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

Циклы и итерации

Циклы - простой способ сделать какое-то действие несколько раз. Эта глава руководства JavaScript Guide познакомит вас с различными операторами доступными в JavaScript.

Вы можете представить цикл в виде компьютеризированной версии игры, где вы говорите кому-то сделать X шагов в одном направлении, затем Y шагов в другом; для примера, идея игры "Иди 5 шагов на восток" может быть выражена в виде цикла:

```JS
var step;
for (step = 0; step < 5; step++) {
  // Запускается 5 раз, с шагом от 0 до 4.
  console.log("Идём 1 шаг на восток");
}
```

Существует множество различных видов циклов, но все они по сути делают тоже самое: повторяют какое-либо действие несколько раз (не забывайте про нулевой раз повторения, отсчёт в массиве начинается с 0). Различные по строению циклы предлагают разные способы для определения начала и окончания цикла. Для различных задач программирования существуют свои операторы цикла, с помощью которых они решаются намного проще.

Операторы предназначенные для организации циклов в JavaScript:

- Цикл for
- Цикл do...while
- Цикл while
- Метка (label)
- break
- continue
- for...in
- for...of

Цикл for

Цикл **for** повторяет действия, пока не произойдёт какое-либо специальное событие

М _

```
for ([начало]; [условие]; [шаг]) выражения
```

При его выполнении происходит следующее:

1. Выполняется выражение `начало`, если оно указано. Это выражение обычно инициализирует один или несколько счётчиков, но синтаксис позволяет выражению быть любой сложности. Также используется для объявления переменных.
2. Выполняется `условие`. Если `условие` истинно, то выполняются `выражения`. Если оно ложно, цикл `for` прерывается. Если же `условие` полностью пропущено, то оно считается истинным.
3. Выполняются `выражения`. Чтобы выполнить несколько выражений, используются блок-выражение `{ ... }` для группировки выражений.
4. Обновляется `шаг`, если он есть, а затем управление возвращается к шагу 2.

Пример

В следующей функции есть цикл `for`, который считает количество выбранных жанров в списке прокрутки (элемент `<select>`, который позволяет выбрать несколько элементов). Цикл `for` объявляет переменную `i` и задаёт ей значение 0. Также он

проверяет, что `i` меньше количества элементов в элементе `<select>`, выполняет оператор `if` и увеличивает `i` на один после каждого прохода цикла.

HTML

```
<form name="selectForm">
  <p>
    <label for="musicTypes"
     >Выберите некоторые жанры музыки, а затем нажмите на кнопку ниже:</label
    >
    <select id="musicTypes" name="musicTypes" multiple="multiple">
      <option selected="selected">R&B</option>
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classical</option>
      <option>Opera</option>
    </select>
  </p>
  <p><input id="btn" type="button" value="Как много выбрано?" /></p>
</form>

<script>
  function howMany(selectObject) {
    var numberSelected = 0;
    for (var i = 0; i < selectObject.options.length; i++) {
      if (selectObject.options[i].selected) {
        numberSelected++;
      }
    }
    return numberSelected;
  }

  var btn = document.getElementById("btn");
  btn.addEventListener("click", function () {
    alert("Выбрано элементов: " + howMany(document.selectForm.musicTypes));
  });
</script>
```

Цикл do...while

Цикл [do...while](#) повторяется пока заданное условие истинно. Оператор `do...while` имеет вид:

```
do
  выражения
while (условие);
```

`выражения` выполняются пока `условие` истинно. Чтобы использовать несколько выражений, используйте блок-выражение `{ ... }`, чтобы сгруппировать их. Если `условие` истинно, `выражения` выполнятся снова. В конце каждого прохода `условие` проверяется. Если `условие` ложно, выполнение приостанавливается и управление передаётся выражению после `do...while`.

