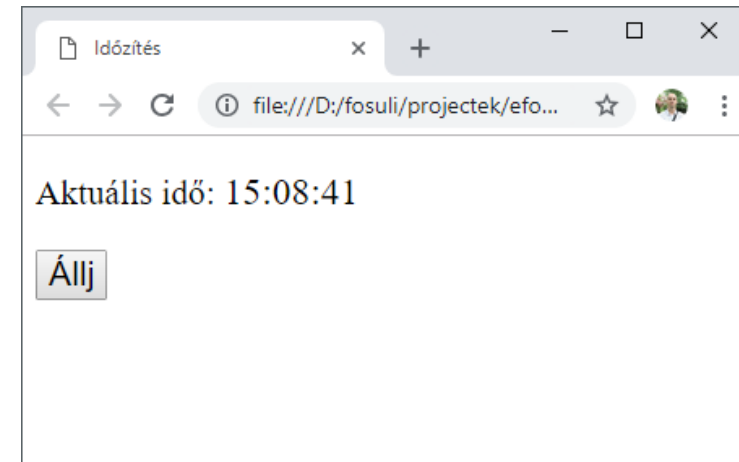
A végrehajtás megszakítható

`clearInterval(időzítő)`

IDŐZÍTÉS

```
<p id="ido"></p>  
<button onclick="clearInterval(t)">Állj</button>
```

```
function aktIdo() {  
    var d = new Date();  
    document.getElementById("ido").innerHTML =  
        d.toLocaleTimeString();  
}  
var t = setInterval(aktIdo(), 1000);
```



FELADAT

1. Készítsen olyan szkriptet, amely megnyit egy új ablakot, beleír egy szöveget, majd 5 másodperc múlva bezárja!
2. Készítsen olyan függvényt, amely az oldal címsorának fejlécét adott időközönként átszínezi!
3. Készítsen olyan függvényt, amely az oldal státusz sorában egy úszó szöveget jelenít meg (beúszik balról, ki jobbról, majd be újra balról, stb.)

ESEMÉNYEK

- A HTML elemekhez események rendelhetők, amelyek bekövetkezésekor valamilyen műveletet végezhet
- Az eseményt a HTML tag-ben kell definiálni és megadni, hogy mit végezzen az eseménykor
- A művelet lehet egyetlen utasítás is, de jellemzően függvényhívás

```
<p onclick="window.close()">Bezár</p>
```

```

```

ESEMÉNYEK

Definiálható fontosabb események

<code>blur</code>	elveszti a fókuszt
<code>change</code>	megváltozik a tartalma
<code>click</code>	egér klikk
<code>dblclick</code>	dupla klikk
<code>drag</code>	drag közben
<code>dragstart</code>	drag kezdetekor
<code>dragend</code>	drag végén
<code>drop</code>	a drag elem elengedésekor

ESEMÉNYEK

Definiálható fontosabb események

<code>focus</code>	amikor fókuszban van
<code>focusin</code>	fókuszt kap
<code>focusout</code>	elveszti a fókuszt
<code>keypress</code>	billentyű lenyomáskor
<code>keydown</code>	billentyű lenyomása
<code>keyup</code>	billentyű felengedése
<code>load</code>	betöltődéskor

ESEMÉNYEK

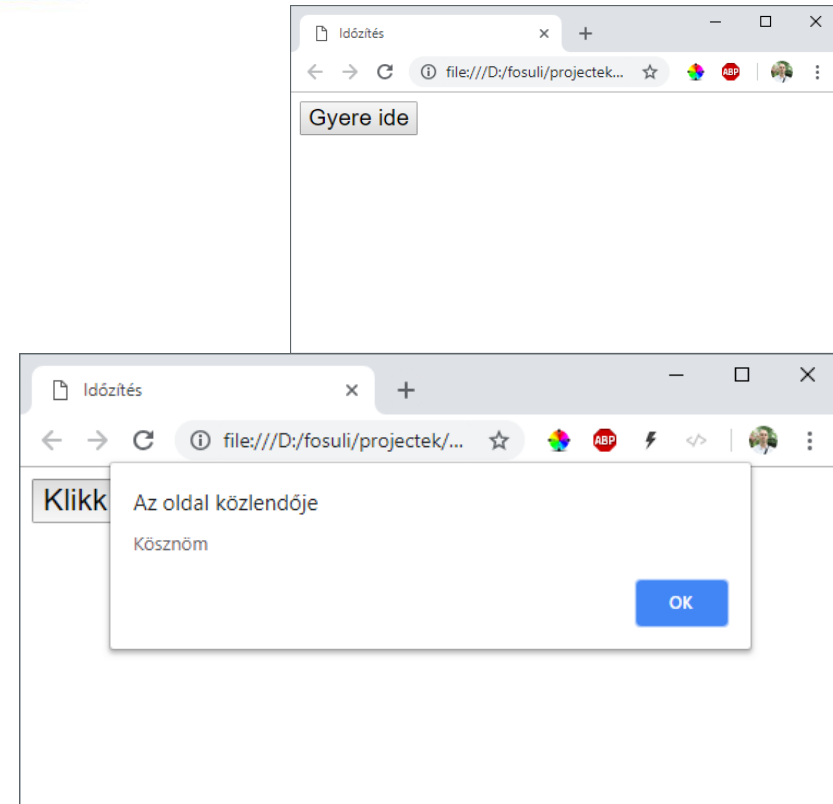
Definiálható fontosabb események

<code>mouseover</code>	egérkurzor fölötte
<code>mouseout</code>	egérkurzor lekerül róla
<code>mousedown</code>	egérgomb lenyomása
<code>mouseup</code>	egérgomb felengedése
<code>resize</code>	a dokumentum átméretezése
<code>scroll</code>	elem görgetése
<code>select</code>	a tartalmából valamit kijelölnek
<code>submit</code>	űrlap elküldése
<code>unload</code>	oldal elhagyása
<code>wheel</code>	egér görgetése

ESEMÉNYEK

```
function hello() {  
    alert("hello");  
}  
function vált(obj,s) {  
    obj.innerHTML = s;  
}  
function kosz() {  
    alert("Kösznöm");  
}
```

```
<button onmouseover="vált(this,'Klikk ide')"  
        onmouseout="vált(this,'Gyere ide')"  
        onclick="kosz()">  
Gyere ide</button>
```



FELADAT

1. Készítsen egy pizza rendelő űrlapot a szokásos adatokkal, változatos űrlapelemekkel! Készítsen olyan szkriptet, amely megjeleníti egy szöveges területen a rendelés adatait egy szövegbe összefűzve és bármely adat módosításakor frissíti a szöveget is!

FELADAT

2. Készítsen betűk egy listáját illetve egy szövegterületet!
Készítsen olyan szkriptet, amely segítségével ha a felhasználó valamelyik betűre klikkel a listában, azt hozzáfűzi a szöveghez. Lehesse a szöveg végéről törölni egy karaktert, illetve az egész szöveget!

FELADAT

3. Készítsen egy memória játékot képekkel! Minden kép kétszer szerepel. Legyenek lefordítva a képek indulásként! Jelölje, hogy melyik fölött van az egérkurzor (például keretvonallal)! Amelyikre klikkel a játékos, azt felfordítja, de csak párosával. Ha már kettőt felfordított, akkor a következő klikkelésre azokat visszafordítja. Amikor egy párt fordít fel egyszerre, azokat már ne fordítsa vissza! A játéknak vége, ha minden kép felfordítva van.

DOM FELÉPÍTÉS

A DOM az oldal elemeiből egy fa struktúrát épít fel

A fa csúcsai a dokumentum alkotóelemei

A DOM a csúcsok tulajdonságait és funkcionalitását specifikálja

A feldolgozást a fa struktúrán haladva végzi

A csúcsok lehetnek

- HTML elem
- Statikus szöveg
- Komment
- Attribútum – törölték, de még működik

DOM FELÉPÍTÉS

A fában lévő összes csúcs elérhető

Új csúcsot lehet létrehozni és befűzni a fába

A csúcsokat lehet módosítani

Meglévő csúcsot lehet törölni

A csúcsok viszonya lehet:

- Szülő amelyikből származik, csak egy van
- Gyerek belőle származó csúcs
- Testvér azonos szülővel rendelkezik

DOM FELÉPÍTÉS

Példa

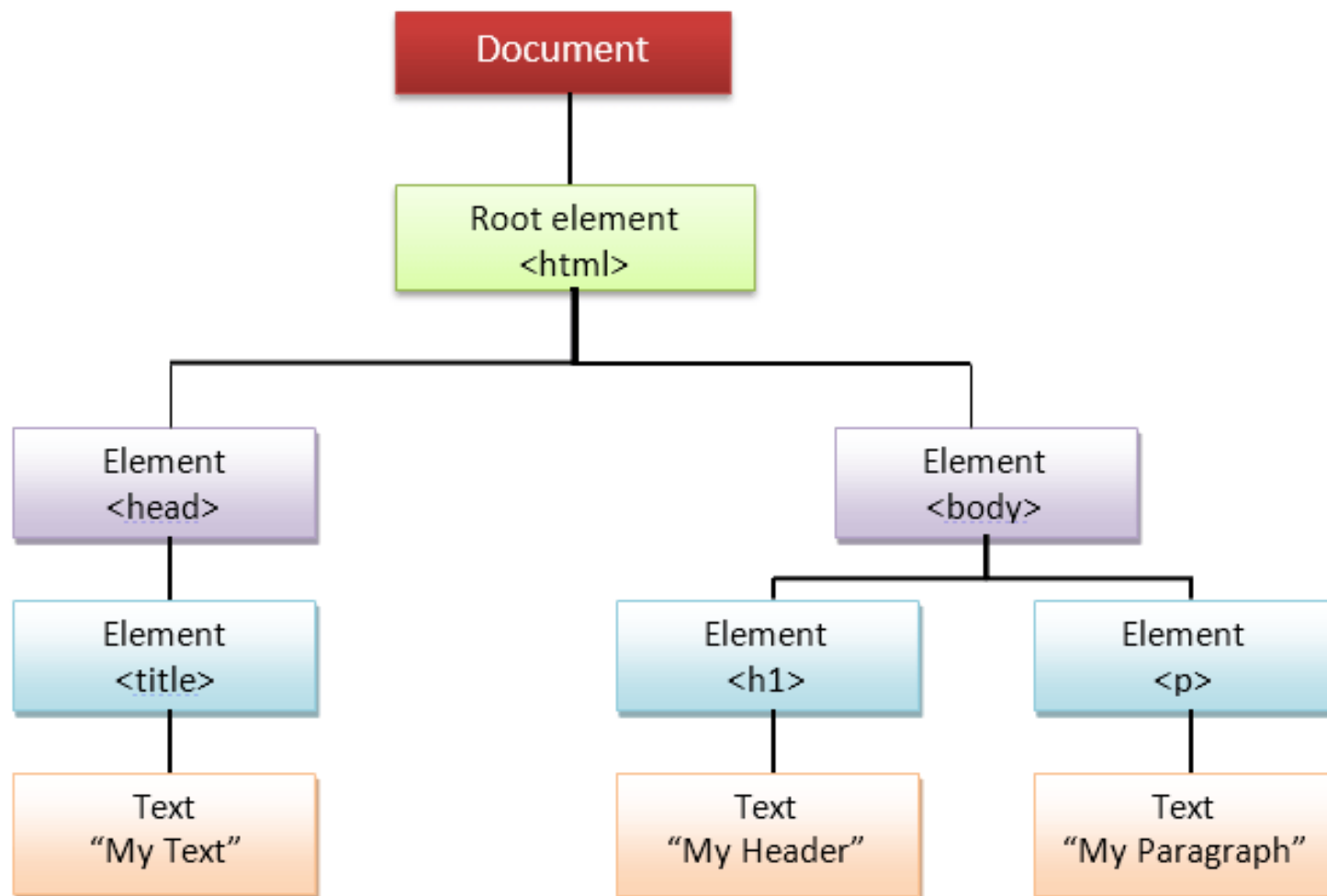
```
<html>
<head>
  <title>My Text</title>
</head>
<body>
  <h1>My Header</h1>
  <p>My Paragraph</p>
</body>
</html>
```

DOM FELÉPÍTÉS

Példa

- A `html` a gyökérelem, nincs szülőeleme
- A `html` elemnek két gyerekeleme van: `head`, `body`
- A `head` és a `body` elemek egymás testvérei
- A `html` elem a `head` és a `body` szülő eleme
- A `body` elemnek két gyerekeleme van: `h1` és `p`
- A `p` elemnek egy gyerekeleme van: egy `text` elem ("My Paragraph")

DOM FELÉPÍTÉS



Node tulajdonságok

nodeName

tag neve	HTML elem esetén (nagybetűs)
attribútum neve	attribútum esetén
#text	text node esetén
#comment	megjegyzés node esetén
#document	document node esetén

Node tulajdonságok

`nodeValue`

<code>null</code>	HTML elem és document esetén
attribútum értéke	attribútum esetén
szövegtartalom	text node esetén
szövegtartalom	megjegyzés node esetén

DOM CSÚCS

Node tulajdonságok

`nodeType`

A csúcs típusát azonosító szám, a legfontosabbak:

- | | |
|---|------------------------|
| 1 | HTML elem esetén |
| 2 | attribútum esetén |
| 3 | text node esetén |
| 8 | megjegyzés node esetén |
| 9 | document node esetén |

DOM CSÚCS

Node tulajdonságok

`tagName`

A tag nevét adja csupa nagybetűs formátumban
Csak HTML elemre használható

`innerHTML`

A csúcs HTML tartalmát jelenti, azaz a nyitó és záró tag közötti részt

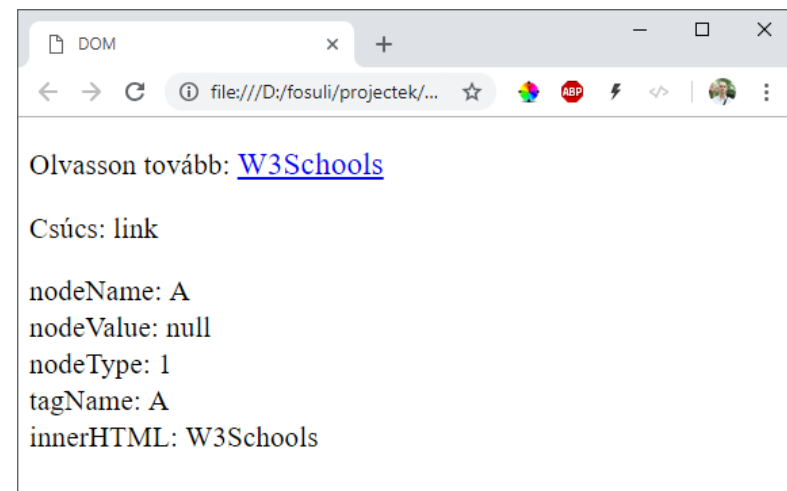
`innerText`

Csak a csúcs szöveged tartalmát jelenti, elhagyva a HTML tartalmat (tag-eket)

DOM CSÚCS

```
<p id="tanacs">Olvasson tovább:  
<a id="w3" href="http://w3schools.com">  
    W3Schools</a>  
</p>
```

```
var x = document.getElementById("w3");  
x.nodeName      // A  
x.nodeValue     // null  
x.nodeType      // 1  
x.tagName       // A  
x.innerHTML     // W3Schools
```



DOM CSÚCS

Node metódusok

`hasAttribute()`

Igazat ad vissza, a csúcsnak létezik ilyen attribútuma

`setAttribute()`

Hozzáad egy attribútumot a csúcshoz

`getAttribute()`

Visszaadja a csúcs adott attribútumának az értékét, ha nincs ilyen attribútuma, `null` értéket ad

`removeAttribute()`

Törli a csúcsnak a megadott attribútumát

DOM CSÚCS

```
var x = document.getElementById("w3");  
if (!x.hasAttributum("title")) {  
    x.setAttributum("title", "W3Schools");  
}  
else {  
    if (x.getAttributum("title") !=  
        "W3Schools") {  
        x.removeAttributum("title");  
    }  
}
```

DOM NAVIGÁCIÓ

Egy csúcsból el lehet érni a környező csúcsokat

`parentNode` szülő csúcs

Mivel minden csúcsnak egy szülő csúcs van, így ez egy
Node objektumot ad vissza

DOM NAVIGÁCIÓ

`childNodes` gyerek csúcsok

A csúcs gyerek csúcsait egy `NodeList` objektumban adja vissza.

