Bevezetés a JavaScript használatába

1. rész

Hatwágner F. Miklós

Széchenyi István Egyetem, Győr

2015. november 11.

A JavaScript fontosabb tulajdonságai

- Web oldalba ágyazható → dinamikus HTML (de: PDF, asztali widget, szerver oldali szolgáltatások is)
- Objektum-orientált, de nincs öröklődés! (objektum alapú, vagy prototípus alapú)
- Gyengén típusos, dinamikus típusosság (változók futásidőben típust válthatnak) → kényelmes, de veszélyes az implicit típuskonverziók miatt
- Böngésző által értelmezett (interpreter) nyelv (script)
- HTML DOM (Document Object Model) elérése
- Eseménykezelés
- Dokumentáció: ECMA, Mozilla, Microsoft, W3Schools, ...

Történelem

- 1995: Brendan Eich, Netscape, eredeti elképzelés: séma értelmező → Mocha, majd LiveScript (Navigator 2.0b) és JavaScript (Navigator 2.0b3), Microsoft: JScript, VBScript
- 1996: Szabványosítás megkezdődik, ECMA (European Computer Manufacturers Association) → ECMAScript
- 1997: ECMAScript v1
- 1998: ECMAScript v2
- 1999: ECMAScript v3, MS: XMLHTTP Active X (AJAX kezdete)
- 2000: Mozilla: XMLHTTPRequest (XHR, AJAX)
- 2002: JSON (JavaScript Object Notation)
- 2004: Firefox v1 megjelenik, Safari: XMLHTTPRequest támogatás
- 2005: Opera: XMLHTTPRequest támogatás
- 2006: MS: XMLHTTPRequest támogatás (IE7+), megjelenik a JQuery, GWT (Google Web Toolkit), Firebug
- 2008: ECMAScript v4-et elvetik
- 2009: ECMAScript v5 megjelenik → 2010: JS 1.8.5, kompatibilitás

Mire használható?

- '90-es évek: felugró ablakok, médialejátszás, objektumok mozgatása, státuszsor tartalmának változtatása, . . . → „játéknyelv”
- De jól használható űrlapok ellenőrzéséhez!
- Szerver oldali ellenőrzés is kell: JS kikapcsolható, hackerek
- az AJAX (Asynchronous JavaScript and XML) új értelmet ad a JS-nek
  - Előnyök: a munka nagy részét a böngészők végzik, a lehető legkevesebb adatot kell továbbítani
  - Hátrány: az oldalak *sokkal* bonyolultabbak lesznek
- Probléma: inkompatibilis böngészők
- Ma már nem csak XML lehet az adat: JSON, nyers szöveg
- Példa: regisztráció egy weblapon

Keretrendszerek (framework)

- Előnyök:
  - böngészőfüggetlenség
  - kódismétlés elkerülése
  - új funkciók, pl. dátumválasztó
- Hátrányok:
  - ált. sokkal több kódot kell letölteni, mint amire szükség van
  - emiatt gyakran lassabb is
  - ettől még ugyanúgy meg kell tanulni a JS-et is
- Példák:
  - script.aculo.us, (Yahoo! User Interface), jQuery, MooTools, ExtJS, The Dojo Toolkit, Prototype ...

JavaScript elhelyezése a weboldalon

- script HTML elem segítségével
- type attribútum értéke: text/javascript. HTML5: a *type* attribútum elhagyható
- A JS kód elhelyezhető a nyitó és záró *script* elem között, vagy az src attribútumban (de egyszerre csak az egyikben). Pl. <script src="forras.js"></script>
- A forrás megadható abszolút és relatív úttal is
- Tetszőleges számú *script* elem használható egy oldalon,
- de több fájl letöltése növeli a letöltési időt!
- A kód azonnal lefut, amint a böngésző elér hozzá (kivéve függvények, amiket hívni kell)

JavaScript elhelyezése a weboldalon

- Bárhol elhelyezhető, de célszerű az oldal végére tenni:
  - az oldal megjelenítését nem lassítja a JS letöltése, feldolgozása
  - az oldal betöltése közben a DOM tartalma változik!
- A JS kód a HTML elemek eseménykezelést beállító attribútumában is beállíthatóak, de nem ajánlott, mert
  - a HTML nehezebben áttekinthetővé válik,
  - nehéz lesz karbantartani az oldalakat,
  - amiken sok, ismétlődő kódrészlet szerepel majd.
  - nem vezet szemantikus webhez, mert a JS kód nem az adat/struktúra része
  - `<input type="button" value="Kattints rám!" onClick="alert('Hahó!')">`
- Ajánlott:
  - külső fájl használata,
  - mert áttekinthető, és a böngésző gyorstárába kerül a fájl, így nem kell többször letölteni

