WEBES ALKALMAZÁSFEJLESZTÉS 1.

Horváth Győző

Egyetemi adjunktus

1117 Budapest,
Pázmány Péter sétány 1/C, 2.420
Tel: (1) 372-2500/1816

Bevezetés

Bevezetés

- Tendenciák
- Webes jelen
- Sokszínűség
- Architektúra variánsok
  - LAMP
  - MEAN
  - Mikro keretrendszerek
- Motiváció

Tantárgy célja

- (Viszonylag) időtálló ismeretek
- Programozási minták
- Tipikus megközelítések

- Szükségszerűen elavuló ismeretek
  - Nyelv
  - Keretrendszer

Tantárgy tematikája

- Kliensoldali világ (JavaScript)
- Szerveroldali világ (PHP)

Követelmények

Korábbi tárgyak (BSc)

- Webfejlesztés 1.
  - Statikus weboldalak készítése
  - HTML, CSS
  - Ergonómia
- Webfejlesztés 2.
  - Javascript
  - PHP
- További tárgyak
  - ASP.NET

Média blokk tárgyai (MSc)

- Honlapszerkesztés
- Webes alkalmazásfejlesztés 1.
- Webes alkalmazásfejlesztés 2.
- Adatbázisok a weben
- Webanimáció

Webes alkalmazásfejlesztés 1.

- JavaScript
  - Modern kliensoldali webprogramozás
- PHP
  - Korszerűbb szerveroldali webprogramozás

Honlap

- http://webprogramozas.inf.elte.hu/weaf1.php

Feltételezett ismeretek

- HTTP protokoll alapszintű ismerete
- Kliens-szerver architektúra áttekintése
- HTML és CSS ismerete
- JavaScript nyelvi elemek alapszintű ismerete
  - Vezérlési szerkezetek, egyszerű típusok, tömb, függvény
  - BOM, DOM, eseménykezelés
- PHP alap nyelvi elemek ismerete
  - Egyszerű típusok, tömb
  - Vezérlési szerkezetek, Függvények
  - Űrlapelemek feldolgozása

Tárgy felépítése

- A tárgy 1+1-es
- Előadás
  - Kedd 11:00-12:00, 0-817 Dudich Endre terem
- Gyakorlat
  - 2 órás gyakorlatok minden második héten
  - 1. csoport: kedd 12:00-14:00, PC9 (?)

Követelmények

- Előadásra nagyon ajánlott bejárni
- Gyakorlatokra kötelező bejárni
  - Legfeljebb 3 hiányzás
- Két zh
  - JavaScript
  - PHP
- Beadandó
  - Komplex feladat

Követelmények

- Beadandó
  - A feladatokat fel kell tölteni a webprogramozas.inf.elte.hu szerverre
  - Értékelése: 1..5
- Zh
  - Feladatok megoldása
  - Tipikusan 4 feladat

Határidők

- JavaScript zh
  - 2015. március 31., PC9
- PHP zh
  - 2013. május 19. kedd, 12-15, Lovarda (PC9?)
- Beadandó
  - 2013. május vége
- Pótzh
  - 2013. május 27. szerda, 9-15, PC3

Regisztráció – webszerver

- A kurzusra jelentkezettek számára létrehozunk felhasználói fiókokat a webprogramozas.inf.elte.hu szerveren
- Felhasználónév: hatjegyű neptun kód kisbetűvel (neptunkod)
- Jelszó: almafa1
- A jelszó belépés után a passwd paranccsal változtatható meg (Putty)
- Fájlok feltöltése SCP vagy SFTP klienssel történhet (WinSCP)
- A dokumentumokat a public_html mappába kell helyezni
- Az ide elhelyezett dokumentumok a http://webprogramozas.inf.elte.hu/~neptunkod/ címről érhetőek el

Regisztráció – MySQL

- Mindenkinek létrehozunk saját adatbázis hozzáférést és adatbázist
- MySQL szerver
- Felhasználónév: neptunkod
- Jelszó: almafa1
- Adatbázis: weaf1_neptunkod
- Elérni a webprogramozas.inf.elte.hu szerverről localhostban lehet a 3306-os porton (tunnelezés lehetséges)

