1. Области видимости переменных.

ES5 *(12.2 Variable Statement):*
«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 *(13.2 Declarations and the Variable Statement):*
«A let and const declarations define variables 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 the variable's LexicalBinding is evaluated. A variable definded 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 undefinded when the LexicalBinding is evaluated»

Фактически, была добавлена функциональная область видимости, т.е. все переменные, объявленные с ключевым словом **var** будут видны в любом месте функции. Для объеявления переменных с блочной областью видимости в новую версию было добавлено ключевое слово **let.**

Я считаю, что это было сделано для еще большего ограничения области видимости переменных, и, соответственно, повышения безопасности кода.

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

  console.log(i); // 3
}

f(-1)
```

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

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

f(-1)
```

2. Классы.

ES5 *(4.2.1 Objects)*
«ECMAScript does not use classes such as those in C++, Smalltalk, or Java. Instead objects may be created in various ways including via a literal notation or via constructors which create objects and then execute code that initialises all or part of them by assigning initial values to their properties. Each constructor is a function that has a property named "`prototype`" that is used to implement prototype-based inheritance and shared properties»

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] *ClassHeritage*opt { *ClassBody*opt }

ClassHeritage[Yield] :
    extends *LeftHandSideExpression*[?Yield]

ClassBody[Yield] :
    ClassElementList[?Yield]

ClassElementList[Yield] :
    ClassElement[?Yield]
    ClassElementList[?Yield] *ClassElement*[?Yield]

ClassElement[Yield] :
    MethodDefinition[?Yield]
    static *MethodDefinition*[?Yield]
    ;

NOTE A *ClassBody* is always strict code.

В новой версии ECMAScript добавлена реализация классов в чистом виде, т.е. именно такая какой ее привыкли видеть в других языках программирования. Я считаю, что это необходимо для более удобной работы с классами, вместо длинной цепочки прототипов.

```
class Person {
   constructor(name) {
      this.name = name;
   }

   describe() {
      return "Person called " + this.name;
   }
}

// Subtype
class Employee extends Person {
   constructor(name, title) {
      super.constructor(name);
      this.title = title;
   }

   describe() {
      return super.describe() + " (" + this.title + ")";
   }
}
```

3. Модули.

ES5

В предыдущей версии ECMAScript отсутствовала возможность импорта-экспорта модулей в ядре языка и присутствовала только в виде надстройки пад JS (типа Node.js).

ES6

В новой версии ECMAScript появилась интегрированная в язык реализация модулей с возможностью их импорта и экспорта в код.

Это нужно, я считаю, для оптимизации кода, для возможного сокращения размера надстроек типа Node.js. Создатели, фактически, стараются сделать язык более удобным и оптимизированным для программирования.

```
module Math {
   export function sum(x, y) {
      return x + y;
   }

   export var pi = 3.141593;

   // Не видна снаружи
   function internal() {
      ...
   }
}
```

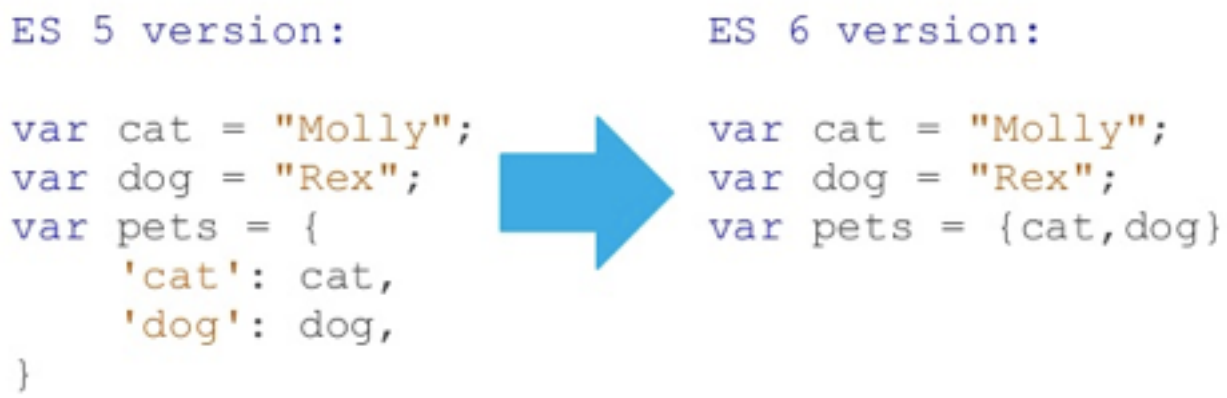
Импортирование модуля:

```
import Math.{sum, pi};
alert("2π = " + sum(pi, pi));
```

Можно использовать *, чтобы импортировать всё:

```
import Math.*;
alert("2π = " + sum(pi, pi));
```

4. Упрощенное определение объектов.



В новой версии ECMAScript была добавлена возможность упрощенной инициализации объектов, позволяющее опускать ключи.

Как и все предыдущие изменения, оно было добавлено для оптимизации, упрощения и ускорения разработки.

Дополнительные источники:

1) http://www.slideshare.net/Solution4Future/javascript-17363650
2) http://frontender.info/es6-modules/
3) http://habrahabr.ru/post/175371/