Ez hasonló a tömbhöz, hivatkozhatunk az elemeire index szerint és lekérdezhető a hossza a `length` tulajdonsággal

A csúcsok olyan sorrendben vannak, ahogy a HTML forrásban

`hasChildNodes()` igazat ad, ha van gyerekcsúcsa

DOM NAVIGÁCIÓ

`children`

gyerek HTML elem csúcsok

A csúcs gyerek csúcsait egy `HTMLCollection` objektumban adja vissza.

Ez hasonló a tömbhöz, hivatkozhatunk az elemeire `index` szerint és lekérdezhető a hossza a `length` tulajdonsággal

A csúcsok olyan sorrendben vannak, ahogy a HTML forrásban

A `children` csak a HTML elemeket tartalmazza, a `childNodes` az összes csúcsot!

DOM NAVIGÁCIÓ

`firstChild`

első gyerekcsúcs

Ugyanaz, mint a `childNodes[0]`

`firstElementChild`

első HTML elem gyerekcsúcs

Ugyanaz, mint a `children[0]`

DOM NAVIGÁCIÓ

`lastChild`

utolsó gyerekcsúcs

Ugyanaz, mint a `childNodes[childNodes.length-1]`

`lastElementChild`

utolsó HTML elem gyerekcsúcs

Ugyanaz, mint a `children[children.length-1]`

DOM NAVIGÁCIÓ

`nextSibling`

következő testvércsúcs

`nextElementSibling`

következő HTML elem
testvércsúcs

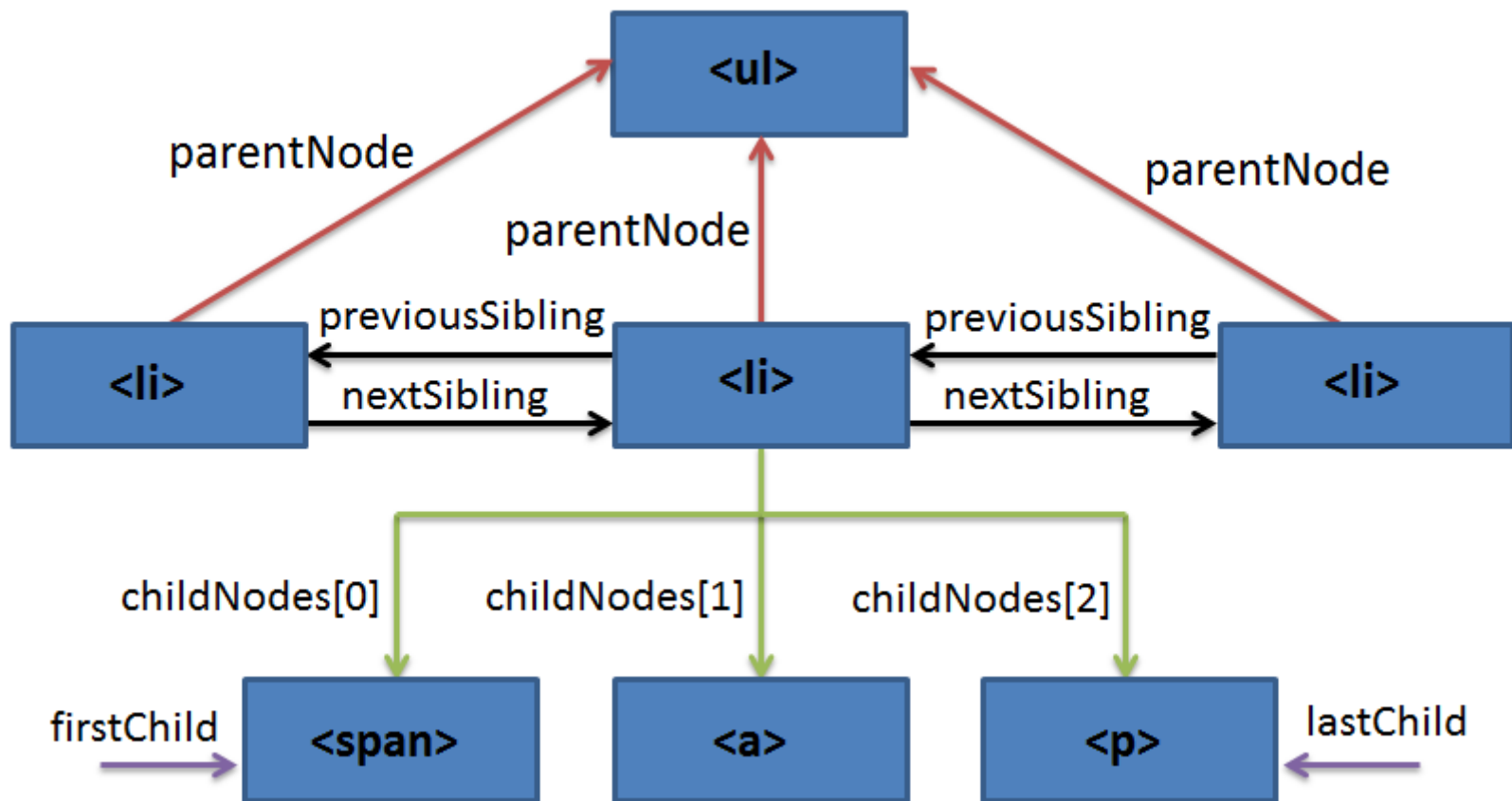
`previousSibling`

előző testvércsúcs

`previousElementSibling`

előző HTML elem
testvércsúcs

DOM NAVIGÁCIÓ



DOM NAVIGÁCIÓ

Példa a 178. diáról

```
var x = document.getElementById("w3");  
var y = document.getElementById("tanacs");
```

```
x.parentNode           // bekezdés csúcs  
x.hasChild()           // true - szövegcsúcs  
y.hasChild()           // true  
y.firstChild           // szövegcsúcs  
y.firstElementChild    // link  
y.lastChild            // link  
y.lastElementChild     // link
```

DOM NAVIGÁCIÓ

<code>y.childNodes</code>	<code>// szöveg, link</code>
<code>y.children</code>	<code>// link</code>
<code>x.previousSibling</code>	<code>// szöveg</code>
<code>x.previousElementSibling</code>	<code>// null</code>
<code>x.nextSibling</code>	<code>// null</code>
<code>x.nextElementSibling</code>	<code>// null</code>

DOM CSÚCS MANIPULÁLÁS

DOM csúcs létrehozás

Új DOM csúcsot először létre kell hozni, majd beállítani a szükséges tulajdonságait végül befűzni a DOM fába a megfelelő helyre

Új DOM csúcs létrehozása

HTML tag

```
document.createElement(tagnév)
```

Szövegcsúcs

```
document.createTextNode(szöveg)
```

DOM CSÚCS MANIPULÁLÁS

DOM csúcs klónozása

`cloneNode()`

Lemásolja a csúcsot és visszaadja az új csúcs objektumot

Megadható egy logikai paraméter, amely ha igaz, a gyerekcúcsokat is lemásolja, ha hamis, csak az attribútumokat másolja

DOM CSÚCS MANIPULÁLÁS

Csúcs hozzáadása

Egy elemhez, mint gyerekelem

```
appendChild()
```

Az elemet legutolsó gyereknek illeszti be

Az elem lehet HTML elem csúcs és szöveg csúcs is

```
szülő.appendChild(újcsúcs)
```

DOM CSÚCS MANIPULÁLÁS

Csúcs hozzáadása

Egy elem gyerekelemként egy adott gyerekelem elé, mint testvér csúcs

```
insertBefore()
```

Az elem lehet HTML elem csúcs és szöveg csúcs is

```
szülő.insertBefore(újcsúcs,gyerek)
```

Szövegcsúcs mellé újabb szövegcsúcsot beszúrni nincs értelme, azok összevonhatóak

DOM CSÚCS MANIPULÁLÁS

Csúcs törlése

Egy szülő csúcstól lehet egy gyerekcsúcsot törölni

```
removeChild()
```

Egy csúcs törléséhez ismerni kell a szülő csúcsot is. Ez a parentNode segítségével meghatározható

```
szülő.removeChild(gyerek)
```

DOM CSÚCS MANIPULÁLÁS

Csúcs lecserélése

Egy csúcsnak valamely gyerekcsúcsát le lehet cserélni egy másik csúcsra

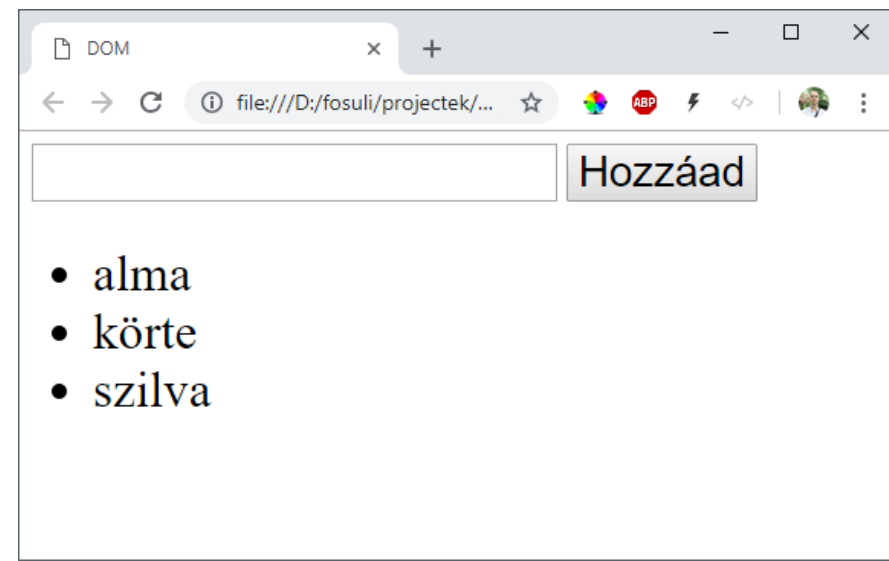
```
replaceChild()
```

A másik csúcs lehet egy új vagy egy létező csúcs is

```
szülő.replaceChild(régicsúcs, újcsúcs)
```

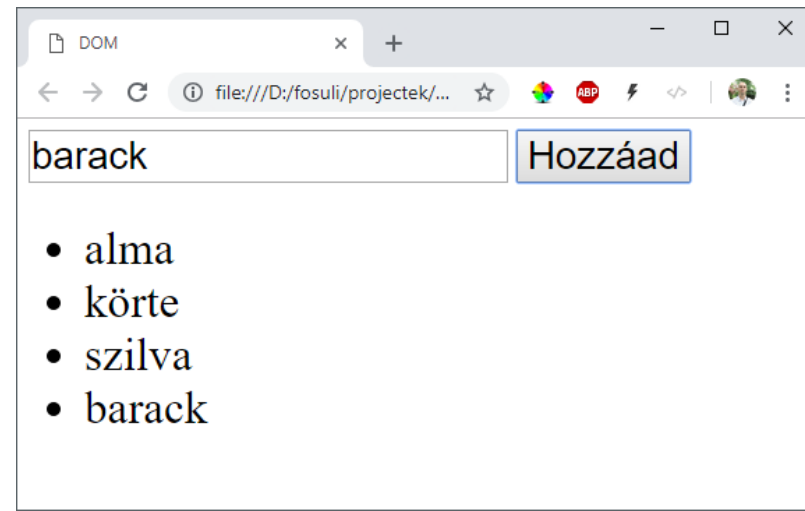
DOM CSÚCS MANIPULÁLÁS

```
<input type="text" id="uj">
<input type="button" value="Hozzáad"
      onclick="add()">
<ul id="lista">
  <li onclick="del(this)">alma</li>
  <li onclick="del(this)">körte</li>
  <li onclick="del(this)">szilva</li>
</ul>
```



DOM CSÚCS MANIPULÁLÁS

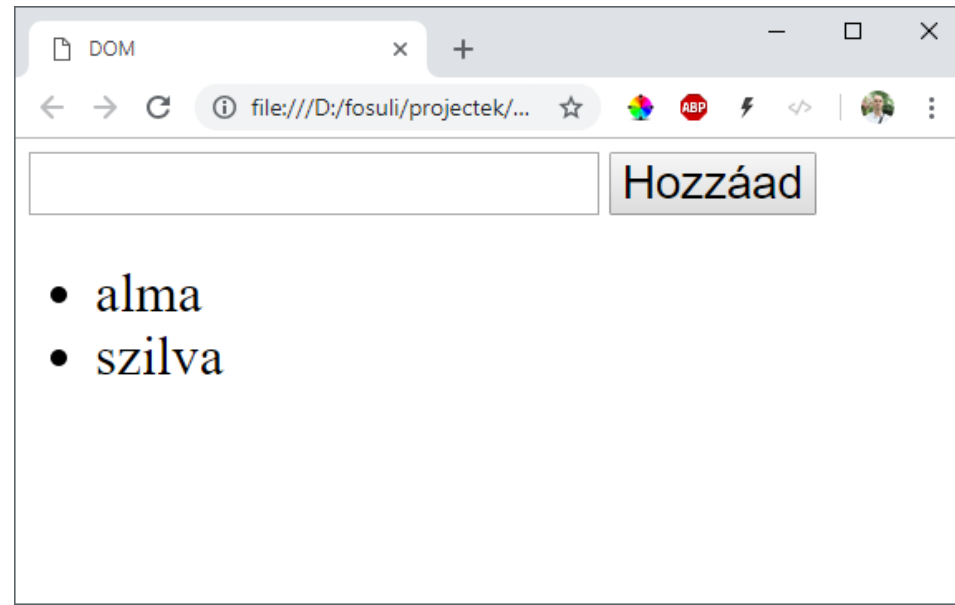
```
function add() {  
    var ujLi = document.createElement("li");  
    var uj = document.getElementById("uj").value;  
    var ujSzoveg = document.createTextNode(uj);  
    ujLi.appendChild(ujSzoveg);  
    ujLi.setAttribute("onclick", "del(this)");  
    var lista = document.getElementById("lista");  
    lista.appendChild(ujLi);  
}
```



DOM CSÚCS MANIPULÁLÁS

A listaelemre dupla klikkeléssel lehet dinamikusan törölni az elemet a listából

```
function del(obj){  
    obj.parentNode.removeChild(obj);  
}
```



FELADAT

1. Készítsen két listát! Bármelyik listából egy elemre klikkel a felhasználó, az kerüljön át a másik listába!
2. Készítsen egy fa struktúra létrehozó alkalmazást! Legyen indulásként egy gyökérelem! Majd minden csúcshoz lehessen újabb levélelemet hozzáadni, melynek a szövegét bekéri az alkalmazás! Lehessen a levélelemeket törölni a fából!

