> Эта страница была переведена с английского языка силами сообщества. Вы тоже можете внести свой вклад, присоединившись к русскоязычному сообществу MDN Web Docs.

let

Директива `let` объявляет переменную с блочной областью видимости с возможностью инициализировать её значением.

Синтаксис

```
let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]];
```

Параметры

`var1`, `var2`, ⋯, `varN`

Имя переменной. Может использоваться любой допустимый идентификатор.

`value1`, `value2`, ⋯, `valueN`

Значение переменной. Любое допустимое выражение.

Описание

Директива `let` позволяет объявить локальную переменную с областью видимости, ограниченной текущим блоком кода . В отличие от ключевого слова `var` (en-US), которое объявляет переменную глобально или локально во всей функции, независимо от области блока.

Объяснение, почему было выбрано название "let" можно найти здесь .

Правила области видимости

Областью видимости переменных, объявленных ключевым словом `let`, является блок, в котором они объявлены, и все его подблоки. В этом работа директива `let` схожа с работой директивы `var`. Основная разница заключается в том, что областью видимости переменной, объявленной директивой `var`, является вся функция, в которой она объявлена:

JS

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2; // та же переменная!
    console.log(x); // 2
  }
  console.log(x); // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2; // другая переменная
    console.log(x); // 2
  }
```

```
  console.log(x); // 1
}
```

Чище код во вложенных функциях

`let` иногда делает код чище при использовании вложенных функций.

JS

```
var list = document.getElementById("list");

for (let i = 1; i <= 5; i++) {
  let item = document.createElement("li");
  item.appendChild(document.createTextNode("Item " + i));

  item.onclick = function (ev) {
    console.log("Item " + i + " is clicked.");
  };
  list.appendChild(item);
}

// чтобы получить такой же эффект с использованием 'var'
// необходимо создать новый контекст
// используя замыкание, чтобы сохранить значение неизменённым
for (var i = 1; i <= 5; i++) {
  var item = document.createElement("li");
  item.appendChild(document.createTextNode("Item " + i));

  (function (i) {
    item.onclick = function (ev) {
      console.log("Item " + i + " is clicked.");
    };
  })(i);
  list.appendChild(item);
}
```

Пример выше будет выполнен как и ожидается, так как пять экземпляров внутренней функции (анонимной) будут ссылаться на пять разных экземпляров переменной `i`. Пример будет выполнен неверно, если заменить директиву `let` на `var`, или удалить переменную `i` из параметров вложенной функции и использовать внешнюю переменную `i` во внутренней функции.

На верхнем уровне скриптов и функций `let, в отличии от var, не создаёт свойства на глобальном объекте`. Например:

JS

```
var x = "global_x";
let y = "global_y";
console.log(this.x); // 'global_x'
console.log(this.y); // undefined
```

В выводе программы будет отображено слово "global_x" для `this.x`, но `undefined` для `this.y`.

Эмуляция приватных членов

При взаимодействии с конструкторами можно использовать выражение `let` чтобы открыть доступ к одному или нескольким приватным членам через использование замыканий:

JS

```
var SomeConstructor;

{
  let privateScope = {};

  SomeConstructor = function SomeConstructor() {
```

```
    this.someProperty = "foo";
    privateScope.hiddenProperty = "bar";
  };

  SomeConstructor.prototype.showPublic = function () {
    console.log(this.someProperty); // foo
  };

  SomeConstructor.prototype.showPrivate = function () {
    console.log(privateScope.hiddenProperty); // bar
  };
}

var myInstance = new SomeConstructor();

myInstance.showPublic();
myInstance.showPrivate();

console.log(privateScope.hiddenProperty); // error
```

Эта техника позволяет получить только "статичное" приватное состояние - в примере выше, все экземпляры полученные из конструктора `SomeConstructor` будут ссылаться на одну и ту же область видимости `privateScope`.

Временные мёртвые зоны и ошибки при использовании `let`

Повторное объявление той же переменной в том же блоке или функции приведёт к выбросу исключения SyntaxError.

JS

```
if (x) {
  let foo;
  let foo; // SyntaxError thrown.
}
```

В стандарте ECMAScript 2015 переменные, объявленные директивой let, переносятся в начало блока. Но если вы сошлётесь в блоке на переменную, до того как она объявлена директивой let, то это приведёт к выбросу исключения ReferenceError (en-US), потому что переменная находится во "временной мёртвой зоне" с начала блока и до места её объявления. (В отличии от переменной, объявленной через `var`, которая просто будет содержать значение `undefined`)

JS

