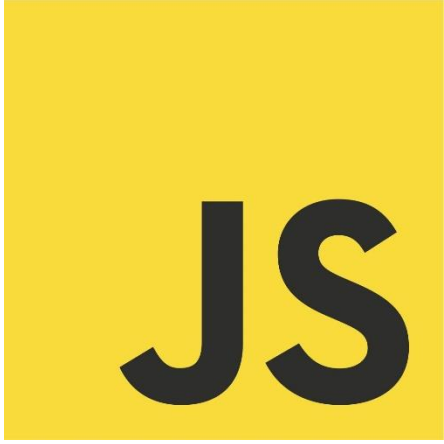


JavaScript bevezető

Szabó Ádám

szabo.adam@bmeautsoft.hu

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Mi lesz ma?

- JavaScript bevezető, és egy rövid gyakorlat
- Node.js alapok
  - > Gyakorlat
- NPM és hasznos Node.js alkalmazások
- Express web keretrendszer
- Komplex REST alkalmazás fejlesztése Express-el
- Loopback előadás és gyakorlat

Történelem

- Brendan Eich, a Netscape mérnöke alkotta meg eredetileg Mocha, később LiveScript néven 1995-ben
- Cél egy olyan egyszerű, nem csak hivatásos fejlesztők által használható szkript nyelv megalkotása volt, amely a Java nyelvet kiegészítve lehetővé teszi interaktív weboldalak készítését
- Az évek során a nyelvnek számos változata alakult ki:
  - **JScript:** 1996 augusztusában a Microsoft által a jogi problémák elkerülésére más néven kiadott dialektus. Az Internet Explorer 9-ben található JScript 9.0 a JavaScript 1.8.1 és az ECMA-262 5. változatával kompatibilis.
  - **ECMAScript**: az ECMA által az ECMA-262 szám alatt szabványosított változat, amely a védjegy bejegyzések miatt kapta ezt a nevet. Az 1. változat 1997. júniusában jelent meg, az 5.1 változat pedig 2011. júniusában. Mind a JavaScript, mind pedig a JScript a szabványhoz képest további funkciókat biztosít.

Jelen

- Minden böngészőben található egy igen jó JavaScript motor
  - **Evergreen** böngészők elterjedése
  - Nagyon gyors és stabil JavaScript fordítók jelentek meg
- Asztali böngészőkből továbbfejlett egy teljesen önálló platformmá
- Nagy és aktív open-source közösség alakult köré
- Flash halálával az egyedüli eszköz a dinamikus, illetve vastag kliens webes alkalmazások készítéséhez
- Node.js meghatározó szerver oldali platform lett

Complie to JavaScript

- Absztakciós szint JavaScript fölött, cél könnyebben érthető, és átlátható kód írása szebb nyelven
- Normál JavaScript-re fordul, így minden JS képes platformon támogatott
- Megközelítések
  - Típusos JavaScript
    - TypeScript
    - Dart
    - Google Web Toolkit
    - Java2Script
  - Szebb és könnyebb struktúra
    - CoffeeScript

Miért szeretjük a JavaScrip-et?

- Hagyományosan a Web böngészők nyelve
  - DOM kezelése
  - Multiplatform
    - Hardver szinten
    - OS szinten
    - Böngésző szinten
  - Szabványos

- Nagyon gyorsan tanulható
- Egyszerűen fejleszthető, nem kell komplex és nagy SDK-kat IDE-ket használni
- Erős fejlesztői közösség
  - Rengeteg open source framework/lib/snippet érhető el
  - Szinte minden kérdésre és problémára lehet megoldást találni a fejlesztői fórumokon

Miért szeretjük a JavaScripet?

- Dinamikus és gyenge típus kezelés
- C szintaktika
- Esemény vezért és Funkcionális programozás támogatása
- Aszinkron működés támogatása
  - Szinte az összes blokkoló szolgáltatás csak aszinkron érhető el
- JavaScript Object Notation - JSON

Miért nem szeretjük a JavaScripet?