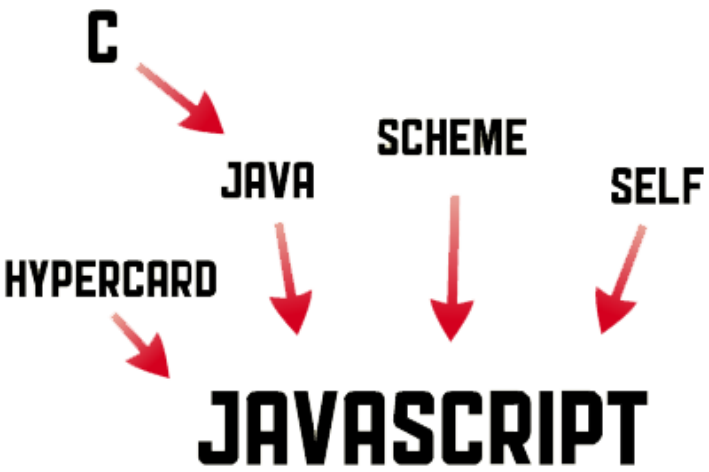
JavaScript története

A JavaScript története

- 1996: Netscape cég felkéri Brendan Eich-et, hogy fektesse le egy olyan programozási nyelv alapjait, amelyekkel interaktívvá tehetők weboldalak
- Cél: Java pluginek elérése nem Javás programozóknak
- Súlyos időhiány: hetek alatt tervezték meg
- →átgondolatlan és hibás részek kerültek a nyelvbe

A JavaScript története

- Mintaként szolgáló nyelvek



- C és Java: szintaktika
- Scheme: kifejezésekre épülő funkcionális prog. nyelv
- Self: prototípus alapú objektum-orientált prog. nyelv
- HyperCard: eseménykezelés, GUI → DOM

A JavaScript története

- Elnevezések
  - LiveScript
  - JavaScript: marketing miatt a Java programozók átcsábítására
- Szabvány: ECMAScript
  - Európai Informatikai és Kommunikációs Rendszerek Szabványosítási Szövetsége (ECMA)
- Microsoft
  - JScript

A JavaScript története

- Böngészők háborúja: Netscape vs. Microsoft
  - Más-más irányba mennek, kölcsönösen nem fogadják el egymás javaslatait
  - Nyelv és böngészők gyors fejlődése
  - CSS
  - Szabványosítás elmaradt → inkompatibilitási problémákhoz vezet
- 1999: Sötét középkor
  - Semmi nem történik kliensoldalon
  - Szerveroldali technológiák fejlődése

A JavaScript története

- 2005: JavaScript újrafelfedezése
  - AJAX (Jesse James Garrett)
  - 6 éves technológia
- Professzionális nyelv

Eszközök

Eszközök

- IDE
  - Notepad++
  - Sublime Text 3
  - Netbeans IDE
  - Eclipse
  - WebStorm
- Futtató környezet
  - Böngészők
  - Node.js
- Webfejlesztői eszköztárak és konzolok
- Dokumentáció
  - MDN
  - Stackoverflow

Eszközök

- JavaScript könyvek
  - Eloquent JavaScript
  - JavaScript: The Good Parts
  - JavaScript Patterns
- Együttműködés
  - Git, local development, pull requests

Eszközök

- HTTP kérések csökkentése
  - UglifyJS
  - Closure Compiler
  - JSMin
  - RequireJS optimizer
- Automatizálás
  - Grunt, Gulp
  - NPM, Bower, JSPM
- Kódminőség
  - JSLint
  - JSHint
- Tesztelés
  - Mocha
  - Chai
  - Sinon.js
  - Jasmine
  - Karma

JavaScript konzolok


```
var valtozo = 'wanna';
console.log(valtozo);                    // => "wanna", a paraméterül kapott
                                                változót kiírja
console.log(1, valtozo, 'rock');         // => "1 wanna rock", vesszővel ellátva
                                                több különböző típusú paraméter
                                                is átadható
console.error('Hiba :(');                // => Kiírja, hogy "Hiba :(", majd egy
                                              teljes stack trace-et ad, ami
                                              alapján vissza tudjuk követni, hogy
                                              milyen függvényeket is hívtunk meg,
                                              mire ide jutottunk
console.trace();                         // A stack trace kézi lekérdezése
console.time("Ciklus sebessége");
for (var i=0, j=0; i<1000; i++) j = j + i;
console.timeEnd("Ciklus sebessége");          // => "Ciklus sebessége: 2ms",
                                                 sebesség mérése, a time és a
                                                 timeEnd közötti időt
                                                 mérhetjük le
console.group('Csoport');
console.log('Csoportba fűzhetőek az egyes üzeneteink!');
console.groupEnd('Csoport');
```

