Для многократного повторения одного участка кода предусмотрены циклы.

**[Цикл «while»](https://learn.javascript.ru/while-for" \l "tsikl-while)**

while (condition) {

// код

// также называемый "телом цикла"

}

Код из тела цикла выполняется, пока условие condition истинно.

Одно выполнение тела цикла по-научному называется итерация.

Любое выражение или переменная может быть условием цикла, а не только сравнение: условие while вычисляется и преобразуется в логическое значение.

**Фигурные скобки не требуются для тела цикла из одной строки**

Если тело цикла состоит лишь из одной инструкции, мы можем опустить фигурные скобки {…}:

let i = 3;

while (i) alert(i--);

## [Цикл «do…while»](https://learn.javascript.ru/while-for" \l "tsikl-dowhile)

Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

do {

// тело цикла

} while (condition);

Цикл сначала выполнит тело, а затем проверит условие condition, и пока его значение равно true, он будет выполняться снова и снова.

Например:

let i = 0;

do {

alert( i );

i++;

} while (i < 3);

Такая форма синтаксиса оправдана, если вы хотите, чтобы тело цикла выполнилось **хотя бы один раз**, даже если условие окажется ложным.

## [Цикл «for»](https://learn.javascript.ru/while-for" \l "tsikl-for)

Более сложный, но при этом самый распространённый цикл — цикл for.

Выглядит он так:

for (начало; условие; шаг) {

// ... тело цикла ...

}

Давайте разберёмся, что означает каждая часть, на примере. Цикл ниже выполняет alert(i) для i от 0 до (но не включая) 3:

for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2

alert(i);

}

Рассмотрим конструкцию for подробней:

| **часть** |  |  |
| --- | --- | --- |
| начало | let i = 0 | Выполняется один раз при входе в цикл |
| условие | i < 3 | Проверяется *перед* каждой итерацией цикла. Если оно вычислится в false, цикл остановится. |
| тело | alert(i) | Выполняется снова и снова, пока условие вычисляется в true. |
| шаг | i++ | Выполняется *после* тела цикла на каждой итерации *перед* проверкой условия. |

В целом, алгоритм работы цикла выглядит следующим образом:

Выполнить начало

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ ...

То есть, начало выполняется один раз, а затем каждая итерация заключается в проверке условия, после которой выполняется тело и шаг.

**Встроенное объявление переменной**

В примере переменная счётчика i была объявлена прямо в цикле. Это так называемое «встроенное» объявление переменной. Такие переменные существуют только внутри цикла.

Вместо объявления новой переменной мы можем использовать уже существующую:

let i = 0;

for (i = 0; i < 3; i++) { // используем существующую переменную

alert(i); // 0, 1, 2

}

alert(i); // 3, переменная доступна, т.к. была объявлена снаружи цикла

### [Пропуск частей «for»](https://learn.javascript.ru/while-for" \l "propusk-chastey-for)

Любая часть for может быть пропущена.

Для примера, мы можем пропустить начало если нам ничего не нужно делать перед стартом цикла.

Вот так:

let i = 0; // мы уже имеем объявленную i с присвоенным значением

for (; i < 3; i++) { // нет необходимости в "начале"

alert( i ); // 0, 1, 2

}

Можно убрать и шаг:

let i = 0;

for (; i < 3;) {

alert( i++ );

}

Это сделает цикл аналогичным while (i < 3).

А можно и вообще убрать всё, получив бесконечный цикл:

for (;;) {

// будет выполняться вечно

}

При этом сами точки с запятой ; обязательно должны присутствовать, иначе будет ошибка синтаксиса.

## [Прерывание цикла: «break»](https://learn.javascript.ru/while-for" \l "preryvanie-tsikla-break)

Обычно цикл завершается при вычислении условия в false.

Но мы можем выйти из цикла в любой момент с помощью специальной директивы break.

Директива break в строке (\*) полностью прекращает выполнение цикла и передаёт управление на строку за его телом, то есть на alert.

Вообще, сочетание «бесконечный цикл + break» – отличная штука для тех ситуаций, когда условие, по которому нужно прерваться, находится не в начале или конце цикла, а посередине или даже в нескольких местах его тела.

## [Переход к следующей итерации: continue](https://learn.javascript.ru/while-for" \l "continue)

Директива continue – «облегчённая версия» break. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно true).

Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

Например, цикл ниже использует continue, чтобы выводить только нечётные значения:

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

// если true, пропустить оставшуюся часть тела цикла

if (i % 2 == 0) continue;

alert(i); // 1, затем 3, 5, 7, 9

}

**Директива continue позволяет избегать вложенности**

Цикл, который обрабатывает только нечётные значения, мог бы выглядеть так:

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

if (i % 2) {

alert( i );

}

}

С технической точки зрения он полностью идентичен. Действительно, вместо continue можно просто завернуть действия в блок if.

Однако мы получили дополнительный уровень вложенности фигурных скобок. Если код внутри if более длинный, то это ухудшает читаемость, в отличие от варианта с continue.

**Нельзя использовать break/continue справа от оператора „?“**

Обратите внимание, что эти синтаксические конструкции не являются выражениями и не могут быть использованы с тернарным оператором ?. В частности, использование таких директив, как break/continue, вызовет ошибку.

Например, если мы возьмём этот код:

if (i > 5) {

alert(i);

} else {

continue;

}

…и перепишем его, используя вопросительный знак:

(i > 5) ? alert(i) : continue; // continue здесь приведёт к ошибке

…то будет синтаксическая ошибка.

Это ещё один повод не использовать оператор вопросительного знака ? вместо if.

## [Метки для break/continue](https://learn.javascript.ru/while-for" \l "metki-dlya-break-continue)

Бывает, нужно выйти одновременно из нескольких уровней цикла сразу.

Обычный break лишь прервёт внутренний цикл

Достичь желаемого поведения можно с помощью меток.

Метка имеет вид идентификатора с двоеточием перед циклом:

labelName: for (...) {

...

}

Вызов break <labelName> в цикле ниже ищет ближайший внешний цикл с такой меткой и переходит в его конец.

outer: for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

if (…) break outer;

}

}

В примере выше это означает, что вызовом break outer будет разорван внешний цикл до метки с именем outer.

Таким образом управление перейдёт со строки, помеченной (\*), к alert('Готово!').

Можно размещать метку на отдельной строке:

outer:

for (let i = 0; i < 3; i++) { ... }

Директива continue также может быть использована с меткой. В этом случае управление перейдёт на следующую итерацию цикла с меткой.

**Метки не позволяют «прыгнуть» куда угодно**

Метки не дают возможности передавать управление в произвольное место кода.

Например, нет возможности сделать следующее:

break label; // не прыгает к метке ниже

label: for (...)

Директива break должна находиться внутри блока кода. Технически, подойдет любой маркированный блок кода, например:

label: {

// ...

break label; // работает

// ...

}

…Хотя в 99.9% случаев break используется внутри циклов, как мы видели в примерах выше.

К слову, continue возможно только внутри цикла.

## [Итого](https://learn.javascript.ru/while-for" \l "itogo)

Мы рассмотрели 3 вида циклов:

* while – Проверяет условие перед каждой итерацией.
* do..while – Проверяет условие после каждой итерации.
* for (;;) – Проверяет условие