

# Знакомство с ES6

2009 -2015

- ЕСМА-262 5th Edition в декабре 2009
- ЕСМА-262 5.1 Edition в июне 2011
- ЕСМА-262 6th Edition в июне 2015

ES6

## Блочная область видимости (let и const)

### Оператор let

- Позволяет объявить локальную переменную с областью видимости, ограниченной текущим блоком кода

## Блочная область видимости

```
1  var apples = 5;
2  ☐ if (true) {
3      var apples = 10;
4      console.log(apples); //10 (внутри блока)
5  }
6  console.log(apples); //10 (снаружи блока то же самое)
```

```
1  let apples = 5;
2  ☐ if (true) {
3      let apples = 10;
4      console.log(apples); //10 (внутри блока)
5  }
6  console.log(apples); //5 (снаружи блока значение не изменилось)
```

# Блочная область видимости

```
1 console.log(a); //undefined
2 var a = 5;
3
4 console.log(b); //ReferenceError: a is not defined
5 let b = 5;
6
7 let x;
8 let x; //SyntaxError: Identifier 'x' has already been declared
9
10 if (true) {
11     let apples = 10;
12     console.log(apples); //10 (внутри блока)
13 }
14 console.log(apples); //ReferenceError: apples is not defined
15
```

# Блочная область видимости

```
1 //ES5
2
3 for(var i=0; i<10; i++) {
4     console.log(i); //0, 1, 2, 3, 4 ... 9
5 }
6 console.log(i); //10
7
8
9
10
11
12 //ES6
13
14 for (let i = 0; i<10; i++) {
15     console.log(i); //0, 1, 2, 3, 4 ... 9
16 }
17 console.log(i); //i is not defined
18
19
20
```

# Блочная область видимости

## Оператор const

- Значение констант не может быть изменено новым присваиванием и не может быть переопределено. Константы подчиняются области видимости уровня блока



## Блочная область видимости

```
1  const lit = 4;
2  lit = 5; //TypeError: Литерал изменить нельзя
3  const obj = { a: 1 };
4  obj.a = 2; //Значения внутри объекта изменить можно
5  console.log(obj); //{ a: 2 }
6  obj = { a: 3 }; //TypeError: Ссылку менять нельзя
7  const arr = [1, 2, 3];
8  arr.push(4); //Значения внутри массива изменить можно
9  console.log(arr); // [1, 2, 3, 4]
10 arr = [4, 3, 2, 1]; //TypeError: Ссылку менять нельзя
```

# Деструктуризация

Деструктуризация- особый синтаксис присваивания, при котором можно присвоить массив или объект сразу нескольким переменным, разбив его на части

- Деструктуризация позволяет привязывать данные при совпадении паттерна
- Поддерживается для массивов и объектов
- Деструктуризация устойчива к ошибкам и во многом похожа на поиск поля в объекте
- Возвращает `undefined`, если что-то пошло не так

## Деструктуризация массива

```
1 let [firstName, lastName] = ['Иван', 'Иванов'];  
2 console.log(firstName); //Иван  
3 console.log(lastName); //Иванов
```

# Деструктуризация массива

```
1 //первый и второй элементы не нужны
2 let [, , middleName] = "Иванов Иван Иванович (1971)".split(" ");
3 console.log(middleName); //Иванович
4 //Первый и второй, а также все элементы после третьего
5 // никуда не записались, они были отброшены
```

# Деструктуризация

## Оператор spread

- Значением `rest` будет массив из оставшихся элементов
- Вместо `rest` можно использовать другое имя переменной
- Оператор `spread` - троеточие
- `Spread` должен стоять только перед последним элементом

# Деструктуризация массива

```
1 let [first, last, ...rest] = "Все ДЗ должны быть сданы".split(" ");
2 console.log(first); //Все
3 console.log(last); //ДЗ
4 console.log(rest); //["должны", "быть", "сданы"]
```

# Значения по умолчанию

```
1 let [firstName, lastName] = [];  
2 console.log(firstName); //undefined
```

```
1 let [firstName="Гость", lastName="Анонимный"] = [];  
2 console.log(firstName); //Гость  
3 console.log(lastName); //Анонимный
```

```
1 function defaultLastName() {  
2     return Date.now() + '-visitor';  
3 }  
4  
5 //lastName получит значение, соответствующее текущей дате:  
6 let [firstName, lastName=defaultLastName()] = ["Вася"];  
7 console.log(firstName); //Вася  
8 console.log(lastName); //1436...-visitor
```

## Отделение объявления от присваивания

```
1  let a, b;  
2  [a, b] = [1, 2];  
3  console.log(a); //1  
4  console.log(b); //2
```



# Swapping

```
1 let a = 1;  
2 let b = 3;  
3 [a, b] = [b, a];  
4 console.log(a); // 3  
5 console.log(b); // 1
```

## Использование возвращающей функции

```
1 function f() {  
2     return [1, 2];  
3 }  
4 let a, b;  
5 [a, b] = f();  
6 console.log(a); //1  
7 console.log(b); //2
```

# Работа с другими коллекциями

```
1 //Строки
2 var [a, b, c] = 'xyz';
3 console.log(a, b, c); //x y z
```

```
1 //Коллекции DOM элементов
2 var [link1, link2] = document.links;
3 console.log(link1.tagName); //A
4 console.log(link2.textContent); //Стандарт ECMAScript 6
```

