**Форум.**

1. **Примитивные объекты**

В ECMAScript 6 появится объект WeakMap.

Объект, ключами которого могут быть объекты. Вроде как не боится циклических ссылок внутри себя. Имеет методы get, set, has, delete.

ES6: var wm1 = new WeakMap(),

wm2 = new WeakMap();

var o1 = {},

o2 = function(){},

o3 = window;

wm1.set(o1, 37);

wm1.set(o2, "azerty");

wm2.set(o1, o2); // любое значение, включая функции и объекты

wm2.set(o3, undefined);

wm2.set(wm1, wm2); // ключами и значениями могут быть любые объекты, даже WeakMaps

wm1.get(o2); // "azerty"

wm2.get(o2); // undefined, т.к. нет такого ключа

wm2.get(o3); // undefined, т.к. там хранится именно это значение

wm1.has(o2); // true

wm2.has(o2); // false

wm2.has(o3); // true (несмотря на значение "undefined")

wm1.has(o1); // true

wm1.delete(o1);

wm1.has(o1); // false

1. **Функции**

В функциях добавилась возможность объявлять у параметров значения по умолчанию:

ES6: function setLevel(newLevel = 0) {

...

}

setLevel(); // newLevel = 0

setLevel(5); // newLevel = 5

setLevel(undefined); // newLevel = 0

ES5: function setLevel(newLevel) {

newLevel = newLevel || 0;

...

}

setLevel(); // newLevel = 0

setLevel(5); // newLevel = 5

setLevel(undefined); // newLevel = 0

В функциях также появилась возможность указывать именованные параметры:

ES6: function foo({ from, to = 10 }) {

...

}

foo({ from: 1, to: 5 });

foo({ to: 5, from: 1 });

foo({ from: 1 });

ES5: function foo(args){

console.log(args.from);

...

}

foo({from: 1, to: 5});

foo({ to: 5, from: 1 });

foo({ from: 1 });

//передача аргумента с помощью объекта

В ECMAScript 6 появятся arrow functions:

ES6: let squares = [ 1, 2, 3 ].map(x => x \* x);

ES5: let squares = [ 1, 2, 3 ].map(function (x) { return x \* x });

//вызов анонимной функции

Arrow-функции немножко отличаются от обычных функций. В первую очередь тем, что в arrow-функциях this привязан к вышестоящему контексту. Т.е.

ES6: let jane = {

name: "Jane",

sayHello: function (friends) {

friends.forEach(friend => { console.log(this.name + " says hello to " + friend) });

}

}

jane.sayHello([ 'Mark', 'John' ]);

Выведет:

Jane says hello to Mark

Jane says hello to John

ES5: let jane = {

name: "Jane",

sayHello: function (friends) {

friends.forEach(function(friend) { console.log(this.name + " says hello to " + friend) });

}

}

Выведет:

says hello to Mark

says hello to John

1. **Массивы**

В ECMAScript 5 появится функция массива forEach():

ES5: var ar = [1,2,3];

ar.forEach(

function(element, index, array){

console.log(element, index, array);

});

Вывод:

1 0 [1, 2, 3]

2 1 [1, 2, 3]

3 2 [1, 2, 3]

Подсчитываем сумму всех элементов:

ES5: var ar = [1,2,3];

var sum = 0;

ar.forEach(

function(value){

sum += value;

});

console.log(sum);

Вывод: 6

В ECMAScript 5 появится функция массива map():

Массив, все элементы которого на 1 больше, чем у исходного

ES5: var arA = [1,2,3];

var arB = arA.map(function(value){ return value + 1;

});

console.log(arA);

console.log(arB);

Вывод:

[1,2,3] - исходный массив не изменился

[2,3,4] - новый массив, после вызова map()

В ECMAScript 5 появится функция массива filter():

Оставить элементы меньшие 5

ES5: var arA = [1,2,3,4,5,6,7,8,9];

var arB = arA.filter(function(value){ return value < 5 ? true : false;

});

console.log(arA);

console.log(arB);

Вывод:

[1,2,3,4,5,6,7,8,9] - исходный массив не изменился

[1,2,3,4] - новый массив, после вызова filter()

1. **Циклы**

let arr = [ "blue", "green" ];

arr.notAnIndex = 123;

Array.prototype.protoProp = 456;

ES5: for(var x in arr) {

console.log(x); // Напечатает blue, green, notAnIndex, protoProp

}

ES6: for(var x of arr) {

console.log(x); // Напечатает только blue, green

}

1. **ООП**

В ECMAScript 6 появятся классы:

ES6: // Supertype

class Person {

constructor(name) {

this.name = name;

}

describe() {

return "Person called " + this.name;

}

}

// Subtype

class Employee extends Person {

constructor(name, title) {

super.constructor(name);

this.title = title;

}

describe() {

return super.describe() + " (" + this.title + ")";

}

}

Что позволит использовать эти классы:

let jane = new Employee("Jane", "CTO");

jane instanceof Person; // true

jane instanceof Employee; // true

jane.describe(); // 'Person called Jane (CTO)'

Всего того же можно было добиться с помощью прототипов:

ES5: // Supertype

function Person(name) {

this.name = name;

}

Person.prototype.describe = function () {

return "Person called " + this.name;

};

// Subtype

function Employee(name, title) {

Person.call(this, name);

this.title = title;

}

Employee.prototype = Object.create(Person.prototype);

Employee.prototype.constructor = Employee;

Employee.prototype.describe = function () {

return Person.prototype.describe.call(this) + " (" + this.title + ")";

};