JQUERY

A jQuery egy szabadon felhasználható JavaScript függvénykönyvtár.

Használatával könnyebben lehet weboldalakhoz olyan műveleteket elkészíteni, amelyek csak JavaScript használatával hosszabb programozás feladata lenne.

A jQuery minden (elismertebb, jelenlegi verziójú) böngészőben egyformán működik pc-n és mobilon egyaránt.

A jQuery open source



JQUERY

jQuery főbb szerepe

- HTML elemek (DOM) kezelése
- Stílusok (CSS) kezelése
- HTML események kezelése
- Effektek és animációk
- egyéb

A jQuery-t folyamatosan fejlesztik, bővítik az újabb igényeknek megfelelően.

JQUERY

jQuery használata

Letölthető a jQuery weboldaláról

<https://jquery.com/download/>

Majd a weboldalhoz hozzákapcsolható, mint lokális JavaScript állomány

```
<script src="jquery-3.3.1.min.js">  
</script>
```

Előnye, hogy független más szervertől

jQuery használata

Használható letöltés nélkül a szerverre hivatkozással (CDN – Content Delivery Network)

Előnye, hogy ha már korábban betöltött egy felhasználó olyan oldalt, ahol ez a jQuery használva volt, akkor az állomány megtalálható a böngésző lokális tárterületén, így nem tölti le újra, ezáltal csökkentve a hálózati forgalmat és az oldal betöltődését

JQUERY

Hivatkozható több helyről

Google

```
<script src="https://ajax.googleapis.com/ajax/  
libs/jquery/3.3.1/jquery.min.js"></script>
```

Microsoft

```
<script src="https://ajax.aspnetcdn.com/ajax/  
jQuery/jquery-3.3.1.min.js"></script>
```

CDNJS

```
<script src="https://cdnjs.cloudflare.com/ajax/  
libs/jquery/3.3.1/jquery.min.js"></script>
```

JQUERY

Két fő típusa azonos funkcionalitással

- Teljes verzió – *uncompressed: jquery.js*

Fejlesztéshez van, könnyen olvasható formátumú, kommentezett kód

- Csökkentett méretű – *minimized: jquery.min.js*

Tömörített verzió, ahol törölve vannak a megjegyzések, white-space karakterek, ezáltal csökkentve a letöltés és értelmezés sebességét

Szintaktika

Az utasítás formája

```
$ ( kijelölő ) . művelet ( )
```

Ahol a *kijelölő* meghatározza azt (azokat) az elemet, amivel az adott *műveletet* végre kell hajtani.

Például

```
$ ("div.box") .toggle ( )
```

JQUERY

Kijelölő

A jQuery kijelölés megegyezik a CSS kijelöléssel

Elemkijelölés

```
$ ("p")
```

Osztálykijelölés

```
$ (".osztaly")
```

Azonosító kijelölés

```
$ ("#azon")
```

Stb. (például)

```
$ ("header ul > li.subM")
```

Események

A jQuery események a böngésző és a weboldallal történt műveletek "kezelése"

Az események jó része megegyezik a DOM eseményekkel

Az eseményekre műveleteket készítünk, aminek végre kell hajtódnia az esemény bekövetkezésekor, ezeket függvényként adjuk meg

```
$ ("button").click(function() {  
    utasítások;  
});
```

JQUERY

Események (csak példák)

Böngésző

```
$("window").resize()
```

Document

```
$("document").ready()
```

Űrlap

```
$("form input.szoveg").focus()
```

Billentyűzet

```
$("#textField").keypress()
```

Egér

```
$("button").click()
```

Stb.

JQUERY

Célszerű biztosítani, hogy a jQuery kód csak akkor fusson le, amikor az oldal már betöltődött, ne próbáljon egy olyan elemmel műveletet végezni, ami még létre se jött (nem töltődött be)

Minden műveletet be kell tenni egy eseménybe

```
$ ("document").ready(function() {  
    ...  
})
```

Rövidebb formában ugyanez

```
$(function() {  
    ...  
})
```

JQUERY

Effektek

A HTML elemeken számos effektet lehet végrehajtani

Megjelenés

`hide()`, `show()`, `toggle()`

`fadeIn()`, `fadeOut()`, `fadeToggle()`

`slideDown()`, `slideUp()`, `slideToggle()`

Például

```
$("div.foMenu").click(function() {  
    $("div.alMenu").toggle(500);  
})
```

JQUERY

Effektek

Animáció

Minden CSS tulajdonságot lehet animálni

```
animate({tulajdonságok})
```

Például

```
$("button#anim").click(function() {  
    $("#box").animate({  
        height : '200px',  
        opacity : '0'  
    });  
});
```

JQUERY

Effektek

Az effektekhez megadható paraméterek (egyik se kötelező)

- végrehajtás sebessége (slow, fast, ezredmásodperc)

```
hide(sebesség)
```

- callback függvény, ami a befejezés után fut le

```
hide(függvény())
```

Példa

```
$("box").hide(500, function() {  
    alert("Befejeződött");  
})
```


JQUERY

CSS tulajdonságok kezelése

`css (tulajdonság, érték)`

Példa

```
$ ("p").css ("color", "red")
```

Osztályok

`addClass ()`

`removeClass ()`

`toggleClass ()`

Példa

```
$ ("h1").addClass ("cimsor")
```

JQUERY

DOM manipulálás

Elemek tulajdonságának kezelése

<code>text()</code>	szövegtartalom
<code>html()</code>	HTML tartalom
<code>val()</code>	érték
<code>attr()</code>	attribútum értéke

Példa

```
$("button").click(function() {  
    var s = $("input#cim").val();  
    $("#link").attr("href",s);  
})
```

DOM manipulálás

Elemek hozzáadása

`append()` hozzáad a végéhez

`prepend()` hozzáad az elejére

`after()` hozzáad utána

`before()` hozzáad elé

Elemek törlése

`remove()` törli az elem(ek)et (és gyerekeit)

`empty()` törli a gyerekelemeket

DOM manipulálása

Példa

```
<ul id="lista">  
  <li>első elem</li>  
  <li>második elem</li>  
  <li>harmadik elem</li>  
</ul>  
<button>Forgat</button>
```

DOM manipulálás

Példa (folytatás)

```
$(document).ready(function() {  
    $("button").click(function() {  
        var s = $("li:first-child").text();  
        var li = document.createElement("li");  
        li.innerHTML = s;  
        $("ul").append(li);  
        var s = $("li:first-child").remove();  
    })  
});
```

DOM manipulálása

Többi elem elérése

<code>parent()</code>	közvetlen szülő
<code>parents()</code>	összes szülő (szűrhető)
<code>children()</code>	közvetlen gyerek (szűrhető)
<code>find()</code>	keresés összes gyerek között

```
$("input").parents("fieldset")
```

```
$("fieldset").children("input[type=text]")
```

JQUERY

DOM manipulálása

Többi elem elérése

`siblings()`

testvérek (szűrhető)

`next()`

következő testvér

`nextAll()`

összes testvér utána

`prev()`

előző testvér

`prevAll()`

összes előző testvér

Példa

```
$("input").siblings("label")
```

```
$("nav li").next()
```

FELADAT

1. Készítsen egy animáltan működő két szintű lenyíló menüt! Használjon jQuery effekteket és CSS tulajdonság meghatározásokat!
2. Készítsen egy lista készítő felületet jQuery felhasználásával! A listához lehessen hozzávenni új elemet, illetve lehessen a listában kijelölni elemet és azt mozgatni a listában előre és hátra, valamint törölni a listából!

FORRÁS



A tananyag a W3C szabványra épül

<https://www.w3.org/standards/webdesign/script>

[w3schools.com](https://www.w3schools.com)

Ajánlott irodalom

<https://www.w3schools.com/js/default.asp>

<https://www.w3schools.com/jquery/default.asp>

SZERVER OLDALI WEBES PROGRAMOZÁS A PHP NYELV

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Szociális
Alap



BEFEKTETÉS A JÖVŐBE

TARTALOM

- A PHP nyelv
- Szintaktikája
- Főbb adattípusok
- Vezérlési szerkezetek
- Főbb osztályai
- Űrlapok kezelése
- Adatbázis kapcsolat
- Session és cookie kezelés

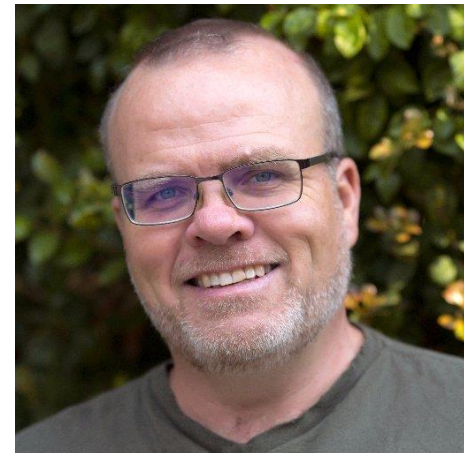
A PHP NYELV

- A PHP egy HTML forrásba ágyazható szerver oldali szkript nyelv
- A beágyazott kódot a webszerverhez kapcsolódó modul értelmezi és azonnal le is futtatja, a szkript eredménye kerül a kliens oldalra küldendő válaszba
- A PHP nyílt forráskódú, szabadon felhasználható
- A PHP kódot tartalmazó állományok kiterjesztése *.php*



A PHP TÖRTÉNETE

- Az első PHP verziót 1995-ben adták ki, csupán saját weboldalak karbantartására (Rasmus Lerdorf)
- 1997-ben megjelenő PHP3-tól nőtte ki magát önálló programnyelvvé
- Az aktuális verzió a 7.3
- A 7.1 előtti verziók már nem támogatottak (<https://www.php.net/supported-versions.php>)



Helye

- A PHP forráskód a HTML kódba a
`<?php ... ?>`
jelöléssel ágyazható be
- Egy oldalon belül több ilyen blokk is lehet, azaz HTML kód (valamint JavaScript) és PHP blokkok egymást szabadon követhetik
- A HTML forráskódú állományba bárhova kerülhet, ha outputja van, akkor az a beágyazás helyére kerül
- A kiértékelése fentről lefele történik sorfolytonosan

SZINTAKTIKA

Példa

Szerver oldal

```
<!doctype html>
<html>
<body>
<?php
echo "Helló Világ";
?>
</body>
</html>
```

Kliens oldal

```
<!doctype html>
<html>
<body>
Helló Világ
</body>
</html>
```

PHP segítségével a HTML kód felépíthető szerver oldalon Akár JavaScript kód is írható PHP kóddal

Szerver oldal

```
<?php $nev = "Mikka Makka";    ?>
<!doctype html>
<html>
<head>
<script>
function udv(){
    alert("Helló <?php echo $nev ?>")
}
</script>
</head>
<body>
<?php
    echo "<h1>Üdvözlés</h1>";
    echo "<p>Helló ".$nev."</p>";
    echo "<button
onclick='udv()'>Helló</button>";
?>
</body>
</html>
```

Kliens oldal

```
<!doctype html>
<html>
<head>
<script>
function udv(){
    alert("Helló Mikka Makka")
}
</script>
</head>
<body>
<h1>Üdvözlés</h1>
<p>Helló Mikka Makka</p>
<button onclick='udv()'>Helló</button>
</body>
</html>
```


SZINTAKTIKA

A tananyag további részében feltételezzük, hogy a példa PHP kódokat egy helyesen megírt HTML forráskódba ágyazzuk be

Például

```
<!doctype html>
<html>
<head></head>
<body>
<?php
// PHP kód ide
?>
</body>
</html>
```

SZINTAKTIKA

- A nyelv foglalt szavait tekintve (pl. `for`, `if`, `stb.`) nem kis-nagybetű érzékeny
- A változó elnevezések tekintetében viszont kis-nagybetű érzékeny
- Minden utasítást ; (pontosvessző) karakterrel kell lezárni
- A kitöltő karaktereket (white space) figyelmen kívül hagyja
- Megjegyzések

```
// egy soros megjegyzés
```

```
/* több soros megjegyzés */
```

SZINTAKTIKA

Példa

```
<?php $nev = "Mekk Elek"; ?>
<!doctype html>
<html>
<body>
<?php
echo "<p>Helló ".$nev."</p>\n";    // helyes
ECHO "<p>Helló ".$nev."</p>\n";    // helyes
echo "<p>Helló ".$NEV."</p>\n";    // nem helyes
?>
</body>
</html>
```

Literálok

Egész szám	100
Valós szám	3.14, 314e-2
Szöveg	"példa szöveg" vagy 'példa szöveg'
Logikai	true, false
null	"üres" érték

OUTPUT

A PHP output a HTML forráskódba kerül a PHP kód helyére, így azt kliens oldalon a böngésző értelmezi. Emiatt az outputba kerülhet statikus szöveg, HTML kód, vagy akár JavaScript forráskód.