JavaScript elhelyezése a weboldalon

- XHTML: a script nem elemezhető, ezért CDATA részbe kell helyezni:

XHTML és JavaScript

```
<script>//<![CDATA[
// Ide kell írni a JavaScript kódot.
//]]></script>
```

Fejlesztési megközelítések

- Feltesszük, hogy mindenhol használható JavaScript
  - rossz ötlet, mert kikapcsolhatja a felhasználó,
  - valamilyen beépülő modul, pl. biztonsági okokból,
  - konzolos eszközökön (pl. `wget`) nincs,
  - de néhány mobil eszközön sincs,
  - szoftver ágensek egy részének (pl. kereső bot) sincs JS értelmezője, stb.
- Elegáns visszalépés (degrade gracefully)
  - feltételezünk bizonyos képességeket, aztán ha ezek nem állnak rendelkezésre, akkor
  - probléma jelzése, pl. <noscript>Nincs JS támogatásod!</noscript>
- Fokozatos fejlődés (progressive enhancement)

Fejlesztési megközelítések

- Fokozatos fejlődés (progressive enhancement)
  - Csak alapfunkciókat biztosítunk, de
  - ha a böngésző képes valamit jobban is megoldani, átváltunk arra.
  - Böngésző típus ellenőrzés vagy funkció ellenőrzés?
- Tartózkodó JavaScript (unobtrusive JS)
  - nem változtatjuk meg a szokásos működést

Változók deklarálása, definiálása

- Gyengén típusos nyelv: az értékadásnál dől el a változó típusa
- Futásidőben bármikor típust válthat
- Deklarálás: var kulcsszóval, vagy anélkül (hatókör!)

Változók deklarálása

```
valt1 = 3;
var valt2 = 4, valt3 = "szöveg";
var valt4;

valt2 = 'béka';
```

Változók deklarálása, definiálása

- Változó neve tartalmazhat:
  - betűket, számokat, hangosközt és néhány egyéb karaktert is, de
  - betűvel, hangosközzel vagy $ jellel kell kezdődnie,
- Nem tartalmazhat pl. szóközt, pontot és még néhány egyéb jelet. Nem használhatóak a foglalt szavak (pl. JavaScript), előredefiniált változók nevei (pl. window)
- A nevek kis- és nagybetűre érzékenyek → konvenció: *camel-case* névadás, mint Java-ban, pl. hallgatoNeve.
- Konstansok: const kulcsszóval hozhatók létre, de nem minden böngésző támogatja! Írásmód: csupa nagybetű, hangosköz szeparátorral, pl. const NEPTUN_HOSSZ = 6;

Típusok

- Egyszerű típusok (literal) létrehozhatók objektumként (object) is, de így lassabb a működés. Bizonyos feladatok implicit objektum létrehozással járnak.
- Egyszerű
  - Number, lebegőpontos számok
  - String, idézett karakter literál
    - 'szöveg', "szöveg"
    - "Guns 'n' Roses", 'ha "idézni" kell'
    - Escape szekvenciák: `\'`, `\"`, `\n`, `\t`, `\r`, `\f`, `\a`, `\b`, `\\`
  - Boolean: true, false
  - null
  - undefined: nem „igazi" típus, deklaráció után, illetve értéket vissza nem adó fv. esetén
- Összetett

Számok

- Aritmetikai operátorok: +, -, *, / (lebegőpontos osztás), % (lebegőpontos maradék képzés)
- Növelés és csökkentés: ++, -- (prefix és postfix formában)
- Hiba esetén: NaN (Not a Number), Infinity
- Létrehozás: var sz = 3; var sz2 = new Number(3.14);
- Gyakran használt metódusok:
  - toFixed() tizedesjegyek száma
  - toPrecision() összes számjegykarakter száma
- Konstansok:
  - MAX_VALUE, MIN_VALUE

Operátorok

Precedencia	Operátor	Megjegyzés
1	. []	tagok elérése
1	new	példányosítás
2	()	fv. hívás
3	++ --	növel, csökkent
4	!	log. nem
4	+ -	előjelek
4	typeof void delete	
5	* / %	aritmetikai op.
6	+ -	összead, kivon
8	< <= > >=	relációk
9	== != === !==	egyezőség
13	&&	log. és
14	||	log. vagy
15	?:	feltételes op.
16	= += -= *= /= %= <<= >>= >>>= &= ^= |=	összetett op.

