ФУНКЦИИ, ОБЪЕКТЫ, МАССИВЫ

Tinkoff.ru

ФУНКЦИИ, ОБЪЕКТЫ, МАССИВЫ

- Глобальный объект
- Строгий режим
- Контекст вызова функции
- Свойства и методы объектов
- Деструктуризация
- Массивы, итерируемые объекты

```
hero = 'Люк';
```

```
hero = 'Люк';

window.hero; // 'Люк'
```

ГЛОБАЛЬНЫЙ ОБЪЕКТ

learn.javascript.ru

```
window.alert('Люк');
window.console.log('Лея');
```

```
var hero = 'Люк';
function force () {};

window.hero; // 'Люк';
window.force; // force () {}
```

```
let hero = 'Люк';
const force = function () {};

window.hero; // undefined
window.force; // undefined
```

ВСЕГДА ИСПОЛЬЗУЙТЕ LET И CONST!

Для большинства переменных используйте const.
Если переменная будет изменяться - let.
И никогда, ни при каких обстоятельствах - var.

STRICT MODE

MDN

```
// file.js

"use strict"

function force () {
  // Строгий режим
}

function b () {
  // Строгий режим
}
```

```
// file.js

function force () {
  "use strict"

  // Строгий режим
}

function b () {
  // Обычный режим
}
```

```
'use strict';

hero = 'Люк'; // ReferenceError:
```

ГЛОБАЛЬНАЯ ОБЛАСТЬ ВИДИМОСТИ

```
// blaster.js

function blaster () {
  console.log(getSound());
}

function getSound () {
  return 'PEW PEW';
}
```

```
// grenade.js

function grenade () {
  console.log(getSound());
}

function getSound () {
  return 'BOOM';
}
```

```js
// blaster.js

function blaster () {
  console.log(getSound());
}

function getSound () {
  return 'PEW PEW';
}
```

```js
// grenade.js

function grenade () {
  console.log(getSound());
}

function getSound () {
  return 'BOOM';
}
```

```html
<script src="blaster.js"></script>
<script src="grenade.js"></script>
<script>
  blaster(); // ???
  grenade(); // ???
</script>
```

```js
// blaster.js

function blaster () {
  console.log(getSound());
}

function getSound () {
  return 'PEW PEW';
}
```

```js
// grenade.js

function grenade () {
  console.log(getSound());
}

function getSound () {
  return 'BOOM';
}
```

```html
<script src="blaster.js"></script>
<script src="grenade.js"></script>
<script>
  blaster(); // 'BOOM'
  grenade(); // 'BOOM'
</script>
```

```js
// blaster.js

const blaster = () => {
  console.log(getSound());
};

const getSound = () => 'PEW PEW';
```

```js
// grenade.js

const grenade = () => {
  console.log(getSound());
};

const getSound = () => 'BOOM';
```

```html
<script src="blaster.js"></script>
<script src="grenade.js"></script> // SyntaxError
```

```
// blaster.js

const blaster = () => {
  const getSound = () => 'PEW PEW';

  console.log(getSound());
};
```

```
// grenade.js

const grenade = () => {
  const getSound = () => 'BOOM';

  console.log(getSound());
};
```

```
<script src="blaster.js"></script>
<script src="grenade.js"></script>
<script>
  blaster(); // 'PEW PEW'
  grenade(); // 'BOOM'
</script>
```

IMMEDIATELY-INVOKED FUNCTION EXPRESSION

```
(function () {
  // code
})()
```

```
(() => {
  // code
})()
```

Mastering Immediately-invoked Function Expressions

```
((global) => {
  'use strict';

  const blaster = () => {
    console.log(getSound());
  };

  const getSound = () => 'PEW PEW';

  // Записываем переменную в глобальную область видимости
  global.blaster = blaster;
})(window)

window.blaster(); // 'PEW PEW'
```