Két legegyszerűbb output utasítás az `echo` és a `print`

Mindkettőnek két használati módja van:

```
echo "Helló Világ";          vagy    echo("Helló Világ");  
print "Helló Világ";        vagy    print("Helló Világ");
```

A főbb különbség a kettő között, hogy az `echo`-nak több paramétere lehet, a `print`-nek egy

```
echo "Helló ", "Világ", " és ", "Emberek";  
print "Helló " . "Világ" . " és " . "Emberek";
```

VÁLTOZÓK

A változókat a \$ karakterrel kell kezdeni

```
$nev = "Mekk Elek"
```

A változó neve csak angol ábécé betűit, számjegyeket és _ (alsó vonás) karaktereket tartalmazhat, de nem kezdődhet számmal. (Kerülendő a csupa nagybetűs valamint az alsó vonással kezdődő változónevek, azok beépített változókra utalnak)

A változó elnevezések kis-nagybetű érzékenyek

A PHP 5.3 verziótól kezdve tartalmaz garbage collector-t, így megszüntetésük nem szükséges

VÁLTOZÓK

A változókat nem kell deklarálni, az első használata (amikor értéket kap) automatikusan deklarálja

```
$a = 10;
```

Egy változó létezése ellenőrizhető

```
isset("változónév")
```

Példa

```
if (isset($a)) {  
    echo $a;  
}  
else {  
    echo "A változó nem létezik";  
}
```

VÁLTOZÓK

Változók érvényességi köre

Egy változó globális változó, ha függvényen kívül deklarált, ekkor kívülről elérhető el csak.

Valamint a változó lokális változó, ha függvényen belül deklarált, ekkor csak a függvényen belül érhető el.

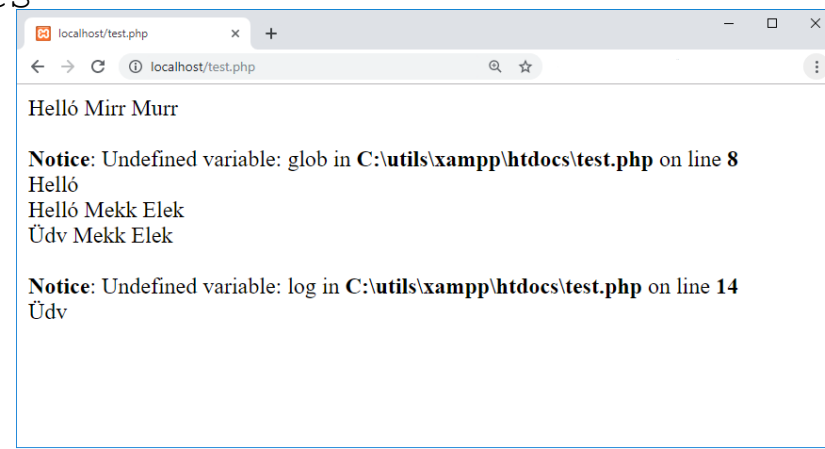
A globális változó bárhol elérhető a beépített `$GLOBALS["változó"]` asszociatív tömb segítségével, ahol az index a változó neve

VÁLTOZÓK

```
$glob = "Mekk Elek"; // globális változó

function hello(){
    $lok = "Mirr Murr"; // lokális változó
    echo " Helló ".$lok."<br>"; // helyes
    echo " Helló ".$glob."<br>"; // hiba
    echo " Helló ".$GLOBALS["glob"]."<br>"; // helyes
}

hello();
echo " Üdv ".$glob."<br>"; // helyes
echo " Üdv ".$log."<br>"; // hiba
```



SZUPER-GLOBÁLIS VÁLTOZÓK

A PHP-nak van néhány ún. szuper-globális változója, amelyet a PHP hoz létre és bárhonnán elérhetők

<code>\$GLOBALS</code>	a globális változók tömbje
<code>\$_SERVER</code>	szerver paraméterek tömbje
<code>\$_POST</code>	<i>post</i> módon átadott űrlap adatok tömbje
<code>\$_GET</code>	<i>get</i> módon átadott űrlap adatok tömbje
<code>\$_REQUEST</code>	kapott adatok tömbje (post, get és cookies)
<code>\$_SESSION</code>	session változók tömbje
<code>\$_COOKIE</code>	süti változók tömbje

KONSTANSOK

Konstans definiálható a `define()` utasítással

A konstans értéke a szkript végéig nem változtatható

A konstansok mindig globálisan elérhetőek

```
define(név, érték, kisnagybetű)
```

Paraméterei

- konstans neve
- konstans értéke
- konstans neve kis-nagybetű érzékeny-e (nem kötelező, alapérték a `false`)

```
define("nev", "Mekk Elek")
```

TÍPUSOK

A PHP lazán típusos nyelv, azaz nem kell deklarálskor megadni a típusát, azt automatikusan kapja az értékadáskor, azaz a típusa változhat

```
$a = 5;
```

```
$a = "abc";
```

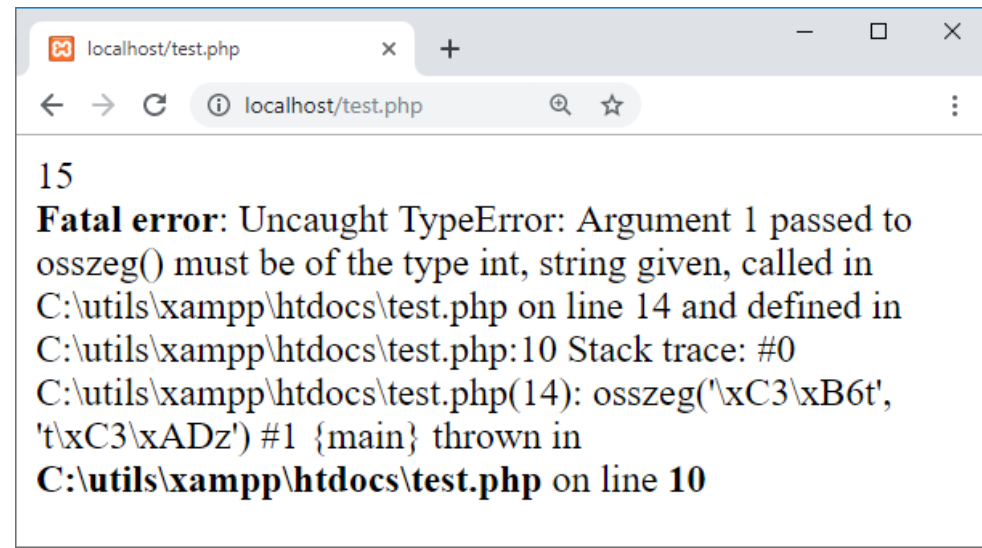
PHP 7-ben bevezettek típusdeklarációt, amikor egy függvény paraméterének és visszatérési értékének megadható a típusa, ehhez meg kell adni a forrás legelején, hogy szigorú típusdeklaráció kötelező

```
<?php declare(strict_types=1); ?>
```

TÍPUSOK

Példa

```
<?php declare(strict_types=1);  
function osszeg(int $a, int $b) {  
    return $a + $b;  
}  
echo osszeg(5,10);           // 15  
echo osszeg("öt","tíz");     // hibaüzenet  
?>
```



TÍPUSOK

Alap típusok

- Egész szám
- Valós szám
- Logikai
- Szöveg
- Tömb
- Objektum
- `null`

A változó típusa lekérdezhető a `gettype()` függvénnnyel
vagy a típusa és az értéke a `var_dump()` függvénnnyel

OPERÁTOROK

Értékadó operátorok

= \$a = \$b

+= \$a += \$b // \$a = \$a + \$b

-= \$a -= \$b // \$a = \$a - \$b

*= \$a *= \$b // \$a = \$a * \$b

/= \$a /= \$b // \$a = \$a / \$b

%= \$a %= \$b // \$a = \$a % \$b

.= \$a .= \$b // \$a = \$a . \$b

maradékképzés
szövegösszefűzés

OPERÁTOROK

Számműveletek

$\$a = 6, \$b = 2$

+	$\$a + \b	= 8	összeg
-	$\$a - \b	= 4	különbség
*	$\$a * \b	= 12	szorzat
/	$\$a / \b	= 3	hányados
%	$\$a \% \b	= 0	maradékképzés
**	$\$a ** \b	= 36	hatványozás
++	$\$a++$	= 7	növelés
--	$\$a--$	= 5	csökkentés

OPERÁTOROK

Logikai műveletek

!	!\$a	tagadás
&& vagy and	\$a && \$b	és
vagy or	\$a \$b	vagy
xor	\$a xor \$b	kizáró vagy

Feltételes operátor

kifejezés ? érték1 : érték2

Ha a kifejezés igaz, érték1-t adja vissza, különben érték2-t

OPERÁTOROK

Szöveg művelet

. (pont) \$a . \$b összefűzés (konkatenáció)

```
$a = "Mekk Elek";
```

```
echo "Helló " . "Világ"; // Helló Világ
```

```
echo "Helló " . $a      // Helló Mekk Elek
```

OPERÁTOROK

Összehasonlítások

`==` egyenlő érték

`===` azonos típus és egyenlő érték

`!=` nem egyenlő

`<` kisebb

`>` nagyobb

`<=` kisebb egyenlő

`>=` nagyobb egyenlő

OPERÁTOROK

Objektumorientált operátorok

A tag lehet adattag és metódus, amelyek elérhetőségét a láthatóság befolyásolja

Objektum operátor

```
objektum->tag
```

Statikus tag operátor

```
objektum::tag
```

Példányosító operátor

```
new osztaly()
```

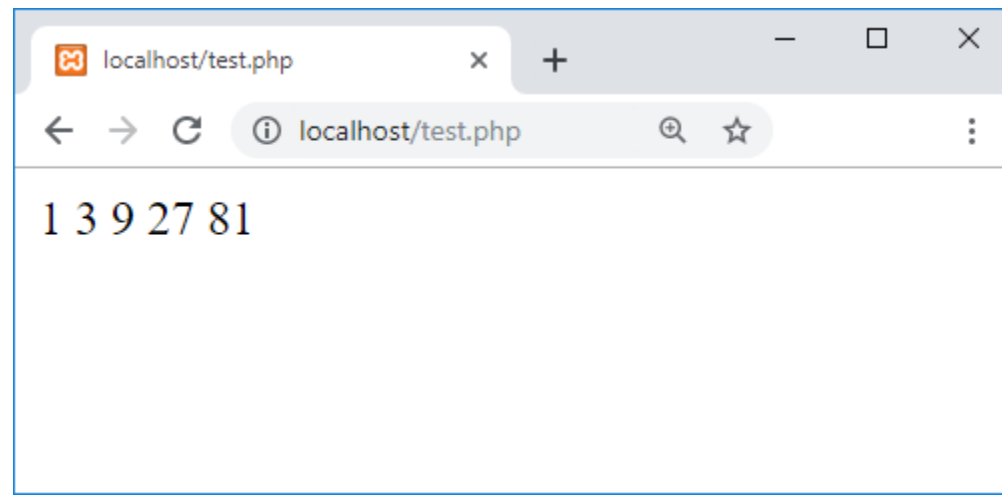
BLOKKUTASÍTÁS

Több utasítás egy blokkba foglalásához a jelölés

```
{  
    utasítás;  
    ...  
}
```

Példa

```
$n = 3;  
$h = 1;  
while ($h < 100){  
    echo $h . " ";  
    $h *= $n;  
}
```



FELTÉTELES UTASÍTÁSOK

Kétirányú elágazás

```
if (feltétel)
    ...          // igaz ág utasítása
else
    ...          // hamis ág utasítása
```

az else ág elhagyható

További elágazások

```
if (feltétel_1)
    ...          // feltétel1 igaz ág utasítása
elseif (feltétel_2)
    ...          // feltétel_1 hamis, feltétel_2 igaz ág utasítása
...             // további lehetséges elseif ágak
else
    ...          // feltétel_2 hamis ág utasítása
```

FELTÉTELES UTASÍTÁSOK

Példa

```
if ($a > 0) {  
    echo $a . " szám pozitív";  
}  
elseif ($a < 0) {  
    echo $a . " szám negatív";  
}  
else {  
    echo $a . " szám a nulla";  
}
```

FELTÉTELES UTASÍTÁSOK

Többirányú elágazás

```
switch (kifejezés) {  
    case érték1: utasítás; break;  
    case érték2: utasítás; break;  
    ...  
    default: utasítás;  
}
```

A kifejezést egyszer értékeli ki, majd sorban összehasonlítja a `case` ágban adott értékekkel

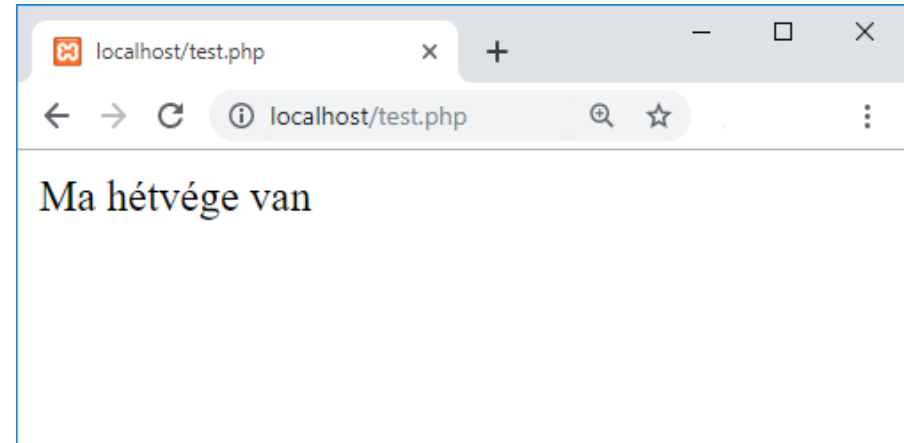
Minden `case` ágat `break` utasítás zár le, hogy egyezés esetén a többi `case` ágat ne vizsgálja

A `default` ág elhagyható

FELTÉTELES UTASÍTÁSOK

Példa

```
echo "Ma ";  
$d = date("w");  
switch ($d){  
    case 1: echo "hétfő"; break;  
    case 2: echo "kedd"; break;  
    case 3: echo "szerda"; break;  
    case 4: echo "csütörtök"; break;  
    case 5: echo "péntek"; break;  
    default: echo "hétvége";  
}  
echo " van";
```



ITERÁCIÓK

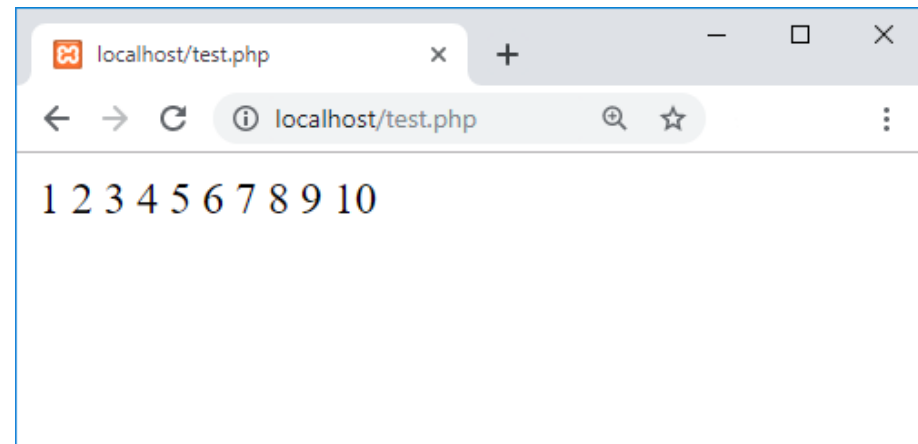
Számláló ciklus

```
for (init; feltétel; léptetés) {  
    utasítások;  
}
```

Az inicializálás után, amíg a feltétel igaz, végrehajtja az utasításokat, majd végrehajtja a léptetést.