- Dinamikus és gyenge típus kezelés
- **Nem klasszikus értelemben objektum orientált**
  - Nincs interfész
  - Öröklés helyett
- Megszokott tervezési mintákat nagyon nehéz implementálni
- Globális változók káosza
- Kevés beépített típus
- JavaScript motorok eltérései

Java vs JavaScript

A Java és JavaScript annyira hasonlít agymásra mint, a Csap és a CsapÁgy

Java	JavaScript
statikusan típusos	dinamikusan típusos
erősen típusos	gyengén típusos
osztály-alapú objektumok	prototípus-alapú objektumok
bájtkódból töltődik be	forráskódból töltődik be
JVM futtatja	JavaScript motor futtatja, és natív kódra fordul

Példakód

```
function Car( model ) {

  this.model = model;
  this.color = "silver";
  this.year = "2012";

  this.getInfo = function () {
    return this.model + " " + this.year;
  };
}

var myCar = new Car("ford");
myCar.year = "2010";
console.log(myCar.getInfo()); //ford 2010
console.log(today); //Tuesday, September 01, 2015
```

Hol futhat JavaScript?

- Böngészőkben
- Böngészőn kívül
  - Helyi gépen / szerveren
- Telefonon
  - Apache Cordova
- Okos eszközökön
  - Smart TV, Tizen, WebOS, stb
- Windows Runtime alkamazásokban
- IoT eszközökben
  - https://samsung.github.io/iotjs/
  - http://devicejs.org/

Nyelvi elemek

- Kommentek: `// egy soros; /* több soros */`
- Aritmetikai operátorok: `+, -, /, *, %, ++, --`
- Értékadás operátorok: `=, +=, -=, *=, /=, %=`
- Bitenkénti operátorok: `&, |, ^, ~, <<, >>, >>>`
- Logikai operátorok: `||, &&`
- Összehasonlító operátorok: `==, ===, !=, !==, <, >, <=, >=`
- Feltétel vizsgálat: `if..else, switch..case (break, default), instanceof, typeof, .. ? .. : ..`
- Ciklusok: `for, for..in, while, do..while, break, continue`
- Hibakezelés: `try..catch..finally, throw`
- Objektumok kezelése: `new, delete`
- Függvények: `function, return`

Típusok

- Az ECMAScript szabvány az alábbi hat adattípust definiálja:
  - String
  - Number
  - Boolean
  - Null
  - Undefined
  - Object

```
var szoveg = 'Gipsz Jakab';
// Használható "" is.
var szam = 10;
var logikai = true;
var obj = {
  a: 10,
  b: 'alma'
}
obj.c; //undefined
```

Logikai kifejezések

- JavaScriptben nem csak Boolean típus lehet *true* vagy *false*.

truthy	falsey
true	false
'0'	0
123 vagy -123	NaN
'valami'	'' (üres string)
[]	null
{}	undefined

- Célszerű az alábbi két szabályt megjegyezni:
  1. A következő hat érték hamisként viselkedik (falsey), minden más igazként: false, 0, NaN, '', null, undefined.
  2. Minden objektum igazként viselkedik.

Változó létezésének vizsgálata

- Gyakori feladat, hogy vizsgálnunk kell, vajon egy adott változó rendelkezik-e „normális” értékkel, például hogy egy függvény bemenő paraméterét a hívó beállította-e. Ilyen esetekben nem kell külön vizsgálnunk, hogy a változó null vagy *undefined*-e, hanem írhatjuk röviden, hogy:

```
if( valtozo ) {
...
}
```

- Vigyázat, ez a rövid kód nem csak null vagy undefined, hanem például 0 és '' érték esetén se fog lefutni!

Alapérték beállítása

- Gyakran kihasználjuk, hogy az *undefined false*-ként viselkedik és a || operátor segítségével így állítunk be alapértéket egy változónak:

```
var afakulcs;
var szokasosafakulcs = 27;
var adokulcs = afakulcs || szokasosafakulcs;
alert(adokulcs); // 27
```

Feltételes kódfuttatás

- A JavaScript lehetővé teszi az alábbi tömör szintakszist feltételes kódfuttatáshoz:

```
var x = '';
x && alert('fut');
```

Összetett típusok

- A JavaScript nyelvben az alábbi összetett típusok találhatók meg, melyek igen szoros kapcsolatban vannak egymással:
  - Array
  - Object
  - Function

