JavaScript

Часть 2

Создание объектов

```
const person = new Object();
person.name = 'Юрий Дудь';
person.age = 31;

const msk = new Object();
msk.country = 'RU';
msk.city = 'Moscow’;

person.location = msk;
```

```
const person = {
    name: 'Юрий Дудь',
    age: 31,
    location: {
        country: 'RU',
        city: 'Moscow'
    }
};
```

```
// Имя: Юрий Дудь, возраст: 31, город: Moscow.
const info = `Имя: ${person.name}, возраст: ${person.age}, город: ${person.location.city}.`;
```

Доступ к свойствам объекта

```
const name = person.name;
const age = person.age;
const country = person.location.country;
const city = person.location.city;
```

```
const name = person['name'];
const age = person['age'];
const country = person['location']['country'];
const city = person['location']['city'];
```

```
person['У попа была собака, он её любил.'] = 42;
```

Доступ к несуществующему свойству

```
let user = {
    firstName: 'Johnny',
    lastName: 'Walker',
    country: undefined
};

let age = user.age; // undefined
let country = user.country; // undefined

let hasAge = 'age' in user; // false
let hasCountry = 'country' in user; // true

delete user.country;
let hasCountryNow = 'country' in user; // false
```

Перебор свойств объекта

```
const user = {
    firstName: 'Johnny',
    lastName: 'Walker',
    age: 41
};

for (let key in user) {
    console.log(key);
    console.log(user[key]);
    console.log('--');
}
```

```
// firstName
// Johnny
// --
// lastName
// Walker
// --
// age
// 41
// --
```

Массивы

```
let companies = ["Apple", "Google", "Amazon"];

console.log(companies.length); // 3
console.log(companies[0]); // Apple

companies[3] = "Intel";

console.log(companies.length); // 4

companies[10] = "ABBYY";

console.log(companies[4]); // undefined
console.log(companies.length); // 11

let months = new Array(12);

console.log(months.length); // 12
```

Массив как стек

```
const drinks = ["beer", "wine"];

const last = drinks.pop();

console.log(last); // wine
console.log(drinks); // [ 'beer' ]

drinks.push("whisky");

console.log(drinks); // [ 'beer', 'whisky' ]
```

Массив как очередь

```
const meals = ["steak", "burger"];

const first = meals.shift();

console.log(first); // steak
console.log(meals); // [ 'burger' ]

meals.unshift("salad");

console.log(meals); // [ 'salad', 'burger' ]
```

Перебор элементов массива

```
const drinks = ["beer", "wine", "whisky"];

for (let i = 0; i < drinks.length; i++) {
    let drink = drinks[i];
    console.log(drink);
}

for (let i in drinks) {
    let drink = drinks[i];
    console.log(drink);
}

for (let drink of drinks) {
    console.log(drink);
}
```

Многомерные массивы

```
const matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

let sum = 0;

for (let i = 0; i < matrix.length; i++) {
    for (let j = 0; j < matrix[i].length; j++) {
        sum += matrix[i][j];
    }
}

console.log(sum); // 45
```

Сортировка массивов

```
const numbers = [5, -1, 14, 8, 1];
numbers.sort();

// [ -1, 1, 14, 5, 8 ]
console.log(numbers);


function compareNumeric(a, b) {
    return a - b;
}

numbers.sort(compareNumeric);

// [ -1, 1, 5, 8, 14 ]
console.log(numbers);
```

Методы для работы с массивами

```
const numbers = '1, 2, 3, 4, 5'.split(', ');
// ['1', '2', '3', '4', '5']

const str = ['1', '2', '3', '4', '5'].join(', ');
// '1, 2, 3, 4, 5'

const numbers = [1, 2, 3];
numbers.reverse();
// [3, 2, 1]

const numbers = [1, 2];
const newNumbers = numbers.concat([3, 4], 5); // то же что numbers.concat(3, 4, 5)
// [1, 2, 3, 4, 5]
```

Методы для работы с массивами

```
const arr = ["zero", "one", "two", "three"];

// удалить два элемента начиная с первого
// и добавить другие вместо них
arr.splice(1, 2, "xxx", "yyy", "zzz");

console.log(arr); // ["zero", "xxx", "yyy", "zzz", "three"]
```