Számpélda

fahrenheit.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Fahrenheit - Celsius átváltó</title>
  </head>
  <body>
    <form id="valtoUrlap">
       <fieldset>
          <legend>Adja meg a Fahrenheit fokot!</legend>
          <div><label for="fahrenheit">Fahrenheit</label>
               <input type="text" name="fahrenheit" id="fahrenheit"
               value="72" pattern="[-+]?[0-9]*\.?[0-9]+"
               required="required" title="lebegõpontos szám" /></div>
          <div><label for="celsius">Celsius</label>
               <input type="output" name="celsius" id="celsius"
               title="számolás eredménye" /></div>
          <div><input type="submit" value="Átvált" id="submit" /></div>
        </fieldset>
    </form>
    <script src="atvalto.js"></script>
  </body>
</html>
```

Számpélda

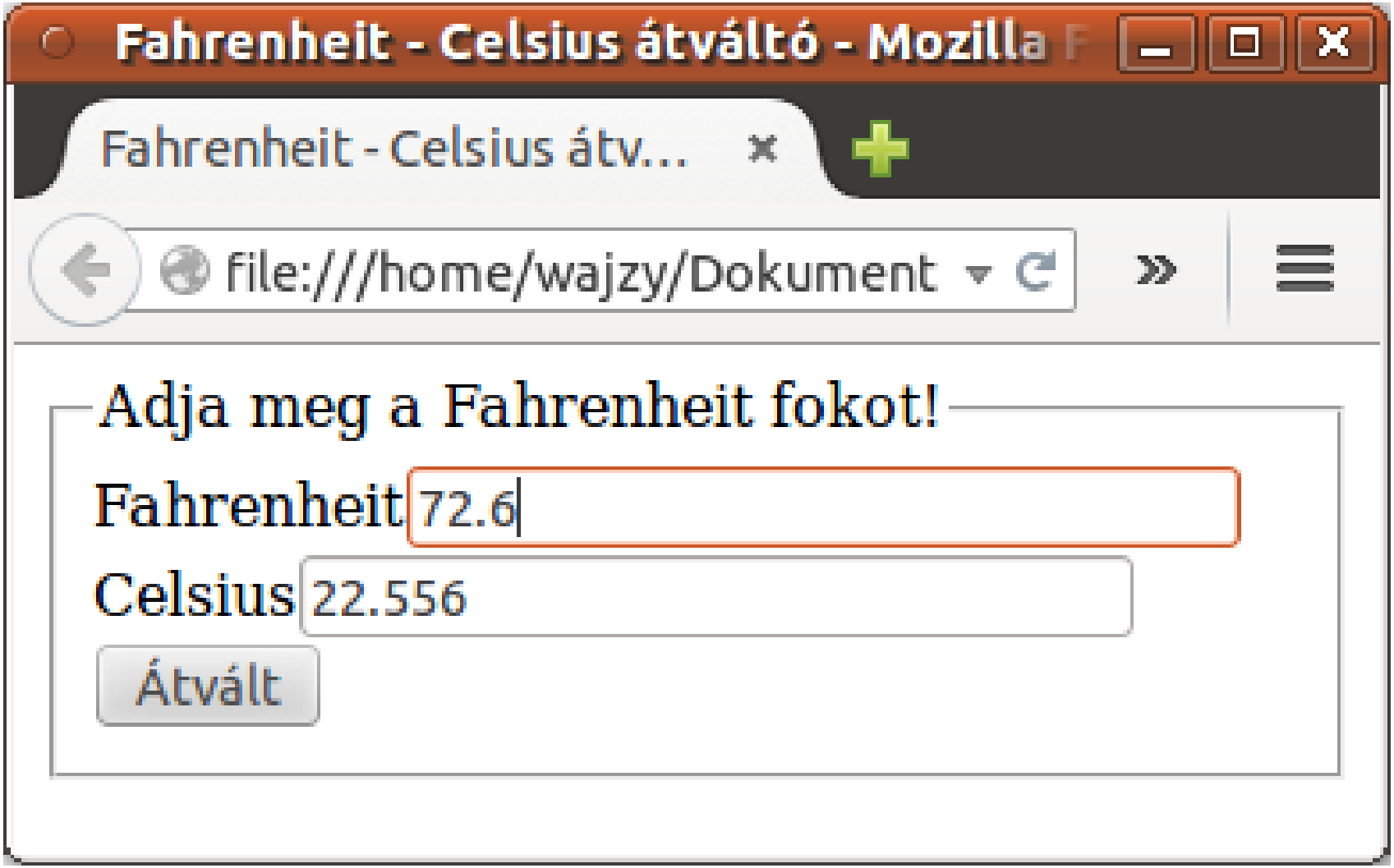
atvalto.js

