WEBES ALKALMAZÁSFEJLESZTÉS 1.

Horváth Győző

Egyetemi adjunktus

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

Függvény JavaScriptben

Függvények

- Olyan objektum, ami meghívható kódot tartalmaz
- Általános tudnivalók
  - visszatérési értékkel rendelkezik
    - return kulcsszó
    - undefined
  - Tetszőleges számú paraméter
- First class object
  - Futási időben létrehozható
  - Objektumok, saját metódussal és tulajdonságokkal
  - Változóban tárolható
  - Paraméterként átadható
  - Visszatérési értékként megadható

Függvény létrehozása

Függvénydeklaráció

- Kötelező név
- Egész futási kontextusban elérhető

```
function multiply(a, b) {
  return a * b;
}
```

```
// Valójában már itt elérhetõ a getHero függvény
var hero = getHero();
assertEquals('Superman', hero);
function getHero() {
    return 'Superman';
}
```

Függvénykifejezés

- Függvény mint kifejezés (függvényliterál):

```
function [name]([param] [, param] [..., param]) {
   statements
}
```

- Névtelen függvénykifejezés:

```
//Névtelen függvénykifejezés átadása egy változónak
var multiply = function(a, b) {
     return a * b;
};
```

- Nevesített függvénykifejezés:

```
//Névtelen függvénykifejezés átadása egy változónak
var multiply = function multiply(a, b) {
     return a * b;
};
```

Fat arrow szintaxis (ES6)


```
//Általános forma
([param[, param[, ... param]]]) => {
   statements
}

//Egyszerűbb eset
param => expression
```

```
var heroes = [
  'Superman',
  'Spiderman',
  'Sugarman'
];
// Ha össze szeretnénk számolni a hősök neveinek hosszát, akkor legegyszerűbb a map függvényt használni:
heroes.map(function(hero) { return hero.length; });
// A Fat arrow operátorral lényegesen expresszívebb kódot kapunk:
heroes.map( hero => hero.length );
```

Function objektum

- Függvénytörzs szövegként
- Dinamikusan generált függvények
- Speciális esetekben használandó
- Egyébként kerüljük!

```
var multiply = new Function('a, b', 'return a * b');
```

Deklaráció vs kifejezés


```
a();
//b(); //nem működik
function a() {
    console.log('a');
}
var b = function () {
    console.log('b');
};
b();
```

```
var c = function c() {
    console.log('c');
};
c();
var d1 = function d2() {
    console.log('d');
};
d1();
//d2(); //nem működik,
csak belül érhető el
```

Változók a függvényekben

Lokális változók

- **var** kulcsszó
- Változókat bárhol deklarálhatunk a függvényen belül
- A függvénytörzs első műveleteként jönnek létre, és értékük undefined.
- Érdemes a legelején létrehozni (JSLint)

Lokális változók viselkedése


```
var valtozoVizsgalat = function() {
    console.log(valtozo);              // => undefined, ha nem hoznánk létre a
                                          következő sorban, akkor itt most
                                          referenceError lenne
    var valtozo = 5;                   // Innen már 5 a valtozo értéke
}

valtozoVizsgalat();

// A fenti függvény valójában ezzel ekvivalens:
var valtozoVizsgalat = function() {
    var valtozo;
    console.log(valtozo);                                  // => undefined
    valtozo = 5;
}
```

Hatókör

- Minden változó hatóköre az a függvény, ahol őt létrehozzuk
- Erre a függvényre nézve lokális változó lesz
- A függvényen belül definiált újabb függvényeken belül el tudjuk érni.
- Belső függvényben definiált ugyanolyan nevű változó elfedi a külső függvénybeli változót
- Belső függvényből módosítható külső függvénybeli változó. Vigyázzunk vele!

Hatókör


```
var outerFunction = function() {
    var a = 1,
        b = 2,
        c = 3,
        innerFunction = function(b) {
            var c = 4,
                d = 5;
            a = 2;
            // Ezen a ponton a globális "b" és "c" változó
            // el van fedve, így "b" undefined, a "c" lokális
            // változó értéke pedig 4. A "d" egy új lokális
            // változó, a függvényen kívül nincs definiálva
        }
        // Ezen a ponton a,b,c értéke a scope-nak
        // megfelelõen 1,2,3 és d változó nem létezik
        innerFunction();
        // A függvény lefutása után az "a" értéke módosul
        // 2-re, hiszen az innerFunction nem fedi el,
        // sõt megváltoztatja azt!
};
```

