JavaScript - условия и циклы

взаимодействие, условия “if/else”, конструкция “switch”, циклы while и for

Взаимодействие: **alert, prompt, confirm**

alert
Она показывает сообщение и ждет, пока пользователь нажмет кнопку «ОК».

```
alert("Hello");
```

prompt
Этот код отобразит модальное окно с текстом, полем для ввода текста и кнопками OK/Отмена.

```
let age = prompt('Сколько тебе лет?', 100);

alert(`Тебе ${age} лет!`); // Тебе 100 лет!
```

confirm
Функция confirm отображает модальное окно с текстом вопроса question и двумя кнопками: OK и Отмена.
Результат – true, если нажата кнопка OK. В других случаях – false.

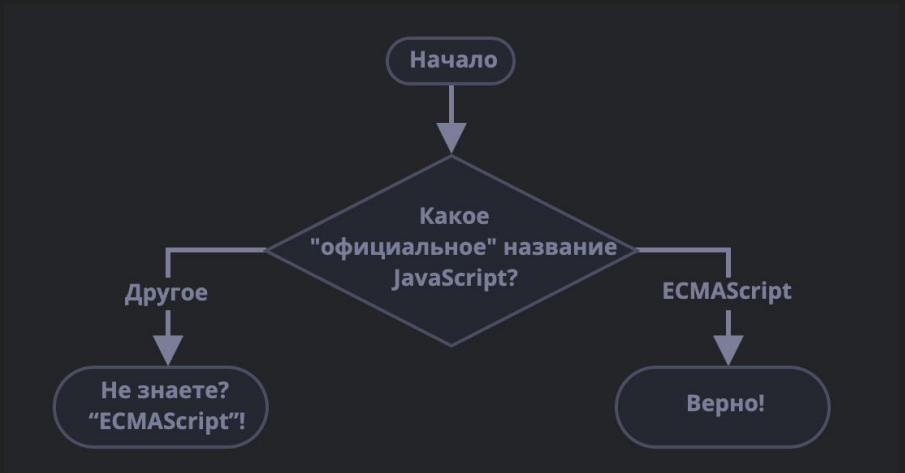
```
let isBoss = confirm("Ты здесь главный?");

alert( isBoss ); // true, если нажата OK
```

Условное ветвление: **if, '?'**

Иногда нам нужно выполнить различные действия в зависимости от условий.

Для этого мы можем использовать инструкцию if и условный оператор ?, который также называют оператором «вопросительный знак».

- Инструкция «if»
- Блок «else» и несколько условий: «else if»
- Условный оператор „?“




```
let value = prompt('Какое "официальное" название JavaScript?', '');

if (value == 'ECMAScript') {
  alert('Верно!');
} else {
  alert('Не знаете? ECMAScript!');
}
```

Инструкция **«if»**

Инструкция if(...) вычисляет условие в скобках и, если результат true, то выполняется блок кода.

Преобразование к логическому типу
Инструкция if (…) вычисляет выражение в скобках и преобразует результат к логическому типу.

- Число 0, пустая строка "", null, undefined и NaN становятся false. Из-за этого их называют «ложными» («falsy») значениями.
- Остальные значения становятся true, поэтому их называют «правдивыми» («truthy»).

```
let year = prompt('В каком году была опубликована спецификация ECMAScript

if (year == 2015) alert( 'Вы правы!' );
```

```
if (year == 2015) {
  alert( "Правильно!" );
  alert( "Вы такой умный!" );
}
```

```
let condition = (year == 2015); // преобразуется к true или false

if (condition) {
  ...
}
```

«else» и «else if»

Блок «else»

Инструкция if может содержать необязательный блок «else» («иначе»). Он выполняется, когда условие ложно.

Несколько условий: «else if»

Иногда нужно проверить несколько вариантов условия. Для этого используется блок else if.

```
let year = prompt('В каком году была опубликована спецификация ECMAScript

if (year == 2015) {
  alert( 'Да вы знаток!' );
} else {
  alert( 'А вот и неправильно!' ); // любое значение, кроме 2015
}
```

```
let year = prompt('В каком году была опубликована спецификация ECMAScript

if (year < 2015) {
  alert( 'Это слишком рано...' );
} else if (year > 2015) {
  alert( 'Это поздновато' );
} else {
  alert( 'Верно!' );
}
```

Условный оператор „?“

Иногда нам нужно определить переменную в зависимости от условия.
Так называемый «условный» оператор «вопросительный знак» позволяет нам сделать это более коротким и простым способом.

Оператор представлен знаком вопроса ?. Его также называют «тернарный», так как этот оператор, единственный в своем роде, имеет три аргумента.

let result = условие ? значение1 : значение2;

Сначала вычисляется условие: если оно истинно, тогда возвращается значение1, в противном случае – значение2.

Смысл оператора «вопросительный знак» ? – вернуть то или иное значение, в зависимости от условия. Пожалуйста, используйте его именно для этого. Когда вам нужно выполнить разные ветви кода – используйте if.

```
let accessAllowed;
let age = prompt('Сколько вам лет?', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);
```

```
let accessAllowed = (age > 18) ? true : false;
```

Конструкция **"switch"**

Конструкция switch заменяет собой сразу несколько if. Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

Синтаксис
Конструкция switch имеет один или более блок case и необязательный блок default.

