ОБЪЕКТНЫЙ ТИП

ФУНКЦИИ, МАССИВЫ, ОБЪЕКТЫ

ФУНКЦИИ

```
// инструкция объявления функции
// function declaration statement

function идентификатор(параметр, параметр) {
  инструкция;
  return выражение;
}

// выражение вызова функции

выражение(аргумент, аргумент);
```

```
// выражение определения функции
// function definition expression

let f = function(lang) {
  let upperLang = lang.toUpperCase();

  return `I ♥ ${upperLang}!`;
};

f("js");                  // I ♥ JS!
```

```
// стрелочные функции
// [fat] arrow functions

let singleParam = num => 10 + num;

let twoParams = (num1, num2) => num1 + num2;

let noParams = () => "no params";

let withFBody = (num1, num2) => {
  let result = num1 + num2;

  return `Result is: ${result}`;
}
```

Ограничения стрелочных функций

- нет доступа к this, super, arguments, new.target
- нельзя вызывать с new
- нет свойства prototype
- нельзя изменить значение this
- нет объекта arguments
- нет повторяющихся именованных параметров в строгом и нестрогом режимах

ПАРАМЕТРЫ И АРГУМЕНТЫ ФУНКЦИЙ

```
function f(a, b) {
  let result = `Result is: ${a} ${b}`;
  return result;
}

f(1, 2);                        // "Result is: 1 2"

f(1);                           // "Result is: 1 undefined"
f(1, 2, 3);                     // "Result is: 1 2"
```

```
function f(a, b) {
  let first = a || "a not passed",
      second = b !== undefined ? b : "b not passed",
      result = `Result is: ${first} ${second}`;

  return result;
}

f(1);                          // "Result is: 1 b not passed"
```

```
function f(a = "a default", b = "b default", c = b) {
  return `Result is: ${a} ${b}`;
}

f();                         // "Result is: a default b default b defaul
```

```
// arguments

function f() {
  let result = "";

  for(let i = 0, l = arguments.length; i < l; i++) {
    result += `${arguments[i]} `;
  }

  return result;
}

f("a", 1, true, "end");              // "a 1 true end"

// в стрелочных функциях нет arguments
// arguments не массив и не имеет встроенных методов
```

```
// rest parameters

function f(...params) {
  let result = params.reduce((p, c) => `${p} ${c}`);

  return result;
}

f("a", 1, true, "end");              // "a 1 true end"
```

```
let f = (...params) => params.reduce((p, c) => `${p} ${c}`);
```

Ограничения rest параметров

```
let f1 = (...rest) => rest;                  // ok
let f2 = (a, ...rest) => rest;               // ok

let f3 = (a, ...rest, b) => rest;            // ERROR
// Uncaught SyntaxError: Rest parameter must be last formal parameter
```

```
let func = (a, b, c) => `${a} ${b} ${c}`;

func(1, 2, 3)                                  // "1 2 3"
```

```
let arr = [5, 6, 7];

func(arr[0], arr[1], arr[2]);
func.apply(this, arr);
```

```
// spread operator ...

func(...arr);
```

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

```
// ключевое слово return

function f(name) {
  return `My name is ${name}`;
}

f("Boo");

let booName = f("Boo").toUpperCase();    // "MY NAME IS BOO"
booName.toLowerCase();                   // "my name is boo"
```

```
let f = name => `My name is ${name}`;
```

```
function f1(a, b) {
  let result = a + b;
}

let f2 = (a, b) => {
  let result = a + b;
}

f1(1, 2);                                   // undefined
f2(1, 2);                                   // undefined

let res = f2(1, 2);
typeof res;                                 // "undefined"
```

```
let func = arg => {
  let res = `Arg is ${arg}`;

  return res;

  res = res.split(" ");
}

function func(importantArgument) {
  if(importantArgument === undefined) return;

  // ...

  return importantArgument;
}
```

СВОЙСТВА ФУНКЦИЙ

```
// length
let f1 = (a, b) => b,
    f2 = () => " ",
    f3 = (...b) => b,
    f4 = (a, ...b) => b;

f1.length                       // 2
f2.length                       // 0
f3.length                       // 0
f4.length                       // 1
```

```
// name
let f1 = a => a;
function f2(a) { return a; }
let f3 = function(a) { return a; };
let f4 = function f5(a) { return a; }

console.log(f1.name);                    // f1
console.log(f2.name);                    // f2
console.log(f3.name);                    // f3
console.log(f4.name);                    // f5
```

ФУНКЦИИ АРГУМЕНТЫ И ВОЗВРАЩАЕМЫЕ ФУНКЦИИ

```
// функции-аргументы
// callbacks

function someFunc(a, b, anotherFunc) {
  let result = a + b;

  anotherFunc(result);
}

someFunc(1, 2, function(num) {
  return `Result is: ${num}`;
});
```

```
let fn = num => `Result is: ${num}`;

someFunc(5, 6, fn);
```

```
let addName = function(name) {
  let myName = `Hi, my name is ${name}`;

  return function(lastName) {
    let fullName = `${myName} ${lastName}`;

    return fullName;
  };
};

let addLastName = addName("Boo");
let getFullName = addLastName("Foo");

console.log(getFullName);                    // "Hi, my name is Boo Fo
addName("Boo")("Foo");
```