Példa

```
for ($i = 1; $i <= 10; $i++) {  
    echo $i." ";  
}
```



ITERÁCIÓK

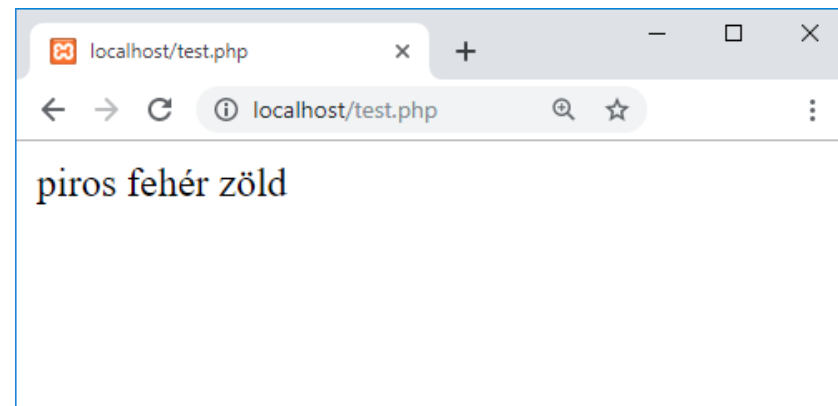
foreach

A tömbök esetén használható a `foreach` iteráció a tömb elemeinek a bejárásához

```
foreach ($tomb as $ertekek) {  
    utasítások;  
}
```

Példa

```
$t = array("piros", "fehér", "zöld");  
foreach ($t as $szin){  
    echo $szin." ";  
}
```



ITERÁCIÓK

Feltételes ciklusok

Elöl tesztelő ciklus

```
while (feltétel) {  
    utasítások;  
}
```

Hátul tesztelő ciklus

```
do {  
    utasítások;  
} while (feltétel);
```

Mindkettő addig hajtja végre az utasításokat, amíg a feltétel értéke igaz.

ITERÁCIÓK

Példa

```
$x = 10;  
$i = 5;  
$j = 5;  
while ($i>0){  
    echo ($x % $i) ." ";  
    $i--;  
}  
do {  
    echo ($x % $j) ." ";  
    $j--;  
} while ($j>0);
```

[Mi a két megoldás között a különbség?]

ITERÁCIÓK

Ciklus megtörése

`break`

ezen a ponton kilép a ciklusból (vagy `switch`-ből) és a következő utasítással folytatja

Ciklusmag kihagyása

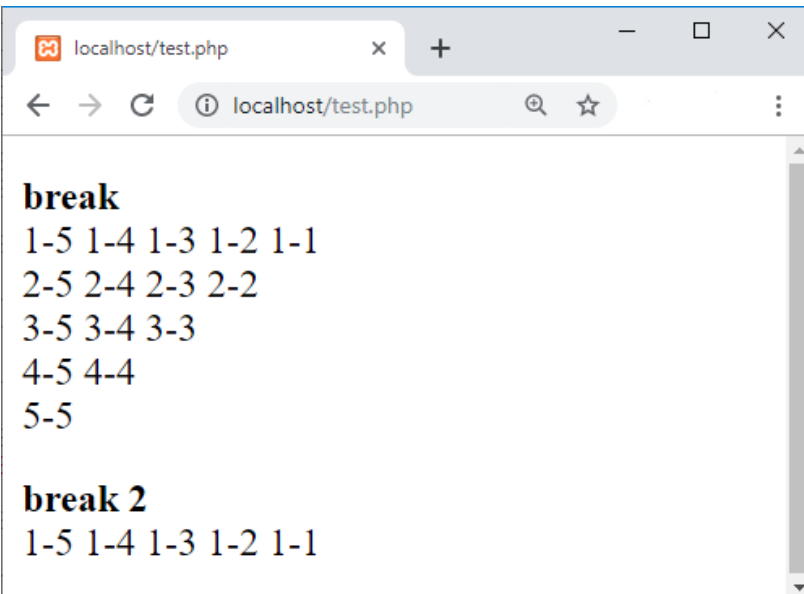
`continue`

kihagyja a ciklusmag hátralévő részét és a ciklus feltételéhez ugrik

ITERÁCIÓK

Mindkettő paraméterezhető, hogy hány iterációs szintre vonatkozik (alapértelmezett szint az 1)

```
for ($i=0; $i<5; $i++) {  
    for ($j=5; $j>0; $j--){  
        echo $i . "-" . $j . " ";  
        if ($i == $j) {  
            break 2;           // mindkét ciklusból kiugrik  
        }  
    }  
    echo "<br>";  
}
```

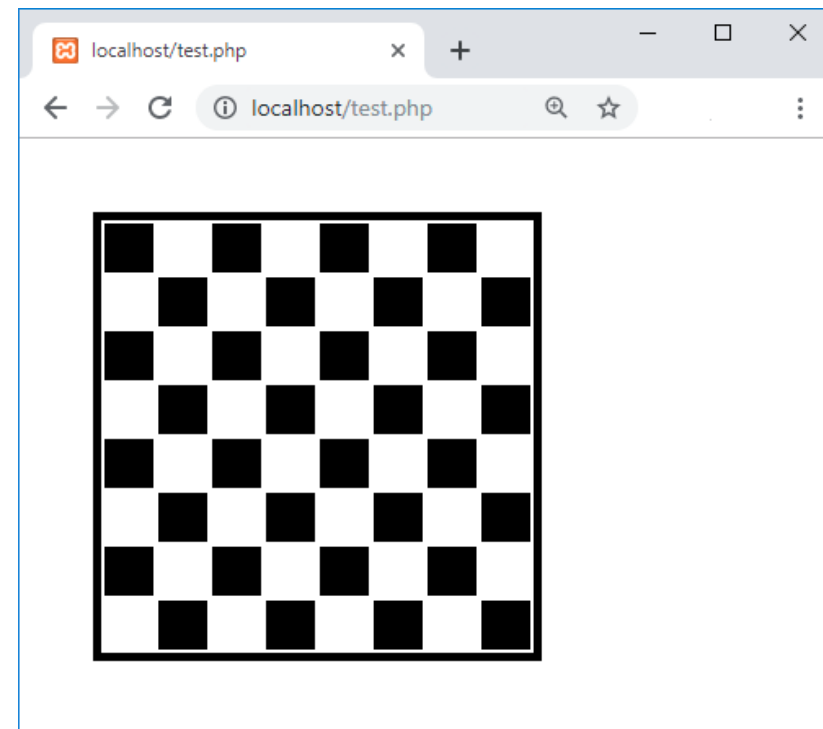


```
localhost/test.php  
localhost/test.php  
  
break  
1-5 1-4 1-3 1-2 1-1  
2-5 2-4 2-3 2-2  
3-5 3-4 3-3  
4-5 4-4  
5-5  
  
break 2  
1-5 1-4 1-3 1-2 1-1
```

FELADAT

Készítsen kódot, amely legenerál egy HTML sakktáblát az oldalra!

Készítse el a kódot úgy, hogy a tábla mérete könnyen változtatható legyen!



FÜGGVÉNYEK

Függvény készítése

```
function függvéynév(paraméterek) {  
    utasítások;  
}
```

A függvény neve egyedi, nem kezdődhet számmal és nem kis-nagybetű érzékeny (de célszerű tartani)

Függvény hívása a függvény nevével és paramétereivel

```
function hello($nev) {  
    echo "Helló " . $nev;  
}  
hello("Mekk Elek");  
hello("Mikka Makka");
```

FÜGGVÉNYEK

A függvény visszatérési értéke megadható (nem kötelező megadni)

`return érték`

A `return` hatására a függvényből kiugrik, az utána lévő utasításokat már nem hajtja végre

```
function fuggveny() {  
    echo "lefut";  
    return 0;  
    echo "már nem fut le";  
}
```

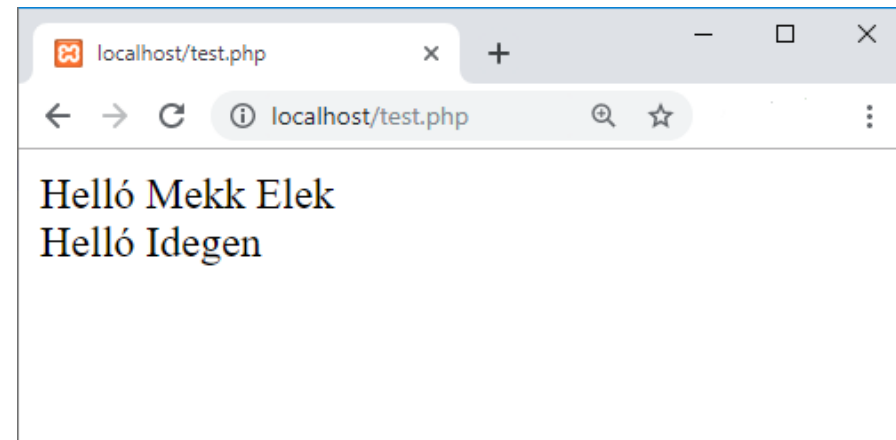
FÜGGVÉNYEK

A paramétereknek megadható alapértelmezett érték

```
function függvénynév(paraméter = érték)
```

Ezt akkor használja a függvény, ha a híváskor nem kap aktuális paramétert

```
function hello($nev = "Idegen") {  
    echo "Helló " . $nev;  
}  
hello("Mekk Elek");  
hello();
```



FÜGGVÉNYEK

A függvények paramétereinek és visszatérési értékének típusai megadható a `strict` mód bekapcsolásával

```
<?php declare(strict_types=1);  
function osszeg(int $a, int $b): int {  
    // return "összeg";    // hiba  
    return $a + $b;  
}  
echo osszeg(5, 10);  
echo osszeg("5", "10");    // hiba  
?>
```

FELADAT

1. Készítsen olyan weboldalt, ahol PHP függvénnnyel formázottan kiírja egy szám adott hatványát! A függvény paramétere az alap és a kitevő.
2. Készítsen olyan PHP függvényt, amely paraméterben adott méretű div-t jelenít meg paraméterben adott színnel! Készítsen weboldalt, amely több különböző méretű és színű div-t jelenít meg az oldalon!

OBJEKTUMORIENTÁLT PROGRAMOZÁS

A PHP5-től kezdve lehetőség van objektumorientáltan programozni a PHP-ban

Osztály definiálása

```
class osztály {  
    private $adattag = érték;  
    public fuggveny() {  
        utasítások;  
    }  
}
```

Példányosítás

```
$obj = new osztály();
```

OBJEKTUMORIENTÁLT PROGRAMOZÁS

Ha osztály metódusából hivatkozunk saját adattagra, akkor a `$this` kulcsszóval meg kell adni, hogy az adott objektum adattagjáról van szó

`$this->adatag`

Példa

```
class ember {  
    private $nev = "Valaki";  
    public function hello(){  
        echo "Helló " . $this->nev;  
    }  
}
```

OBJEKTUMORIENTÁLT PROGRAMOZÁS

Öröklődés

Az öröklődést az `extends` kulcsszóval adjuk meg

```
class osztály extends ősosztály { ... }
```

A PHP-ban minden osztály legfeljebb egy osztályból származhat

Az öröklődéskor az utód örököli az ős adattagjait és metódusait. Ezek használatát befolyásolja az ősosztályban definiált láthatóság

OBJEKTUMORIENTÁLT PROGRAMOZÁS

Konstruktor

A konstruktor speciális metódus, amely a példányosításkor fut le. Minden osztály tartalmaz alap konstruktort, amely csak a példányosítást végzi el.

Lehet saját konstruktort írni:

```
function __construct(paraméterek)
```

Példa

```
class ember {  
    private $nev;  
    function __construct($nev) {  
        $this->nev = $nev;  
    }  
}  
  
$obj = new ember("Mekk Elek");
```

OBJEKTUMORIENTÁLT PROGRAMOZÁS

Származtatott osztály konstruktorában gondoskodni kell az ősosztály konstruktorának hívásáról.

Ősosztály konstruktorának meghívása

```
parent::__construct()
```

Példa

```
class negyzet {
    private $a;
    function __construct($oldal){
        $this->a = $oldal;
    }
}

class teglalap extends negyzet{
    private $b;
    function __construct($aoldal, $boldal){
        parent::__construct($aoldal);
        $this->b = $boldal;
    }
}
```

OBJEKTUMORIENTÁLT PROGRAMOZÁS

Láthatóság

Három láthatósági szint:

<code>private</code>	csak a saját osztály
<code>protected</code>	saját és leszármazott osztályok
<code>public</code>	bárhonnan

Ha egy metódushoz nincs megadva láthatóság,
alapértelmezetten `public` láthatóságot kap

FELADAT

Készítsen PHP osztályt, amely leír egy felhasználót azonosítóval és jelszóval! Származtasson ebből az osztályból egy újabb hallgató osztályt a hallgató nevével! Készítse el az adatkezelő és kiíró metódusokat! Példányosítsa a hallgató osztályt és írja ki az adatait egy bekezdésbe!

HIBAKEZELÉS

A PHP-nak saját beépített hibakezelője van a szkriptek futása során keletkező hibákhoz. A hibát detektálja és szükség esetén leállítja a program futását. Továbbá a hibaüzeneteket két függvénnyel lehet beállítani

<code>error_reporting()</code>	meghatározza, hogy mely hibákról adjon értesítést
<code>error_log()</code>	a hibaüzeneteket hova irányítsa (pl. fájl, e-mail, stb.)