Tömbök

```
var napok = [ 'hétfő', 'kedd', 'szerda' ];
var evszakok = new Array( 'tavasz', 'nyár', 'ősz', 'tél' );

alert( typeof napok ); // 'object'
alert( typeof evszakok ); // 'object'
```

- A tömbök a szokásos módon viselkednek: nullától indexelődnek és a *length* tulajdonsággal kérdezhető le a méretük. A szokásostól eltérő viselkedés azonban, hogy a *length* tulajdonság írható is, azaz egy kisebb érték beállításával csonkolhatjuk a tömböt.

```
for( var i = 0; i < napok.length; i++ ) {
  alert( napok[ i ] );
  // hétfő, kedd, szerda
}
```

```
for( var i in napok ) {
  alert( napok[ i ] );
  // hétfő, kedd, szerda
}
```

Függvények

- JavaScriptben nincs overloading, azaz ha két azonos nevű függvényt hozunk létre, akkor a második felül fogja írni az elsőt

```
function kiir(szoveg) {
  alert(szoveg);
}
function osszead( a, b ) {
  return a + b;
}
kiir('Ébresztő!');
var osszeg = osszead( 2, 3 );
alert(osszeg);
```

- A függvényeket tetszőleges számú paraméterrel meg lehet hívni.
  Ha egy bemenő paraméternek a híváskor nem adunk értéket, akkor a függvényen belül *undefined* lesz
- Minden függvényen belül elérhető egy *arguments* nevű lokális változó, amelyen keresztül is elérhetjük a hívótól kapott paramétereket

Objektumok

- Az objektumok olyan összetett változók, amelyek kulcs-érték párokat fognak össze. Egyszerűbb esetben ezek tulajdonságok:

```
var gj = {
  nev: 'Gipsz Jakab',
  kor: 35,
  kiir: function() {
    alert( this.nev + ' ' + this.kor + ' éves.' );
  }
};
gj.kiir();
```

Gyakran használt beépített objektumok

- `Date`: JavaScriptben nincs dátum adattípus, helyette a Date objektumot használhatjuk dátumok leírására és dátum műveletek végzésére.
- `Math`: matematikai műveletek és konstansok.
- `RegExp`: reguláris kifejezések.
- `Error`: hiba (kivétel) leírása.

Öröklés - prototype chain

- Nincs osztály, csak objektum!
  - Struktúra tagváltozókkal, illetve függvényekkel
- Minden objektumban található hivatkozás egy „ős” objektumra, amely a *prototype* változón érhető el.
  - Végső szülő objektum: *Object*
- Tagváltozók öröklése
  - Ha nem található az objektumban az adott property, akkor az ősben folytatódik a keresés
  - Ez addig folytatódik, amíg van ős, tehát az *Object* beépített objektumig.
- Függvények öröklése
  - Ugyan úgy működik mint a tagváltozók
  - A függvény kontextusa (*this*) mindig a leszármazottra mutat

Változók láthatósága

- A JavaScript nyelvben nincs közvetlen lehetőség a privát és publikus tagok megjelölésére,
- csak és kizárólag a függvény blokkok határozzák meg a láthatóságot, más néven hatókört (functional scoping)

```
var x = 5;
function kulso() {
  var x = 8;
  function belso() {
    var x = 9;
    alert( 'Belül: ' + x ); // Első: 9
  }
  belso();
  alert( 'Kívül: ' + x ); // Második: 8
}
kulso();
alert( 'Legkívül: ' + x ); // Harmadik: 5
```

Closures

- JavaScriptben a függvények teljes értékű típusként viselkednek, ezért közvetlenül inicializálhatók objektum tagváltozó értékeként:

```
var kulso = function() {
  var x = 8;
  var belso = function() {
    alert( x );
  };
  return belso;
};

var b = kulso();
b(); // 8
```

Névtelen függvények

- Függvények hasonlóan a változókhoz, közvetlenül értékül adhatók más függvényeknek vagy változóknak.

```
setTimeout(function() { alert('2 sec elapsed!' ); }, 2000);
```

Self-executing (self-invoking) functions

- JavaScript nyelvben definiálhatunk automatikusan lefutó függvényeket
- Ezek segítségével könnyen lehet külön modulokba szervezni a JS kódjainkat
- Elkerülhetjük vele a globális scope fölösleges használatát

```
var fv = function( nev ) {
  alert( 'Szia ' + nev );
};
fv( 'Világ' );
```

```
(function( nev ) {
  alert( 'Szia ' + nev );
})( 'világ' );
```

