Условные операторы. Циклы

Оглавление

[Операторы и операнды 0](#_Toc447573385)

[Приоритет операций 1](#_Toc447573386)

[Побитовые операторы 6](#_Toc447573387)

[Используемая литература 7](#_Toc447573388)

# Условные операторы

Иногда, в зависимости от условия, нужно выполнить различные действия. Для этого используется оператор if.

Например:

|  |
| --- |
| var year = prompt('В каком году появилась спецификация ECMA-262 5.1?', '');  if (year != 2011) alert( 'А вот и неправильно!' ); |

Оператор if («если») получает условие, в примере выше это year != 2011. Он вычисляет его, и если результат – true, то выполняет команду.

Если нужно выполнить более одной команды – они оформляются блоком кода в фигурных скобках:

|  |
| --- |
| if (year != 2011) {  alert( 'А вот..' );  alert( '..и неправильно!' );  } |

Оператор if (...) вычисляет и преобразует выражение в скобках к логическому типу.

В логическом контексте:

Число 0, пустая строка "", null и undefined, а также NaN являются false,

Остальные значения – true.

# Тернарный оператор

Иногда нужно в зависимости от условия присвоить переменную. Например:

|  |
| --- |
| var access;  var age = prompt('Сколько вам лет?', '');  if (age > 14) {  access = true;  } else {  access = false;  }  alert(access); |

Оператор вопросительный знак '?' позволяет делать это короче и проще.

Он состоит из трех частей:

|  |
| --- |
| условие ? значение1 : значение2 |

Проверяется условие, затем если оно верно – возвращается значение1, если неверно – значение2, например:

|  |
| --- |
| access = (age > 14) ? true : false; |

Оператор '?' выполняется позже большинства других, в частности – позже сравнений, поэтому скобки можно не ставить, но когда скобки есть – код лучше читается. Так что рекомендуется их писать.

# Оператор множественного выбора

Конструкция switch заменяет собой сразу несколько if.

Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

|  |
| --- |
| switch(x) {  case 'value1': // if (x === 'value1')  ...  [break]  case 'value2': // if (x === 'value2')  ...  [break]  default:  ...  [break]  } |

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.

При этом case называют вариантами switch.

# Логические операторы

Для операций над логическими значениями в JavaScript есть || (ИЛИ), && (И) и ! (НЕ).

Хоть они и называются «логическими», но в JavaScript могут применяться к значениям любого типа и возвращают также значения любого типа.

**|| (ИЛИ)**

Оператор ИЛИ выглядит как двойной символ вертикальной черты:

|  |
| --- |
| result = a || b; |

Логическое ИЛИ в классическом программировании работает следующим образом: "если хотя бы один из аргументов true, то возвращает true, иначе – false". В JavaScript, как мы увидим далее, это не совсем так, но для начала рассмотрим только логические значения.

Получается следующая «таблица результатов»:

|  |
| --- |
| alert( true || true ); // true  alert( false || true ); // true  alert( true || false ); // true  alert( false || false ); // false |

Если значение не логического типа – то оно к нему приводится в целях вычислений. Например, число 1 будет воспринято как true, а 0 – как false.

Обычно оператор ИЛИ используется в if, чтобы проверить, выполняется ли хотя бы одно из условий, например:

|  |
| --- |
| var hour = 9;  if (hour < 10 || hour > 18) {  alert( 'Офис до 10 или после 18 закрыт' );  } |

JavaScript вычисляет несколько ИЛИ слева направо. При этом, чтобы экономить ресурсы, используется так называемый «короткий цикл вычисления».

Допустим, вычисляются несколько ИЛИ подряд: a || b || c || .... Если первый аргумент – true, то результат заведомо будет true (хотя бы одно из значений – true), и остальные значения игнорируются.

Это особенно заметно, когда выражение, переданное в качестве второго аргумента, имеет сторонний эффект – например, присваивает переменную.