JSLint


```
function fakt(n) {
  var ered = 1;
  if (n==0) {
      ered = 1;
  }
  else {
    for (var i = 1; i<=n; i++) {
      ered *= i;
    }
  }
  return ered;
}
```

```
function fakt(n) {
    "use strict";
    var ered = 1, i;
    if (n === 0) {
        ered = 1;
    } else {
        for (i = 1; i <= n; i += 1) {
            ered *= i;
        }
    }
    return ered;
}
```

JavaScript nyelvi alapok

JavaScript nyelvi jellemzői

- Interpretált
- Dinamikusan típusos
  - Változó típusa a benne tárolt érték típusától függ
- Gyengén típusos
  - Automatikus konverziók
- Szintaxisa C, Java-szerű
  - Vezérlési szerkezetek, operátorok
- Funkcionális elemek
  - Függvény mint első osztályú objektum

Adattípusok

- Értéktípusok
  - Egyszerű adattípusok
- Referenciatípusok
  - Referencia: mutató egy objektum aktuális helyére
  - Objektumok és tömbök

Literál forma

- Literál: fix érték megjelenítésére használt szimbólum
- JavaScript nyelv minden adattípusához létezik ilyen

Értéktípusok

- Primitív típusok
  - szám (Number)
  - szöveg (String)
  - logikai (Boolean)
  - null
  - undefined
- Értékük megváltoztathatatlan

- Csomagolóobjektumok
  - Number
  - String
  - Boolean

```
var mvv = "teafilter"; // az mvv név nyilván azt jelenti, hogy „megváltoztathatatlan
                       változóm”
console.log( mvv.replace('tea', 'kávé') );  // => kávéfilter
console.log( mvv );                         // => teafilter
mvv = mvv.replace('tea', 'kávé');
console.log( mvv );                         // => kávéfilter
```

Típuslekérdezés

- typeof operátor
- Ha a változó nem létezik → undefined

```
typeof "a";     // => 'string'
typeof 2;       // => 'number'
typeof true;    // => 'boolean'
typeof {};      // => 'object'
typeof asd;     // => 'undefined'
```

- constructor tulajdonság lekérdezése

```
// Ellenőrizzük, vajon a num szöveg-e
if ( num.constructor == String ) {
    // Ha igen, alakítsuk számmá
    num = parseInt(num, 10);
}
```

Típuslekérdezés

Variable	typeof Variable	Variable.constructor
{ an: “object” }	object	Object
[“an”, “array”]	object	Array
function(){}	function	Function
“a string”	string	String
55	number	Number
true	boolean	Boolean
new User()	object	User

Automatikus típuskonverzió

- Lehetőleg kerüljük
- Kézi konvertálás

```
assertEquals(1, 2 / "2");
assertEquals(4, "2" * 2);
assertEquals("11", 1 + "1");
assertEquals(2, 1 + true);
assertEquals(0, null + false);
assertEquals("It is true", "It is " + true);
assertEquals(true, "3" < 4);
```

Számok

Szám literál

- Egyetlen típus: Number
- Háttérben lebegőpontos (double, 64 bit)
- Kifelé tizes számrendszerbeli egész szám, amíg belefér, utána valós

```
5         // szám literál
-2        // negatív szám
5.6       // lebegőpontos (valós) szám literál
.8        // a 0.8 lebegőpontos szám rövidített alakja
0755      // szám nyolcas számrendszerben (10-es számrendszerbeli megfelelője a
          493)
0xFFFFFF  // szám tizenhatos számrendszerben (a 10-es számrendszerbeli 16777215)
Infinity  // típusa szám, azonban értéke végtelen - minden olyan szám végtelen, ami
          akkora, hogy a JavaScript már nem tudja ábrázolni
```

Kerekítési hibák

- Hiba

```
0.1 + 0.2 // 0.30000000000000004
```

- Megoldás: normalizálás
  - Nagy pontosságú számolásoknál

```
var a = 0.1,
    b = 0.2,
    normalizaltOsszeadas = (a*10 + b*10)/10;
assertEquals(0.3, normalizaltOsszeadas);
```