Modul tevezési minta

```
var N = (function(){
  var priValt = 3;
  var priFv = function(){
    alert( 'Privát!' );
  };

  return {
    pubValt: 5,
    pubFv: function(){
      alert( 'Publikus' );
    }
  };
})();
N.pubFv();
alert( N.pubValt );
```

JavaScript Object Notation - JSON

- Probléma:
  - Az objektumoknak az előbbiekben „Object literal”-nak nevezik. Gyakori feladat, hogy egy JavaScript objektumot szerver oldalra kell felküldenünk, amelyhez ez nem használható, hanem az objektumot stringgé kellene sorosítanunk.
- Megoldás: **JSON**

```
'{
"nev": "Gipsz Jakab",
  "kor": 35,
  "szuletes": {
  "hely": "Cegléd",
    "ev": 1978
  }
}'
```

JSON

- Bár formailag a JSON és az Object literal nagyon hasonló, a JSON sokkal szigorúbb és kevesebb információt hordoz:
  - A kulcsoknak idézőjelekbe tett stringeknek kell lenniük.
  - Az érték lehet: *szöveg, szám, objektum (JSON-ban), tömb, true, false, null.*
- Látható, hogy a függvények nem jelennek meg a JSON formátumú leírásban, azaz a JSON csak részhalmaza az Object literalnak.
- JSON sorosítás és parse-olás:

```
var json = JSON.stringify( gj );
alert( json ); // {"nev":"Gipsz Jakab","kor":35,"szuletes":{"hely":"Cegléd","ev":1978}}
var obj = JSON.parse( json );
alert( obj.nev ); // 'Gipsz Jakab'
```

JavaScript motorok

- Feladatuk a JavaScript kód értelmezése, futtatása és monitorozása
  - Natív kódra fordítás
    - JIT
    - Dinamikus JavaScript nyelvből egy statikus objektum orientált kód keletkezik
  - Kód és fordítás optimalizálás
  - Változók, függvények gyorsítótárazása
  - Memória kezelés

JavaScript motorok

- V8
  - Chrome, Chromium
    - Desktom és mobil
  - Node.JS, IO.JS
- JavaScriptCore (Nitro)
  - Webkit-es böngészők
    - Safari
- Rhino
  - Firefox
- OpenLaszlo ☺

ECMA Script 5

- Strict mód
  - Kötelező minden változót inicializálni használat előtt
  - "use strict" –et kell elhelyezni a kódban
  - Kontextusonként lehet engedélyezni

```
"use strict";
x = 3.14; //Error
```

```
"use strict";
myFunction();
function myFunction() {
  y = 3.14;   // Error
}
```

- Legfontosabb újdonságok:
  - https://gist.github.com/sym3tri/2425983
- Kompatibilitás:
  - http://kangax.github.io/compat-table/es5/

Modulok - AMD

- Probléma:
  - Az egyes modulok milyen más modulokra építenek?
  - Milyen sorrendben kell betöltenünk a fájlokat?
  - Az oldal betöltésekor kell letöltenünk az összes fájlt, vagy csak amikor használni szeretnénk?
- Megoldás: **Asynchronous Module Definition (AMD)**

```
define( id?, dependencies?, factory );
```

- Segítségével:
  - Lazán csatolható modulok
  - Modul függőségek egyszerű definiálása
  - Igény szerinti betöltés

Modulok - AMD

```
define( id?, dependencies?, factory );
```

- **id**: a modul neve, nem kötelező megadni.
- **dependencies**: azok a modulok, amelyek ennek a modulnak a futásához szükségesek, nem kötelező megadni.
- **factory**: a függvény, amelyet a betöltő meghív a modul vagy objektum példányosításához.

```
define( ['jQuery', 'knockout-2.2.1'], function( $, ko ) {
  // A jQuery a $, a KnockoutJS pedig a ko változón keresztül érhető el.
});
```