Методы для работы с массивами

```
const arr = [1, 0, false, 1];

console.log(arr.indexOf(0)); // 1
console.log(arr.indexOf(false)); // 2
console.log(arr.indexOf(null)); // -1
console.log(arr.lastIndexOf(1));  // 3
```

Методы для работы с массивами

```
let arr = [1, -1, 2, -2, 3];

function isPositive(number) {
    return number > 0;
}

console.log(arr.filter(isPositive)); // [1, 2, 3]
console.log(arr.every(isPositive)); // false, не все положительные
console.log(arr.some(isPositive)); // true, есть хоть одно положительное
```

Методы для работы с массивами

```
let names = ['HTML', 'CSS', 'JavaScript'];
let nameLengths = names.map(name => name.length);

console.log(nameLengths); // 4, 3, 10
```

```
let arr = [1, 2, 3, 4, 5]

// для каждого элемента массива запустить функцию,
// промежуточный результат передавать первым аргументом далее
let result = arr.reduce(function (sum, current, index, array) {
    return sum + current;
}, 0);

console.log(result); // 15
```

Перебор элементов массива (ещё один)

```
arr.forEach((item, i, arr) =>
{
   // item - элемент
   // i - индекс
   // arr - весь массив
});
```

Получение свойств объекта

```
const user = {
   name: 'John',
   age: 41,
   country: 'US'
}

const keys = Object.keys(user);
// ['name', 'age', 'country']
```

Self-Invoking Functions

```
(function() {
    let x = 100;
    console.log(x); // 100
})();

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

Лексическое окружение

```
function foo(x, y) {
    let z = 41;

    // LexicalEnvironment:
    // { x: [значение_x], y: [значение_y], z: 42, bar: [функция_bar] }

    function bar() { /* do nothing */ }
}

// LexicalEnvironment (глобальный объект):
{ foo: [функция_foo] }
```

Все переменные и параметры функций являются свойствами объекта LexicalEnvironment.
Каждый запуск функции создает новый такой объект. На верхнем уровне им является «глобальный объект».

Лексическое окружение и [[Scope]]

```
(function run() {
    let greeting = 'Hello’;

    // LexicalEnvironment: { greeting: 'Hello', sayHello: [function sayHello] }

    function sayHello(name) { // [[Scope]] функции == LexicalEnvironment функции run

        // LexicalEnvironment: { name: 'Martin' }
        console.log(`${greeting}, ${name}!`);
    }
    sayHello('Martin');
})();
```

При создании функция получает скрытое свойство [[Scope]], которое ссылается на лексическое окружение, в котором она была создана.
Интерпретатор, при доступе к переменной, сначала пытается найти переменную в текущем LexicalEnvironment, а затем, если её нет – ищет во объекте переменных, на который ссылается [[Scope]].

Лексическое окружение (пример)

```
let x = 1;

(function first() {
    let y = 2;

    (function second() {
        let x = 111;
        let z = 3;

        console.log(x, y, z); // 111, 2, 3
    })();

    console.log(x, y); // 1, 2
})();

console.log(x); // 1
```

Передача функции как аргумента

```
function sayHello(name) {
    console.log(`Hello, ${name}!`);
}

function greeting(helloFunc, name) {
    helloFunc(name);
    console.log('How are you?');
}

greeting(sayHello, 'John');
```

Возврат функции из другой функции

```
function makeGreeting() {
    return function (name) {
        console.log(`Hello, ${name}!`);
    };
}

let greeting = makeGreeting();
greeting('John');
```

Фунарг-проблема

```
let x = 10;

function foo() {
    // CallStack: foo > bar > Program
    console.log(x);
}

function bar(funArg) {
    let x = 20;
    funArg();
}

bar(foo); // 10 или 20?
```

Фунарг-проблема

```
function foo() {
    let x = 10;

    function bar() {
        return x;
    }

    return bar;
}

let x = 20;
let barFunc = foo();

let value = barFunc();

console.log(value); // 10 или 20?
```

Замыкание

Замыкание — это функция, *захватывающая лексическое окружение* того контекста, где она *создана*.
В дальнейшем это окружение используется для *разрешения идентификаторов*.

Методы объектов и this

```
let name = 'John';

let user = {
    name: 'Steve',
    print: function() {
        console.log(name); // John
        console.log(this.name); //Steve
    }
}

user.print();


console.log(this.name); // undefined
```

Что выведет console.log?

