Знакомство с ES6

2009 -2015

- ECMA-262 5th Edition в декабре 2009
- ECMA-262 5.1 Edition в июне 2011
- ECMA-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

```
var apples = 5;
if (true) {
    var apples = 10;
    console.log(apples); //10 (внутри блока)
}
console.log(apples); //10 (снаружи блока то же самое)
```

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

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

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

Значения по умолчанию

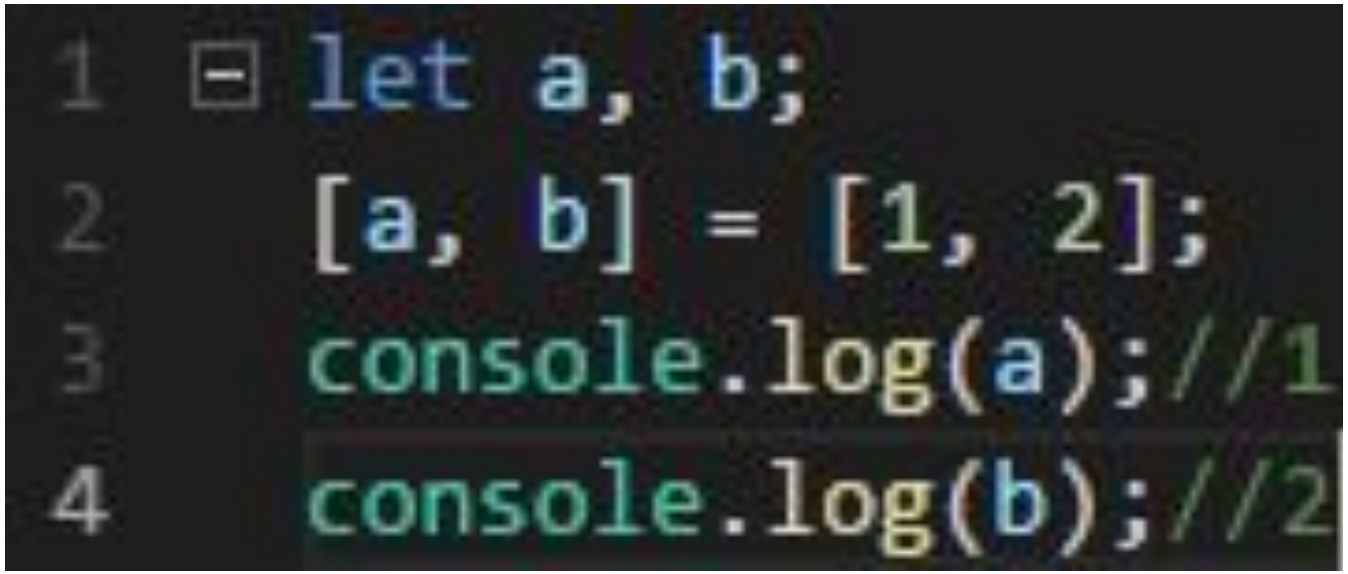
```
let [firstName, lastName] = [];
console.log(firstName);//undefined
```

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

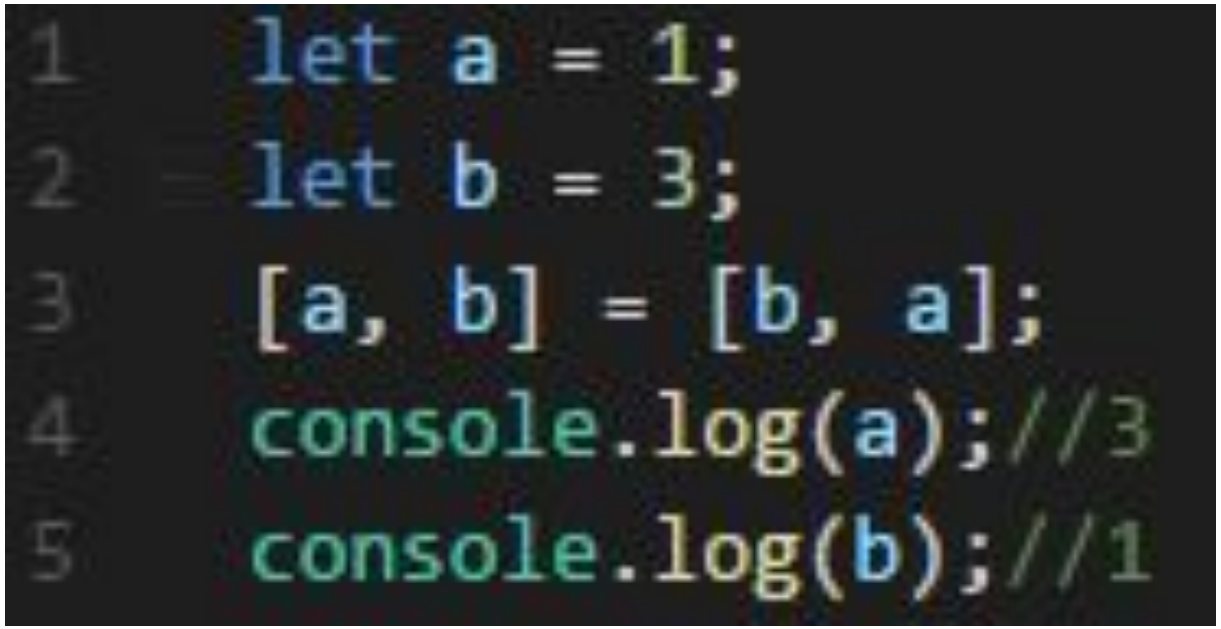
```
function defaultLastName() {
    return Date.now() + '-visitor';
}

//lastName получит значение, соответствующее текущей дате:
let [firstName, lastName=defaultLastName()] = ["Вася"];
console.log(firstName);//Вася
console.log(lastName);//1436...-visitor
```