МОДУЛИ ES6

```
// blaster.mjs
const blaster = () => {
  console.log(getSound());
};

const getSound = () => 'PEW PEW';

export default blaster;
```

```
// grenade.js
const grenade = () => {
  console.log(getSound());
};

const getSound = () => 'BOOM';

export default grenade;
```

```
<script type="module">
  import blaster from './blaster.mjs';
  import grenade from './grenade.js';

  blaster(); // 'foo'
  grenade(); // 'bar'
</script>
```

Модули ES в картинках

ОБЪЕКТЫ

MDN

СВОЙСТВА И МЕТОДЫ

```
const hero = {
  name: 'Хан Соло',     // Это свойство
  quote: function () { // А это метод
    return 'Я хоть раз вас обманывал?!';
  }
};
```

ОБРАЩЕНИЕ К СВОЙСТВАМ

```
const wookie = {
  name: 'Чубакка'
};

wookie.weapon = 'Арбалет';

wookie.name; // Чубакка
wookie.weapon; // Арбалет
```

ВЫЧИСЛЯЕМЫЕ СВОЙСТВА

```
const wookie = {
  ['name']: 'Чубакка'
};

wookie['na' + 'me']; // Чубакка
```

```
const family = {
  [1]: 'Люк',
  [2]: 'Лея',
  [3]: 'Дарт Вейдер'
};

family[1]; // Люк
family['1']; // Люк
family[1 + 1]; // Лея
```

Вычисляемые имена свойств всегда приводятся к строкам (кроме Symbol).

```
const weirdObject = {
  [{ foo: 1 }]: 'Stop',
  [{ bar: 2 }]: 'Please'
};

weirdObject[{ baz: 3}]; // Please
weirdObject['[object Object]']; // Please
```

СИМВОЛЫ

```
const name = Symbol();

const jedi = {
  [name]: 'Йода'
};

jedi.name; // undefined
jedi[name]; // Йода
```

КОРОТКИЙ СИНТАКСИС

```
const name = 'Мейс Винду';

const jedi = {
  name: name,
  quote: function () {
    return 'Спектакль окончен.'
  }
};
```

```
const name = 'Мейс Винду';

const jedi = {
  name,
  quote () {
    return 'Спектакль окончен.'
  }
};
```

ДЕСТРУКТУРИЗАЦИЯ

```
const jedi = {
  name: 'Депа Биллаба',
  height: 1.68
};

const name = jedi.name;
const age = jedi.age;

name; // 'Депа Биллаба'
age;// undefined
```

```
const jedi = {
  name: 'Депа Биллаба',
  height: 1.68
};

const { name, age } = jedi;


name; // 'Депа Биллаба'
age;// undefined
```

MDN

```
const jedi = {
  name: 'Коулман Требор',
  deceased: '22 ДБЯ'
};

const getName = ({ name }) => name

getName(jedi); // Коулман Требор
```

```
const jedi = {
  name: 'Коулман Требор',
  deceased: '22 ДБЯ'
};

const getName = ({ name: jediName }) => jediName

getName(jedi); // Коулман Требор
```

```
const getName = ({ name = 'Имярек' }) => name

getName({}); // Имярек
```

```
const ship = {
  name: 'Защитник',
  size: {
    length: 1137,
    width: 548
  }
};

const getShipLength = ({ size: { length } }) => length

getShipLength(ship); // 1137
```

```
const waitWhat = ({
  myValue: {
    inner = 'other value',
    otherInner: {
      andAnother: {
        andAnother: lol
      } = {
        andAnother: 'lol'
      }
    }
  },
  kek: [chebureck]
} = {
  myValue: {
    inner: null
  }
}) => [lol, chebureck]
```

УДАЛЕНИЕ СВОЙСТВ

```
const jedi = {
  name: 'Йаддль',
  born: '509 ДБЯ'
}

delete jedi.born;

// В объекте останется одно свойство
{
  name: 'Йаддль'
}
```