```
let fullname = 'Иван Иванов’;

let obj = {
    fullname: 'Пётр Петров',
    prop: {
        fullname: 'Сидр Сидоров',
        getFullname: function () {
            return this.fullname;
        }
    }
};

let test1 = obj.prop.getFullname();
console.log(test1); // ?

let test2 = obj.prop.getFullname;
console.log(test2()); // ?
```

Конструкторы

```
function User(name, age) {
    this.name = name;
    this.age = age;
}

let user = new User('Steve', 33); // { name: 'Steve', age: 33 }

// Steve 33
console.log(user.name, user.age);
```

Что выведет console.log?

```
function Func(val){
    let value = val;
    this.print = function () {
        console.log(value);
    };
}

let obj1 = new Func(1);
let obj2 = new Func(2);

obj1.print(); // ?
obj2.print(); // ?

obj1.print = obj2.print;

obj1.print(); // ?
obj2.print(); // ?
```

Что выведет console.log?

```
function Func(val){
    this.value = val;
    this.print = function () {
        console.log(this.value);
    };
}

let obj1 = new Func(1);
let obj2 = new Func(2);

obj1.print(); // ?
obj2.print(); // ?

obj1.print = obj2.print;

obj1.print(); // ?
obj2.print(); // ?
```

Прототипы

```
let rectDef = {
    getArea: function() {
        return this.width * this.height;
    }
};

let rectImpl1 = {
    width: 10,
    height: 20,
    __proto__: rectDef
};

let rectImpl2 = {
    width: 5,
    height: 2,
    __proto__: rectDef
};
```

```
// 200 10
console.log(
    rectImpl1.getArea(),
    rectImpl2.getArea()
);
```

Прототипы

```
let rectDef = {
    getArea: function() {
        return this.width * this.height;
    }
};

let rectImpl1 = {
    width: 10,
    height: 20
};
Object.setPrototypeOf(rectImpl1, rectDef);

let rectImpl2 = {
    width: 5,
    height: 2
};
Object.setPrototypeOf(rectImpl2, rectDef);
```

Конструкторы

```
function Rectangle(width, height) {
    this.width = width;
    this.height = height;
}

Rectangle.prototype.getArea = function() {
    return this.width * this.height;
};

let rectImpl1 = new Rectangle(10, 20);
let rectImpl2 = new Rectangle(5, 10);
```

Наследование через прототипы

```
function Shape(width, height) {
    this.width = width;
    this.height = height;
}

Shape.prototype.sayArea = function () {
    console.log(this.width * this.height);
};

function ColoredShape(width, height, color) {
    Shape.call(this, width, height);
    this.color = color;
}

ColoredShape.prototype = Object.create(Shape.prototype);

ColoredShape.prototype.sayColor = function() {
    console.log(this.color);
};
```

```
let coloredShape =
        new ColoredShape(2, 5, 'red');
coloredShape.sayArea(); // 10
coloredShape.sayColor(); // red
```

Наследование через классы

```
class Shape {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }

    sayArea() {
        console.log(this.width * this.height);
    }
}

class ColoredShape extends Shape {
    constructor(width, height, color) {
        super(width, height);
        this.color = color;
    }

    sayColor() {
        console.log(this.color);
    };
}
```

```
let coloredShape =
        new ColoredShape(2, 5, 'red');
coloredShape.sayArea(); // 10
coloredShape.sayColor(); // red
```

Деструктуризация массивов

```
let str = "каждый охотник желает знать где сидит фазан";
let arr = str.split(' ');
let [red, , yellow, green, ...rest] = arr;

console.log(red); // каждый
console.log(yellow); // желает
console.log(green); // знать
console.log(rest); // [ 'где', 'сидит', 'фазан' ]
```

```
let name = ['John'];
let [firstName, lastName = 'Smith'] = name;
console.log(firstName); // John
console.log(lastName); // Smith
```

Деструктуризация объектов

```
let user = {
    name: 'John',
    age: 33
};

let { name, age } = user;

// John 33
console.log(name, age);
```

```
let user = {
    name: 'John',
    location: {
        country: 'UK'
    }
};

let {
    name: firstName,
    age = 33,
    location: { country }
} = user;

// John 33 UK
console.log(firstName, age, country);
```