<https://www.tutorialspoint.com/es6/index.htm>

**ES-6**

Az ECMAScript (ES) egy szkriptnyelv-specifikáció, amelyet az ECMAScript International szabványosított. Az olyan nyelveket, mint a JavaScript, a Jscript és az ActionScript, ez a specifikáció szabályozza.

The Strict Mode

// Whole-script strict mode syntax

"use strict";

v = "Hi! I'm a strict mode script!"; // ERROR: Variable v is not declared

function f1() {

"use strict";

var v = "Hi! I'm a strict mode script!";

}

## JavaScriptVariable Scope

**Global Scope: bárhonnan elérhetőek.**

**Local Scope** : csak abban a függvényben, ahol deklaráltuk

var num = 10

function test() {

var num = 100

console.log("value of num in test() "+num)

}

console.log("value of num outside test() "+num)

test()

## The Let and Block Scope

A var kulcsszóval függvény scope-ot kapunk, a let kulcsszóval pedig blokk scope-ot. (A legközelebbi befoglaló blokk).

"use strict"

function test() {

var num = 100

console.log("value of num in test() "+num)

{

console.log("Inner Block begins")

let num = 200

console.log("value of num : "+num)

}

}

test()

## The const

A const kulcsszóval read-only változót kapunk

const x = 10

x = 12 // will result in an error!!

# ES6 - Functions

Függvény definíció:

function function\_name() {

// function body

}

Függvény hívás:

function\_name()

Példa:

//define a function

function test() {

console.log("function called")

}

//call the function

test()

Classification of Functions

### Returning functions

function function\_name() {

//statements

return value;

}

function retStr() {

return "hello world!!!"

}

var val = retStr()

console.log(val)

### Parameterized functions

function func\_name( param1,param2 ,…..paramN) {

......

......

}

function add( n1,n2) {

var sum = n1 + n2

console.log("The sum of the values entered "+sum)

}

add(12,13)

### Default function parameters

function add(a, b = 1) {

return a+b;

}

console.log(add(4))

function add(a, b = 1) {

return a + b;

}

console.log(add(4,2))

Rest Parameters

function fun1(...params) {

console.log(params.length);

}

fun1();

fun1(5);

fun1(5, 6, 7);

Anonymous Function

var res = function( [arguments] ) { ... }

var f = function(){ return "hello"}

console.log(f())

var func = function(x,y){ return x\*y };

function product() {

var result;

result = func(10,20);

console.log("The product : "+result)

}

product()

The Function Constructor

Egy másik módja a dinamikus függvény létrehozásnak. Az utolsó paramétere a függvény törzse.=> Függvény is objektum JS-ben.

var variablename = new Function(Arg1, Arg2..., "Function Body");

var func = new Function("x", "y", "return x\*y;");

function product() {

var result;

result = func(10,20);

console.log("The product : "+result)

}

product()

Recursion and JavaScript Functions

**Recursion**

function factorial(num) {

if(num<=0) {

return 1;

} else {

return (num \* factorial(num-1) )

}

}

console.log(factorial(6))

Lambda Functions

A lambda függvények egy másik módja az anoním függvények lértehozásának. Ezeket „arrow function”-nak is nevezzük.

3 része van a lambda függvényeknek:

* paraméterek: a függvénynek lehetnek paraméterei (nem kötelező)
* „fat arrow notation/lambda notation” (=>)
* utasítások: a függvény utasításai (függvény törzse)

([param1, parma2,…param n] )=>statement;

var foo = (x)=>10+x

console.log(foo(10))

### Lambda Statement

( [param1, parma2,…param n] )=> {

//code block

}

var msg = ()=> {

console.log("function invoked")

}

msg()

### Syntactic Variations

var msg = x=> {

console.log(x)

}

msg(10)

var disp = ()=>console.log("Hello World")

disp();

Immediately Invoked Function Expression

*. AZ IIFE függvények ahol deklaráltuk őket, ott meg is hívódnak.*

Immediately Invoked Function Expression (IIFE)

(function() {

var msg = "Hello World"

console.log(msg)

})()

var main = function() {

var loop = function() {

for(var x = 0;x<5;x++) {

console.log(x);

}

}();

console.log("x can not be accessed outside the block scope x value is :"+x);

}

main();

var main = function() {

(function() {

for(var x = 0;x<5;x++) {

console.log(x);

}

})();

console.log("x can not be accessed outside the block scope x value is :"+x);

}

main();