Самовызывающаяся анонимная функция

Self-invoked anonymous function

Immediately-invoked function expression (IIFE)

```
let result = function (name) {
  return `Name is ${name}`;
}("Boo");

console.log(result);                    // "Name is Boo"
```

```
let result = (function (name) {
  return `Name is ${name}`;
}("Boo"));

(name => `Name is ${name}`)("Foo");
```

Рекурсия

```
(function func(num) {
  console.log(num);
  if(num < 10) func(++num);
})(0);
```

ЦЕПОЧКА ОБЛАСТЕЙ ВИДИМОСТИ
SCOPE CHAIN

```
var n = 10;                          // глобальная

(function() {
  var n = 20;                        // локальная
})();

for(let i = 0; i < 10; i++) {
  let p = i + 1;                     // блочная
}
```

```
var n = "global";
function local() {
  var n = "local";
  console.log(n);                    // "local"

  function inner() {
    var n = "inner";
    console.log(n);                  // "inner"
  }

  console.log(n);                    // "local"
  inner();
}

local();
```

```
var n = "global";
function local() {
  var n = "local";
  console.log(n);                    // "local"

  function inner() {
    n = "inner";
    console.log(n);                  // "inner"
  }

  console.log(n);                    // "inner"
  inner();
}

local();
```

Подъем (hoisting)

```
function outer() {
  var n = "outer";

  function inner() {
    console.log(n);

    var n = "inner";
  }
}
```

Подъем (hoisting)

```
function outer() {
  var n = "outer";

  function inner() {
    var n;
    console.log(n);

    n = "inner";
  }
}
```

Подъем с let

```
(function() {
  console.log(someLet);
  console.log(someConst);

  let someLet = 10;
  const someConst = 20;
})();
```

declaration statement
vs. function expression

```
// hoisting

console.log(declaration);
console.log(expression);

function declaration() { return; }
var expression = function() { return; };
```

declaration statement
vs. function expression

```
// block level declaration

if (true) {
  function func() {
    return;
  }

  func();
}

func();
```

ЗАМЫКАНИЯ
CLOSURES

```
function fn() {
  let a = "fn";

  return function() {
    console.log(a);
  }
}

function otherFunc() {
  let a = "otherFunc",
      getFn = fn();

  console.log(getFn());
}

otherFunc();
```

```
function fn() {
  var str = "Some string";

return function(add) {
    str += ` ${add}`;
    console.log(str);
  };
}

let addStr = fn();

addStr("call 1st time");
addStr("call 2nd time");
```

МАССИВЫ

```
[element0, element1, ..., elementN]
Array(element0, element1[, ...[, elementN]])
Array(arrayLength)

Array.of(element0[, ...[, elementN]])
Array.from(arrayLike[, mapFn[, thisArg]]);

let arr = ["0", 1, true, {k: 1}, [0, 1]];
let arrConstructor = Array(1, 2, {k: 1});
let arrLength = Array(5);
let arrOf = Array.of(5);

typeof arr;                         // "object"
Array.isArray(arr);                 // true
```

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

// доступ к элементу по индексу
arr[0]                                  // 1

// свойство length
arr.length;                             // 5
arr[arr.length];                        // undefined
arr[arr.length - 1]                     // 5

// добавление/изменение элементов по индексу
arr[0] = 6;
arr[10] = 1;
arr[arr.length] = arr.length;
```

Перебор массива

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

arr.forEach(function(e, i, array) {
  console.log(`index is ${i} and element is ${e}`);
});
arr.every((e, i, array) => e === 5);
arr.some(e => e === 5);
arr.filter(e => e % 2);                  // [1, 3, 5]
arr.find(e => !(e % 6));                 // undefined
arr.findIndex(e => !(e % 3));
arr.map((e, i) => e + i);
arr.reduce((p, c) => p - c);
arr.reduce((p, c) => p - c, 0);
arr.reduceRight((p, c) => p - c);
```

Не мутирующие методы

```
arr.concat(6, [7, 8, 9]);
arr.includes(6);
arr.join("::");
arr.slice(2, -1);
arr.indexOf(6);                    // -1
arr.lastIndexOf(2);                // 1
```

Методы мутирующие массив

```
arr.fill(1, 0, 2);                          // [1, 1, 3, 4, 5]
arr.pop();                                  // [1, 1, 3, 4]
arr.shift();                                // [1, 3, 4]
arr.push(5, 6);                             // [1, 3, 4, 5, 6]
arr.unshift(0);                             // [0, 1, 3, 4, 5, 6]
arr.reverse();                              // [6, 5, 4, 3, 1, 0]
arr.sort();                                 // [0, 1, 3, 4, 5, 6]
arr.sort((a, b) => { /* ... */ });
arr.splice(2, 0, 2);                        // [0, 1, 2, 3, 4, 5, 6]
arr.splice(0, 2, 100);                      // -> [0, 1]
                                            //  [100, 2, 3, 4, 5, 6]
arr.splice(1, 1);                           // -> [2]
                                            // [100, 3, 4, 5, 6]
```

ПОЛЕЗНЫЕ ССЫЛКИ

- CodeFights
- Самый адекватный гид по функциональному программированию от профессора Фрисби
- You Don't Know JS: Scope & Closures
- YouTube канал "Sorax"
- Airbnb JavaScript Style Guide
- Библиотека Lodash