Operátorok

- Operátorok megpróbálják az operandusokat számmá konvertálni
  - Kivétel a + operátor → szöveg

```
assertEquals(true, 5 < 6);
assertEquals(false, 5 > 6);
assertEquals(true, 5 === (4+1));
assertEquals(6, 2 * 3);
assertEquals(2, 6 / 3);
// Maradékos osztás:
assertEquals(1, 5 % 2);
// Automatikus típuskonverzió sztringre:
assertEquals("11", 1 + "1");
assertEquals("11", "1" + 1);
```

Sztringek számokká konvertálása

- var intValue = parseInt(string[, radix])
  - Mindig adjuk meg a számrendszert jelző 2. paramétert
- var floatValue = parseFloat(string)
- ES5: automatikusan 10-es számrendszerbe

```
assertEquals(755,  parseInt("755"));
assertEquals(7,    parseInt("7.55"));
assertEquals(493,  parseInt("755", 8));
assertEquals(755,  parseInt("755px"));
assertEquals(NaN,  parseInt("Px is 755"));
assertEquals(755,  parseFloat("755"));
assertEquals(7.55, parseFloat("7.55"));
assertEquals(2,    parseInt("1") + 1);
```

Konstansok

- A Number három kiemelt konstans értékkel rendelkezik:
  - Infinity
  - -Infinity
  - NaN

```
assertEquals('number', typeof Infinity);
assertEquals('number', typeof -Infinity);
assertEquals('number', typeof NaN);
```

NaN – Not a Number

- Speciális szám
  - typeof NaN === ”number”
- Minden művelet, ami kivezet a számok halmazából, ezt eredményezi
- Toxikus: ha kifejezésben szerepel, akkor az egész kifejezés NaN lesz
- Vizsgálata: isNaN()

```
var mitCsinalsz = "alma" / 2;
mitCsinalsz;          // => NaN
mitCsinalsz * 2;      // => NaN

NaN === NaN;          // => false

isNaN(NaN);           // => true
isNaN(665);           // => false
isNaN('alma');        // => true
isNaN('665');         // => false
```

Szám-e?

- isNaN()
  - Automatikus konverzió
  - Végtelen konstansokat is számnak veszi
- isFinite()
  - Automatikus konverzió
- Saját megoldás

```
var isNumber = function (value) {
  return typeof value === 'number' && isFinite(value);
}
```

Számok mint objektum

- Number a csomagolóobjektum
- Number tulajdonságai
  - Number.MAX_VALUE
  - Number.MIN_VALUE
  - Number.POSITIVE_INFINITY
  - Number.NEGATIVE_INFINITY
- Számpéldányok metódusai
  - toExponential
  - toFixed
  - toPrecision
  - toString

Számok mint objektum


```
var szam = 12.98;
assertEquals("12.98", szam.toString());
// A toPrecision az adott hosszhoz igazítja a számot:
assertEquals("12.980", szam.toPrecision(5));
assertEquals("13.0", szam.toPrecision(3));
assertEquals("13", szam.toPrecision(2));
// A toFixed adott tizedesjegyre kerekíti a számot:
assertEquals("12.98000", szam.toFixed(5));
assertEquals("12.98", szam.toFixed(2));
assertEquals("13.0", szam.toFixed(1));
// A műveletek során nem változott meg a szam értéke:
assertEquals(12.98, szam);
```

Matematikai műveletek

- Math objektum statikus metódusain keresztül

```
assertEquals(3.141592653589793, Math.PI);
assertEquals(5, Math.abs(-5));
assertEquals(3, Math.floor(3.5));
assertEquals(4, Math.ceil(3.5));
// A random 0 és 1 közötti véletlenszámot generál:
Math.random();
// Kis trükkel generálhatunk más számok között is
// Pl. 1 és 10 között:
Math.ceil(Math.random()*10);
```

Szövegek

Szövegliterál

- JavaScriptben egyetlen karakter, karaktertömbábrázolás létezik
- Indexelés 0-tól

```
„Hello világ"
'Kéne valami új'
"mert a 'Hello világ' unalmas”
'talán lehetne az, hogy: "Hello Béla"!'
'Persze a JavaScriptben is escape-elhetjük a(z) \' karaktert!'
'A \n egy sztringbeli sortörés, a \t a vízszintes tabulátor, a \\ pedig a
fordított perjel'
'hivatkozhatunk egy karakterre is, elvégre "karaktertömbök" vagyunk'[1]
```

