**Блочная область видимости (block scope)**

ES5: A variable statement declares variables that are created as defined in 10.5. Variables are initialised to**undefined** when created. A variable with an *Initialiser* is assigned the value of its *AssignmentExpression* when the *VariableStatement* is executed, not when the variable is created.

ES6: A let and const declarations define variables that are scoped to the running execution context’s LexicalEnvironment. The variables are created when their containing Lexical Environment is instantiated but may not be accessed in any way until the variable’s LexicalBinding is evaluated. A variable defined by a LexicalBinding with an Initializer is assigned the value of its Initializer’s AssignmentExpression when the LexicalBinding is evaluated, not when the variable is created. If a LexicalBinding in a let declaration does not have an Initializer the variable is assigned the value undefined when the LexicalBinding is evaluated.

В текущей версии JavaScript присутствует функциональная область видимости. Это означает, что все переменные, объявленные c помощью ключевого слова var, будут видны в любом месте функции (даже если они объявлены внутри блока). А let лишь в текущем блоке.

function f(a) {

if (a < 0) { var i = 3; }

console.log(i); // 3

}

В новой версии появится ключевое слово let, которое позволит объявлять переменные с блочной областью видимости:

function f(a) {

if (a < 0) { let i = 3; }

console.log(i); // ReferenceError: i is not defined

}

**Классы**

ES5:

prototype

object that provides shared properties for other objects

NOTE When a constructor creates an object, that object implicitly references the constructor’s “prototype” property for the purpose of resolving property references. The constructor’s “prototype” property can be referenced by the program expression constructor.prototype, and properties added to an object’s prototype are shared, through inheritance, by all objects sharing the prototype. Alternatively, a new object may be created with an explicitly specified prototype by using the Object.create built-in function.

ES6:

14.5 Class Definitions

Syntax

ClassDeclaration[Yield, Default] : class BindingIdentifier[?Yield, ?Default] ClassTail[?Yield]

ClassExpression[Yield,GeneratorParameter] : class BindingIdentifier[?Yield]opt ClassTail[?Yield,?GeneratorParameter]

ClassTail[Yield,GeneratorParameter] : [~GeneratorParameter] ClassHeritage[?Yield]opt { ClassBody[?Yield]opt } [+GeneratorParameter] ClassHeritageopt { ClassBodyopt }

ClassHeritage[Yield] : extends LeftHandSideExpression[?Yield]

ClassBody[Yield] : ClassElementList[?Yield]

ClassElementList[Yield] : ClassElement[?Yield]

ClassElementList[?Yield] ClassElement[?Yield]

В ECMAScript 6 появятся классы:

class Person {

constructor(name) {

this.name = name;

}

describe() {

return "Person called " + this.name;

}

}

Всего того же можно было добиться с помощью прототипов:

// Supertype

function Person(name) {

this.name = name;

}

Person.prototype.describe = function () {

return "Person called " + this.name;

};

// Subtype

function Employee(name, title) {

Person.call(this, name);

this.title = title;

}

Employee.prototype = Object.create(Person.prototype);

Employee.prototype.constructor = Employee;

Employee.prototype.describe = function () {

return Person.prototype.describe.call(this) + " (" + this.title + ")";

};