ES6 let


```
//Function scope
function funcExample() {
  for (var i = 0; i<3; i++) {
    console.log(i);
  }
  assertEquals(i, 3);
}
funcExample();

//Block scope
function blockExample() {
  for (let i = 0; i<3; i++) {
    console.log(i);
  }
  assertEquals(i, undefined);
}
blockExample();
```

Változók élettartama

- Függvényekben létrehozott változók élettartama addig tart, amíg azok használatba vehetőek (mutat rájuk referencia)
- Alapesetben a függvény futási ideje
- Garbage collector
  - Szükséges lexikális scope-pal rendelkező nyelvek esetén (ld. closure)

Függvényparaméterek

Paraméterek száma

- Aktuális paramétereinek száma eltérhet a formálisakétól
  - Ha kevesebb aktuális van, akkor a maradék formális undefined
  - Ha több aktuális van, akkor is elérhető (arguments, rest parameters)

Paraméterek száma


```
var parProba = function (a, b) {
    console.log('a = ', a);
    console.log('b = ', b);
    console.log('arguments = ', arguments);
}

parProba(1, 2);
    // a = 1
    // b = 2
    // arguments = [1, 2]
parProba(1);
    // a = 1
    // b = undefined
    // arguments = [1]
parProba(1, 2, 3);
    // a = 1
    // b = 2
    // arguments = [1, 2, 3]
```

Paraméterlista dinamikus kezelése

- Minden függvény rendelkezik egy **arguments** nevű tömbszerű objektummal
- Változó paraméterszám kezelésére szolgál
- Nem tömb, csak tömbszerű
  - length tulajdonság
  - Nincsenek tömbmetódusok
  - EcmaScript 5 tömbszerűbbé teszi
- arguments objektum módosításával a paraméterek értékei is változnak – oda-vissza igaz (ES3)

arguments példa


```
var osszegzo = function() {
    var osszeg = 0;
    for (var i = 0, l = arguments.length; i < l; i += 1) {
        osszeg += parseInt(arguments[i], 10);
    }
    return osszeg;
}

osszegzo(1,2);     // => 3
osszegzo(2,3,4);   // => 9
```

arguments tömbösítése

- Külön függvénnyel
- Tömbfüggvény kölcsönvételével

```
var args = Array.prototype.slice.call(arguments, 0);

//vagy

var args = [].slice.call(arguments, 0);
```

ES6 paraméterkezelés

- Alapértelmezett értékek (tetszőleges kifejezés):

```
function defPars(a, b = 42) {
  return [a, b];
}
assertEquals( [1, 2],          defPars(1, 2) );
assertEquals( [1, 42],         defPars(1) );
assertEquals( [undefined, 42], defPars() );
```

- Változó hosszúságú paraméterlisták:

```
function restPars(a, ...others) {
  return others;
}
assertEquals( [2, 3], defPars(1, 2, 3) );
assertEquals( [],     defPars(1) );
assertEquals( [],     defPars() );
```

Konfigurációs objektum

- Függvénynek objektumliterál átadása
- Függvénytörzsben ellenőrizni

```
var jsAblak = function (opts) {
    if (!opts) opts = {};              // ha nem adunk meg opts paramétert, legyen
                                           üres objektum
    opts = {                           // alapértékek megadása
        szelesseg:  opts.szelesseg  || 500,
        magassag:   opts.magassag || 300,
        cim:        opts.cim  || 'Névtelen ablak'
    }
    //...
}

jsAblak({
    szelesseg: 100,
    cim: 'Ablakom'
});
```

Konfigurációs objektum

- Előnyei
  - Nem kell a paraméterek sorrendjére emlékezni
  - Opcionális paraméterek kihagyhatók
  - Olvashatóbb, karbantarthatóbb kód
  - Könnyebb hozzáadni, elvenni paramétereket
- Hátrányai
  - Emlékezni kell a paraméterek nevére
  - Nem tömöríthető (minify)
- Hasznos
  - DOM elemek készítésénél
  - CSS tulajdonságok megadásánál

Függvénykifejezés használata

Kifejezés használata


```
//literál
6



//értékadás
var szam = 6;



//Objektum adattagja
var obj = {
    szam: 6
};



//Függvényparaméter
var f = function (p) {
    return p;
}
f(6);
```

```
//Visszatérési érték
var inic = function () {
    return 6;
}



//Használva
6



//Kifejezés
if (6 === 6) {
    console.log('IGAZ');
}
```