Szöveg mint objektum

- String a csomagolóobjektum
- Példánytulajdonság: length
- Példánymetódus sok van

```
'Fortissima'.length;                    // => 10, sztring hossza

'csak a'.toUpperCase();                 // => 'CSAK A'
'MÁV'.toLowerCase();                    // => 'máv'

'Hólapátolás'.replace('Hó','Tó');       // => 'Tólapátolás', egyszerű csere
'Hólapátolás'.replace(/Hó|lás/gi,'Tó'); // => 'TólapátoTó', csere reguláris kifejezéssel

'Hólapátolás'.search('pát');            // => 4, a keresett karakterek kezdőindex-e
'Hólapátolás'.search(/pát/gi);          // => 4, keresés reguláris kifejezéssel
'Hólapátolás'.search('pátosz');         // => -1, ha nem találni ilyet

'Hólapátolás'.substr(2);          // => 'lapátolás', sztring a 3. (2 indexű) karaktertől
'Hólapátolás'.substr(2,5);        // => 'lapát', sztring a 3. karaktertől, az innen számított 5.
                                        karakterig
'Hólapátolás'.substr(-5);         // => 'tolás', az utolsó 5 karakter

'Hólapátolás'.slice(2,5);         // => 'lap', sztring a 3. karaktertől a 6.-ig
```

Operátorok

- Konkatenáció: +
- Számok összeadására is szolgál!

```
'a' + 'lma';    // => 'alma', sztringek összefűzése, konkatenáció
5 + 'ödölő';    // => '5ödölő', sztringhez számot adva az automatikusan
                   sztringgé konvertálódik
'lak' + 6 + 5; // => 'lak65', elég egy sztring a műveletsorban, a
                   végeredmény sztring lesz
'lak' + (6 + 5); // => 'lak11'
6 + '';            //  =>  '6',  egyszerű  módszer  számok  sztringgé
                   konvertálására
```

Többsoros szövegek

A) Hibás sztring

```
'<div>
    <p>A lovagok, akik azt mondják:
        <strong>NI</strong>
    </p>
 </div>';                                                    // => HIBÁS SZTRING
```

B) Megoldás konkatenációval

```
'<div>'+
'   <p>A lovagok, akik azt mondják:'+
'        <strong>NI</strong>' +
'   </p>' +
'</div>';
```

C) Megoldás levédett sortörésekkel

```
'<div>\
    <p>A lovagok, akik azt mondják:\
        <strong>NI</strong>\
    </p>\
 </div>';
```

Logikai értékek

Logikai literál és operátorok


```
true              // => igaz
false             // => hamis

!true             // => false, tagadás
!false            // => true
!!true       // => true, dupla tagadás, jó módszer bármilyen adattípus
                        logikai értékké való konvertálásához

true && true      // => true
true && false     // => false
false && true     // => false
false && false    // => false

true || true      // => true
true || false     // => true
false || true     // => true
false || false    // => false
```

Kifejezések logikai értéke

- Minden kifejezés logikai kifejezéssé alakítható
- Hat kis hamis szabály
  - false
  - '' (üres szöveg)
  - 0
  - null
  - undefined
  - NaN
- Minden egyéb kifejezés igaz.

Kifejezések logikai értéke


```
true  ? 'Igaz' : 'Hamis';                          // => 'Igaz'
false ? 'Igaz' : 'Hamis';                          // => 'Hamis'
0             ? 'Igaz' : 'Hamis';      // => 'Hamis'
'0'           ? 'Igaz' : 'Hamis';      // => 'Igaz'
0x455         ? 'Igaz' : 'Hamis';      // => 'Igaz'
0455          ? 'Igaz' : 'Hamis';      // => 'Igaz'
0x000         ? 'Igaz' : 'Hamis';      // => 'Hamis', mivel 0
'false'       ? 'Igaz' : 'Hamis';      // => 'Igaz'
('alma' / 2) ? 'Igaz' : 'Hamis';      // => 'Hamis', mivel NaN
(function() {})    ? 'Igaz' : 'Hamis'; // => 'Igaz', egy függvény literál nem hamiskás
                                          érték
(function() {})() ? 'Igaz' : 'Hamis'; // => 'Hamis', ekkor a függvényt deklarálás után
                                          egyből    meghívtuk,    return    hiányában    a
                                          visszatérési értéke undefined, ami hamis

10 < 100              // => true
1000 == 1000          // => true, de a dupla egyenlőségjel kerülendő! (2.17. Példa)
100 === 1000          // => false
'macska' !== 'kutya'  // => true
```