Generator Functions

*A yield után átadja az irányítást a hívónak, utána visszaveszi.*

"use strict"

function\* rainbow() {

// the asterisk marks this as a generator

yield 'red';

yield 'orange';

yield 'yellow';

yield 'green';

yield 'blue';

yield 'indigo';

yield 'violet';

}

for(let color of rainbow()) {

console.log(color);

}

function\* ask() {

const name = yield "What is your name?";

const sport = yield "What is your favorite sport?";

return `${name}'s favorite sport is ${sport}`;

}

const it = ask();

console.log(it.next());

console.log(it.next('Ethan'));

console.log(it.next('Cricket'));

# ES6 - Objects

Object Initializers

var identifier = {

Key1:value, Key2: function () {

//functions

},

Key3: [“content1”,” content2”]

}

var person = {

firstname:"Tom",

lastname:"Hanks",

func:function(){return "Hello!!"},

};

//access the object values

console.log(person.firstname)

console.log(person.lastname)

console.log(person.func())

The Object() Constructor

objektum definíció:

var obj\_name = new Object();

obj\_name.property = value;

OR

obj\_name["key"] = value

property elérése:

Object\_name.property\_key

OR

Object\_name["property\_key"]

var myCar = new Object();

myCar.make = "Ford"; //define an object

myCar.model = "Mustang";

myCar.year = 1987;

console.log(myCar["make"]) //access the object property

console.log(myCar["model"])

console.log(myCar["year"])

var myCar = new Object();

myCar.make = "Ford";

console.log(myCar["model"])

var myCar = new Object()

var propertyName = "make";

myCar[propertyName] = "Ford";

console.log(myCar.make)

Constructor Function

function function\_name() {

this.property\_name = value

}

var Object\_name= new function\_name()

//Access the property value

Object\_name.property\_name

function Car() {

this.make = "Ford"

this.model = "F123"

}

var obj = new Car()

console.log(obj.make)

console.log(obj.model)

function Car() {

this.make = "Ford"

}

var obj = new Car()

obj.model = "F123"

console.log(obj.make)

console.log(obj.model)

The Object.create Method

var roles = {

type: "Admin", // Default value of properties

displayType : function() {

// Method which will display type of role

console.log(this.type);

}

}

// Create new role type called super\_role

var super\_role = Object.create(roles);

super\_role.displayType(); // Output:Admin

// Create new role type called Guest

var guest\_role = Object.create(roles);

guest\_role.type = "Guest";

guest\_role.displayType(); // Output:Guest

The Object.assign() Function

Object.assign(target, ...sources)

"use strict"

var det = { name:"Tom", ID:"E1001" };

var copy = Object.assign({}, det);

console.log(copy);

for (let val in copy) {

console.log(copy[val])

}

**Merging Objects**

var o1 = { a: 10 };

var o2 = { b: 20 };

var o3 = { c: 30 };

var obj = Object.assign(o1, o2, o3);

console.log(obj);

console.log(o1);

Eredmény:

{ a: 10, b: 20, c: 30 }

{ a: 10, b: 20, c: 30 }

var o1 = { a: 10 };

var obj = Object.assign(o1);

obj.a++

console.log("Value of 'a' in the Merged object after increment ")

console.log(obj.a);

console.log("value of 'a' in the Original Object after increment ")

console.log(o1.a);

Eredmény:

Value of 'a' in the Merged object after increment

11

value of 'a' in the Original Object after increment

11

Deleting Properties

// Creates a new object, myobj, with two properties, a and b.

var myobj = new Object;

myobj.a = 5;

myobj.b = 12;

// Removes the ‘a’ property

delete myobj.a;

console.log ("a" in myobj) // yields "false"

Comparing Objects

**Different Object References**

var val1 = {name: "Tom"};

var val2 = {name: "Tom"};

console.log(val1 == val2) // return false

console.log(val1 === val2) // return false

**Single Object Reference**

var val1 = {name: "Tom"};

var val2 = val1

console.log(val1 == val2) // return true

console.log(val1 === val2) // return true

Object De-structuring

**destructuring:** Ilyenkor az entitás stuktúráját szétbontjuk (kis egységekre). Itt az fogja jelenteni, hogy az objektumot változókra bontjuk szét (az objektum property-jei egy-egy változó lesz)

var emp = { name: 'John', Id: 3 }

var {name, Id} = emp

console.log(name)

console.log(Id)

