Есть 3 способа объявления переменных

1. let
2. const
3. var

let и const ведут себя одинаково по отношению к лексическому окружению, области видимости.

**[Для «var» не существует блочной области видимости](https://learn.javascript.ru/var" \l "dlya-var-ne-suschestvuet-blochnoy-oblasti-vidimosti)**

Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом.

function sayHi() {

var phrase = "Привет"; // локальная переменная, "var" вместо "let"

alert(phrase); // Привет

}

sayHi();

alert(phrase); // Ошибка: phrase не определена

Но var выходит за пределы блоков if, for и подобных. за пределами блоков данных конструкций var переменные будут доступны. Это происходит потому, что на заре развития JavaScript блоки кода не имели лексического окружения. Поэтому можно сказать, что var – это пережиток прошлого.

Например:

if (true) {

var test = true; // используем var вместо let

}

alert(test); // true, переменная существует вне блока if

Так как var игнорирует блоки, получается, что test это глобальная переменная.

Если бы мы использовали let test вместо var test, тогда переменная была бы видна только внутри if:

if (true) {

let test = true; // используем let

}

alert(test); // Error: test is not defined

Аналогично для циклов: var не может быть блочной или локальной внутри цикла:

for (var i = 0; i < 10; i++) {

// ...

}

alert(i); // 10, переменная i доступна вне цикла, т.к. является глобальной переменной

Если блок кода находится внутри функции, то var становится локальной переменной в этой функции:

function sayHi() {

if (true) {

var phrase = "Привет";

}

alert(phrase); // срабатывает и выводит "Привет"

}

sayHi();

alert(phrase); // Ошибка: phrase не определена (видна в консоли разработчика)

## [«var» допускает повторное объявление](https://learn.javascript.ru/var" \l "var-dopuskaet-povtornoe-obyavlenie)

Если в блоке кода дважды объявить одну и ту же переменную let, будет ошибка:

let user;

let user; // SyntaxError: 'user' has already been declared

Используя var, можно переобъявлять переменную сколько угодно раз. Повторные var игнорируются:

var user = "Pete";

var user; // ничего не делает, переменная объявлена раньше

// ...нет ошибки

alert(user); // Pete

Если дополнительно присвоить значение, то переменная примет новое значение:

var user = "Pete";

var user = "John";

alert(user); // John

## [«var» обрабатываются в начале запуска функции](https://learn.javascript.ru/var" \l "var-obrabatyvayutsya-v-nachale-zapuska-funktsii)

Объявления переменных var обрабатываются в начале выполнения функции или запуска скрипта, если переменная является глобальной.

Другими словами, переменные var считаются объявленными с самого начала исполнения функции вне зависимости от того, в каком месте функции реально находятся их объявления (при условии, что они не находятся во вложенной функции).

Т.е. этот код:

function sayHi() {

phrase = "Привет";

alert(phrase);

var phrase;

}

sayHi();

…Технически полностью эквивалентен следующему (объявление переменной var phrase перемещено в начало функции):

function sayHi() {

var phrase;

phrase = "Привет";

alert(phrase);

}

sayHi();

…И даже коду ниже т.к. блочная область видимости игнорируется):

function sayHi() {

phrase = "Привет"; // (\*)

if (false) {

var phrase;

}

alert(phrase);

}

sayHi();

Это поведение называется «hoisting» (всплытие, поднятие), потому что все объявления переменных var «всплывают» в самый верх функции.

В примере выше if (false) условие никогда не выполнится. Но это никаким образом не препятствует созданию переменной var phrase, которая находится внутри него, поскольку объявления var «всплывают» в начало функции. Т.е. в момент присвоения значения (\*) переменная уже существует.

**Объявления переменных «всплывают», но присваивания значений – нет.**

Например:

function sayHi() {

alert(phrase); // объявление переменной срабатывает вначале...

var phrase = "Привет";

}

sayHi();

Строка var phrase = "Привет" состоит из двух действий:

1. Объявление переменной var
2. Присвоение значения в переменную =.

Объявление переменной обрабатывается в начале выполнения функции («всплывает»), однако присвоение значения всегда происходит в той строке кода, где оно указано. Т.е. код выполняется по следующему сценарию:

function sayHi() {

var phrase; // объявление переменной срабатывает вначале...

alert(phrase); // undefined

phrase = "Привет"; // ...присвоение - в момент, когда исполнится данная строка кода.

}

sayHi();

Поскольку все объявления переменных var обрабатываются в начале функции, мы можем ссылаться на них в любом месте. Однако, переменные имеют значение undefined до строки с присвоением значения.

В обоих примерах выше вызов alert происходил без ошибки, потому что переменная phrase уже существовала. Но её значение ещё не было присвоено, поэтому мы получали undefined.

## [IIFE](https://learn.javascript.ru/var" \l "iife)

В прошлом, поскольку существовал только var, а он не имел блочной области видимости, программисты придумали способ её эмулировать. Этот способ получил название «Immediately-invoked function expressions» (сокращенно IIFE). (Немедленно вызываемые функциональные выражения)

IIFE выглядит следующим образом:

(function() {

var message = "Hello";

alert(message); // Hello

})();

Здесь создаётся и немедленно вызывается Function Expression. Так что код выполняется сразу же и у него есть свои локальные переменные.

Function Expression обёрнуто в скобки (function {...}), потому что, когда JavaScript встречает "function" в основном потоке кода, он воспринимает это как начало Function Declaration. Но у Function Declaration должно быть имя, так что такой код вызовет ошибку:

// Пробуем объявить и сразу же вызвать функцию

function() { // <-- SyntaxError: Function statements require a function name

var message = "Hello";

alert(message); // Hello

}();

Даже если мы скажем: «хорошо, давайте добавим имя», – это не сработает, потому что JavaScript не позволяет вызывать Function Declaration немедленно.

// ошибка синтаксиса из-за скобок ниже

function go() {

}(); // <-- нельзя вызывать Function Declaration немедленно

Так что скобки вокруг функции – это трюк, который позволяет объяснить JavaScript, что функция была создана в контексте другого выражения, а значит, что это Function Expression: ей не нужно имя и её можно вызвать немедленно.

Помимо круглых скобок существуют и другие способы сообщить JavaScript, что мы имеем в виду Function Expression:

// Способы создания IIFE

(function() {

alert("Круглые скобки вокруг функции");

})();

(function() {

alert("Круглые скобки вокруг всего выражения");

}());

!function() {

alert("Выражение начинается с логического оператора НЕ");

}();

+function() {

alert("Выражение начинается с унарного плюса");

}();

Дело в том, что все переменные и функции, созданные внутри самовызывающейся функции, доступны только внутри неё — это позволяет создавать, например, геттеры и сеттеры без использования ООП.