ПРОВЕРКА НАЛИЧИЯ СВОЙСТВ В ОБЪЕКТЕ

```
const jedi = {
  name: 'Пло Кун'
};

jedi.name !== undefined; // true
jedi.age !== undefined; // false
```

```
const jedi = {
  name: 'Пло Кун',
  age: undefined
};

jedi.name !== undefined; // true
jedi.age !== undefined; // false
```

```
const jedi = {
  name: 'Пло Кун',
  age: undefined
};

'age' in jedi; // true
'born' in jedi; // false
```

```
const jedi = {
  name: 'Пло Кун',
  age: undefined
};

jedi.hasOwnProperty('age'); // true
jedi.hasOwnProperty('born'); // false
```

```
jedi.hasOwnProperty('hasOwnProperty'); // false
'hasOwnProperty' in jedi; // true
```

ПЕРЕБОР СВОЙСТВ ОБЪЕКТА

```
const jedi = {
  name: 'Ади Галлия',
  deceased: '20 ДБЯ'
};

for (const key in jedi) {
  console.log(key, jedi[key])
}

// name, Ади Галлия
// deceased, 20 ДБЯ
```

```
const jedi = {
  name: 'Ади Галлия',
  deceased: '20 ДБЯ'
};

Object.keys(jedi);
// ['name', 'deceased']
```

```
const jedi = {
  name: 'Ади Галлия',
  deceased: '20 ДБЯ'
};

Object.values(jedi);
// ['Ади Галлия', '20 ДБЯ']
```

ДЕСКРИПТОРЫ СВОЙСТВ

learn.javascript.ru

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

console.log(Object.getOwnPropertyDescriptor(ship, 'name'));

// {
//   configurable: true,
//   enumerable: true,
//   value: "Тысячелетний сокол",
//   writable: true
// }
```

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

Object.defineProperty(ship, 'owner', {
  writable: false
});

ship.owner = 'Ландо'; // TypeError!
```

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

Object.defineProperty(ship, 'owner', {
  enumerable: false
});

for (let key in ship) {
  console.log(key);
}

// "name", "model"
```

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

Object.defineProperty(ship, 'owner', {
  enumerable: false
});

Object.keys(ship); // ['name', 'model']
```

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

console.log(`Look at my ${ship}!`);
// Look at my [object Object]!
```

Обычные объекты приводятся к строке `[object Object]`.

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300',
  toString () {
    return 'YT-1300'
  }
};

console.log(`Look at my ${ship}!`);
// Look at my YT-1300!
```

Это поведение можно переопределить с помощью метода `toString()`.

```
const ship = { ... };

Object.keys(ship);
// ['name', 'owner', 'model', 'toString']
```

```
const ship = { ... };

Object.defineProperty(ship, 'toString', {
  enumerable: false
});

Object.keys(ship);
// ['name', 'owner', 'model']
```

```
const ship = { ... };

Object.defineProperty(ship, 'owner', {
  configurable: false
});

Object.defineProperty(ship, 'owner', {
  configurable: true
}); // TypeError!

Object.defineProperty(ship, 'owner', {
  enumerable: false
}); // TypeError!

delete ship.owner; // TypeError!
```

Object.preventExtensions()

```
const ship = {
  name: 'Тысячелетний сокол',
  owner: 'Хан Соло',
  model: 'YT-1300'
};

Object.preventExtensions(ship);

ship.serialNumber = '492727ZED'; // TypeError!
```

Object.seal()

```
const ship = { ... };

Object.seal(ship);

Object.getOwnPropertyDescriptor(ship, 'owner');

// {
//   configurable: false,
//   enumerable: true,
//   value: "Тысячелетний сокол",
//   writable: true
// }
```

Object.freeze()

```
const ship = { ... };

Object.freeze(ship);

Object.getOwnPropertyDescriptor(ship, 'owner');

// {
//   configurable: false,
//   enumerable: true,
//   value: "Тысячелетний сокол",
//   writable: false
// }
```

СРАВНЕНИЕ

	Extend	Configure	Write	Read
Object.preventExtension	❌	✅	✅	✅
Object.seal	❌	❌	✅	✅
Object.freeze	❌	❌	❌	✅