Függvénykifejezés használata


```
//literál
function (a, b) {
    return a * b;
}

//értékadás
var szam = function (a, b) {
    return a * b;
};

//Objektum adattagja
var obj = {
    szam: function (a, b) {
        return a * b;
    }
};

//Függvényparaméter
var f = function (fv, a, b) {
    return fv(a, b);
}
f(function (a, b) {
        return a * b;
    }, 2, 3);
```

```
//Visszatérési érték
var inic = function () {
    return function (a, b) {
        return a * b;
    };
}

//Használva
(function (a, b) {
    return a * b;
})(2, 3);

//Kifejezés
if ((function (a, b) { return a * b; })(2, 3) === 6) {
    console.log('IGAZ');
}
```

Önkioldó függvény

Névterek, privát névtér, névtér függvény

Globális névtér

- Minden függvényen kívül létrehozott változó a globális névtér része
- Ez a böngésző esetében a window objektum
- Minél kevesebb globális változónk legyen a névütközések, felüldefiniálások elkerülése végett

```
var mit = function() {
    window.globalis = 'Juppi, globális vagyok'; // közvetlen a globális
                                                névtérbe hozzuk létre
    return 'azt';
}

mit();                          // => 'azt'
window.mit();                   // => 'azt'
console.log(globalis);          // => 'Juppi, globális vagyok'
console.log(window.globalis); // => 'Juppi, globális vagyok'
```

Globális névtér


```
/* TELJESEN EGYEDI ALKALMAZÁSOM */
var megy = function() {
        return 'ÉS MÉG MEGY!';
    },
    megall = function() {
        return '...ÉS MEGÁLL';
    };
megy();    // => 'ÉS MÉG MEGY!'
megall();  // => '...ÉS MEGÁLL'

// ....
// Később, ugyanabban a kódban előfordulhat:

/* XY. BÉLA GONOSZ ALKALMAZÁSA */
var megy = function() {
    return 'Béla a király!';
}
megy();    // => 'Béla a király!', felülírta az én alkalmazásom
                függvényét
```

Saját névtér

- Változó hatóköre az őt deklaráló függvény
- Ezzel névterek létrehozhatók
- Függvényen belül definiált változók, függvények nem befolyásolják a program ezen kívüli részét

```
/* MAJDNEM EGYEDI NÉVTÉRBEN TELJESEN EGYEDI ALKALMAZÁSOM */
var duracellNyuszi = function() {
    var megy = function() { return 'ÉS MÉG MEGY!'; },
        megall = function() { return '...ÉS MEGÁLL'; };
    megy();    // => 'ÉS MÉG MEGY!'
    megall();  // => '...ÉS MEGÁLL'
}
duracellNyuszi();

// Most már akárhogy gonoszkodhat XY. Béla, nem tud belenyúlni az ő
alkalmazásával a miénkbe.
```

Önkioldó függvény

- Ha csak a névtér miatt hozzuk létre a függvényt, akkor felesleges nevesíteni, hiszen csak a globális névteret szennyezzük vele
- → önkioldó névtelen függvény: olyan függvény, amit létrehozás után rögtön futtatunk is
- Később nem tudunk rá hivatkozni
- Példa
  - Oldalbetöltődéskor eseménykezelők definiálása, objektumok létrehozása
  - Temporális változók

Önkioldó függvény

A) Önkioldó függvény definiálása

```
/* TELJESEN EGYEDI NÉVTÉRBEN TELJESEN EGYEDI ALKALMAZÁSOM */
(function() {
    var megy = function() { return 'ÉS MÉG MEGY!'; },
        megall = function() { return '...ÉS MEGÁLL'; };
    megy();    // => 'ÉS MÉG MEGY!'
    megall();  // => '...ÉS MEGÁLL'
}());
```

B) Önkioldó függvény paraméterezése

```
(function(a, b) {
    console.log(a+b);
}(1,2));       // => 3
```

C) Önkioldó függvény visszatérési értéke

```
var ize = (function(str) {
    return str;
}('bize'));
console.log(ize);    // => 'bize'
```

Függvény mint paraméter

Callback minta

Függvény mint paraméter

- A függvények objektumok, így átadhatók paraméterként
- Sokrétű, általános logikájú függvények kaphatók
- A paraméterként megkapott függvényt visszahívja → callback függvény, callback minta
- Felhasználása
  - Eseménykezelők
  - Időzítők
  - Függvénykönyvtárak

