**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

}

**Циклы**

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/#sec-9.9)(*experValue*).
5. Let *V* = empty.
6. Repeat
   1. Let *P* be the name of the next property of *obj* whose [[Enumerable]] attribute is **true**. If there is no such property, return (normal, *V*, empty).
   2. Let *lhsRef* be the result of evaluating the *LeftHandSideExpression* ( 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 evaluating *VariableDeclarationNoIn*.
2. Let *exprRef* be the result of evaluating the *Expression*.
3. Let *experValue* be [GetValue](http://www.ecma-international.org/ecma-262/5.1/#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/#sec-9.9)(*experValue*).
6. Let *V* = empty.
7. Repeat
   1. Let *P* be the name of the next property of *obj* whose [[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/#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*.

Es6:

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 )

perIterationBindings has any elements, then 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 multiple levels of nested parenthesizes surround Expression.

Цикл for-in в JavaScript итерирует по всем полям объекта (включая наследованных). Т.е. итерироваться по значениям массива можно, но опасно:

let arr = [ "blue", "green" ];

arr.notAnIndex = 123;

Array.prototype.protoProp = 456;

for(var x in arr) {

console.log(x); // Напечатает blue, green, notAnIndex, protoProp

}

В ECMAScript 6 появится цикл for-of, который решит данную проблему:

for(var x of arr) {

console.log(x); // Напечатает только blue, green

}