- Modul betöltés:

```
require( dependencies, callback );
```

ECMA Script 6

- Szabvány idén júniusban lett véglegesítve
- JavaScript motorok még nem támogatják teljesen
- Legfontosabb újdonságok:
  - https://babeljs.io/docs/learn-es2015/
- Kompatibilitás:
  - http://kangax.github.io/compat-table/es6/

ES6 – Function shorthand

```
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);

nums.forEach(v => {
  if (v % 5 === 0)
  fives.push(v);
});

var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
  this._friends.forEach(f => console.log(this._name + " knows " + f));
}
};
```

ES6 - Class

```
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
  update(camera) {
    //...
    super.update();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
  }
}
```

ES6 – Template strings

```
// Basic literal string creation
`In ES5 "\n" is a line-feed.`

// Multiline strings
`In ES5 this is
not legal.`

// Interpolate variable bindings
var name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`

// Construct an HTTP request prefix is used to interpret
// the replacements and construction
GET`http://foo.org/bar?a=${a}&b=${b}
  Content-Type: application/json
X-Credentials: ${credentials}
{ "foo": ${foo},
  "bar": ${bar}}`(myOnReadyStateChangeHandler);
```

ES6 - Generators

```
var fibonacci = {
  [Symbol.iterator]: function*() {
    var pre = 0, cur = 1;
    for (;;) {
      var temp = pre;
      pre = cur;
      cur += temp;
      yield cur;
    }
  }
}

for (var n of fibonacci) {
  if (n > 1000)
    break;
  console.log(n);
}
```

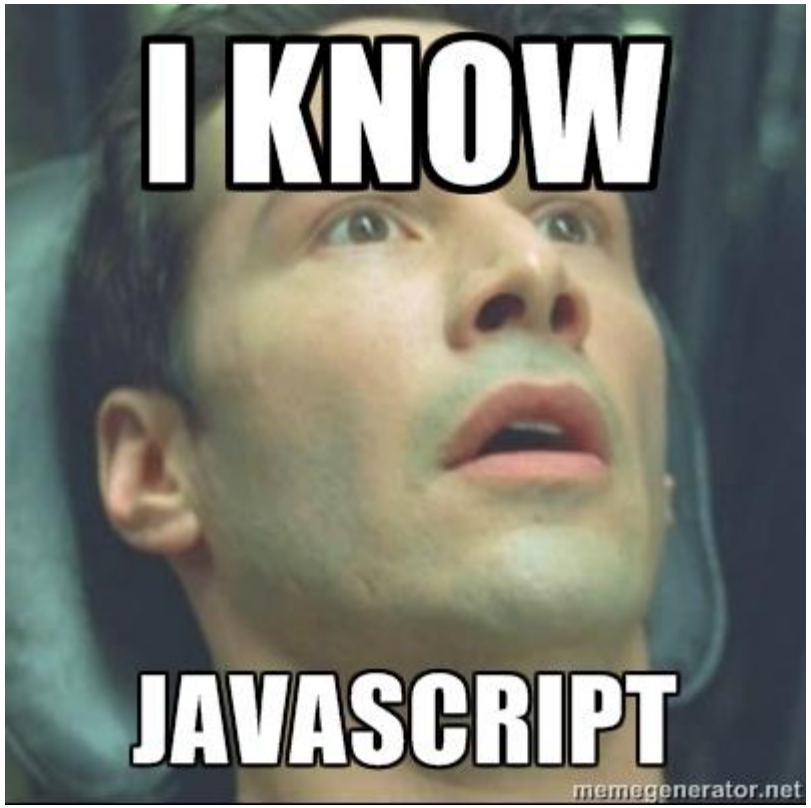
ES6 - Promises

```
function timeout(duration = 0) {
  return new Promise((resolve, reject) => {
      setTimeout(resolve, duration);
})
}

var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
  throw new Error("hmm");
}).catch(err => {
  return Promise.all([timeout(100), timeout(200)]);
})
```

ECMA Script 7

- A jövő, még korai szakaszban van
- További komplex nyelvi elemek
- Cél:
  - Kódizoláció fejlesztése
  - Erősebb kontroll az események fölött
  - Beépített szolgáltatások bővítése
  - Új struktúrák
  - ...
- Kompatibilitás:
  - http://kangax.github.io/compat-table/es7/



JavaScript gyakorlat