Egyenlőség vagy identitás

- Automatikus konverziók miatt kerüljük a == használatát
- Mindig érték és típus szerint vizsgálódjunk
- Identitásvizsgálat === operátorral

```
0 == 0                  // => true
0 == '0'                // => true, ???
0 == ''                 // => true, ???
null == undefined       // => true, ???

0 === 0                 // => true
0 === '0'               // => false
0 === ''                // => false
null === undefined      // => false
```

Guard operátor

- és operátorlánc addig értékelődik ki, amíg az éppen vizsgált kifejezés igazat ad
- Létezésvizsgálat

```
var book = {
  getISBN: function()
};
// A book egy objektum, ami logikai kontextusban igaz, hiszen
// nincs a falsy értékek között, ugyanez igaz a book getISBN
metódusára
if (book && book.getISBN) {
  book.getISBN();
}
// Mivel nincs fly metódusa egy könyvnek, így azt sosem fogja
meghívni
book && book.fly && book.fly();
```

Default operátor

- vagy operátorlánc: első olyan kifejezés értékével tér vissza, ami igaz, vagy az utolsóéval

```
var book = function(name, datas) {
    // A név vagy a megadott név volt vagy egy üres sztring lesz
    name = name || '';
    datas = datas || ;
    datas.isbn = datas.isbn || '0-00-000000-0';
    datas.pages = datas.pages || 0;
    datas.author = datas.author || 'Author not provided';
    datas.name = name;
    return datas;
};
var bookDatas = book('Hally Pottel', {
        isbn: 0,
        pages: 50,
        name: 'Something else'
    });
```

Egyéb értékek

Beépített értékek

- undefined
  - Ami más nyelvben a null, az itt az undefined
  - Nem létező változóra hivatkozáskor
  - Visszatérési érték nélküli függvény esetén
  - Vigyázat! Felüldefiniálható!
- null
  - Ne használjuk

Reguláris kifejezések

Reguláris kifejezése létrehozása


```
// Illeszkedik az "a string" sztringre:
var regexp = /a string/;

var suffix = "string";
var regexp = new RegExp('a ' + suffix);

var regexp = /string is found/;
assertEquals(true, regexp.test('True when string is found'));
assertEquals(false, regexp.test('False when string is not found'));
```

Reguláris kifejezések használata


```
assertEquals('John Wayne', 'John Smith'.replace(/\w*$/, 'Wayne'));
// Alakítsuk át a CSS attribútumok JavaScriptesített neveit
// eredeti formájukba, ahogy egy CSS tagbe illeszthetőek:
assertEquals("font-family", "fontFamily".replace(/([A-Z])/g, "-
$1").toLowerCase());
// A replace második paraméterében $1, $2, stb. néven
// hivatkozhatunk a backreference-ekre:
assertEquals('Smith John', 'John Smith'.replace(/(\w+)\s(\w+)$/, '$2 $1'));

// Találjuk meg azokat a számokat amelyek egymás
// után többször szerepelnek:
assertEquals(['22', '444'], '06301223444'.match(/(\d)\1+/gi));

var countPartsOfPhoneNumber(number) {
    return number.split(/[ ()-]+/gi).length;
}
assertEquals(4, countPartsOfPhoneNumber('+36 (30) 481-9873'));
```

Összetett típusok

Összetett típusok

- Referenciatípusok
- Valódi objektumok
  - Nincs ideiglenes csomagolóobjektum
- Értékadáskor referencia szerint adódnak át, így bármelyiket módosítjuk, a másik is változik
- Típusok
  - Objektumok
  - Tömbök
  - Függvények

Objektumok

Objektum-orientáltság

- JavaScript objektum-orientált nyelv a szó szoros értelmében
- Nincsen osztályok, csak objektumok
- Nincsenek példányok, minden objektum egy egyedi példány, vagy másolata egy meglevőnek.
- JavaScriptben az elemi típusokon kívül minden objektum