```
function do_something() {
  console.log(bar); // undefined
  console.log(foo); // ReferenceError: foo is not defined
  var bar = 1;
  let foo = 2;
}
```

Вы можете столкнуться с ошибкой в операторах блока switch (en-US), так как он имеет только один подблок.

JS

```
switch (x) {
  case 0:
    let foo;
    break;

  case 1:
    let foo; // Выброс SyntaxError из-за повторного объявления переменной
    break;
}
```

Использование `let` в циклах `for`

Вы можете использовать ключевое слово `let` для привязки переменных к локальной области видимости цикла `for`. Разница с использованием `var` в заголовке цикла `for`, заключается в том, что переменные объявленные `var`, будут видны во всей функции, в которой находится этот цикл.

JS

```
var i = 0;
for (let i = i; i < 10; i++) {
  console.log(i);
}
```

Правила области видимости

JS

```
for (let expr1; expr2; expr3) statement;
```

В этом примере *expr2*, *expr3, statement *заключены в неявный блок, который содержит блок локальных переменных, объявленных конструкцией `let expr1`. Пример приведён выше.

Примеры

let vs var

Когда let используется внутри блока, то область видимости переменной ограничивается этим блоком. Напомним, что отличие заключается в том, что областью видимости переменных, объявленных директивой var, является вся функция, в которой они были объявлены.

JS

```
var a = 5;
var b = 10;

if (a === 5) {
  let a = 4; // The scope is inside the if-block
  var b = 1; // The scope is inside the function

  console.log(a); // 4
  console.log(b); // 1
}

console.log(a); // 5
console.log(b); // 1
```

let в циклах

Вы можете использовать ключевое слово `let` для привязки переменных к локальной области видимости цикла `for`, вместо того что бы использовать глобальные переменные (объявленные с помощью `var`).

JS

```
for (let i = 0; i < 10; i++) {
  console.log(i); // 0, 1, 2, 3, 4 ... 9
}

console.log(i); // i is not defined
```

Нестандартизированные расширения `let`

let блок

> **Предупреждение:** Поддержка `let` блоков была убрана в Gecko 44 Firefox bug 1023609 .

`let` блок предоставляет способ, ассоциировать значения с переменными внутри области видимости этого блока, без влияния на значения переменных с теми же именами вне этого блока.

Синтаксис

```
JS
let (var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]]) block;
```

Описание

`let` блок предоставляет локальную область видимости для переменных. Работа его заключается в привязке нуля или более переменных к области видимости этого блока кода, другими словами, он является блоком операторов. Отметим, что область видимости переменных, объявленных директивой `var` , в блоке **`let`** , будет той же самой, что и если бы эти переменные были объявлены вне блока **`let`** , иными словами областью видимости таких переменных по-прежнему является функция. Скобки в блоке **`let`** являются обязательными. Опускание их приведёт к синтаксической ошибке.

Пример

```
JS
var x = 5;
var y = 0;

let (x = x+10, y = 12) {
  console.log(x+y); // 27
}

console.log(x + y); // 5
```

Правила для этого блока кода аналогичны как и для любого другого блока кода в JavaScript. Он может содержать свои локальные переменные, объявленные `let` .

Правила области видимости

Областью видимости переменных, объявленных директивой `let` , в блоке **`let`** является сам блок и все подблоки в нем, если они не содержат объявлений переменных с теми же именами.

/\/

> **Предупреждение:** Поддержка `let` выражений была убрана в Gecko 41 Firefox bug 1023609 .

`let выражение` позволяет объявить переменные с областью видимости ограниченной одним выражением.

Синтаксис

```
let (var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]]) expression;
```

Пример

Вы можете использовать let для объявления переменных, областью видимости которых является только одно выражение:

```
JS
var a = 5;
let(a = 6) console.log(a); // 6
console.log(a); // 5
```

Правила области видимости

В данном `let` выражении:

```
JS
let (decls) expr
```

* `expr` *оборачивается в неявный блок.

Спецификации

Specification
ECMAScript Language Specification # sec-let-and-const-declarations

Поддержка браузерами

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	
let	Chrome 49	Edge 14	Firefox 44	Opera 17	Safari 10	Chrome 49 Android	Firefox 44 for Android	Opera 18 Android	

Tip: you can click/tap on a cell for more information.

Full support Partial support See implementation notes. Has more compatibility info.

This page was last modified on 28 нояб. 2023 г. by MDN contributors.