Пример

В следующем примере, цикл `do` выполнится минимум 1 раз и запускается снова, пока `i` меньше 5.

```
JS
```

```
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

Цикл while

Цикл `while` выполняет выражения пока условие истинно. Выглядит он так:

```
while (условие)
  выражения
```

Если `условие` становится ложным, выражения в цикле перестают выполняться и управление переходит к выражению после цикла.

`Условие` проверяется на истинность до того, как выполняются `выражения` в цикле. Если `условие` истинно, выполняются `выражения`, а затем условие проверяется снова. Если `условие` ложно, выполнение приостанавливается и управление переходит к выражению после `while`.

Чтобы использовать несколько выражений, используйте блок выражение `{ ... }`, чтобы сгруппировать их.

Пример 1

Следующий цикл `while` работает, пока `n` меньше трёх:

JS

```
var n = 0;
var x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

С каждой итерацией, цикл увеличивает `n` и добавляет это значение к `x`. Поэтому, `x` и `n` получают следующие значения:

- После первого прохода: `n = 1` и `x = 1`
- После второго: `n = 2` и `x = 3`
- После третьего прохода: `n = 3` и `x = 6`

После третьего прохода, условие `n < 3` становится ложным, поэтому цикл прерывается.

Пример 2

Избегайте бесконечных циклов. Убедитесь, что условие цикла в итоге станет ложным; иначе, цикл никогда не прервётся. Выражения в следующем цикле `while` будут выполняться вечно, т.к. условие никогда не станет ложным:

JS

```
while (true) {
  console.log("Hello, world");
}
```

Метка (label)

Метка представляет собой оператор с идентификатором, который позволяет вам ссылаться на какое-то место в вашей программе. Например, вы можете использовать метку, чтобы обозначить цикл, а затем использовать операторы `break` или `continue`, чтобы указать, должна ли программа прерывать цикл или продолжать его выполнение.

Синтаксис метки следующий:

```
метка :
   оператор
```

Значение *метки* может быть любым корректным JavaScript идентификатором, не являющимся зарезервированным словом. `Оператор`, указанный вами после метки может быть любым выражением.

Пример

В этом примере, метка `markLoop` обозначает цикл `while`.

```
JS
markLoop: while (theMark == true) {
  doSomething();
}
```

break

Используйте оператор `break`, чтобы прерывать цикл, переключать управление или в сочетании с оператором метка.

- Когда вы используете `break` без метки, он прерывает циклы `while`, `do-while` и `for` или сразу переключает управление к следующему выражению.
- Когда вы используете `break` с меткой, он прерывает специально отмеченное выражение.

Синтаксис оператора может быть таким:

1. `break;`
2. `break Метка;`

Первая форма синтаксиса прерывает цикл совсем или переключает управление; вторая прерывает специально обозначенное выражение.

Пример 1

Следующий пример проходит по элементам в массиве, пока не найдёт элемент, чьё значение - `theValue` :

```
JS
```

```
for (i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

Пример 2: Прерывание метки

```
JS
```

```
var x = 0;
var z = 0;
labelCancelLoops: while (true) {
  console.log("Внешний цикл: " + x);
  x += 1;
  z = 1;
  while (true) {
    console.log("Внутренний цикл: " + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelCancelLoops;
    } else if (z === 10) {
      break;
    }
  }
}
```

continue

Оператор `continue` используется, чтобы шагнуть на шаг вперёд в циклах `while`, `do-while`, `for` или перейти к метке.

- Когда вы используете `continue` без метки, он прерывает текущую итерацию циклов `while`, `do-while` и `for` и продолжает выполнение цикла со следующей итерации. В отличие от `break`, `continue` не прерывает выполнение цикла полностью. В цикле `while` он прыгает к условию. А в `for` увеличивает `шаг`.
- Когда вы используете `continue` с меткой, он применяется к циклу с этой меткой.

Синтаксис `continue` может выглядеть так:

1. `continue;`
2. `continue Метка ;`

Пример 1

Следующий пример показывает цикл `while` с оператором `continue`, который срабатывает, когда значение `i` равно 3. Таким образом, `n` получает значения 1, 3, 7 и 12.

JS