Objektumok

- A JavaScript objektum névvel ellátott értékek (kulcs-érték párok) gyűjteménye
- Más nyelvekben
  - Asszociatív tömb
  - Hash
  - Táblázat
- Dinamikusak: bővíthető, szűkíthető
- Név
  - Sztring
  - Azonosító
- Érték
  - Tetszőleges kifejezés
  - Ha futtatható (függvény), akkor metódus (method)
  - Egyébként tulajdonság, attribútum, adattag (property, attribute)

Objektumok csoportosítása

- Natív objektumok (ECMAScript szabvány által definiált)
  - Beépített objektumok (String, Number, Date, stb.)
  - Felhasználó által definiált (var o = {};)
- Futtatókörnyezet objektumai (pl. böngésző)
  - Pl. window, DOM, stb

Objektumliterál – felhasználói obj.

- Literálforma: { … }

```
var uresObjektum = {};

var objektum = {
    tulajdonsag: 'Ez egy tulajdonság',
    'tulajdonság2': 'Ez egy másik tulajdonság',
    metodus: function() {
        return 'Metódus vagyok';
    }
};
```

Hivatkozás az adattagokra

- Kétféle módon
  - obj.prop
  - obj['prop']

```
var uresObjektum = {};

var objektum = {
    tulajdonsag: 'Ez egy tulajdonság',
    'tulajdonság2': 'Ez egy másik tulajdonság',
    metodus: function() {
        return 'Metódus vagyok';
    }
};
```

```
objektum.tulajdonsag;      // => 'Ez egy tulajdonság'
objektum['tulajdonság2']; // => 'Ez egy másik tulajdonság'
objektum.metodus();        // => 'Metódus vagyok'
```

Adattagok módosítása

- Bármelyik adattag szabadon módosítható
  - Kivétel néhány beépített objektum adattagja

```
objektum.tulajdonsag = 'Más lettem';
objektum.tulajdonsag;      // => ‘Más lettem'

objektum.metodus = function() {
    return ‘Más metódus vagyok';
}
objektum.metodus();        // => ‘Más metódus vagyok'
```

Objektum bővítése

- Tetszőlegesen bővíthető mindegyik objektum

```
objektum.ujTulajdonsag = 'Na, most meg dinamikusan bővítettek...';
objektum.ujTulajdonsag;   // => 'Na, most meg dinamikusan bővítettek...'

objektum.ujMetodus = function() {
    return 'Elég legyen mára!';
}
objektum.ujMetodus();     // => 'Elég legyen mára!';
```

Objektumok egymásba ágyazása


```
objektum.belsoObjektum = {
    belsoTulajdonsag: 'Kezd érdekes lenni!',
    legbelsoObjektum: {
        legbelsoTulajdonsag: 'Tovább és tovább...'
    }
};

objektum.belsoObjektum.belsoTulajdonsag;
       // => 'Kezd érdekes lenni!'
objektum.belsoObjektum.legbelsoObjektum.legbelsoTulajdonsag;
       // => 'Tovább és tovább...'
objektum.belsoObjektum['legbelsoObjektum']['legbelsoTulajdonsag'];
       // => 'Tovább és tovább...'
```

Adattag törlése

- delete operátor
- undefined lesz az értéke
- Csak adattagot töröljünk!

```
delete azObjektum.eletErtelme;
console.log(azObjektum2.eletErtelme); // => undefined
```

Objektum, a referenciatípus

- Objektumot értékül adva egy másik változónak, mindkét változó ugyanarra az objektumra mutat.
- Módosítva az egyiken, a másik is változik.

```
var azObjektum = { eletErtelme: 42 };

var azObjektum2 = azObjektum;
azObjektum2.eletErtelme = 43;

azObjektum.eletErtelme;                    // => 43
```

this

- Egy objektumra saját metódusában a this kulcsszóval hivatkozhatunk

```
var azObjektum = {
    eletErtelme: 42,
    csalas: function(i) {
        this.eletErtelme = i;
    },
    miAzEletErtelme: function() {
        return this.eletErtelme;
    }
};

azObjektum.csalas(43);
azObjektum.miAzEletErtelme();     // => 43
```

Beépített objektumok

- Hasznos objektumok, amiket mindenképpen érdemes megismerni
  - Math: matematikai műveletek
  - Date: dátum és időkezeléssel kapcsolatos műveletek
  - RegExp: reguláris kifejezések