Függvény mint paraméter


```
var showHero = function(name, callback) {
        console.log("I'm " + name);
        if (callback) {
            callback();
        }
    };
showHero('Superman', function() {
    console.log('Callback invoked!');
});
// => "I'm Superman"
// => "Callback invoked!"
```

Függvény mint paraméter


```
var egyenletMegoldo = function(a, b, muvelet) {
    var vegeredmeny;
    if (!muvelet || typeof muvelet !== 'function') {
        console.log('Nem adott meg műveletet!');
        return false;
    }
    console.log('Művelet a(z)', a, 'és a', b, 'értékeken!');
    vegeredmeny = muvelet(a,b);
    console.log('Az eredmény: ', vegeredmeny);
    return vegeredmeny;
}
egyenletMegoldo(3,4); // => false, a logban: 'Nem adott meg műveletet!'

// Egyszerű szorzás
egyenletMegoldo(3,4,
    function(a,b) {
        return a * b;
    });
// => 10, a logban: Művelet a(z) 3 és a 4 értékeken!  Az eredmény: 12
```

Függvény mint paraméter

- Időzítőben
- Külön függvényként csak a referenciát kell átadni
  - Eseménykezelőknél is

```
//Külön függvényként
var elkesem = function () {
    console.log('500ms-ot késtem...');
};
setTimeout(elkesem, 500);

//Függvényliterállal
setTimeout(function () {
        console.log('500ms-ot késtem...');
    }, 500);
```

Paraméteres callback


```
var numArray = [1,2,3].map(function(el) {
  return el * 2;
});
assertEquals([2, 4, 6], numArray);
```

Függvény mint visszatérési érték

Függvény mint visszatérési érték

- Függvények objektumok, ezért visszaadhatóak
- Függvény visszatérési értéke függvény
- Függvénygenerátorok

```
var setup = function () {
    console.log(1);
    return function () {
        console.log(2);
    };
};

var fv = setup(); // 1
fv(); //  2
```

Függvény mint visszatérési érték

- Mi van abban az esetben, ha a külső függvény egy lokális változójára hivatkozom?

```
var setup = function () {
    var lokalis = 1;
    return function () {
        console.log(lokalis);
    };
};

var fv = setup();
fv(); //  ???
```

Closure

Closure

- Zárlat, lexikális scope
- Belső függvény hivatkozhat a külső függvény változóira (függvény scope) az után is, hogy a külső függvény már lefutott
- Egy változó addig marad életben, amíg esély van arra, hogy bárki használja őket
- Closure egy környezet, amiben a változók definiálva vannak
- Szemétgyűjtő szabadítja fel, ha nincs rá referencia

Closure - példák

- Függvény mint visszatérési érték

```
var setup = function () {
    var lokalis = 1;
    return function () {
        console.log(lokalis);
    };
};

var fv = setup();
fv(); //  1
```

- Időzítő

```
var valogatott = function() {
    var statusz = 'Győztünk!';
    var miTortent = function() {
        console.log(statusz);
    }
    setTimeout(miTortent, 1000);
}

valogatott();
// 1mp múlva megjelenik: 'Győztünk!'
```

Függvényalkotó

- Egymáshoz hasonló feladatoknál a paraméter

```
//Sokszor ugyanolyan paraméterrel használt függvény
var hozzaAd = function (mihez, mit) {
    return mihez + mit;
}
console.log(hozzaAd(1000, 42));  //1042
console.log(hozzaAd(1000, 111)); //1111

//Függvényalkotó
var specialisHozzaadotAlkoto = function(mit) {
    return function(mihez) {
        return mihez + mit;
    }
}

var hozzaAd1000 = specialisHozzaadotAlkoto(1000);
console.log(hozzaAd1000(42));    //1042
console.log(hozzaAd1000(111));    //1111
```