A definiált hibaszintek megtalálhatóak a következő linken:

<https://www.php.net/manual/en/errorfunc.constants.php>

HIBAKEZELÉS

Ha a beépített hibakezelés nekünk nem megfelelő vagy nem biztonságos, lehetőség van saját hibakezelést használni, amelynek több módja van:

1. Egyszerű futásmegszakítással
2. Saját hibakezelő függvényekkel
3. Kivételkezeléssel

HIBAKEZELÉS

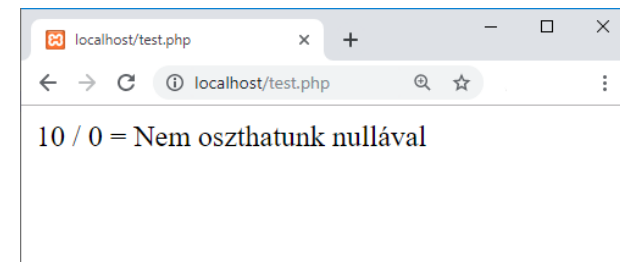
1. Futásmegszakítás: `die()`

A legegyszerűbb lehetőség, ha a hiba esetleges előfordulását előre leellenőrizzük és a fennállása esetén megszakítjuk a programfutást. Erre való a

`die(üzenet)`

függvény, amely kiírja az adott hibaüzenetet és megszakítja a programfutást

```
function hanyados($a, $b) {  
    echo $a . " / " . $b . " = ";  
    if ($b == 0) {  
        die("Nem oszthatunk nullával");  
    }  
    echo ($a/$b);  
}  
  
hanyados(10, 0);
```



HIBAKEZELÉS

2. Saját hibakezelő függvény

Lehetőség van saját hibakezelő függvényt írni

```
hibaFuggveny(szint, üzenet, fájl, sor, változók)
```

Paraméterek

szint a hiba szintje

üzenet hibaüzenet

fájl mely fájlban keletkezett a hiba

sor a fájl mely sorában keletkezett a hiba

változók az aktuális változók tömbje az értékükkel

Az első két paraméter kötelező, a többi opcionális

HIBAKEZELÉS

Kezelhető hibaszintek

E_WARNING (2)	Nem fatális hiba, a futás nem áll le
E_NOTICE (8)	Futásidejű értesítés
E_USER_ERROR (256)	Programozó által generált fatális hiba
E_USER_WARNING (512)	Programozó által generált figyelmeztetés
E_USER_NOTICE (1024)	Programozó által generált értesítés
E_RECOVERABLE_ERROR (4096)	Elkapható fatális hiba
E_ALL (8191)	Minden hiba és figyelmeztetés

HIBAKEZELÉS

Mivel a PHP-nak saját hibakezelő függvénye van, be kell állítani a saját függvényünket hibakezelőnek:

```
set_error_handler("függvénynév")
```

Ha csak a függvény nevét adjuk meg paraméterként, akkor minden hibaszintet ez kezel. Viszont megadható hibaszint is, amely meghatározza, hogy csak mely szinteket kezelje ez a függvény

HIBAKEZELÉS

A szkript futása során bárhol előidézhető hiba:

```
trigger_error(hibaüzenet, szint)
```

A hibaüzenet a hibakezelő függvénynek átadott üzenet. A szint a hiba szintje, amely lehet

- E_USER_ERROR
- E_USER_WARNING
- E_USER_NOTICE.

A szint megadása nem kötelező, ekkor E_USER_NOTICE az alapértelmezett szint

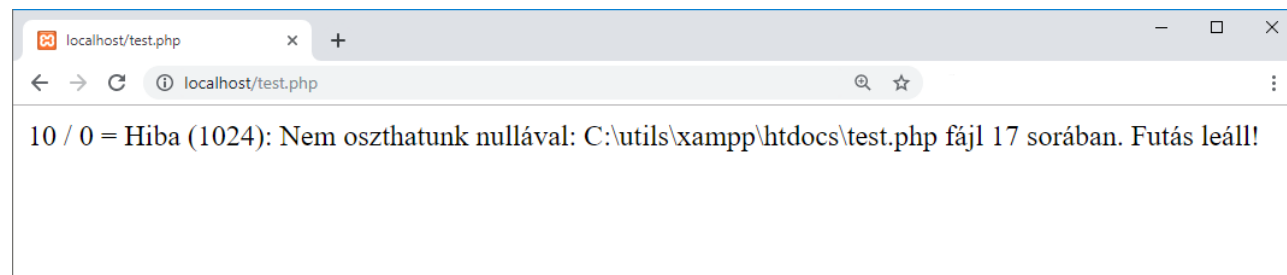
HIBAKEZELÉS

Példa

```
function hibakezelo($level, $msg, $f, $line){
    echo "Hiba (".$level.): ".$msg.: ".$f." fájl ".$line." sorában."
    echo "Futás leáll!";
    die();
}

function hanyados($a, $b){
    echo $a . " / " . $b . " = ";
    if ($b == 0){
        trigger_error("Nem oszthatunk nullával");
    }
    echo ($a/$b);
}

set_error_handler("hibakezelo");
hanyados(10,0);
```



3. Kivételkezelés

Kivételkezeléssel lehetőség van egy strukturált hibakezelésre.

Megváltoztatható a program futásának iránya bizonyos hibák (kivételek) előfordulásakor.

A hiba bekövetkezésekor a program jelenlegi állapotát elmenti és átadja a vezérlést a megfelelő hibakezelő pontra, amely meghatározza, hogy a program futása hogyan folytatódik.

HIBAKEZELÉS

Hibakezelő blokk

```
try {  
    ...           // hiba lehetséges előfordulásának helye  
}  
catch (kivétel) {  
    ...           // hibakezelő utasítások  
}  
... // további catch blokkok  
finally {  
    ...  
}
```

Több `catch` blokk is lehet egymás után

A `finally` ág mindenképp lefut. Ez az ág elhagyható

HIBAKEZELÉS

Kivétel dobása

A programból is dobható kivétel, amely elkapható a `catch` blokkal. Ehhez az `Exception` osztályt kell példányosítani.

```
throw new Exception(hibaüzent)
```

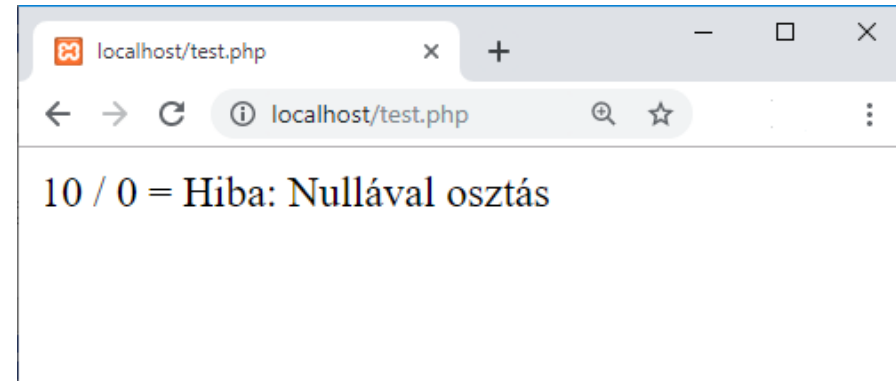
A kivételobjektum `getMessage()` metódusával lehet a hibaüzenete kiolvasni

```
try {  
    throw new Exception("hibaüzenet");  
}  
catch (Exception $e) {  
    echo $e->getMessage();  
}
```

HIBAKEZELÉS

Példa

```
function hanyados($a, $b){  
    echo $a . " / " . $b . " = ";  
    try {  
        if ($b == 0){  
            throw new Exception("Nullával osztás");  
        }  
        echo ($a / $b);  
    }  
    catch (Exception $e){  
        echo "Hiba: " . $e->getMessage();  
    }  
}  
hanyados(10, 0);
```



HIBAKEZELÉS

Felső szintű hibakezelő

Lehetőség van saját felső szintű hibakezelő függvényt írni. Ezt be kell állítani:

```
set_exception_handler("függvélynév")
```

Ez fogja lekezelni az összes nem elkapott hibát (ami nincs try-catch blokkal lekezelve)

```
function hibakezelo($e){  
    hibakezelés;  
}  
set_exception_handler("hibakezelo");  
throw new Exception("hibaüzenet");
```

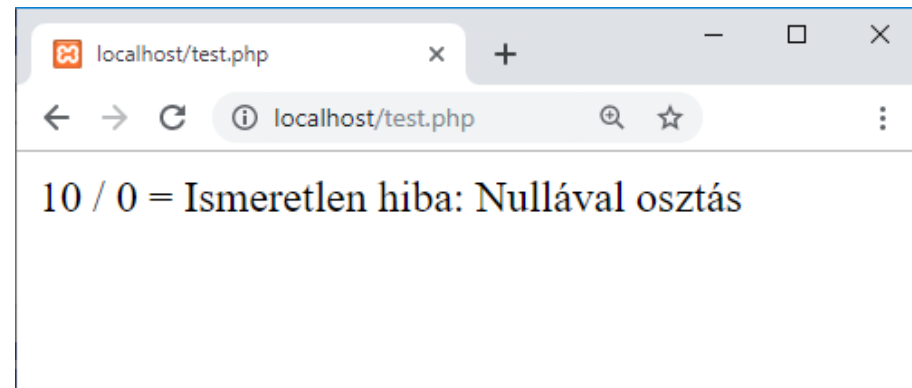
HIBAKEZELÉS

Példa

```
function hibakezelo($e){
    echo "Ismeretlen hiba: " . $e->getMessage();
}

function hanyados($a, $b){
    echo $a . " / " . $b . " = ";
    if ($b == 0){
        throw new Exception("Nullával osztás");
    }
    echo ($a / $b);
}

set_exception_handler("hibakezelo");
hanyados(10, 0);
```



TÖMBÖK

A tömb több különböző érték egy változóban tárolására szolgál

Index szerinti tömb

```
$st = array("Tercsi", "Fercsi")
```

az indexelés 0-tól kezdődik

```
echo $st[0] // Tercsi
```

elem hozzáadása a végére

```
$st[] = "Kata"
```

elem megadása index szerint, ha nincs ilyen, hozzáadja

```
$st[3] = "Klára"
```

TÖMBÖK

A tömb elemek lehetnek nem sorrendben is

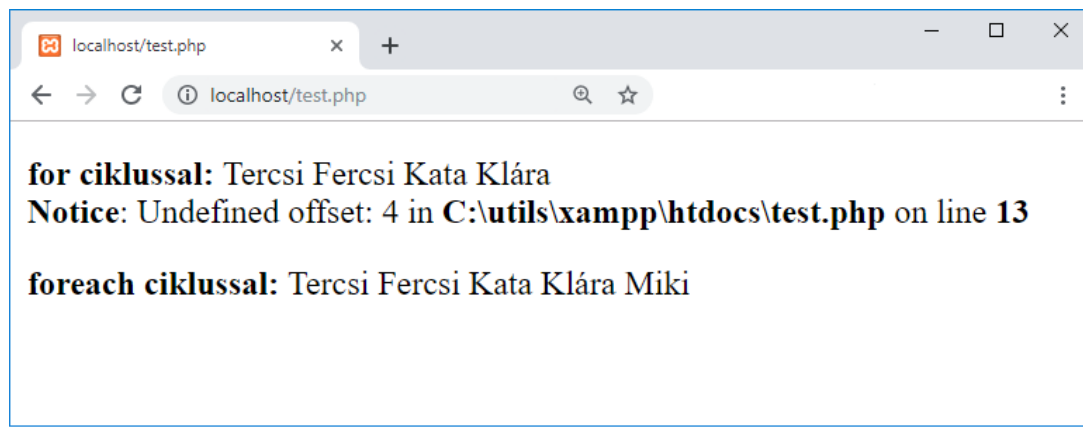
```
$st[10] = "Miki"
```

Ekkor a köztes elemek nem léteznek, így figyelni kell a tömb bejárására, nem alkalmas a szokásos `for` ciklus

```
for ($i=0;$i<count($st);$i++){  
    echo $st[$i] . " ";  
}
```

Ekkor hasznosabb a `foreach`

```
foreach ($st as $nev){  
    echo $nev . " "  
}
```



Asszociatív tömb

Az asszociatív tömbben az elemekhez nem sorszámot, hanem azonosítókulcsot rendelünk

```
$nev = array("veznev"=>"Mekk", "kernev"=>"Elek");
```

vagy

```
$nev["veznev"] = "Mekk";
```

```
$nev["kernev"] = "Elek";
```

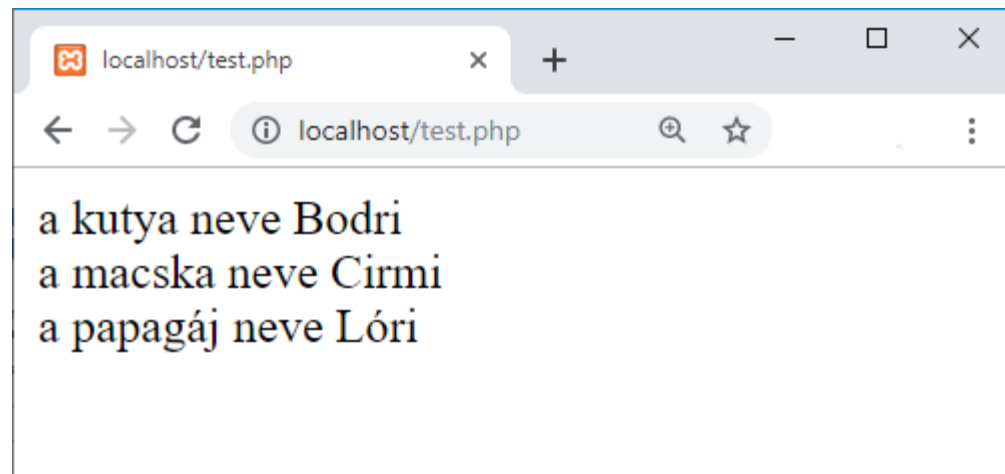
```
echo $nev["veznev"]. " ". $nev["kernev"];
```

Az azonosítónak egyedinek kell lenni

TÖMBÖK

Az asszociatív tömb bejárható a `foreach` ciklussal úgy, hogy az azonosító kulcsokat és az értékeket is kiolvassuk

```
$allatok = array("kutya"=>"Bodri",  
                "macska"=>"Cirmi",  
                "papagáj"=>"Lóri");  
foreach ($allatok as $kulcs => $ertek) {  
    echo "a " . $kulcs . " neve " . $ertek . "<br>";  
}
```



TÖMBÖK

Több dimenziós tömb

Több dimenziós tömböt kapunk, ha tömbbe tömböt ágyazunk

```
$allatok = array(array("kutya","macska"),  
                  array("sas","kolibri"))
```

A beágyazott tömbök elemszáma eltérhet

```
$allatok[] = array("csuka","keszeg","ponty");
```

Bejárása dimenzió szerinti indexeléssel

```
for ($i=0;$i<count($allatok);$i++) {  
    for ($j=0;$j<count($allatok[$i]);$j++) {  
        echo $allatok[$i][$j] . " ";  
    }  
    echo "<br>";  
}
```

TÖMBÖK

Tömb függvények (néhány fontosabb)

