1. **Примитивные типы.**

**Блочная область видимости (block scope)**

ES5:

function f(a) {

if (a < 0) {

var i = 3;

}

console.log(i); // 3

}

f(-1)

Комментарии: В текущей версии JavaScript присутствует функциональная область видимости. Это означает, что все переменные, объявленные c помощью ключевого слова var, будут видны в любом месте функции (даже если они объявлены внутри блока).

ES6:

function f(a) {

if (a < 0) {

let i = 3;

}

console.log(i); // ReferenceError: i is not defined

}

f(-1)  
Комментарии: В новой версии появится ключевое слово let, которое позволит объявлять переменные с блочной областью видимости.

**Const**

Const max\_size = 100;

// нельзя перезапивать (var, let и т.д.)

let max\_size = 100; // error

const a = 15;

console.log(a); // 15

a = 1;

//объявляет переменную и присваивает ей значение, которое потом не может быть изменено.

1. **Примитивных объектов**

**Новые методы базовых объектов.**

ES6:

String

String.prototype.contains

‘too much’.contains(‘too’); //true

String.prototype.startsWith

‘too much’. startsWith (‘t’); //true

String.prototype.endsWith

‘too much’. endsWith (‘t’); //false

Object

Object.assign({a: 1, b: 3}, {b: 2, c: 3}); // a: 1, b: 2, c: 3

Array

Array.prototype.find

[1, 3, 15, 7, 15, 23].find(function(item))

{return item/3===5;}); //15

Array.prototype.findIndex

[1, 3, 15, 7, 15, 23].findIndex(function(item))

{return item/3===5;}); //2

**Обмен двух переменных без третьей.**

ES5

Let x = 10;

Let y = 20;

Let z = let x;

Let x = let y;

Let y = let x;

ES6:

Let x = 10;

Let y = 20;

[x, y] = [y, x]

**деструктуризирование при присваивании**ES6:

let { first: f, last: l } = { first: 'Jane', last: 'Doe' };

console.log(f); // 'Jane'

console.log(l); // 'Doe'

**//** при разворачивании объекта нужно указать только ключи, которые нужны.

1. **Функций**

**Краткий синтаксис функций , функции-блоки**

ES5:

Обычная функция

[1,2,3].map(function (x) {return x\*x}); //1,4,9

ES6:

Функция-блок

[1,2,3].map{|x| x\*x}; //1,4,9

Комментарии:

- опциональный return;

- |x| вместо function;

- необязательные скобки вызова.

**Значения по умолчанию.**

ES5:

function setLevel(newLevel) {

newLevel = newLevel || 0; ... }

setLevel(); // newLevel = 0

setLevel(5); // newLevel = 5

setLevel(undefined); // newLevel = 0

ES6:

function setLevel(newLevel = 0) {...}

setLevel(); // newLevel = 0

setLevel(5); // newLevel = 5

setLevel(undefined); // newLevel = 0

Комментарии:

В функциях добавилась возможность объявлять у параметров значения по умолчанию.

**Именованные параметры функции.**

ES5:

Попытка переписать объект аргументов приведёт к ошибке:

arguments = [...]; // not allowed  
Определение нескольких аргументов с одинаковым названием приведет к ошибке

function(Foo, Foo) {} // error  
ES6:

В функциях также появилась возможность указывать именованные параметры:

function foo({ from, to = 10 }) {...}

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

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

foo({ from: 1 });

Именованные параметры можно комбинировать с обычным (позиционными параметрами):

function foo(positional, { named1, named2 }) {...}

foo(123, { named1: 'abc', named2: 'def' })

foo(123, { named2: 'def', named1: 'abc' })

**Arrow-функции**  
ES6:

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

ES5:

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

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  
  
Другие отличия arrow-функций от обычных функций:

* Нельзя использовать arrow-функции как конструкторы (new (() => {}) кинет ошибку)
* Arrow-функции не могут обратиться к переменной arguments (да и незачем)

1. **Массивов**

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. **Циклов**

**Цикл for-of**

ES5:

let tags = [‘#404fest’, ‘wstdays’, ‘dconf’];

for (let tag in tags){console.log(tag);} //0, 1, 2

ES6:

for (let tag of tags){console.log(tag);} // ‘#404fest’, ‘wstdays’, ‘dconf’

Комментарии:

Цикл for-in в JavaScript итерирует по всем полям объекта (включая наследованных). Т.е. итерироваться по значениям массива можно, но опасно.

В ECMAScript 6 появится цикл for-of, который решит данную проблему.

1. **ООП**

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 + ")";

};

ES6:

В ECMAScript 6 появятся классы:

// 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 + ")";

}

}

Теперь можно использовать эти классы:

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

jane instanceof Person; // true

jane instanceof Employee; // true

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

Классы могут иметь статические методы:

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

static zero() {

return new Point(0, 0);

}

}

Приватных полей и методов не будет (по крайней мере, в ECMAScript 6). Однако некоторое сокрытие данных все же появится. Через модули.