Closure hátrányai

- Például ciklusnál

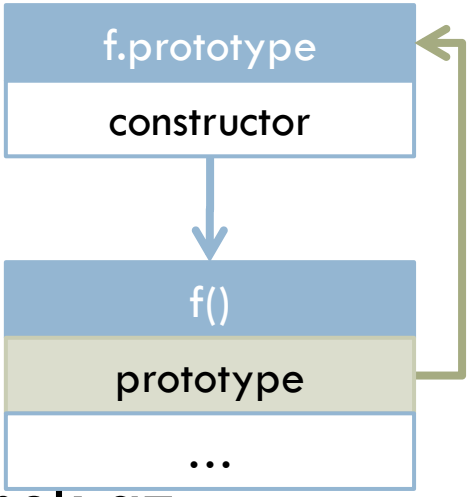
```
(function() {
    var i;
    for (i = 0; i < 3; i += 1) {
        setTimeout(function () {
            console.log(i);
        }, 500);
    }
})();

//Output
// 3
// 3
// 3
```

Függvény mint objektum

Függvény mint objektum

- A függvények is objektumok: vannak adattagjaik és metódusaik
  - prototype: egy objektum, aminek egyetlen constructor nevű adattagja a függvényre mutat
  - length: függvény formális paramétereinek száma
  - apply(): ld. később
  - call(): ld. később
- Két rejtett attribútum
  - függvény kontextusa (ld. closure)
  - függvény kódja
- Az egyetlen különlegessége a függvényeknek az objektumokhoz képest, hogy meghívhatók



Függvény mint objektum


```
var fgvObj = function () {
    return 'Meghívtál, visszatértem';
};

//Saját adattagok hozzáadása
fgvObj.adat = 'Adat vagyok';
fgvObj.metodus = function () {
    return this.adat;
};

console.log(fgvObj());           //'Meghívtál, visszatértem'
console.log(fgvObj.metodus());   //'Adat vagyok'
```

Memoization minta

- Költséges művelet eredményének tárolása

```
var myFunc = function () {
  var cachekey = JSON.stringify(Array.prototype.slice.call(arguments)),
    result;
  if (!myFunc.cache[cachekey]) {
    result = {};
    // ... expensive operation ...
    myFunc.cache[cachekey] = result
  }
  return myFunc.cache[cachekey];
};
// cache storage
myFunc.cache = {};
```

```
var myFunc = function (param) {
  if (!myFunc.cache[param]) {
    var result = {};
    // ... expensive operation ...
    myFunc.cache[param] = result;
  }
  return myFunc.cache[param];
};
// cache storage
myFunc.cache = {};
```

A this jelentése

Függvényhívási minták

- Függvények hívásakor minden függvény két további paramétert kap a deklarált paraméterek mellé
  - this
  - arguments
- A this értéke a függvényhívás módjától függ
- Függvényhívási minták
  - Metódushívási minta
  - Függvényhívási minta
  - Alkalmazkodó függvényhívási minta
  - Konstruktorhívási minta

Globális kontextus


```
// A böngészõkben a globális objektum a window
assertEquals(this, window);
this.hero = 'Superman';
assertEquals('Superman', window.hero);
```

Metódushívási minta

- Ha a függvény egy objektum adattagja, akkor metódusnak hívjuk
- A this az adott objektumra mutat

```
var facebook = {
    tagok: ['Tamás', 'Thomas', 'Tuomas', 'Etelka'],
    regisztraltak: function() {
        return this.tagok;
    }
}

//Metódushívás:
facebook.regisztraltak();
// => ["Tamás", "Thomas", "Tuomas", "Etelka"]
```

Függvényhívási minta

- Ha egy függvény nem metódusa egy objektumnak, akkor a globális névtérben jön létre
- „Hagyományos” függvények
- A globális objektum (window) része lesz
- this: undefined(strict mode) vagy globális (egyéb)

```
var fgv = function () {
    console.log(this);
}

//Függvényhívási minta
fgv();      //window

console.log(fgv === window.fgv);
```

```
window.fgv = function () {
    console.log(this);
}

//Függvényhívási minta
window.fgv();      //window

console.log(fgv === window.fgv);
```

Hívás módja a lényeg


```
var hero = {
    name: 'Superman'
};
function getName() {
    return this.name;
};
// A függvényre mutató referenciát később adjuk hozzá a
// hero objektumhoz, ha így, metódusként hívjuk meg
// akkor a this az objektumra fog mutatni:
hero.getName = getName;
assertEquals('Superman', hero.getName());
assertEquals(undefined, getName());
```