Object.assign()

```
const hero = { name: 'Хан Соло' };
const weapon = { weapon: 'бластер' };

Object.assign(hero, weapon);

hero; // { name: 'Хан Соло', weapon: 'бластер' }
```

```
const stub = { model: 'B1', type: 'Battle droid' };
const weapons = { blaster: 'E-5' };

const assembleDroid = () => {
  return Object.assign({}, stub, weapons);
};

assembleDroid();
// Вернет новый объект
{
  model: 'B1',
  type: 'Battle droid',
  blaster: 'E-5',
}
```

Object.is()

```
NaN === NaN; // false
-0 === 0; // true
```

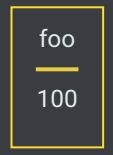
```
Object.is(NaN, NaN); // true
Object.is(-0, 0); // false
```

SameValue

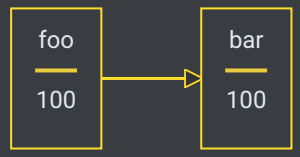
ССЫЛОЧНАЯ ПРИРОДА ОБЪЕКТОВ

В переменных сохраняются:
для примитивов — значения
для объектов — ссылки

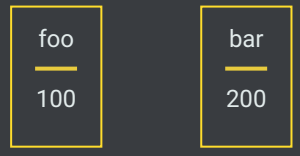
```
let foo = 100;
foo; // -> 100
```



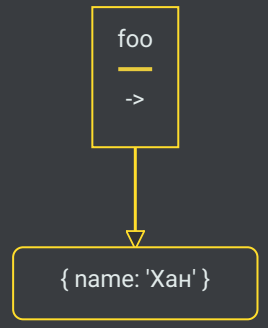

```
let bar = foo;
foo; // -> 100
bar; // -> 100
```



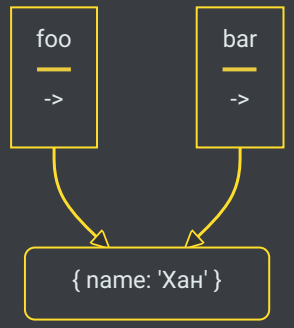

```
bar = 200;
foo; // -> 100
bar; // -> 200
```



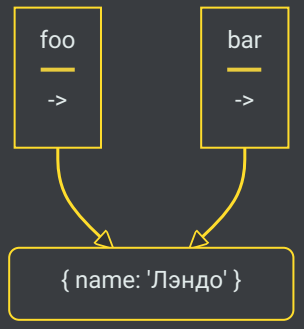
```
let foo = { name: 'Хан' };
foo.name; // -> Хан
```




```
let bar = foo;
foo.name; // -> Хан
bar.name; // -> Хан
```




```
bar.name = 'Лэндо';
foo.name; // -> Лэндо
bar.name; // -> Лэндо
```



```
const destroyers = ['Бесстрашный', 'Решительный', 'Защитник'];
```

```
const logFirstShip = (ship) => {
  console.log(ship.shift());
}
```

MDN: shift()

```
const destroyers = ['Бесстрашный', 'Решительный', 'Защитник'];

logFirstShip(destroyers); // 'Бесстрашный'
```