```
function atvalt() {
  'use strict';
  var cels;
  var fahr = document.getElementById("fahrenheit").value;
  cels = 5/9*(fahr-32);
  document.getElementById("celsius").value = cels.toFixed(3);
  return false;
}

function init() {
  'use strict';
  var urlap = document.getElementById("valtoUrlap");
  urlap.onsubmit = atvalt;
}

window.onload = init;
```

Számpélda


Fahrenheit - Celsius átváltó - Mozilla F
Fahrenheit - Celsius átv...
file:///home/wajzy/Dokument
Adja meg a Fahrenheit fokot!
Fahrenheit 72.6
Celsius 22.556
Átvált

Math objektum

Konstansok:

- PI, 3.1415...
- E, 2.71...

Metódusok:

- abs(), abszolút értéket adja vissza
- ceil(), felfelé kerekít
- floor(), lefelé kerekít
- min(), a tetszőleges számú paraméter közül a legkisebbet szolgáltatja
- max(), mint *min()*, csak a legnagyobbat adja
- pow(), hatványozás (alap, kitevő)
- round(), kerekítés a legközelebbi egészre
- random(), véletlen szám a [0;1) intervallumból
- sin(), cos(), tan(), trigonometrikus fv.-ek, argumentum radiánban
- asin(), acos(), atan(), az előzőek inverzei
- exp(), log(), exponenciális fv., természetes alapú logaritmus
- sqrt(), négyzetgyök vonása

String objektum

Immutable object, mint Java-ban
Létrehozás:

- var str = new String("egy");
- var str2 = "kettő";
- var str3 = 'három';

Tulajdonság (nyilvános konstans adattag):

- length, a karakterlánc hossza

Metódusok:

- charAt(), adott indexű karaktert ad vissza, pl.
  "JavaScript".charAt(0); // "J"

String objektum

Metódusok:

- indexOf(), rész-karakterlánc keresése, pl.
```
"JavaScript".indexOf("a"); // 1
"JavaScript".indexOf("Script"); // 4
"JavaScript".indexOf("a", 2); // 3
"JavaScript".indexOf("A"); // -1
```
- lastIndexOf(), rész-karakterlánc keresése visszafelé, pl.
```
"JavaScript".lastIndexOf("a"); // 3
```
- slice(), rész-karakterlánc visszaadása, pl.
```
"JavaScript".slice(4); // "Script"
"JavaScript".slice(0, 4); // "Java"
"JavaScript".slice(0, -6); // "Java"
"JavaScript".slice(-6); // "Script"
```
- substring(), mint *slice()*, kis eltérésekkel
- substr(), mint *slice()*, de itt a karakterek darabszámát várja, nem a záró elem indexét, pl.
```
"JavaScript".substr(4,6); // "Script"
```

String objektum

Metódusok:

- concat(), összefűzés, pl.

```
"Java".concat("Script", " 1.8.5"); // "JavaScript 1.8.5"
var str = "Java" + "Script"; // "JavaScript"
str += " 1.8.5"; // "JavaScript 1.8.5"
```

- toLowerCase(), kisbetűsre alakítás, pl.

```
"JavaScript".toLowerCase(); // "javascript"
```

- toUpperCase(), nagybetűsre alakítás, pl.

```
"JavaScript".toUpperCase(); // "JAVASCRIPT"
```

- trim(), kezdő és záró fehér karakterek törlése, JS 1.8.1+, pl.

```
" a ".trim(); // "a"
```

- split(), elválasztó jel mentén tördelés, tömb készítés, pl.

```
"a b c".split(" "); // [ "a", "b", "c" ]
```

Típusok közötti átalakítások

Probléma: gyenge típusosság és a + operátor kettős működése

- 12.3 + 4 // 16.3
- "12.3" + 4 // 12.34
- parseFloat("12.3") + 4 // 16.3
- parseInt("12.3") + 4 // 16
- parseInt("12szamár") // 12
- parseInt("szamár12") // NaN
- +"12.3" + 4 // 16.3
- parseInt("0xFF", 16) // 255
- parseInt("FF", 16) // 255
- var sz = 5; sz.toString() // "5"
- new Number(5).toString() // "5"

Feltételek, *if*

- if(*feltétel*) *utasítás*;
- if(*feltétel*) {
  // utasítások
  }
- if(*feltétel*) {
  // igaz ág utasításai
  } else {
  // hamis ág utasításai
  }

- Mikor teljesül a feltétel, és mikor nem?
  - false
  - 0
  - ""
  - NaN
  - null
  - undefined

Operátorok

Feltételes operátorok

- < kisebb
- <= kisebb vagy egyenlő
- > nagyobb
- >= nagyobb vagy egyenlő
- == egyenlő
- != nem egyenlő
- === értéke és típusa is azonos
- !== értéke vagy típusa nem azonos

Logikai operátorok

- && és
- || vagy
- ! nem

Feltételek, *switch*