Tömbök

Tömbök

- A tömb is az objektum egy speciális fajtája (Array)
  - Adattagokat 0-tól kezdve indexelhetünk a [] operátorral
  - Saját metódusok (pl. push())
  - Saját tulajdonságok (pl. length)
- Bármilyen adattípust tartalmazhat, minden eleme lehet más típusú
- Futási időben módosíthatók az elemei, újak tehetők bele, törölhetők meglévők
- Adatok sorrendje kötött az objektummal szemben

Tömb literál és használat

- Literálforma: […]

```
var uresTomb = [];
var tomb = ['A', { minoseg: 'legjobb' }, function() { return 'böngésző' }, 'az IE!'];

tomb[0];            // => 'A'
tomb[1].minoseg;    // => 'legjobb'
tomb[2]();          // => 'böngésző'
tomb.length;        // => 4
```

- Módosítás, bővítés

```
tomb[0] = 'Nem a';
tomb[0];              // => 'Nem a'

tomb.push('5. elem');
tomb[tomb.length] = 'Most én vagyok az utolsó!';
tomb.length;          // => 6

tomb[50] = 'Hú de messze vannak a többiek!';
tomb.length;          // => 51
tomb[49];             // => undefined
```

Elem törlése

- delete operátor
  - Tömb hossza nem változik
  - undefined lesz
- splice metódus
  - Kivágja a megadott részt

```
var t = ['Tro', 'lo', 'lolo'];
delete t[1];
console.log(t);              // => ['Tro', undefined, 'lolo']
console.log(t.length);       // => 3

var t = ['Tro', 'lo', 'lolo'];
t.splice(1,1);               // => 'lo'
console.log(t);              // => ['Tro', 'lolo']
console.log(t.length);       // => 2
```

Tömb metódusok

- pop
- push
- reverse
- shift
- sort
- splice
- unshift
- concat
- join
- slice

Tömb és sztring

- Hatékony szövegösszefűzés

```
var html = [];

html.push('<div>');
for (var i=0, part; i<3; i++) {
    html.push('<p>'+i+'</p>');    // csak a nagy sztringekhez való konkatenáció lassú, így
                                  a könnyebb olvashatóság kedvéért a nyugodtan
                                  használjunk normál konkatenációt a rész-sztringekben
}
html.push('</div>');

html.join(''); // => '<div><p>0</p><p>1</p><p>2</p></div>'
```

- Join és split

```
var gyumolcsok = ["alma", "barack", "papaya"],
    gyumiString = gyumolcsok.join(','),        // => 'alma,barack,papaya'
    gyumiArray1 = gyumiString.split(','),      // => ["alma", "barack", "papaya"]
    gyumiArray2 = gyumiString.split('');       // => ["a", "l", "m", "a", "," ...]
```

Műveletek


```
var numbers = [1,2,3];
// A forEach egyszerűen végigiterál az összes elemen
var minimumValue = Number.MAX_VALUE;
numbers.forEach(function(item) {
    if (item < minimumValue) {
        minimumValue = item;
    }
});
assertEquals(1, minimumValue);
// A map az átadott függvény visszatérési értékeiből
// egy új tömböt épít
var multiplied = numbers.map(function(item) {
    return item * 2;
});
assertEquals([2,4,6], multiplied);
// A filter az átadott függvény visszatérési értékei alapján
// épít fel egy új tömböt
var smallNumbers = numbers.filter(function(item) {
    if (item < 3) {
        return true;
    }
    return false;
});
assertEquals([1,2], smallNumbers);
```

Műveletek


```
// Az every eldönti, hogy a tömb összes eleme megfelel-e az átadott
// metódus igazságvizsgálatára
var isNumbers = numbers.every(function(item) {
    return typeof item === 'number';
});
assertEquals(true, isNumbers);
// Az every-hez hasonló some esetén elég egyetlen igaz visszatérési
// érték is, a végeredmény igaz lesz
var hasThree = numbers.some(function(item) {
    return item === 3;
});
assertEquals(true, hasThree);
// A reduce egy értéket állít elő az iteráció végén.
// Az előállítandó értéket a második paraméterben inicializálhatjuk,
// minden iterációs lépésnek átadja az eddig kiszámolt értékeket
// és az aktuális elemet. A visszatérési érték lesz átadva a
// következő hívásnak
var sum = numbers.reduce(function(previous, item) {
    return previous + item;
}, 0);
assertEquals(6, sum);
```