```
var i = 0;
var n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
}
```

Пример 2

Выражение, отмеченное `checkiandj` содержит выражение отмеченное `checkj`. При встрече с `continue`, программа прерывает текущую итерацию `checkj` и начинает следующую итерацию. Каждый раз при встрече с `continue`, `checkj` переходит на следующую итерацию, пока условие возвращает `false`. Когда возвращается `false`,

после вычисления остатка от деления `checkiandj`, `checkiandj` переходит на следующую итерацию, пока его условие возвращает `false`. Когда возвращается `false`, программа продолжает выполнение с выражения после `checkiandj`.

Если у `continue` проставлена метка `checkiandj`, программа может продолжиться с начала метки `checkiandj`.

JS

```
checkiandj: while (i < 4) {
  console.log(i);
  i += 1;
  checkj: while (j > 4) {
    console.log(j);
    j -= 1;
    if (j % 2 != 0) {
      continue checkj;
    }
    console.log(j + " чётное.");
  }
  console.log("i = " + i);
  console.log("j = " + j);
}
```

for...in

Оператор `for...in` проходит по всем перечислимым свойствам объекта. JavaScript выполнит указанные выражения для каждого отдельного свойства. Цикл `for...in` выглядит так:

```
for (variable in object) {
  выражения
}
```

Пример

Следующая функция берёт своим аргументом объект и его имя. Затем проходит по всем свойствам объекта и возвращает строку, которая содержит имена свойств и их значения.

JS

```
function dump_props(obj, obj_name) {
  var result = "";
  for (var i in obj) {
    result += obj_name + "." + i + " = " + obj[i] + "<br>";
  }
  result += "<hr>";
  return result;
}
```

Для объекта `car` со свойствами `make` и `model`, *результатом* будет:

JS

```
car.make = Ford;
car.model = Mustang;
```

Пример №2

Также, по ключу можно выводить значение:

```
let obj = {model: 'AUDI A8', year: '2019', color: 'brown'}

for (key in obj) {
  console.log(`${key} = ${obj[key]}`);
}
// model = AUDI A8
// year = 2019
// color = brown
```

Массивы

Хотя, очень заманчиво использовать `for...in` как способ пройтись по всем элементам `Array`, этот оператор возвращает имя свойств определённых пользователем помимо числовых индексов. Таким образом лучше использовать стандартный `for` для числовых индексов при взаимодействии с массивами, поскольку оператор `for...in` проходит по определённым пользователем свойствам в дополнение к элементам массива, если вы изменяете массив, например, добавляете свойства и методы.

Пример

```
let arr = ['AUDI A8', '2019', 'brown'];
arr.cost = '$100.000';

for (key in arr) {
  console.log(`${key} = ${arr[key]}`);
}

// 0 = AUDI A8
// 1 = 2019
// 2 = brown
// cost = $100.000
```

for...of

Оператор `for...of` создаёт цикл, проходящий по перечислимым объектам (en-US) (включая `Array`, `Map` (en-US), `Set`, объект `arguments` (en-US) и так далее), вызывая на каждой итерации функцию с выражениями, которые надо выполнить для получения значения каждого отдельного свойства.

```
for (variable of object) {
  выражения
}
```

Следующий пример показывает разницу между циклами `for...of` и `for...in`. Тогда как `for...in` проходит по именам свойств, `for...of` проходит по значениям свойств:

JS

```
let arr = [3, 5, 7];
arr.foo = "hello";

for (let i in arr) {
  console.log(i); // выводит "0", "1", "2", "foo"
}

for (let i of arr) {
  console.log(i); // выводит "3", "5", "7"
}
```

This page was last modified on 7 авг. 2023 г. by MDN contributors.