Lexical Environment

- The official ES5 docs defines Lexical Environment (an abstract concept really) as the place where “the association of Identifiers to specific variables and functions based upon the lexical nesting structure of ECMAScript code” is stored.

```
function makeCounter() {
    var currentCount = 1;
    return function () {
        return currentCount++;
    };
};
var counter = makeCounter();//каждый вызов увеличивает счётчик и возвращает результат
console.log(counter()); //1
console.log(counter()); //2
console.log(counter()); //3
var counter2 = makeCounter(); //если создать другой счётчик, он будет независим от первого
console.log(counter2()); //1

```

Lexical Environment

```
var y = 5;
var x = function(){
    return y;
};
var z = function(t){
    var y = 10;
    return t();
};
z(x);//5
```

```
var y = 5;
var x = function(){
    return y;
};
var z = function(t){
    var y = 10;
    return t();
};
y = 15;
z(x);//15
```

```
var y = 5;
var x = function(){
    return y;
};
var z = function(t){
    y = 10;
    return t();
};
z(x);//10
```

this

Simple call:

```
1. function f() {
2.     console.log(this === window); // true
3. }
4. f();
```

Self-invoking function:

```
1. (function () {
2.     console.log(this === window); // true
3. })();
```

Constructor-like call:

```
1. function f() {
2.     this.x = 5;
3.     console.log(this === window); // false
4. }
5. var o = new f();
6. console.log(o.x === 5); // true
```

this

Object param method call:

```
1.  var o = {
2.      f: function() {
3.          return this;
4.      }
5.  }
6.  console.log(o.f() === o);//true
7.  var o2 = {f: o.f};
8.  console.log(o2.f() === o2);//true
```

Use ‘Call’ and ‘Apply’:

```
1.  function f() {
2.  }
3.  f.call(window); // this внутри функции f будет ссылаться на объект window
4.  f.call(f); //this внутри f будет ссылаться на f
```

OOP on JS

- Function like
- Prototype like

Function like OOP

```
function Person(name, age){
    this.name = name;
    this.age = age;
    this.getName = function(){
        return this.name;
    }
    this.getAge = function(){
        return this.age;
    }
}
```

Creating object

```
var person = new Person('Matthew', 99);
console.log(person.getName());//Matthew
console.log(person.getAge()); //99
console.log(person instanceof Object);//true
console.log(person instanceof Person);//true
```

Order of creation

- Object creation
- Assigning a new object to the this variable of the constructor (after which this points to a new object)
- Execution of code inside the constructor (adding properties to the new object)
- Return a new object

Problem

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.getName = function() {
        return this.name;
    }
}

var p1 = new Person('Ira', 23);
var p2 = new Person('Max', 25);

console.log(p1.getName()); //Ira
console.log(p2.getName()); //Max
console.log(p1.getName == p2.getName); //false
```

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.getName = new Function('return this.name;');
}

var p1 = new Person('Ira', 23);
var p2 = new Person('Max', 25);

console.log(p1.getName());//Ira
console.log(p2.getName());//Max
console.log(p1.getName == p2.getName); //false
```

Resolution

```
1  function Person(name, age) {
2      this.name = name;
3      this.age = age;
4      this.getName = getName
5  }
6
7  function getName(){
8      return this.name;
9  }
10
11 var p1 = new Person('Ira', 23);
12 var p2 = new Person('Max', 25);
13 console.log(p1.getName == p2.getName); //true
```

Getter and setter

```
function Person(name, age) {
    var fAge = 'лет';
    this.name = name;
    this.getAge = function(){
        return age + ' ' + fAge;
    }
}
var person = new Person(‘Matt', 99);
person.name = 'Matthew';
console.log(person.name);//Matthew
console.log(person.getAge());//99 лет
```

Inheritance

```
function Person(name, age){
        this.name = name;
        this.age = age;
        this.getName = function(){
        return this.name;
        }
        this.getAge = function(){
        return this.age;
        }
}

function Student(course, group) {

    Person.call(this);

    this.course = course;

    this.group = group;

}

var student = new Student(1, 1);

student.name = 'Peter';

student.age = 24;

console.log(student.name);//Peter

console.log(student.group);//1
```

Method override

```
function Student(course, group) {
        Person.call(this);
        this.course = course;
        this.group = group;
        this.getAge = function () {
        return this.age + ' лет';
        }
}
var student = new Student(1, 1);
student.age = 20;
console.log(student.getAge()); //20 лет
```

Static members

```
function Person(name, age){
    this.name = name;
    this.age = age;
    this.getName = function(){
        return this.name;
    }
    this.getAge = function(){
        return this.age;
    }
    Person.counter++;
}

Person.counter = 0;
Person.getCount = function(){
    return Person.counter;
}

new Person('Max', 26);
new Person('Yulia', 21);
console.log(Person.getCount());//2
```

Event loop