**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 ключевое слово var которое определяет локальность переменных на уровне функций. (функциональная область видимости) Это означает, что все переменные, объявленные c помощью var, будут видны в любом месте функции (даже если они объявлены внутри блока).

function f(a) {

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

console.log(i); // 3

}

Но этого было явно мало, и подчас приводило к ошибкам, поэтому в новой версии появится ключевое слово let, которое позволит объявлять переменные с блочной областью видимости (блок это всё что угодно внутри фигурных скобок {} ):

Пример 1.

function f(a) {

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

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

}

Пример 2.

var tag = '#wstdays';

if (true) {

let tag = '#404fest';

console.log(tag); // "#404fest"

}

console.log(tag); // "#wstdays"

Так же появится ключевое слово const , которое позволяет определять константы (переменные, которые имеют не могут быть переопределены / read only ).

const a = 10;

a = 15;

// SyntaxError: Assignment to constant variable.

var a = 15;

// SyntaxError: Variable 'a' has already been declared

Область видимости у const такая же как и у let – только внутри блока.

var a = 15;

if (true) {

const a = 10;

console.log(a); // 10

}

console.log(a); // 15

**For-of**

**ES5:**

The Production IterationStatement: for (LeftHandSideExpression in Expression) Statement is evaluated as follows:

1. Let exprRef be the result of evaluating the Expression.
2. Let experValue be [GetValue](http://www.ecma-international.org/ecma-262/5.1/#sec-8.7.1)(exprRef).
3. If experValue is null or undefined, return (normal, empty, empty).
4. Let obj be [ToObject](http://www.ecma-international.org/ecma-262/5.1/" \l "sec-9.9)(experValue).
5. Let V = empty.
6. Repeat
   1. Let P be the name of the next property of objwhose [[Enumerable]] attribute is true. If there is no such property, return (normal, V, empty).
   2. Let lhsRef be the result of evaluating theLeftHandSideExpression (it may be evaluated repeatedly).
   3. Call [PutValue](http://www.ecma-international.org/ecma-262/5.1/#sec-8.7.2)(lhsRef, P).
   4. Let stmt be the result of evaluating Statement.
   5. If stmt.value is not empty, let V = stmt.value.
   6. If stmt.type is break and stmt.target is in the current label set, return (normal, V, empty).
   7. If stmt.type is not continue || stmt.target is not in the current label set, then
      1. If stmt is an [abrupt completion](http://www.ecma-international.org/ecma-262/5.1/#sec-8.9), return stmt.

The production IterationStatement : for (var VariableDeclarationNoIn in Expression ) Statement is evaluated as follows:

1. Let varName be the result of evaluatingVariableDeclarationNoIn.
2. Let exprRef be the result of evaluating the Expression.
3. Let experValue be [GetValue](http://www.ecma-international.org/ecma-262/5.1/" \l "sec-8.7.1)(exprRef).
4. If experValue is null or undefined, return (normal,empty, empty).
5. Let obj be [ToObject](http://www.ecma-international.org/ecma-262/5.1/" \l "sec-9.9)(experValue).
6. Let V = empty.
7. Repeat
   1. Let P be the name of the next property of objwhose [[Enumerable]] attribute is true. If there is no such property, return (normal, V, empty).
   2. Let varRef be the result of evaluating varName as if it were an Identifier [Reference](http://www.ecma-international.org/ecma-262/5.1/#sec-8.7) ([11.1.2](http://www.ecma-international.org/ecma-262/5.1/#sec-11.1.2)); it may be evaluated repeatedly.
   3. Call [PutValue](http://www.ecma-international.org/ecma-262/5.1/" \l "sec-8.7.2)(varRef, P).
   4. Let stmt be the result of evaluating Statement.
   5. If stmt.value is not empty, let V = stmt.value.
   6. If stmt.type is break and stmt.target is in the current label set, return (normal, V, empty).
   7. If stmt.type is not continue || stmt.target is not in the current label set, then
      1. If stmt is an [abrupt completion](http://www.ecma-international.org/ecma-262/5.1/#sec-8.9), return stmt.