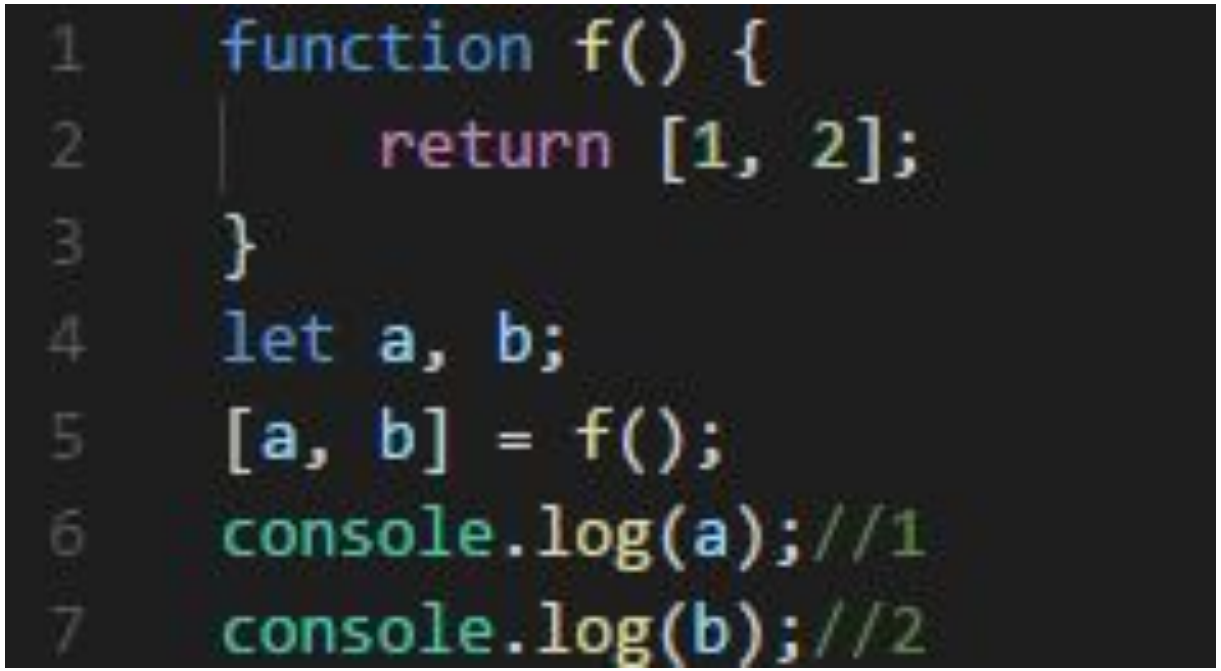
Отделение объявления от присваивания



Swapping



Использование возвращающей функции



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

```
let options = {
    title: "Меню",
    width: 100,
    height: 200
};
let {width: w, height: h, title} = options;
console.log(title);//Меню
console.log(w);//100
console.log(h);//200
```

Значение по умолчанию

```
let options = {
    title: "Меню"
};
let {width=100, height=200, title} = options;
console.log(title);//Меню
console.log(width);//100
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

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

Вложенные деструктуризации

```
let options = {
    size: {
        width: 100,
        height: 200
    },
    items: ["Пончик", "Пирожное"]
};

let { title="Меню",
      size: {width, height},
      items: [item1, item2]
    } = options;

//Меню 100 200 Пончик Пирожное
console.log(title);//Меню
console.log(width);//100
console.log(height);//200
console.log(item1);//Пончик
console.log(item2);//Пирожное
```

Функции

Параметры по умолчанию

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

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

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

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

```
function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min)) + min;
}
function print(number = getRandomInt(10, 100)){
    console.log(number);
}
print();//29
```

Оператор spread

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

```
function printName(firstName, lastName, ...rest) {
    console.log(firstName + ' ' + lastName + ' - ' + rest);
    console.log(rest);
}
printName('Иван', 'Иванов', 'Иванович', '1959');
//Иван Иванов - Иванович,1959
//["Иванович", "1959"]
```

Свойство name

```
function getRandomInt(min, max)
{
    return Math.floor(Math.random() * (max - min)) + min;
}
console.log(getRandomInt.name);//getRandomInt
getRandomInt.name = 'f';
console.log(getRandomInt.name);//getRandomInt
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: (param1, param2, paramN) => 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

```
var Specialization = [
    "PFE",
    "FE"
  ];

  var a = faculties.map(function(s){ return s.length });//[3,2]
  var b = faculties.map( s => s.length);//[3,2]
```

Пример

```
let square = x => x*x;
console.log(square(3));//9

let sum = (x, y) => x + y;
console.log(sum(3,4));//7

let getObject = () => ({ brand: 'BMW' });
console.log(getObject());//Object {brand: "BMW"}

let func = () => 77;
console.log(func());//77
```

Отсутствие Arguments

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