```
switch(kifejezés) {
    case érték1:
        // utasítások
        break;
    case érték2:
    case érték3:
        // utasítások
        break;
    default:
        // utasítások
        break;
}
```

Az értéknek *és* a típusnak is egyeznie kell!
A *default* ág elhagyható.

Feltételek, három operanduszú operátor

(*kifejezés*) ? *igaz_ág* : *hamis_ág*;

var megisszuk = (ital=="sör") true : false;

Összehasonlítási nehézségek

Lebegőpontos ábrázolás miatt néha az értékek nem egyeznek. Lehetséges megoldások:

- *toFixed()*
- mindig egész számok használatára törekedni, pl. áfa 1.27 helyett 127

De if (NaN === NaN) sosem teljesül! Megoldás:

- isNaN()
- isFinite()

typeof

```
if(typeof valtozo == "string") ...
```

Típus	Eredmény
Undefined	undefined
Null	object
Boolean	boolean
Number	number
String	string
tömb	object
objektum	object

Ciklusok

```
for(előkészítő_kifejezések; ismétlési_feltétel;
      ciklusmag_utáni_kifejezések) {
      // Ciklusmag utasításai
}

while(ismétlési_feltétel) {
      // Ciklusmag utasításai
}

do {
      // Ciklusmag utasításai
} while (ismétlési_feltétel);
```

break, continue

Date objektum

Példány létrehozása:

- new Date(): a kliens órájának állását veszi fel
- new Date(*ezredmp*): a Unix-időszámítás kezdetétől számított ennyi ezredmásodperces időbélyeget reprezentálja
- new Date(*dátumString*): karakterláncként adott időbélyeg ábrázolása, formátum
- new Date(*év*, *hónap*, *nap* [, *óra*, *perc*, *másodperc*, *ezredmp*]): külön-külön megadott óra, perc, ... értékek megadása. Az utolsó négy közül tetszőleges számú elhagyható, de csak hátulról előre, összefüggő egységként

Figyelem!

- hónap [0;11]
- nap [1;...]
- óra [0;23]

Date objektum

Metódusok

- get / setTime(), Date.now()*: időbélyeg
- get / setFullYear(): év
- get / setMonth(): hónap, január = 0
- get / setDate(): hónapon belüli nap
- getDay(): a hét napja, vasárnap = 0
- get / setHours(): óra
- get / setMinutes(): perc
- get / setSeconds(): másodperc
- get / setMilliseconds(): ezredmp.

*ECMAScript 5

Date objektum

Exportálás különleges formátumokra:

- toDateString() // Wed Oct 24 2012
- toISOString()* // 2012-10-24T15:53:35.970Z
- toJSON()* // 2012-10-24T15:53:35.970Z
- toLocaleDateString() // 2012-10-24
- toLocaleString() // 2012. okt. 24., szerda, 17.53.35 CEST
- toLocaleTimeString() // 17.53.35
- toString() // Wed Oct 24 2012 17:53:35 GMT+0200 (CEST)
- toTimeString() // 17:53:35 GMT+0200 (CEST)
- toUTCString() // Wed, 24 Oct 2012 15:53:35 GMT
  (ld. getTimezoneOffset())

*ECMAScript 5

Ellenőrzés: if(datum.toJSON) ...

Tömbök létrehozása

Objektumként:

- var tomb = new Array();
- var tomb = new Array(1, 2, 3);
- var tomb = new Array(3.14, "Hahó", false);

Literál:

- var tomb = [];
- var tomb = [1, 2, 3];
- var tomb = [3.14, "Hahó", false];

Tömbök használata

```
var tomb = new Array(1, 2, 3); // [1, 2, 3]
console.log(tomb.length); // 3; A tömb 3 elemű(nek tűnik)
console.log(tomb[1]); // 2; Az 1 indexű elem értéke 2
tomb[3] = 4;
console.log("Kiegészített tömb: " + tomb);
    // Kiegészített tömb: 1,2,3,4
tomb[9] = 10;
console.log("Definiálatlan elemek: " + tomb);
    // Definiálatlan elemek: 1,2,3,4,,,,,,10
console.log(tomb.length); // 10
console.log(tomb[8]); // undefined
tomb[] = 11; // SyntaxError: syntax error

tomb = new Array(3);
console.log(tomb);
    // [undefined, undefined, undefined]
```

Tömbök használata

Hiányos tömbök

```
var ht = [1, , 3]; // [1, undefined, 3]
ht = [1, , 3, ]; // [1, undefined, 3]
ht = [1, undefined, 3, undefined]; // [1, undefined, 3, undefined]
```

Keresés (ECMAScript 5)

```
var sor = ["Kozel", "HB", "Dreher"];
sor.indexOf("HB"); // 1
sor.indexOf("Soproni"); // -1
sor[sor.length] = "HB";
sor.lastIndexOf("HB"); // 3
// Itt is használható lenne az opcionális második paraméter
```

Tömbök használata

Törlés

```
var sor = ["Kozel", "HB", "Dreher", "Kőbányai"];
delete sor[3];
console.log(sor); // ["Kozel", "HB", "Dreher", undefined]
```

Hozzáadás, elvétel...