# ES6 - Classes

### Declaring a Class

class Class\_name {

}

### Class Expressions

var var\_name = new Class\_name {

}

### Declaring a class

class Polygon {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

### Class Expression

var Polygon = class {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

Creating Objects

var object\_name= new class\_name([ arguments ])

var obj = new Polygon(10,12)

Accessing Functions

//accessing a function

obj.function\_name()

### Putting them together

'use strict'

class Polygon {

constructor(height, width) {

this.h = height;

this.w = width;

}

test() {

console.log("The height of the polygon: ", this.h)

console.log("The width of the polygon: ",this. w)

}

}

//creating an instance

var polyObj = new Polygon(10,20);

polyObj.test();

The Static Keyword

'use strict'

class StaticMem {

static disp() {

console.log("Static Function called")

}

}

StaticMem.disp() //invoke the static method

The instanceof operator

'use strict'

class Person{ }

var obj = new Person()

var isPerson = obj instanceof Person;

console.log(" obj is an instance of Person " + isPerson);

Class Inheritance

class child\_class\_name extends parent\_class\_name

'use strict'

class Shape {

constructor(a) {

this.Area = a

}

}

class Circle extends Shape {

disp() {

console.log("Area of the circle: "+this.Area)

}

}

var obj = new Circle(223);

obj.disp()

Öröklődés:

* egyszeres: egy osztálynak egy szülője van
* többszörös: egy osztálynak lehet több szülője is (ES6 ezt nem támogatja)
* többszintű: egy osztály egy másik leszármazottja és egy harmadik szülője is lehet

'use strict'

class Root {

test() {

console.log("call from parent class")

}

}

class Child extends Root {}

class Leaf extends Child {}

//indirectly inherits from Root by virtue of inheritance {}

var obj = new Leaf();

obj.test()

Class Inheritance and Method Overriding

'use strict' ;

class PrinterClass {

doPrint() {

console.log("doPrint() from Parent called… ");

}

}

class StringPrinter extends PrinterClass {

doPrint() {

console.log("doPrint() is printing a string…");

}

}

var obj = new StringPrinter();

obj.doPrint();

Eredmény:

doPrint() is printing a string…

The Super Keyword

'use strict'

class PrinterClass {

doPrint() {

console.log("doPrint() from Parent called…")

}

}

class StringPrinter extends PrinterClass {

doPrint() {

super.doPrint()

console.log("doPrint() is printing a string…")

}

}

var obj = new StringPrinter()

obj.doPrint()

Eredmény:

doPrint() from Parent called.

doPrint() is printing a string.

# ES6 - Error Handling

Throwing Exceptions

### Syntax: Throwing a generic exception

throw new Error([message])

OR

throw([message])

### Syntax: Throwing a specific exception

throw new Error\_name([message])

Exception Handling

try {

// Code to run

[break;]

} catch ( e ) {

// Code to run if an exception occurs

[break;]

}[ finally {

// Code that is always executed regardless of

// an exception occurring

}]

var a = 100;

var b = 0;

try {

if (b == 0 ) {

throw(’Divide by zero error.’);

} else {

var c = a / b;

}

}

catch( e ) {

console.log("Error: " + e );

}

Custom Errors

### Example 1: Custom Error with default message

function MyError(message) {

this.name = 'CustomError';

this.message = message || 'Error raised with default message';

}

try {

throw new MyError();

} catch (e) {

console.log(e.name);

console.log(e.message); // 'Default Message'

}

### Example 2: Custom Error with user-defined error message

function MyError(message) {

this.name = 'CustomError';

this.message = message || 'Default Error Message';

} try {

throw new MyError('Printing Custom Error message');

}

catch (e) {

console.log(e.name);

console.log(e.message);

}

# ES6 - Modules

JavaScript fájlokat szeparálhatjuk. Egyik JS osztályt, függvényt használhatunk egy másik JS fájlban. Nem kell <script src=”..”> használni.

Exporting a Module

### Export a single value or element - Use export default

export default element\_name

### Export multiple values or elements

export {element\_name1,element\_name2,....}

Importing a Module

### Import a single value or element

import element name from module\_name

### Import multiple values or elements

import {element\_name1,element\_name2,....} from module\_name

Példa:

export default printMsg

import printMsg from './Message.js'

### Example: Defining and Using ES6 modules

**Defining a module: Message\_module.js**

function display\_message() {

console.log("Hello World")

}

export default display\_message

**Importing the module: consume\_module.js**

import display\_message from './MessageModule.js'

display\_message()

Promise, then, async, await

Aszinkron programozás. Olyan esetek lekezelése, amikor várni kell az eredményre.