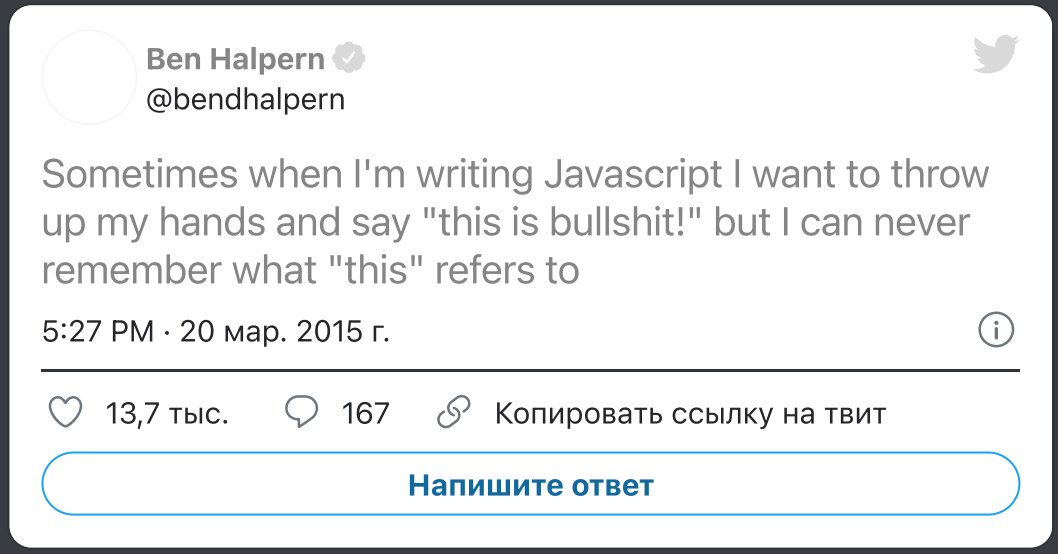
```
const destroyers = ['Бесстрашный', 'Решительный', 'Защитник'];

logFirstShip(destroyers); // 'Бесстрашный'

destroyers; // ['Решительный', 'Защитник'];
```

`Shift` — мутирующий метод!

THIS


Ben Halpern
@bendhalpern
Sometimes when I'm writing Javascript I want to throw up my hands and say "this is bullshit!" but I can never remember what "this" refers to
5:27 PM · 20 мар. 2015 г.
13,7 тыс.
167
Копировать ссылку на твит
Напишите ответ

1. ГЛОБАЛЬНЫЙ ОБЪЕКТ

```
this;
// Window {postMessage: ƒ, …}
```

```
const getThis = function () {
  return this;
}

getThis();
// Window {postMessage: ƒ, …}
```

СТРОГИЙ РЕЖИМ

```
'use strict';
const getThis = function () {
  return this;
}

getThis(); // undefined
```

2. КОНТЕКСТ МЕТОДА

```
const starDestroyer = {
  shipName: 'Забота',
  report () {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
};

starDestroyer.report();
// "Забота" готов к атаке!
```

```
const starDestroyer = {
  shipName: 'Забота',
  report () {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
};

const { report } = starDestroyer;

report();
// "undefined" готов к атаке!
```

```
const starDestroyer = {
  shipName: 'Забота',
  report () {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
}

const delay = (fn) => setTimeout(fn, 1000);

delay(starDestroyer.report);
// "undefined" готов к атаке!
```

```
'use strict';
const starDestroyer = {
  shipName: 'Забота',
  report () {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
};

const { report } = starDestroyer;

report();
// TypeError: Cannot read property 'shipName' of undefined
```

```
const report = function () {
  console.log(`"${this.shipName}" готов к атаке!`);
}

const starDestroyer = {
  shipName: 'Забота'
};

starDestroyer.report = report;
starDestroyer.report();
// "Забота" готов к атаке!
```

3. CALL И APPLY

```
const report = function () {
  console.log(`"${this.shipName}" готов к атаке!`);
}

const starDestroyer = { shipName: 'Забота' };
report.call(starDestroyer); // "Забота" готов к атаке!

const fighter = { shipName: 'T-70' }:
report.call(fighter); // "T-70" готов к атаке!
```

```
const attack = function (targetName, targetClass) {
  return `"${this.shipName}" нападает на ${targetClass} "${targetName}"!`;
}

const starDestroyer = { shipName: 'Забота' };

attack.call(starDestroyer, 'истребитель', 'X-wing');
// "Забота" нападает на истребитель "X-wing"!
```

```
const attack = function (targetName, targetClass) {
  return `"${this.shipName}" нападает на ${targetClass} "${targetName}"!`;
}

const fighter = { shipName: 'T-70' };

attack.apply(fighter, ['звездный разрушитель', 'Забота']);
// "T-70" нападает на звездный разрушитель "Забота"!
```