<code>count()</code>	tömb elemszáma
<code>print_r()</code>	tömb jól olvasható kiírása (nem csak tömbre használható)
<code>array_pop()</code>	törli és visszaadja az utolsó elemet
<code>array_push()</code>	új eleme(ke)t szúr be a tömb végére
<code>array_shift()</code>	törli és visszaadja az első elemet
<code>array_unshift()</code>	új eleme(ke)t szúr be a tömb elejére
<code>array_reverse()</code>	visszaadja a tömb elemeket fordított sorrendben
<code>array_search()</code>	keres egy elemet és visszaadja a helyét

TÖMBÖK

Tömb függvények (folytatás)

<code>array_walk()</code>	minden elemre meghív egy adott függvényt
<code>in_array()</code>	keres egy adott értéket a tömbben
<code>implode()</code>	szöveggé fűzi össze az elemeket adott szeparátorral
<code>sort()</code>	növekvő sorrendbe rendez
<code>rsort()</code>	csökkenő sorrendbe rendez
<code>shuffle()</code>	összekeveri a tömb elemeit

Asszociatív tömb függvények (néhány fontosabb)

<code>array_key_exists()</code>	keres egy kulcsot a tömbben
<code>array_keys()</code>	visszaadja a kulcsokat
<code>array_values()</code>	visszaadja az értékeket
<code>asort()</code>	növekvő sorrendbe rendez (érték)
<code>ksort()</code>	növekvő sorrendbe rendez (kulcs)
<code>arsort()</code>	csökkenő sorrendbe rendez (érték)
<code>krsort()</code>	csökkenő sorrendbe rendez (kulcs)
<code>compact()</code>	változókból egy tömböt csinál

FELADAT

1. Készítsen PHP tömböt a hónapok neveivel! Hozzon létre egy HTML listát a tömbben lévő nevekből!
2. Készítsen PHP asszociatív tömböt, amelyben tárolja egy verseny versenyszámainak nyerteseinek nevét. Például futás: Pisti, távolugrás: Józsi, stb. Írja ki formázottan a verseny nyerteseit!
3. Készítsen két dimenziós PHP tömböt, ahol a sorok egy asszociatív tömb csapatok nevével azonosítva, a sorokban pedig a csapattagok nevei vannak tárolva indexelt tömbbel! Írja ki egy HTML táblázatba a csapatokat és tagjaikat! Nem biztos, hogy minden csapatnak ugyanannyi tagja van.

SZÖVEGEK

Stringet létre lehet hozni aposztróf vagy idézőjel használatával.

```
$s = "Helló Világ"
```

vagy

```
$s = 'Helló Világ'
```

A szövegben speciális karakterek szerepeltetéséhez a \ jelentést elnyomó (escape) karakter használata szükséges

```
$s = 'Rock \'n\' Roll'
```

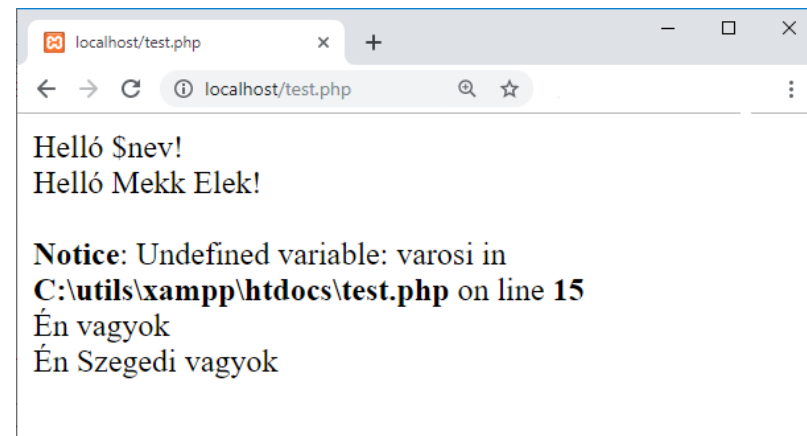
Különbség

Az idézőjellel létrehozott szövegben szereplő \$ jel esetén megpróbál változót felismerni és azt behelyettesíteni.

```
$nev = "Mekk Elek";  
echo 'Hello $nev!';  
echo "Helló $nev!";
```

Ha nem egyértelmű a változó neve, akkor { } zárójeleket kell használni

```
$varos = "Szeged";  
echo "Én $varosi vagyok"; // hibás  
echo "Én ${varos}i vagyok";
```



SZÖVEGEK

Függvények (néhány fontosabb)

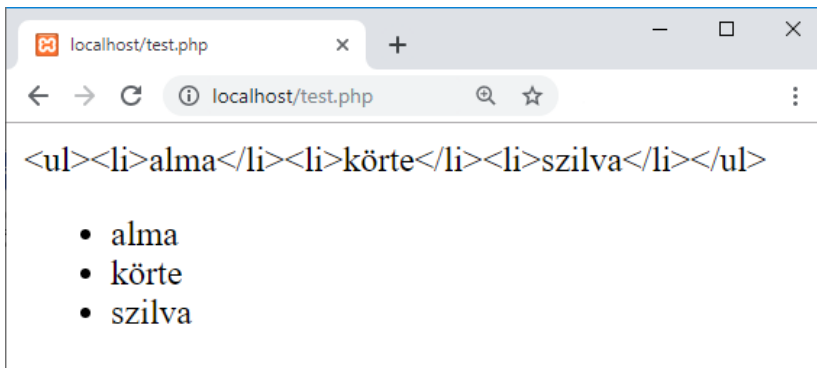
<code>echo()</code>	kiír egy vagy több szöveget
<code>print()</code>	kiír egy vagy több szöveget
<code>printf()</code>	formázottan kiír egy vagy több szöveget
<code>strlen()</code>	szöveg hosszát adja vissza
<code>strrev()</code>	a szöveg visszafele
<code>strpos()</code>	szövegben keres szöveget, első előfordulás
<code>strspos()</code>	szövegben keres szöveget, utolsó előfordulás
<code>substr()</code>	szöveg egy részét adja vissza
<code>trim()</code>	adott karaktereket vág le az elejéről és a végéről (alapból white-space karaktereket)

Függvények (folytatás)

<code>implode()</code>	egy tömb elemeit összefűzi szöveggé
<code>explode()</code>	egy szövegből tömböt készít
<code>htmlspecialchars()</code>	speciális karaktereket entitássá konvertálja
<code>htmlspecialchars_decode()</code>	az entitásokat karakterekké konvertálja
<code>md5()</code>	a szöveg md5 kódját adja
<code>sha1()</code>	a szöveg sha1 kódját adja
<code>strip_tags()</code>	törli a tag-eket a szövegből

FELADAT

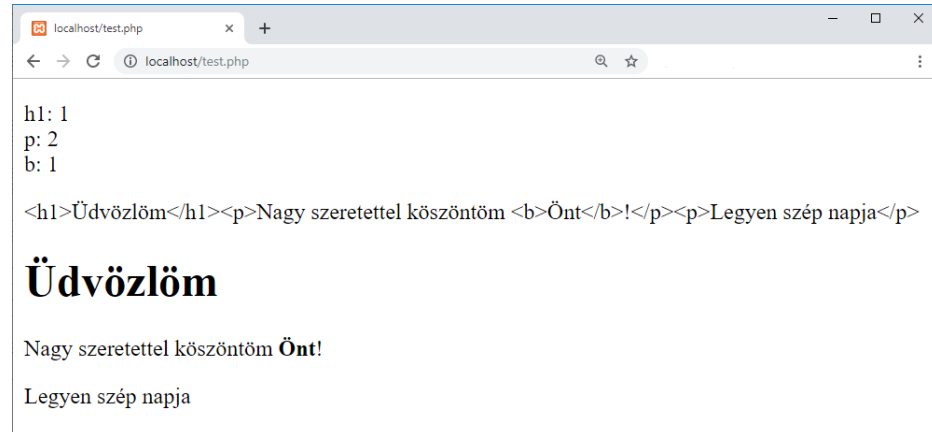
1. Adott egy tömb, amelynek az elemeiből készítsen egy HTML listát! Írja ki a lista HTML kódját és a hozzá létre a listát!
2. Egy változóban HTML forráskód található! Összesítse, hogy mely tag-ből hány darab található a kódban, írja ki a HTML forráskódot, illetve a HTML eredményét!



A screenshot of a web browser window with the address bar showing 'localhost/test.php'. The main content area displays the HTML code: `almakörteszilva`. Below the code, the rendered HTML is shown as an unordered list with three items: 'alma', 'körte', and 'szilva'.

```
<ul><li>alma</li><li>körte</li><li>szilva</li></ul>
```

- alma
- körte
- szilva



A screenshot of a web browser window with the address bar showing 'localhost/test.php'. The main content area displays the HTML code: `<h1>Üdvözlöm</h1><p>Nagy szeretettel köszöntöm Önt!</p><p>Legyen szép napja</p>`. Below the code, the rendered HTML is shown as a greeting message: 'Üdvözlöm', 'Nagy szeretettel köszöntöm Önt!', and 'Legyen szép napja'.

```
<h1>Üdvözlöm</h1><p>Nagy szeretettel köszöntöm <b>Önt</b>!</p><p>Legyen szép napja</p>
```

Üdvözlöm

Nagy szeretettel köszöntöm **Önt**!

Legyen szép napja

ŰRLAPOK

A HTML űrlapokon keresztül a felhasználó adatokat tud küldeni szerver oldalra. Az adatátadás két módon lehetséges:

`get` láthatóan az URL-ben

Az URL végéhez hozzáfűzi az átadott értékeket

```
http://www.cim.hu/oldal.php?valtozo1=ertek1&valtozo2=ertek2
```

`post` belső, nem látható csatornán

A `post` mód biztonságosabb illetve nincs korlátozva az átküldött adat mennyisége

Az elküldés módját a `form` tag `method` attribútuma határozza meg

```
<form method="get/post">...</form>
```

ŰRLAPOK

A szerver oldalon mindkét módon átküldött adatot egyformán tudjuk kezelni egy-egy beépített szuper-globális tömb változóval:

```
$_GET
```

```
$_POST
```

Mindkét változó olyan asszociatív tömb, ahol az elemek azonosítója az űrlap elemeinek a neve, az értékek pedig az űrlap elemeinek az értékei.

```
$_POST["nev"]
```

```
$_GET["nev"]
```

ÜRLAPOK

Bármely módon küldött adatok kiolvashatók egy harmadik szuper-globális változóból

```
$_REQUEST
```

amely szintén asszociatív tömbként működik

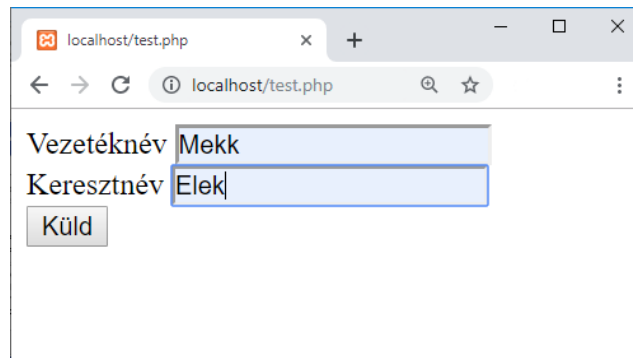
```
$_REQUEST["nev"]
```

Biztonsági okból ennek használata nem javasolt, mert nem ellenőrizhető, hogy milyen módon jött át az adat

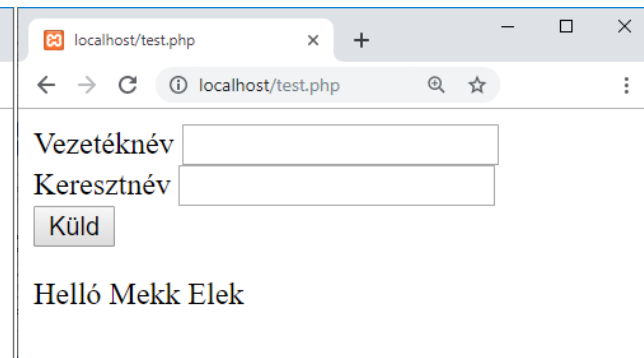
ŰRLAPOK

Példa

```
<form method="post" action="test.php">
<label for="veznev">Vezetéknév <input type="text"
    name="veznev" id="veznev"><br>
<label for="kernev">Keresztnév <input type="text"
    name="kernev" id="kernev"><br>
<input type="submit" value="Küld">
</form>
<?php
if (isset($_POST["veznev"]) && isset($_POST["kernev"])) {
    echo "<p>Helló " . $_POST["veznev"] . "
        " . $_POST["kernev"] . "</p>";
}
?>
```



A screenshot of a web browser window showing a form at localhost/test.php. The form contains two text input fields: 'Vezetéknév' with the value 'Mekk' and 'Keresztnév' with the value 'Elek'. Below the inputs is a 'Küld' button. The browser's address bar shows 'localhost/test.php'.



A screenshot of a web browser window showing the same form at localhost/test.php after submission. The input fields are now empty. Below the 'Küld' button, the text 'Helló Mekk Elek' is displayed. The browser's address bar shows 'localhost/test.php'.

ŰRLAPOK

Lehetőség van adatokat tömbként átadni, ehhez a megfelelő űrlapelemeknek azonos nevet kell adni használva a tömböt jelölő szögletes zárójeleket

```
<input type="text" name="nevek[]" id="veznev">  
<input type="text" name="nevek[]" id="kerneve">
```

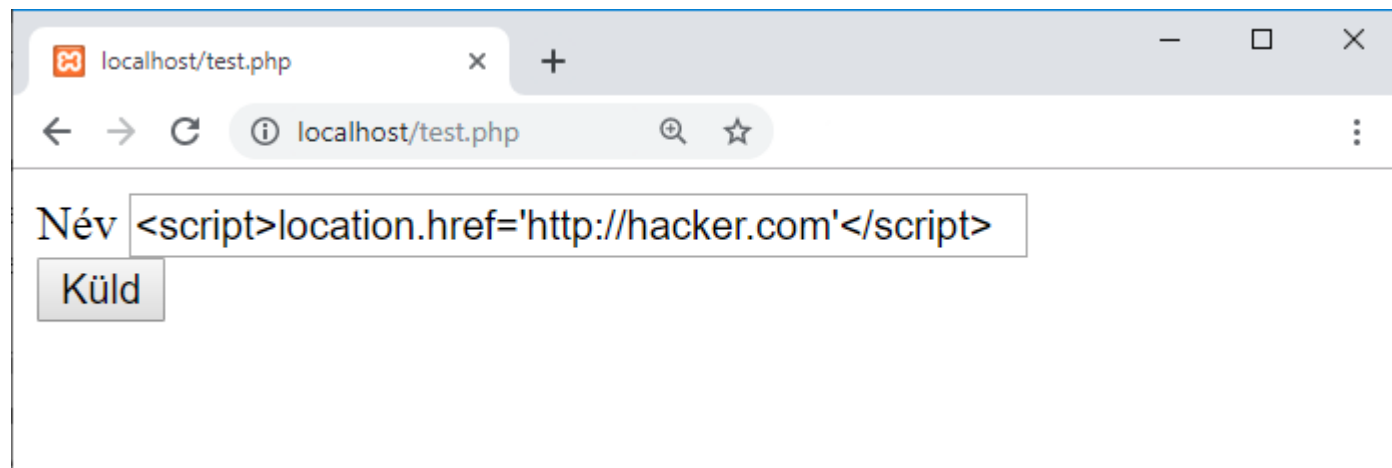
Ekkor az ilyen nevű PHP változó tömbként tartalmazza a megfelelő űrlapelemek átküldött értékeit

```
foreach ($_POST["nevek"] as $nev) {  
    echo $nev . " ";  
}
```

Validáció

Az űrlapokon keresztül támadásokat indíthatnak a weboldalunk ellen úgy, hogy a mezők értékébe scriptet, SQL utasítást vagy HTML kódot írnak.

