JAVASCRIPT

ECMAScript 6

ECMAScript 6

поддержка констант

immutable variables - неизменяемые переменные

переменные, которые не могут быть повторно назначены новым значением

```
> const test = {my: 1}
<- undefined
> test.by = 2
<- 2
> test
<- ▶ Object {my: 1, by: 2}
> test = 2
✖ ▶ Uncaught TypeError: Assignment to constant variable.
      at <anonymous>:1:6
```

Замечание: если константа имеет значение объекта, то сам объект может быть изменен

ECMAScript 6

Переменные (и константы) с фиксированной областью видимости без подъема

```
let callbacks = []
for (let i = 0; i <= 2; i++) {
    const MY_LOCAL_CONST = 'my local value'
    callbacks[i] = function () { return i * 2 }
}
```

```
console.log(callbacks[0]() === 0)
console.log(callbacks[1]() === 2)
console.log(callbacks[2]() === 4)
```

```
{
    function foo () { return 1 }
    foo() === 1
    {
        function foo () { return 2 }
        foo() === 2
    }
    foo() === 1
}
```

ECMAScript 6

Стрелочные методы/функции

```
odds  = evens.map(function (v) {
    return v + 1;
});


pairs = evens.map(function (v) {
    return { even: v, odd: v + 1 };
});


nums  = evens.map(function (v, i) {
    return v + i;
});
```

```
odds  = evens.map(v => v + 1)


pairs = evens.map(v => ({ even: v, odd: v + 1 }))


nums  = evens.map((v, i) => v + i)
```

```
nums.forEach(function (v) {
   if (v % 5 === 0)
       fives.push(v);
});
```

```
nums.forEach(v => {
   if (v % 5 === 0)
       fives.push(v)
})
```

ECMAScript 6

Значения параметров по умолчанию

Простые и интуитивные значения по умолчанию для параметров функции

```
function f (x, y = 7, z = 42) {
    return x + y + z
}
f(1) === 50
```

Агрегация оставшихся аргументов в один параметр

```
function f (x, y, ...a) {
    return (x + y) * a.length
}
console.log(f(1, 2, "hello", true, 7) === 9)
```

Оператор распространения

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ]

function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, ...params) === 9
```

```
var str = "foo"
var chars = [ ...str ]
```

ECMAScript 6

Литералы шаблонов

Интерполяция строк

Интуитивная интерполяция выражения для однострочных и многострочных строк.

```
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 }

var message = `
Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?
`
```

ECMAScript 6

Расширенные свойства объекта

Более короткий синтаксис для определения общего свойства объекта

```
//ES5
obj = { x: x, y: y };
```

```
//ES6
obj = { x, y }
```

Поддержка вычисляемых имен в определениях свойств объекта.

```
//ES5
var obj = {
    foo: "bar"
};
obj[ "baz" + quux() ] = 42;
```

```
//ES6
let obj = {
    foo: "bar",
    [ "baz" + quux() ]: 42
}
```

```
//ES5
obj = {
    foo: function (a, b) {
        …
    },
    bar: function (x, y) {
        …
    }
};
```

```
//ES6
obj = {
    foo (a, b) {
        …
    },
    bar (x, y) {
        …
    }
}
```

ECMAScript 6

гибкое деструктурирование массивов в отдельные переменные

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
[ b, a ] = [ a, b ]
```

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
undefined
[ a, , b ]
▶ (3) [1, undefined × 1, 3]
[ a, b ]
▶ (2) [1, 3]
[ b, a ] = [ a, b ]
▶ (2) [1, 3]
[ b, a ]
▶ (2) [1, 3]
[ a, b ]
▶ (2) [3, 1]
```

ECMAScript 6

гибкое деструктурирование массивов и объектов в отдельные параметры во время вызова функций

```
function f ([ name, val ]) {
    console.log(name, val)
}

f([ "bar", 42 ])
```

```
function g ({ name: n, val: v }) {
    console.log(n, v)
}

g({ name: "foo", val:  7 })
```

```
function h ({ name, val }) {
    console.log(name, val)
}

h({ name: "bar", val: 42 })
```

ECMAScript 6

Экспорт / импорт значений

```
//  lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

ECMAScript 6

Определение и наследование класса

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
```

```
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```

ECMAScript 6

Геттеры и Сеттеры

```
class Rectangle {
    constructor (width, height) {
        this._width  = width
        this._height = height
    }
    set width  (width)  { this._width = width                }
    get width  ()       { return this._width                 }
    set height (height) { this._height = height              }
    get height ()       { return this._height                }
    get area   ()       { return this._width * this._height }
}

var r = new Rectangle(50, 20)
r.area === 1000
```

ECMAScript 6

Статические методы

```
class Rectangle extends Shape {
    …
    static defaultRectangle () {
        return new Rectangle("default", 0, 0, 100, 100)
    }
}
class Circle extends Shape {
    …
    static defaultCircle () {
        return new Circle("default", 0, 0, 100)
    }
}
var defRectangle = Rectangle.defaultRectangle()
var defCircle    = Circle.defaultCircle()
```

JAVASCRIPT

ECMAScript 6