4. BIND

```
const report = function () {
  console.log(`"${this.shipName}" готов к атаке!`);
}

const fighter = { shipName: 'T-70' };
const fighterReport = report.bind(fighter);

fighterReport(); // "T-70" готов к атаке!
report(); // "undefined" готов к атаке!
```

```
const starDestroyer = {
  shipName: 'Забота',
  report () {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
}

const delay = (fn) => setTimeout(fn, 1000);

delay(starDestroyer.report.bind(starDestroyer));
// "Забота" готов к атаке!
```

5. ФУНКЦИИ-КОНСТРУКТОРЫ

```
function Destroyer (name) {
  this.shipName = name;
  this.type = 'Звездный разрушитель';
}

const dauntless = new Destroyer('Бесстрашный');

dauntless.name; // Бесстрашный
dauntless.type; // Звездный разрушитель
```

Контекст вызова функции зависит от способа вызова функции.

Подробнее о контексте

```
const obj = {
  a: 1,
  foo () {
    console.log(this.a);
  }
};

obj.foo.call({ a: 2 }); // ???
```

```
const obj = {
  a: 1,
  foo () {
    console.log(this.a);
  }
};

obj.foo.call({ a: 2 }); // 2
```

```
const obj = {
  a: 1,
  foo () {
    console.log(this.a);
  }
};

const foo = obj.foo.bind(obj);

foo.call({ a: 2 }); // ???
```

```
const obj = {
  a: 1,
  foo () {
    console.log(this.a);
  }
};

const foo = obj.foo.bind(obj);

foo.call({ a: 2 }); // 1
```

```
function foo (value) {
  this.value = value;
}

const objA = {};

foo = foo.bind(obj);

const objB = new foo(3);

objA; // ???
objB; // ???
```

```
function foo (value) {
  this.value = value;
}

const objA = {};

foo = foo.bind(objA);

const objB = new foo(3);

objA; // {}
objB; // { value: 3 }
```

1. `new`
2. `bind`
3. `call, apply`
4. `точка`
5. Глобальный объект/undefined

Приоритет

6. СТРЕЛОЧНЫЕ ФУНКЦИИ

```
const starDestroyer = {
  shipName: 'Забота',
  report: () => {
    console.log(`"${this.shipName}" готов к атаке!`);
  }
}

starDestroyer.report(); // "undefined" готов к атаке!
starDestroyer.report.call(starDestroyer); // "undefined" готов к атаке!

const report = starDestroyer.report.bind(starDestroyer);
report(); // "undefined" готов к атаке!
```

У стрелочных функций нет своего контекста. Они всегда используют контекст из замыкания, то есть глобальный или контекст родительской функции.

```
const starDestroyer = {
  shipName: 'Забота',
  delayedReport() {
    setTimeout(function () {
      console.log(`"${this.shipName}" готов к атаке!`);
    }, 1000)
  }
}

starDestroyer.delayedReport();
// "undefined" готов к атаке!
```

```
const starDestroyer = {
  shipName: 'Забота',
  delayedReport() {
    setTimeout(() => {
      console.log(`"${this.shipName}" готов к атаке!`);
    }, 1000)
  }
}

starDestroyer.delayedReport();
// "Забота" готов к атаке!
```

Контекст вызова:

- У обычных функций: задается в момент вызова и зависит от способа вызова.
- У стрелочных функций: не существует, this всегда ссылается на контекст вызова родительской функции

```
function getShipName () {
  return this.shipName;
}

getShipName.shipName = '🤔';

console.log(getShipName());
// ???
```

```
function getShipName () {
  return this.shipName;
}

getShipName.shipName = '🤔';

console.log(getShipName());
// undefined
```

```
function getShipName () {
  return this.shipName;
}

const ship = {
  shipName: 'Бесстрашный',
  getShipName () {
    return getShipName();
  }
}

console.log(ship.getShipName());
// ???
```

```
function getShipName () {
  return this.shipName;
}

const ship = {
  shipName: 'Бесстрашный',
  getShipName () {
    return getShipName();
  }
}

console.log(ship.getShipName());
// undefined
```

```
function getShipName () {
  return this.shipName;
}

const ship = {
  shipName: 'Бесстрашный',
  getShipName () {
    return getShipName.call(this);
  }
}

console.log(ship.getShipName());
// Бесстрашный
```