# Деструктуризация объектов

- Указываем, какие свойства в какие переменные должны перейти
- Объект справа – существующий объект
- Список слева – список переменных, в которые записываются соответствующие свойства

## Деструктуризация объекта

```
1  const o={firstName:'Иван', lastName:'Иванов'};  
2  let {firstName, lastName} = o;  
3  console.log(firstName); //Иван  
4  console.log(lastName); //Иванов
```

# Использование возвращающей функции

```
1  var calc = function(num) {  
2    |   return Math.pow(num, 2);  
3  | };  
4  var { prop: x, pow: y = calc(x)} = { prop: 4 };  
5  console.log(x, y); // 4 16  
6  |
```

## Другое имя переменной

```
1 let options = {
2     title: "Меню",
3     width: 100,
4     height: 200
5 };
6 let {width: w, height: h, title} = options;
7 console.log(title); //Меню
8 console.log(w); //100
9 console.log(h); //200
```

# Значение по умолчанию

```
1  let options = {  
2    |   title: "Меню"  
3  };  
4  let {width=100, height=200, title} = options;  
5  console.log(title); //Меню  
6  console.log(width); //100  
7  console.log(height); //200
```



# Отделение объявления от присваивания

```
1 | var width, height;  
2 | ({width, height} = {width: 100,height: 200});  
3 |  
4 | console.log(width); //100  
5 | console.log(height); //200
```

## Использование конкретного имени

```
1 let key = "title";  
2 let { [key]: t } = {title: "Меню"};  
3 console.log(t); //Меню
```

# Вложенные деструктуризации

```
1 let options = {
2     size: {
3         width: 100,
4         height: 200
5     },
6     items: ["Пончик", "Пирожное"]
7 };
8
9 let { title="Меню",
10     size: {width, height},
11     items: [item1, item2]
12 } = options;
13
14 //Меню 100 200 Пончик Пирожное
15 console.log(title); //Меню
16 console.log(width); //100
17 console.log(height); //200
18 console.log(item1); //Пончик
19 console.log(item2); //Пирожное
```

# Функции

# Параметры по умолчанию

```
1 function [name]([param1[= defaultValue1][, ..., paramN[= dValueN]])
2 {
3     //function's body
4 }
```

```
1 function multiply(a, b = 1) {
2     return a*b;
3 }
4 console.log(multiply(5)); //5
```

```
1 function multiply(a, b = 1) {  
2     return a*b;  
3 }  
4 multiply(5, undefined); //5
```

```
1 function append(value, array = []) {  
2     array.push(value);  
3     return array;  
4 }  
5 append(1); // [1]  
6 append(2); // [2]
```

```
1 function getRandomInt(min, max) {  
2   |   return Math.floor(Math.random() * (max - min)) + min;  
3   | }  
4 function print(number = getRandomInt(10, 100)){  
5   |   console.log(number);  
6   | }  
7 print();//29
```

# Оператор spread

*В нашем случае, rest - это массив, а значит, можно использовать методы map, forEach и т.д.*

```
1 function printName(firstName, lastName, ...rest) {  
2     console.log(firstName + ' ' + lastName + ' - ' + rest);  
3     console.log(rest);  
4 }  
5 printName('Иван', 'Иванов', 'Иванович', '1959');  
6 //Иван Иванов - Иванович,1959  
7 //["Иванович", "1959"]
```



## Свойство name

```
1 function getRandomInt(min, max)
2 {
3     return Math.floor(Math.random() * (max - min)) + min;
4 }
5 console.log(getRandomInt.name); //getRandomInt
6 getRandomInt.name = 'f';
7 console.log(getRandomInt.name); //getRandomInt
8 console.log(f.name); //ReferenceError: f is not defined
```

# Стрелочные функции

# Стрелочные функции

*Выражения имеют более короткий синтаксис, всегда анонимные и лексически привязанные к значению `this`.*

Синтаксис:  $(\text{param1}, \text{param2}, \text{paramN}) \Rightarrow \text{expression}$

# Особенности использования стрелочных функций

- Лексическое связывание. Значения `this`, `super` и `arguments` определяются не тем, как стрелочные функции были вызваны, а тем, как они были созданы
- Неизменяемые `this`, `super` и `arguments`. Значения этих переменных внутри стрелочных функций остаются неизменными на протяжении всего жизненного цикла функции
- Стрелочные функции не могут быть использованы как конструктор
- Недоступность «собственного» значения переменной `arguments`, `this` ...

# Короткая запись

```
1  var Specialization = [  
2      "PFE",  
3      "FE"  
4  ];  
5  
6  var a = faculties.map(function(s){ return s.length });//[3,2]  
7  var b = faculties.map( s => s.length);//[3,2]
```

# Пример

```
1  let square = x => x*x;
2  console.log(square(3)); //9
3
4  let sum = (x, y) => x + y;
5  console.log(sum(3,4)); //7
6
7  let getObject = () => ({ brand: 'BMW' });
8  console.log(getObject()); //Object {brand: "BMW"}
9
10 let func = () => 77;
11 console.log(func()); //77
```

# Отсутствие Arguments

```
1 function foo() {  
2     var f = (i) => arguments[0]+i;  
3     //возмет arguments из функции foo  
4     console.log(f(2));  
5 }  
6 foo(1); //3
```

# Отсутствие запуска с new

```
1  var a = new (function() {});  
2  //переменной a будет присвоено значение экземпляра анонимной функции  
3  var b = new (() => {});  
4  //TypeError: (intermediate value) is not a constructor
```