МОДУЛИ

```
// some-module.js
export var someVar = "var";
export const num = num;
export let arrowFunc = param => param * 2;

export function mult(param) {
  return param * 2;
}

function _private(secret) { return secret; }

export class Person {
  constructor(name) {
    this.name = name;
  }
}
```

```
// sub-module.js
import { someVar } from "some-module.js"

// main.js
import { num, arrawFunc, mult, Person } from "some-module.js"

let person = new Person("John");
console.log(person);

num = 2;                              // error
let mult = null;                      // error
```

```
// module.js
export let num = 1;
export let changeNum = val => num = val;

// main.js
import { num, changeNum } from "module.js";

console.log(num);           // 1
changeNum(2);
console.log(num);           // 2
```

```
// main.js
import * as someModule from "some-module.js"

let result = someModule.mult(2);
console.log(result);
```

```
// some.js
let personAge = 20;
export { personAge as age };

// main.js
import { age } from "some.js";

export let f = () => `Person is ${age} years old`;

// age.js
import { f as getAge } from "main.js";
```

```
// some-module.js
export default function(a, b) {
  return a + b;
}

// main.js
import f from "some-module.js"
let result = f(2);

export { result as default };

// third.js
import res from "main.js"
console.log(res);
```

```
// some-module.js
export default function() { return "default"; };
export var str = "hello there";

// main.js
import { default as f, str } from "some-module.js";
// or
import f, { str } from "some-module.js";
```

DESTRUCTURING

OBJECT DESTRUCTURING

```
let obj = {
  name: "Bill",
  age: 40
};

let { name, age } = obj;

console.log(name);              // "Bill"
console.log(ag);                // 40
```

```
let obj = {
  onTouch: "&",
  hero: "<"
};

let f = (config) => {
  let { onTouch, hero = "Default" } = config;

  return `onTouch: ${onTouch}, hero: ${hero}`;
};
```

```
// переопределение локальных переменных

let obj = {
  prop: "some value"
};

let { prop: localProp, anotherProp: p = "value" } = obj;
console.log(localProp);
console.log(p);
```

```
let o = {
  p: {
    k: "some value",
    l: "another value"
  }
};

let { p: { k }, p: { l: local }, p: { m = "default" } } = o;

console.log(k);
console.log(local);
console.log(m);
```

```
let f = ({
          name = "Unknown",
          age = 18
        }) => `${name} is ${age}`;

let person = { name: "Sam", age: 35 };

console.log(f(person));
```

ARRAY DESTRUCTURING

```
let seq = [1, 2, 3];

let [one, two] = seq;
let [,, three] = seq;

console.log(one);
console.log(two);
console.log(three);
```

```
let nested = [1, [2, 3]];
let [ one, [, three] ] = nested;

console.log(one);
console.log(three);
```

```
let ar = ["big", "bada", "boom"];
let [first, ...rest] = ar;

console.log(first);              // "big"
console.log(rest);               // ["bada", "boom"]
```

```
// clone array

let someArray = [1, 2, 3];
let [ ...anotherArray ] = someArray;

anotherArray[1] = 10;
console.log(someArray[1]);
```

ГЕНЕРАТОРЫ И ИТЕРАТОРЫ

```
function *generatorFunction() {
  yield "some value";
  yield "another value";
  yield 3;
  yield { prop: "value" };
}

let iterator = generatorFunction();

console.log(iterator.next());
console.log(iterator.next().done);
console.log(iterator.next().value);
console.log(iterator.next().value);
console.log(iterator.next());
```

```
let getObjectKeys = function *(obj) {
  for(let prop in obj) {
    yield prop;
  }
}

let o = { name: "Jack", lastName: "Sparrow" };
let getKey = getObjectKeys(o);

console.log(getKey.next().value);       // "name"
console.log(getKey.next().value);       // "lastName"
console.log(getKey.next().value);       // undefined
```

```
function *generate() {
  setTimeout(function() {
    yield "100ms passed";              // Syntax error
  }, 100);
}
```

```
let o = {
  a: [1, 2, 3],
  *getValues() {
    let [...a] = this.a;
    for(let i = 0, l = a.length; i < l; i++) {
      yield a[i];
    }
  }
};

let getValue = o.getValues();
console.log(getValue.next());
console.log(getValue.next());
console.log(getValue.next());
console.log(getValue.next());
```

FOR-OF LOOP

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

for(let num of ar) {
  console.log(num);
}
```

DEVDOCS: SYMBOL

MDN: SYMBOL

```
let ar = [1, 2, 3];

let getElements = ar[Symbol.iterator]();
console.log(getElements.next());
console.log(getElements.next());
console.log(getElements.next());
console.log(getElements.next());
```

```
let o = {
  a: [1, 2, 3],
  *[Symbol.iterator]() {
    let [...a] = this.a;
    for(let el of a) {
      yield `el is ${el}`;
    }
  }
};

for(let el of o) {
  console.log(el);
}
```

```
let person = {
  name: "Paul",
  lastName: "Irish",
  age: 35,
  *[Symbol.iterator]() {
    for(let prop in this) {
      if(this.hasOwnProperty([prop])) yield { [prop]: this[prop] };
    }
  }
};

for(let prop of person) {
  console.log(prop);
}
```

```
let person = {
  name: "Paul",
  lastName: "Irish",
  age: 35,
  *[Symbol.iterator]() {
    for(let prop in this) {
      if(this.hasOwnProperty([prop])) yield [ [prop], this[prop] ];
    }
  }
};

for(let [key, value] of person) {
  console.log(`${key}: ${value}`);
}
```

САМОСТОЯТЕЛЬНО

- Set
- WeakSet
- Map
- WeakMap
- Proxy and Reflect

ПОЛЕЗНЫЕ ССЫЛКИ

- Understanding ECMAScript 6
- Exploring ES6
- JavaScript Allongé (ES5)
- JavaScript Allongé, the "Six" Edition
- Современные возможности ES-2015
- You Don't Know JS: ES6 & Beyond