```
var csajok = ["Rozi", "Marika", "Nárcisz"];
csajok.push("Melinda");
      // 4; ["Rozi", "Marika", "Nárcisz", "Melinda"]
csajok.pop(); // "Melinda"; ["Rozi", "Marika", "Nárcisz"]
csajok.unshift("Melinda");
      // 4; ["Melinda", "Rozi", "Marika", "Nárcisz"]
csajok.shift(); // "Melinda"; ["Rozi", "Marika", "Nárcisz"]
```

Tömbök használata

Többdimenziós tömbök

```
var emberek = [["Rozi", "Marika", "Nárcisz"],
      ["Arnold", "Dávid", "Laci", "Szilárd"]];
console.log(emberek.length); // 2
console.log(emberek[0].length); // 3
console.log(emberek[1].length); // 4
console.log(emberek[1][2]); // "Laci"
```

Új elemek hozzáfűzése

```
emberek.concat(["Zoli", "Tihamér"]);
      // [["Rozi", "Marika", "Nárcisz"],
      // ["Arnold", "Dávid", "Laci", "Szilárd"],
      // "Zoli", "Tihamér"]
```

Tömbök használata

Kivág, beilleszt...

```
var csajok = ["Rozi", "Marika", "Nárcisz"];
console.log(csajok.splice(1, 2)); // ["Marika", "Nárcisz"]
console.log(csajok); // ["Rozi"]
csajok.splice(1, 0, "Marika", "Nárcisz");
      // [ ]; ["Rozi", "Marika", "Nárcisz"]
console.log(csajok.splice(-1, 1)); // ["Nárcisz"]
```

Tehát a a paraméterezés: index, darabszám, új elemek
Módosítja az eredeti tömböt.

Tömbök használata

Kivág, beilleszt...

```
var csajok = ["Rozi", "Marika", "Nárcisz"];
console.log(csajok.slice(0, 2)); // ["Rozi", "Marika"]
console.log(csajok); // ["Rozi", "Marika", "Nárcisz"]
console.log(csajok.slice(1)); // ["Marika", "Nárcisz"]
console.log(csajok.slice(-1)); // ["Nárcisz"]
console.log(csajok.slice(-3, -1)); // ["Rozi", "Marika"]
```

Tehát a a paraméterezés: kezdőIndex, végIndex+1
Nem módosítja az eredeti tömböt.

Tömbök használata

Egyesítés. . .

```
var csajok = ["Rozi", "Marika", "Nárcisz"];
var lista = "<ul><li>";
lista += csajok.join("</li><li>");
lista += "</li></ul>";
```

. . . feldarabolás

```
var csajok = "Rozi,Marika,Nárcisz";
var cst = csajok.split(",");
console.log(cst); // ["Rozi", "Marika", "Nárcisz"]
```

Objektumok

Néhány megjegyzés:

- JS-ben nincs öröklés, az objektumok prototípusok alapján készülnek
- megkönnyíti az objektumok létrehozását, de néhány OO technika nehezen kivitelezhető
- az objektumok adattagokat és metódusokat zárnak egybe
- de ezek sorrendjét nem őrzi meg

Létrehozás módjai:

- `new` operátorral: `var obj = new Object();`
- literál szintakszissal: `var obj = {};` (ajánlott)

Adattagok felvétele

Szintaktika:

- Új adattag: adattag_neve : érték
- Adattagokat , választja el egymástól
- Új adattag utólag is felvehető: auto.szin = "fehér";
- és törölhető: delete auto.szin;

Minta objektum

```
var auto = {
  gyarto: "Hindustan",
  tipus: "Ambassador",
  evjarat: 1958,
  gyartasban: true,
  felszereltseg:
    ["tekerhető kormány", "felhúzható ablak"],
  tulajdonos: {
    vezeteknev: "Gipsz",
    keresztnev: "Jakab"
  }
};
```

Adattagok elérése

Az objektum nevét az adattag nevétől a . operátor választja el.