GETTERS & SETTERS

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло'
};

console.log(`${pilot.firstName} ${pilot.lastName}`);
// "Хан Соло"
```

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло',
  fullName () {
    return `${this.firstName} ${this.lastName}`;
  }
};

console.log(pilot.fullName()); // "Хан Соло"
```

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло'
};

Object.defineProperty(pilot, 'fullName', {
  get () {
    return `${this.firstName} ${this.lastName}`;
  }
});

console.log(pilot.fullName); // "Хан Соло"
```

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло',
  get fullName () {
    return `${this.firstName} ${this.lastName}`;
  }
};

console.log(pilot.fullName); // "Хан Соло"
```

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло',
  get fullName () {
    return `${this.firstName} ${this.lastName}`;
  }
};

pilot.fullName = 'Лэндо Калриссиан';
console.log(pilot.fullName); // Хан Соло
```

```
const pilot = {
  firstName: 'Хан',
  lastName: 'Соло',
  get fullName () {
    return `${this.firstName} ${this.lastName}`;
  },
  set fullName (value) {
    const splitted = value.split(' ');

    this.firstName = splitted[0];
    this.lastName = splitted[1];
  }
};
```

```
pilot.fullName = 'Лэндо Калриссиан';
console.log(pilot.firstName); // Лэндо
console.log(pilot.lastName); // Калриссиан
```

МАССИВЫ

MDN

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

ships[0]; // Бесстрашный
ships[1]; // Решительный
ships[100]; // Undefined
```

ДЕСТРУКТУРИЗАЦИЯ

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

const [dauntless, resolute, defender] = ships;
dauntless; // Бесстрашный
resolute; // Решительный
defender; // Зашитник
```

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

const [, resolute] = ships;
resolute; // Решительный

const [dountless,, defender] = ships;
dauntless; // Бесстрашный
defender; // Зашитник
```

...REST

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

const [dountless, ...otherShips] = ships;
dountless; // Бесстрашный
otherShips; // ['Решительный', 'Защитник']
```

POP/PUSH

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];
const lastShip = ships.pop();

lastShip; // 'Защитник'
ships; // ['Бесстрашный', 'Решительный']

ships.push('Спаситель');
ships; // ['Бесстрашный', 'Решительный', 'Спаситель']
```

SHIFT/UNSHIFT

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];
const firstShip = ships.shift();

firstShip; // 'Бесстрашный'
ships; // ['Решительный', 'Защитник']

ships.unshift('Спаситель');
ships; // ['Спаситель', 'Решительный', 'Защитник']
```

FOREACH

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

ships.forEach(ship => console.log(ship));
// Бесстрашный
// Решительный
// Защитник
```

MAP

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

const capitalized = ships.map(ship => ship.toUpperCase());
capitalized; // ['БЕССТРАШНЫЙ', 'РЕШИТЕЛЬНЫЙ', 'ЗАЩИТНИК']
```

FILTER

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

const adjectives = ships.filter(ship => ship.endsWith('ный'));
adjectives; // ['Бесстрашный', 'Решительный']
```

SOME/EVERY

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

ships.some(ship => ship.endsWith('ный')); // true
ships.every(ship => ship.endsWith('ный')); // false
```

FIND

```
const ships = ['Бесстрашный', 'Решительный', 'Защитник'];

ships.find(ship => !ship.endsWith('ный')); // Защитник
```

REDUCE

```
const jedi = [
  { name: 'Люк', age: 19 },
  { name: 'Йода', age: 896 },
  { name: 'Энакин', age: 42 }
]

const oldestJedi = jedi.reduce((result, nextJedi, jedis) => {
  return result.age > nextJedi.age ? result : nextJedi;
}, jedi[0]);

// OR

const oldestJedi = jedi.reduce((result, nextJedi, jedis) => {
  return result.age > nextJedi.age ? result : nextJedi;
});
```

MAP