The mechanics and order of enumerating the properties (step 6.a in the first algorithm, step 7.a in the second) is not specified. Properties of the object being enumerated may be deleted during enumeration. If a property that has not yet been visited during enumeration is deleted, then it will not be visited. If new properties are added to the object being enumerated during enumeration, the newly added properties are not guaranteed to be visited in the active enumeration. A property name must not be visited more than once in any enumeration.

Enumerating the properties of an object includes enumerating properties of its prototype, and the prototype of the prototype, and so on, recursively; but a property of a prototype is not enumerated if it is “shadowed” because some previous object in the prototype chain has a property with the same name. The values of [[Enumerable]] attributes are not considered when determining if a property of a prototype object is shadowed by a previous object on the prototype chain.

**ES6:**

IterationStatement :

for ( LeftHandSideExpression in Expression ) Statement

for ( LeftHandSideExpression of AssignmentExpression ) Statement

* It is a Syntax Error if LeftHandSideExpression is either an ObjectLiteral or an ArrayLiteral and if the lexical token sequence matched by LeftHandSideExpression cannot be parsed with no tokens left over using AssignmentPattern as the goal symbol.
* If LeftHandSideExpression is either an ObjectLiteral or an ArrayLiteral and if the lexical token sequence matched by LeftHandSideExpression can be parsed with no tokens left over using AssignmentPattern as the goal symbol then the following rules are not applied. Instead, the Early Error rules for AssignmentPattern are used.
* It is a Syntax Error if LeftHandSideExpression is a IdentifierReference that can be statically determined to always resolve to a declarative environment record binding and the resolved binding is an immutable binding.
* It is a Syntax Error if LeftHandSideExpression is neither an ObjectLiteral nor an ArrayLiteral and IsValidSimpleAssignmentTarget of LeftHandSideExpression is false.
* It is a Syntax Error if the LeftHandSideExpression is CoverParenthesizedExpressionAndArrowParameterList : ( Expression ) and Expression derives a production that would produce a Syntax Error according to these rules if that production is substituted for LeftHandSideExpression. This rule is recursively applied.

NOTE The last rule means that the other rules are applied even if parentheses surround Expression.

IterationStatement :

for ( ForDeclaration in Expression ) Statement

for ( ForDeclaration of AssignmentExpression ) Statement

* It is a Syntax Error if the BoundNames of ForDeclaration contains "let".
* It is a Syntax Error if any element of the BoundNames of ForDeclaration also occurs in the VarDeclaredNames of Statement.

Перебор массивов. Проблема в текущем ECMAScript в том, что перебирая массив с помощью for-in, мы получаем не значения элементов массива, а их индексы.

let tags = ['#f8', '#wstdays'];

for (let tag in tags) {

console.log(tag);

}

// 0, 1

Поэтому в ECMAScript 6 появится конструкцию for-of, которая как раз получает значения элементов массива.

let tags = ['#f8', '#wstdays'];

for (let tag of tags) {

console.log(tag);

}

// "#f8", "#wstdays"

Так же он может перебирать элементы коллекций.

let tags = new Set([

'#f8',

'#wstdays',

'#f8'

]);

for (let tag of tags) {

console.log(tag);

}

// "#f8", "#wstdays"

For-of работает также и с Map’ом. Но тут ситуация сложнее, т.к. у map каждое значение состоит из массива, где первый элемент — это ключ, а второй его значение.

let data = new Map();

data.set('f8', {'a': 1, 'b': 0});

data.set('wd', {'a': 0, 'b': 1});

for (let params of data) {

console.log(params);

}

// ["f8", {"a": 1, "b": 0}]

// ["wd", {"a": 0, "b": 1}]