```
console.log(auto.evjarat) // 1958
console.log(auto.tulajdonos.vezeteknev) // Gipsz
console.log(auto.felszereltseg[1]) // felhúzható ablak
```

Adattag létezésének ellenőrzése

- if(auto.szin) { ... // Nem biztonságos, pl. "" → false
- if("szin" in auto) { ... // true

Kapcsolat a(z asszociatív) tömbök és az objektumok között

- auto.tulajdonos.keresztnev
- auto["tulajdonos"]["keresztnev"]

Utóbbi akkor is használható, ha egy változó tartalmazza az adattag nevét.

Adattagok elérése

A for/in ciklussal minden adattag/metódus elérhető.

Objektum kiírása

```
for(var t in auto) {
  console.log(t + ": " + auto[t]);
}
```

Kimenet

```
gyarto: Hindustan
tipus: Ambassador
evjarat: 1958
gyartasban: true
felszereltseg: tekerhető kormány,felhúzható ablak
tulajdonos: [object Object]
szin: fehér
```

Függvények készítése

Általános tudnivalók:

- A fv. nevének képzése a szokásos szabályok szerint történik (nem kezdődhet számjeggyel, de tartalmazhat alfanumerikus karaktereket, hangosközt, $ jelet, stb., nem használhatóak a foglalt szavak, ...)
- Tetszőleges számú paraméter fogadására felkészíthető
- Nem kötelező értékkel visszatérnie

Egy függvény váza

```
function függvény_neve(par1, par2, ...) {
  // Függvény törzse
  return; // elhagyható --> undefined
}
```

Hívás

```
függvény_neve(par1, par2, ...); // Önálló fv. hívása
objektum.metódus(par1, par2, ...); // Obj. met. hív.
```

Függvények paraméterezése

Fontosabb tudnivalók:

- Gyenge típusosság → csak változónevek szerepelnek, típust nem adjuk meg → nincs típusellenőrzés!
- Ha erre szükség van, nekünk kell megoldani!

Négyzetre emelő fv.

```
function negyzet(alap) {
  return alap * alap;
}
console.log(negyzet(3)); // 9
console.log(negyzet("Malacka")); // NaN
```

Négyzetre emelő fv., típusellenőrzéssel

```
function negyzet(alap) {
  if(typeof alap == "number") {
    return alap * alap;
  }
}
console.log(negyzet(3)); // 9
console.log(negyzet("Malacka")); // undefined
```

Függvények paraméterezése

Fontosabb tudnivalók, folyt.:

- A fv.-t tetszőleges számú paraméterrel lehet hívni (formális paraméterek száma nem számít!)
- Kevesebb aktuális paraméter: undefined
- Több: csak az arguments tömb-szerű objektummal érhetőek el
- Átadás módja: objektumok (tömbök) referencia szerint, minden más típus érték szerint

Tetszőleges mennyiségű szám összeadása

```
function osszeg() {
  var osszeg = 0;
  for(var i=arguments.length-1; i>=0; i--) {
    if(typeof arguments[i] == "number") {
      osszeg += arguments[i];
    }
  }
  return osszeg;
}
console.log(osszeg()); // 0
console.log(osszeg(1, 2, 3)); // 6
```

Függvények paraméterezése

Eltérő számú aktuális paraméter

```
function teszt(par1, par2) {
  console.log("par1: " + par1 + ", par2: " + par2);
}
teszt(); // par1: undefined, par2: undefined
teszt("a"); // par1: a, par2: undefined
teszt("a", "b"); // par1: a, par2: b
teszt("a", "b", "c"); // par1: a, par2: b
teszt(undefined, "b"); // par1: undefined, par2: b
```

A paramétereknek nincs alapértelmezett értéke $\rightarrow$ nekünk kell gondoskodni róla!

Alapértelmezett érték

```
function teszt(par1, par2) {
  if(typeof par1 == "undefined") {
    par1 = "n/a";
  }
  console.log("par1: " + par1 + ", par2: " + par2);
}
teszt(undefined, "b"); // par1: n/a, par2: b
```

Függvények paraméterezése

Érték szerinti átadás

```
function novel(sz) {
  sz++;
  console.log("sz: " + sz);
}
var szam = 1;
console.log("szam: " + szam); // szam: 1
novel(szam); // sz: 2
console.log("szam: " + szam); // szam: 1
```

Referencia szerinti átadás

```
function novel2(sz) {
  sz.adat++;
  console.log("sz.adat: " + sz.adat);
}
var objektum = {
    adat: 1
};
console.log("objektum.adat: " + objektum.adat); // objektum.adat: 1
novel2(objektum); // sz.adat: 2
console.log("objektum.adat: " + objektum.adat); // objektum.adat: 2
```

Változók hatóköre

Lokális

```
function fv() {
  console.log("fv() -> szam: " + szam);
  var szam = 2; // A "var" lokális változót vezet be!!!
  console.log("fv() -> szam: " + szam);
}

var szam = 1;
console.log("szam: " + szam);
fv();
console.log("szam: " + szam);
```

Kimenet