Függvényhívási minta

- Belső függvényben a this a globális objektumra mutat (nyelvtervezési hiba → ES5 strict mode)

```
var name = 'Sugarman',
    hero = {
      name: 'Superman',
      getIntroductionHTML: function() {
        var formattedName = function(tag) {
          return '<' + tag + '>' + this.name + '</' + tag + '>';
        };
        return '<div class="name">' +
                  formattedName('span') +
                '</div>';
      }
    };
assertEquals('<div class="name"><span>Sugarman</span></div>',
  hero.getIntroductionHTML());
```

Függvényhívási minta

- Megoldás: this kimentése → _this, that, self
- vagy strict mode → TypeError

```
var name = 'Sugarman',
    hero = {
      name: 'Superman',
      getIntroductionHTML: function() {
        var that = this;
        var formattedName = function(tag) {
          return '<' + tag + '>' + that.name + '</' + tag + '>';
        };
        return '<div class="name">' +
                  formattedName('span') +
                '</div>';
      }
    }
assertEquals('<div class="name"><span>Superman</span></div>',
  hero.getIntroductionHTML());
```

Alkalmazkodó függvényhívási minta

- Függvénynek lehetnek saját metódusai
- A **call()** és **apply()** segítségével megmondhatjuk, hogy a függvény this paramétere melyik objektumra mutasson (első paraméterük)
  - apply() második paramétere argumentumok tömbje
  - call() 2., 3., stb. paramétere a függvény 1., 2., stb. paramétere lesz
- **bind()**: a függvény kontextusát véglegesen a paraméterül megadott objektumhoz köti

Alkalmazkodó függvényhívási minta


```
var facebook = {
    tagok: ['Tamás', 'Thomas', 'Tuomas', 'Etelka'],
    regisztraltak: function() {
        return this.tagok;
    }
}

var iwiw = {
    tagok: ['Csobilla', 'Csende', 'Cseperke']
}

var tagok = ['Géza', 'Gizi'];

facebook.regisztraltak();
// => ['Tamás', 'Thomas', 'Tuomas', 'Etelka']
facebook.regisztraltak.apply(iwiw);
// => ["Csobilla", "Csende", "Cseperke„]
```

Alkalmazkodó függvényhívási minta

- Ha nem számít a this
- Csak a paraméterezés

```
var numbers = [5, 6, 2, 3, 7];
// Keressük meg a legnagyobb értéket a tömbben. A
// Math.max paramétereiben számokat vár, és azok
// közül választja ki a legnagyobbat. Az apply tökéletes
// választás, hiszen a tömböt argumentumokra bontja
// és így hívja meg a Math.max-ot:
var max = Math.max.apply(null, numbers);
assertEquals(7, max);
```

```
//ES6 spread operator
var numbers = [5, 6, 2, 3, 7];
var max = Math.max(...numbers);
assertEquals(7, max);
```

Alkalmazkodó függvényhívási minta


```
var name = 'Sugarman',
    hero = {
      name: 'Superman',
      getIntroductionHTML: function() {
        var formattedName = (function(tag) {
          return '<' + tag + '>' + this.name + '</' + tag + '>';
        }).bind(this);
        return '<div class="name">' +
                  formattedName('span') +
               '</div>';
      }
    }
assertEquals(
  '<div class="name"><span>Superman</span></div>',
  hero.getIntroductionHTML());
```

Lambda kifejezés kontextusa (ES6)

- Mindig az az scope, amiben definiáltuk

```
var hero = {
  name: 'Superman',
  getIntroductionHTML: function() {
    var name = (tag) => '<' + tag + '>' + this.name + '</' + tag + '>';
    return '<div class="name">' +
              name('span') +
           '</div>';
  }
}
assertEquals(
    '<div class="name"><span>Superman</span></div>',
    hero.getIntroductionHTML());
```

Konstruktorhívási minta

- Mivel nincsenek osztályok, ezért az operandusául kapott függvény lefuttatásával új objektumot hoz létre
- this egy olyan objektumra mutat, amelyet a függvény visszaad a new operátornak

```
var Facebook = function () {
    this.tagok = ['Tamás', 'Thomas', 'Tuomas', 'Etelka'];
    this.regisztraltak = function () {
        return this.tagok;
    }
}

var a = new Facebook();
a.regisztraltak();      // => ["Tamás", "Thomas", "Tuomas", "Etelka"]
```

Összefoglalás

- Függvények létrehozása
- Változók a függvényekben
- Függvények paraméterezése
- Függvénykifejezés használata
- Önkioldó függvény
- Függvény mint paraméter
- Függvény mint visszatérési érték
- Closure
- A függvény mint objektum
- A this jelentése