Ez a feldolgozáskor lefut és például átirányítja a felhasználót egy másik oldalra:



The screenshot shows a web browser window with the address bar displaying 'localhost/test.php'. The page content includes a form with a label 'Név' (Name) and a text input field containing the malicious script: `<script>location.href='http://hacker.com'</script>`. Below the input field is a 'Küld' (Send) button.

ŰRLAPOK

Ezek kivédésére érdemes a kapott adatokat átengedni több szöveg konvertáló szűrőn, amely megakadályozza a kód lefutását

<code>trim()</code>	Adott karaktereket levág a szöveg két végéről
<code>htmlspecialchars()</code>	Adott HTML karaktereket entitássá alakítja át
<code>stripslashes()</code>	Törli a visszaper karaktereket (escape karakter)

További lehetőség a kapott érték formai ellenőrzése reguláris kifejezéssel használata

<code>preg_match()</code>	egy szöveget hasonlít egy reguláris kifejezéssel
---------------------------	--

FELADAT

Készítsen egy regisztrációs űrlapot felhasználó névvel, jelszóval, e-mail címmel!

Határozzon meg szabályokat mindhárom mezőre!

Ellenőrizze PHP szkripttel a felvitt adatok érvényességét, hiba esetén írjon hibaüzenetet a megfelelő beviteli mezőhöz!

Helyes adatok esetén írjon üdvözlő szöveget a kapott felhasználónévvel!

ADATBÁZIS

A dinamikus weboldalak az egyik fontos forrása gyakran egy adatbázis, mivel az oldal dinamikus tartalmát adó adatok általában adatbázisban vannak letárolva.

A PHP weboldalak egyik leggyakoribb adatbáziskezelője a MySQL ingyenes adatbázisszerver.

Megjegyzés: Használtak még a NoSQL adatbázisok is, amelyek nem SQL adatbázisban tárolják az adatokat, hanem pl fájlként. Ilyen pl a MongoDB, illetve ilyen fájlformátumok az XML vagy a JSON.

A MySQL adatbázisok kezelésére két függvénykönyvtár használható a PHP-ban:

- MySQLi
- PDO

A MySQLi csak MySQL adatbázishoz jó, míg a PDO több adatbáziskezelőhöz tud csatlakozni, így ha a webalkalmazás átkerül egy másik adatbáziskezelőre, akkor a MySQLi-t használó kód jelentős részét át kell írni, míg a PDO-t használó kód csak kis részét. Ezért a PDO egy hasznos eszköz hosszú távú tervezéskor.

PDO (PHP Data Object)

- Adatbáziskezelő objektum (objektumorientált szemlélet)
- 12 különböző adatbáziskezelő ismerete (MySQL, PostgreSQL, Oracle, MSSQL, stb.)
- Biztonságos – véd az SQL-befecskendezés ellen
- Névvel hivatkozható paraméterek
- Előkészített lekérdezések
- Adatbáziskezelési kivételek

Kapcsolódás

Az adatbázis használatához először kapcsolódni kell hozzá, ami a PDO objektum példányosításakor történik meg:

```
$conn = new PDO('mysql:host=localhost;  
dbname=test', $felh, $jelszo)
```

Az első paraméterben megadjuk az adatbázis típusát (*mysql*), címét (*localhost*) és az adatbázis nevét (*test*). Majd megadjuk a kapcsolódáshoz szükséges felhasználói azonosítót és jelszót.

Kapcsolat megszakításához kiürítjük a változót

```
$conn = null
```

Hibakezelés

A fellépő hibák kezelésére érdemes beállítani a kivétel alapú hibakezelést

```
$conn->setAttribute(PDO::ATTR_ERRMODE,  
PDO::ERRMODE_EXCEPTION)
```

Ezután az adatbázisműveletek során fellépő hibáknál a PDO által dobott kivételek elkaphatók a `try...catch...` blokkal

Nem javasolt saját PDO kivételt dobni!

ADATBÁZIS

```
$felh = "valaki";
$jelszo = "valami";
try {
    $conn = new PDO('mysql:host=localhost; dbname=test',
    $felh, $jelszo);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
    PDO::ERRMODE_EXCEPTION);

    // adatbáziskezelő utasítások

}
catch (PDOException $e){
    echo "Adatbázis hiba: ".$e->getMessage();
}
catch (Exception $e){
    echo "Hiba: ".$e->getMessage();
}
```

Előkészített lekérdezés

Az előkészített lekérdezés során először csak egy paraméterezett lekérdezést küldünk a szervernek, a paraméter : (kettőspont) jellel kezdődik

```
$query = $conn->prepare("SELECT nev FROM User  
WHERE id=:userId")
```

Később a paraméterekhez konkrét értékeket rendelünk típusával, így a lekérdezés többször futtatható, amikor más-más értéket adunk meg paraméterként

```
$query->bindParam(":userId", 1, PDO::PARAM_INT)
```

A lekérdezés lefuttatás

```
$query->execute()
```

Előkészített lekérdezés előnyei

- Az értelmezőnek csak egyszer kell feldolgozni a lekérdezést
- A lekérdezést csak egyszer kell elküldeni, később már csak a paraméterek értékeit kell küldeni
- Véd az SQL befecskendezés ellen
(https://www.w3schools.com/sql/sql_injection.asp)

Természetesen minden SQL utasítást így érdemes kezelni (SELECT, INSERT, UPDATE, DELETE, stb.)

Lekérdezés

A lekérdezést eredményét soronként ki lehet olvasni

```
$row = $query->fetch()
```

Ha nincs több kiolvasható sor, hamissal tér vissza, így jól használható iterációban

```
while ($row = $query->fetch()) { ... }
```

Megadható, hogy az eredményt indexelt, asszociatív, vagy mindkét (alapértelmezett) tömbbel adja vissza

```
fetch(PDO::FETCH_NUM | PDO::FETCH_ASSOC |  
      PDO::FETCH_BOTH)
```

A tömb azonosítói a lekérdezett oszlopok nevei, vagy sorszámai

```
$row[0]
```

```
$row["nev"]
```

A lekérdezés oszlopaihoz változókat lehet rendelni, így a kiolvasáskor a sor adott mezői automatikusan a változóba kerülnek

```
$query->bindColumn("nev", $nev);
```

```
while ($row = $query->fetch(PDO::FETCH_BOUND)) {  
    echo "Név: " . $nev;  
}
```

A kiolvasott sorból objektum is generálható

```
$obj = $query->fetchObject();  
echo $obj->nev
```

A lekérdezés eredménye egyben is kiolvasható egy két dimenziós tömbbe

```
$tomb = $query->fetchAll()
```

Egy oszlop is kiolvasható egy tömbbe (pl. első oszlop)

```
$oszlop = $query->fetchAll(PDO::FETCH_COLUMN, 0)
```

Kiolvasható a lekérdezés eredményének a sorainak a száma

```
$query->rowCount()
```

Példa

```
$jogszin = 1;
$felh = "valaki";
$jelszo = "valami";
try {
    $conn = new PDO('mysql:host=localhost; dbname=test', $felh, $jelszo);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = $conn->prepare("SELECT id, nev FROM USER WHERE jog=:jogszint");
    $query->bindParam(":jogszint", $jogszint, PDO::PARAM_INT);
    $query->bindColumn("id", $id);
    $query->bindColumn("nev", $nev);
    $query->execute();
    while ($row = $query->fetch(PDO::FETCH_BOUND)) {
        echo "Felhasználó: " . $nev . " (" . $id . ")";
    }
}
catch (PDOException $e) {
    echo "Adatbázis hiba: ".$e->getMessage();
}
```

Tranzakció

Több műveltet lehet egy tranzakcióba gyűjteni.

A tranzakciót el kell indítani

```
$conn->beginTransaction()
```

ha nem történt hiba, a tranzakcióba tartozó utasításokat véglegesíteni kell

```
$conn->commit()
```

vagy hiba esetén vissza lehet forgatni a műveleteket

```
$conn->rollback()
```

FELADAT

1. Készítsen egy regisztrációs weboldalt, ahol a felhasználó megadja a felhasználó nevét, jelszavát és teljes nevét! Az adatokat tárolja el adatbázisban! Ellenőrizze, hogy a felhasználó név foglalt-e már!
2. Készítsen egy bejelentkező felületet, ahol a felhasználó megadhatja a felhasználónevet és jelszavát! A helyességet ellenőrizze az adatbázis alapján! Helyes adatok esetén üdvözlje a felhasználót a teljes nevét használva!

SESSION

Amikor valamilyen értékeket kell megjegyezni több oldalon keresztül, akkor azokat el lehet tárolni egy session-ben. A session a szerveren egy ideiglenes tároló hely, amit a szerver kezel és egy böngészőhöz (felhasználóhoz) köt. Így az itt tárolt adatok elérhetőek több oldalon keresztül, amit erről a böngészőről nyitnak meg.

A session akkor szűnik meg, amikor a böngészőt bezárják, vagy a program szünteti meg.

Mivel a szerveren tároltak az adatok, a kliens nem látja őket

SESSION

A session nem automatikus, minden oldalon el kell indítani, ha használni szeretnénk az adatait

```
session_start()
```

A session programból is megszüntethető

```
session_destroy()
```

A session változók egy szuper-globális tömb változóba kerülnek, ahol írhatók és olvashatók

```
$_SESSION["változó"]
```

Az összes session adat törölhető egyben (kiüríti a \$_SESSION változót)

```
session_unset()
```


SESSION

oldal1.php

```
<?php
session_start();
$_SESSION["user"] = "valaki";
?>
```

oldal2.php

```
<?php
session_start();
if (isset($_SESSION["user"])) {
    echo $_SESSION["user"];
}
?>
```

FELADAT

A korábbi bejelentkező felületét fejlessze tovább úgy, hogy sikeres bejelentkezéskor tárolja el session-ben a felhasználó nevét és teljes nevét! Bővítse a site-t további oldalakkal, amelyeket csak akkor tekinthet meg a felhasználó, ha be van jelentkezve (session-ben el van tárolva), egyébként egy hibaüzenettel irányítsa vissza a bejelentkező oldalra!

Készítsen egy kijelentkező funkciót, amelyre törli a session adatokat és visszatér a bejelentkező oldalra!

COOKIE

A cookie-k (sütik) olyan adatok, amelyet a szerver a kliens gépén helyez el. A kliens minden kéréssel együtt ezeket az adatokat is elküldi a szervernek.

Így ha a felhasználó ugyanarról a helyről látogat el az oldalra, a letárolt süti adatok már ismertek lesznek az alkalmazásban.

Mivel a sütik a kliens oldalon vannak tárolva, fontosabb adatot nem célszerű így tárolni. Illetve a felhasználó letilthatja és törölheti is a sütiket, így ezektől az adatoktól függetlenül is működnie kell az alkalmazásnak

COOKIE

Cookie tárolása

Mindenképp minden más output előtt kell szerepeljen (pl. a `<html>` tag előtt)

```
setcookie(név, érték, lejárát, útvonal,  
domain, secure, httponly)
```

```
setcookie("user", "tesztelek", time()+60*60*24,  
"www.pelda.com", "/")
```

Paraméterek

név süti neve

érték süti tárolt értéke

lejárát a süti lejárat ideje, utána törlődik (megadva másodpercben)

COOKIE

Paraméterek

<i>útvonal</i>	a domain-en belül csak mely útvonalra érvényes, "/" jelentése a teljes domain
<i>domain</i>	mely domain-re érvényes
<i>secure</i>	<i>true</i> esetén csak akkor használható a süti, ha HTTPS-n keresztül megy az adat (alapértelmezett a <i>false</i>)
<i>httponly</i>	<i>true</i> esetén csak HTTP protokollon keresztül olvasható, azaz pl JavaScript kóddal nem (alapértelmezett a <i>false</i>)

Csak a név paraméter a kötelező

COOKIE

A tárolt értékek a `$_COOKIE` szuper-globális tömb változóba kerülnek, ahonnan kiolvashatók

Kiolvasáskor a létezésüket először célszerű ellenőrizni az `isset()` utasítással

Úgy lehet ellenőrizni, hogy a kliens engedélyezi-e a sütik tárolását, hogy létrehozunk egy sütit, majd ellenőrizzük, hogy létrejött-e

```
setcookie("user", "tesztelek", time()+60*60*24,  
    "www.pelda.com", "/")  
if (isset($_COOKIE["user"])) {  
    echo "Sikeres tárolás: " . $_COOKIE["user"];  
}
```

COOKIE

Cookie módosítása

Egy tárolt cookie értékének módosításához csak be kell állítani az új értéket a `setcookie()` utasítással

Cookie törlése

A cookie a mentéskor megadott lejáratási idő után automatikusan törlődik. Ha ennél hamarabb szeretnénk törölni, akkor a `setcookie()` utasítással adjunk meg már elmúlt lejáratási időt

```
setcookie("user", "", time() - 3600)
```

FELADAT

A korábbi bejelentkezési felületét egészítse ki úgy, hogy bejelentkezéskor a felhasználó beállíthassa, hogy szeretne bejelentkezve maradni. Ekkor a felhasználó nevét és azonosítóját mentse el sütiként! Ellenőrizze, hogy lehet-e sütit menteni! Ha nem, írjon erről üzenetet a felhasználónak!

Ha a lejáratí idő előtt a felhasználó visszatér az oldalra, automatikusan jelentkeztesse be!

A kijelentkezési funkciót bővítsé úgy, hogy törli a mentett sütit is!

FORRÁS

A tananyag a PHP hivatalos oldalára épül

<https://www.php.net/manual/en/index.php>



Ajánlott irodalom

A W3C hivatalos oktató weboldala

<https://www.w3schools.com/php7/default.asp>

w3schools.com