```
szam: 1
fv() -> szam: undefined
fv() -> szam: 2
szam: 1
```

Változók hatóköre

Globális

```
function fv() {
  console.log("fv() -> szam: " + szam);
  szam = 2; // A globális változó értékét módosítjuk!!!
  console.log("fv() -> szam: " + szam);
}

var szam = 1;
console.log("szam: " + szam);
fv();
console.log("szam: " + szam);
```

Kimenet

```
szam: 1
fv() -> szam: 1
fv() -> szam: 2
szam: 2
```

A függvények objektumok

Ez teszi lehetővé pl. annak lekérdezését is, hogy valamilyen szolgáltatás (objektum, metódus, függvény, stb.) támogatott-e az éppen használt böngészőben.

Függvények létezésének ellenőrzése

```
function min(a, b) {
  return (a<b) ? a : b;
}
console.log(typeof min); // function
console.log(min(1,2)); // 1
console.log(typeof max); // undefined
console.log(typeof Date.now); // function
```

Függvény definíció másképpen

```
var min = function (a, b) {
  return (a<b) ? a : b;
}
console.log(min(1,2)); // 1
```

A függvények objektumok

Emlékezzünk vissza az első JS mintaprogramra!

Eseménykezelő módosítása

```
function init() {
  'use strict';
  var urlap = document.getElementById("valtoUrlap");
  urlap.onsubmit = atvalt;
}
window.onload = init;
```

Ugyanez névtelen (anonymous) függvénnyel

```
window.onload = function () {
  'use strict';
  var urlap = document.getElementById("valtoUrlap");
  urlap.onsubmit = atvalt;
}
```

A függvények objektumok

Névtelen függvényeket szoktunk használni, ha:

- a fv.-t hozzárendeljük egy változóhoz,
- objektum adattagjához, azaz metódust készítünk (ld. később),
- átadjuk egy másik függvénynek paraméterként (ld. később),
- vagy azonnal meg is akarjuk hívni

Azonnal meghívásra kerülő fv.

```
(function() {
  // Függvény törzse
})();
```

A függvények egymásba is ágyazhatóak.

Egymásba ágyazott függvények

```
function kulso() {
  console.log("kívül");
  function belso() {
    console.log("belül");
  }
  belso();
  console.log("kívül");
}
kulso();
```

Kimenet

```
kívül
belül
kívül
```

A függvények objektumok

Függvény paramétere is lehet függvény. Pl. az Array.sort() fogadhat rendezési relációt meghatározó függvényt.

Rendezés növekvő és csökkenő sorrendbe

```
var csajok = ["Rozi", "Marika", "Melinda", "Nárcisz"];
console.log(csajok.sort());
  // ["Marika","Melinda","Nárcisz","Rozi"]
function forditva(a, b) {
  if(a < b) {
    return 1;
  } else if(a == b) {
    return 0;
  } else {
    return -1;
  }
}
console.log(csajok.sort(forditva));
  // ["Rozi","Nárcisz","Melinda","Marika"]
```

A függvények objektumok

Tömbök függvény paramétert fogadó függvényei az ECMAScript 5-ben

- **forEach()** Minden elemet egyesével átad a függvénynek.
- **every()** Ha a tesztelő fv. minden elemre igazzal tér vissza, akkor ez is igazat ad.
- **some()** Ha a tesztelő fv. legalább egy elemre igazzal tért vissza, akkor ez is igazat ad.
- **filter()** Új tömböt készít és ad vissza, amelyben azok az elemek szerepelnek, amelyekre a tesztfüggvény igazat adott.
- **map()** Új tömböt készít és ad vissza, amelynek elemei az eredetinek a paraméter függvénnyel történő átalakításával jöttek létre.
- **reduce()** Balról jobbra összevonja az elemeket egy egyszerű változóba.
- **reduceRight()** Jobbról balra összevonja az elemeket egy egyszerű változóba.

Kis értékű elemek kiszűrése

```
console.log(
  [10, 20, 30].filter(
    function (elem) {
      return elem > 15;
    }
  )
); // [20,30]
```

Metódusok

Metódus: objektum függvénye. Adattagok és metódusok a this segítségével érhetőek csak el!

Téglatest felszínének és térfogatának számítása

```
var teglatest = {
  szel: 2,
  hossz: 3,
  mag: 5,
  felszin: function() {
    return (this.szel*this.hossz +
      this.hossz*this.mag +
      this.mag*this.szel) * 2;
  },
  terfogat: function() {
    return this.szel * this.mag * this.hossz;
  }
};
console.log("Felszín: " + teglatest.felszin());
  // Felszín: 62
console.log("Térfogat: " + teglatest.terfogat());
  // Térfogat: 30
```