```
const ships = new Map();
```

MDN

```
const ships = new Map();

map.set('Бесстрашный', {
  name: 'Бесстрашный',
  length: 1137
});

map.get('Бесстрашный');
// Вернет объект
{
  name: 'Бесстрашный',
  length: 1137
}
```

```js
const map = {};

map[0] = 'foo';
map['0'] = 'bar';

map[0]; // bar
map['0']; // bar
```

```js
const map = new Map();

map.set(0, 'foo');
map.set('0', 'bar');

map.get(0); // foo
map.get('0'); // bar
```

```
const map = {};

map[true] = 'foo';
map['true'] = 'bar';

map[true]; // bar
map['true']; // bar
```

```
const map = new Map();

map.set(true, 'foo');
map.set('true', 'bar');

map.get(true); // foo
map.get('true'); // bar
```

```
const foo = { foo: 'foo' };
const bar = { bar: 'bar' };
```

```
const map = {};

map[foo] = 'foo';
map[bar] = 'bar';

map[foo]; // bar
map[bar]; // bar
```

```
const map = new Map();

map.set(foo, 'foo');
map.set(bar, 'bar');

map.get(foo); // foo
map.get(bar); // bar
```

```
const map = new Map();

map.set(NaN, 'bar');
map.get(NaN); // bar
```

Для сравнения ключей используется алгоритм SameValueZero

```
const shipSizes = new Map();
shipSizes.set('Бесстрашный', 1137);
shipSizes.set('Сокол', 34.7);

shipSizes.forEach(console.log);
// 1137, 'Бесстрашный', Map
// 34.7, 'Сокол', Map
```

```
const shipSizes = new Map();
shipSizes.set('Бесстрашный', 1137);
shipSizes.set('Сокол', 34.7);

shipSizes.has('Сокол'); // true

shipSizes.size; // 2
shipSizes.delete('Сокол');

shipSizes.values();
shipSizes.keys();

shipSizes.clear();
```

SET

```
const ships = new Set();
```

MDN

```
const ships = new Set();

ships.add('Сокол');
ships; // Set(1) {"Сокол"}

ships.add('Сокол');
ships; // Set(1) {"Сокол"}
```

```
const values = [1, 1, 2, 2, 3, 3, 3];

const uniqueValues = new Set(values);

uniqueValues; // Set(3) {1, 2, 3}
```

```
const ships = new Set();
ships.set('Бесстрашный');
ships.set('Сокол');

ships.has('Сокол'); // true

ships.size; // 2
ships.delete('Сокол');

ships.keys();

ships.clear();
```

ЦИКЛ for .. of

```
const ships = new Set();
ships.set('Бесстрашный');
ships.set('Забота');

for (let ship of ships) {
  console.log(ship);
}
// 'Бесстрашный'
// 'Забота'
```

Итераторы и генераторы

```
const ships = ['Бесстрашный', 'Забота'];

for (let ship of ships) {
  console.log(ship);
}
// 'Бесстрашный'
// 'Забота'
```

```
const ship = 'Забота';

for (let letter of ship) {
  console.log(letter);
}
// 'З', 'а', 'б', 'о', 'т', 'а'
```

```
const shipSizes = new Map();
shipSizes.set('Бесстрашный', 1137);
shipSizes.set('Сокол', 34.7);
```

```
for (let ship of shipSizes.keys()) {
  console.log(ship);
}
// 'Бесстрашный'
// 'Забота'
```

```
for (let size of shipSizes.values()) {
  console.log(size);
}
// 1137
// 34.7
```

... spread

```
const log = (...args) => {
  console.log(...args);
};

log(...['a', 'b', 'c']);
```

```
const shipSizes = new Map();
shipSizes.set('Бесстрашный', 1137);
shipSizes.set('Сокол', 34.7);

Math.max(...shipSizes.values()); // 1137
```

```
const destroyers = ['Бесстрашный', 'Забота'];
const copy = [...destroyers];
```

```
const destroyers = ['Бесстрашный', 'Забота'];
const fighters = ['Шип', 'Поток'];

const ships = [...destroyers, ...fighters];
```

Rest и spread - синтаксис, а не операторы!