- Переменная x проверяется на строгое равенство первому значению value1, затем второму value2 и так далее.
- Если соответствие установлено – switch начинает выполняться от соответствующей директивы case и далее, до ближайшего break (или до конца switch).
- Если ни один case не совпал – выполняется (если есть) вариант default.

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
    [break]
}
```

Особенности **"switch"**

Группировка «case»

Несколько вариантов case, использующих один код, можно группировать.

Для примера, выполним один и тот же код для case 3 и case 5, сгруппировав их.

Тип имеет значение

Нужно отметить, что проверка на равенство всегда строгая. Значения должны быть одного типа, чтобы выполнялось равенство.

```
let a = 3;

switch (a) {
  case 4:
    alert('Правильно!');
    break;

  case 3: // (*) группируем оба case
  case 5:
    alert('Неправильно!');
    alert("Может вам посетить урок математики?");
    break;

  default:
    alert('Результат выглядит странновато. Честно.');
}
```

```
let arg = prompt("Введите число?");
switch (arg) {
  case '0':
  case '1':
    alert( 'Один или ноль' );
    break;

  case '2':
    alert( 'Два' );
    break;

  case 3:
    alert( 'Никогда не выполнится!' );
    break;
  default:
    alert( 'Неизвестное значение' );
}
```

Циклы

- Цикл **«while»** – Проверяет условие перед каждой итерацией.
- Цикл **«do…while»** – Проверяет условие после каждой итерации.
- Цикл **«for»** – Проверяет условие перед каждой итерацией, есть возможность задать дополнительные настройки.
- **break**, **continue** и **метки**

При написании скриптов зачастую встает задача сделать однотипное действие много раз.

Например, вывести товары из списка один за другим. Или просто перебрать все числа от 1 до 10 и для каждого выполнить одинаковый код.

Для многократного повторения одного участка кода предусмотрены циклы.

Одно выполнение тела цикла по-научному называется ***итерация***.

Цикл **«while»**

Код из тела цикла выполняется, пока условие condition истинно.

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

```
while (condition) {
  // код
  // также называемый "телом цикла"
}
```

```
let i = 0;
while (i < 3) { // выводит 0, затем 1, затем 2
  alert( i );
  i++;
}
```

Цикл **«do…while»**

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

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

Такая форма синтаксиса оправдана, если вы хотите, чтобы тело цикла выполнилось хотя бы один раз, даже если условие окажется ложным. На практике чаще используется форма с предусловием: while(…) {…}.

```
do {
  // тело цикла
} while (condition);
```

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Цикл «for»

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

```
for (начало; условие; шаг) {
  // ... тело цикла ...
}
```

Встроенное объявление переменной:
Вместо объявления новой переменной мы можем использовать уже существующую:

```
for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // ошибка, нет такой переменной
```

```
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
```

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)
→ ...
```

Пропуск частей «for»

Любая часть for может быть пропущена.

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

Можно убрать и шаг: Это сделает цикл аналогичным while.

А можно и вообще убрать всё, получив бесконечный цикл.

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

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

for (; i < 3; i++) { // нет необходимости в "начале"
  alert( i ); // 0, 1, 2
}
```

```
let i = 0;

for (; i < 3;) {
  alert( i++ );
}
```

```
for (;;) {
  // будет выполняться вечно
}
```

break, continue и метки

- Прерывание цикла: «**break**»
- Переход к следующей итерации: **continue**
- **Метки** для break/continue

Прерывание цикла: **«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 передает 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.

Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока посетитель их вводит, а затем – выдаёт:

```
let sum = 0;

while (true) {

  let value = +prompt("Введите число", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Сумма: ' + sum );
```

Переход к следующей итерации: **continue**

Директива continue – «облегчённая версия» break. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все еще равно true).
Ее используют, если понятно, что на текущем повторе цикла делать больше нечего.

Директива continue позволяет избегать вложенности
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.

Например, цикл ниже использует continue, чтобы выводить только нечетные значения:
Для четных значений i, директива continue прекращает выполнение тела цикла и передает управление на следующую итерацию for (со следующим числом). Таким образом alert вызывается только для нечётных значений.

```
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
}
```

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

  if (i % 2) {
    alert( i );
  }

}
```

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.

```
if (i > 5) {
  alert(i);
} else {
  continue;
}
```

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

Метки для break/continue

```
labelName: for (...) {
  ...
}
```

Бывает, нужно выйти одновременно из нескольких уровней цикла сразу.

Нам нужен способ остановить выполнение, если пользователь отменит ввод.

Обычный break после input лишь прервет внутренний цикл, но этого недостаточно. Достичь желаемого поведения можно с помощью меток.

Метка имеет вид идентификатора с двоеточием перед циклом:

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++) {

    let input = prompt(`Значение на координатах (${i},${j})`, '');

    // если пустая строка или Отмена, то выйти из обоих циклов
    if (!input) break outer; // (*)

    // сделать что-нибудь со значениями...
  }
}

alert('Готово!');
```

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

Метки не позволяют «прыгнуть» куда угодно

```
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.

Ресурсы

Взаимодействие: alert, prompt, confirm - ТЫК

Условное ветвление: if, '?' - ТЫК

Конструкция "switch" - ТЫК

Циклы - ТЫК