При запуске примера ниже присвоение x не произойдёт:

|  |
| --- |
| var x;  true || (x = 1);  alert(x); // undefined, x не присвоен |

А в примере ниже первый аргумент – false, так что ИЛИ попытается вычислить второй, запустив тем самым присваивание:

|  |
| --- |
| var x;  false || (x = 1);  alert(x); // 1 |

**&& (И)**

Оператор И пишется как два амперсанда &&:

|  |
| --- |
| result = a && b; |

В классическом программировании И возвращает true, если оба аргумента истинны, а иначе – false:

|  |
| --- |
| alert( true && true ); // true  alert( false && true ); // false  alert( true && false ); // false  alert( false && false ); // false |

К И применим тот же принцип «короткого цикла вычислений», но немного по-другому, чем к ИЛИ.

Если левый аргумент – false, оператор И возвращает его и заканчивает вычисления. Иначе – вычисляет и возвращает правый аргумент.

Например:

|  |
| --- |
| // Первый аргумент - true,  // Поэтому возвращается второй аргумент  alert( 1 && 0 ); // 0  alert( 1 && 5 ); // 5  // Первый аргумент - false,  // Он и возвращается, а второй аргумент игнорируется  alert( null && 5 ); // null  alert( 0 && "не важно" ); // 0 |

Можно передать и несколько значений подряд, при этом возвратится первое «ложное» (на котором остановились вычисления), а если его нет – то последнее:

|  |
| --- |
| alert( 1 && 2 && null && 3 ); // null  alert( 1 && 2 && 3 ); // 3 |

Итак, оператор && вычисляет операнды слева направо до первого «ложного» и возвращает его, а если все истинные – то последнее значение.

Иначе можно сказать, что "&& запинается на лжи".

**! (НЕ)**

Оператор НЕ – самый простой. Он получает один аргумент. Синтаксис:

|  |
| --- |
| var result = !value; |

Действия !:

Сначала приводит аргумент к логическому типу true/false.

Затем возвращает противоположное значение.

Например:

|  |
| --- |
| alert( !true ); // false  alert( !0 ); // true |

В частности, двойное НЕ используют для преобразования значений к логическому типу:

|  |
| --- |
| alert( !!"строка" ); // true  alert( !!null ); // false |

# Циклы

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.

**Цикл while**

Цикл while имеет вид:

|  |
| --- |
| while (условие) {  // код, тело цикла  } |

Пока условие верно – выполняется код из тела цикла.

Например, цикл ниже выводит i пока i < 3:

|  |
| --- |
| var i = 0;  while (i < 3) {  alert( i );  i++;  } |

Повторение цикла по-научному называется «итерация». Цикл в примере выше совершает три итерации.

Если бы i++ в коде выше не было, то цикл выполнялся бы (в теории) вечно. На практике, браузер выведет сообщение о «зависшем» скрипте и посетитель его остановит.

Бесконечный цикл можно сделать и проще:

|  |
| --- |
| while (true) {  // ...  } |

Условие в скобках интерпретируется как логическое значение, поэтому вместо while (i!=0) обычно пишут while (i):

|  |
| --- |
| var i = 3;  while (i) { // при i, равном 0, значение в скобках будет false и цикл остановится  alert( i );  i--;  } |

**Цикл do…while**

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

|  |
| --- |
| do {  // тело цикла  } while (условие); |

Цикл, описанный, таким образом, сначала выполняет тело, а затем проверяет условие.

Например:

|  |
| --- |
| var i = 0;  do {  alert( i );  i++;  } while (i < 3); |

Синтаксис do..while редко используется, т.к. обычный while нагляднее – в нём не приходится искать глазами условие и ломать голову, почему оно проверяется именно в конце.

**Цикл for**

Чаще всего применяется цикл for. Выглядит он так:

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

Пример цикла, который выполняет alert(i) для i от 0 до 2 включительно (до 3):

|  |
| --- |
| var i;  for (i = 0; i < 3; i++) {  alert( i );  } |

Здесь:

* Начало: i=0.
* Условие: i<3.
* Шаг: i++.
* Тело: alert(i), т.е. код внутри фигурных скобок (они не обязательны, если только одна операция)

Цикл выполняется так:

1. Начало: i=0 выполняется один-единственный раз, при заходе в цикл.
2. Условие: i<3 проверяется перед каждой итерацией и при входе в цикл, если оно нарушено, то происходит выход.
3. Тело: alert(i).
4. Шаг: i++ выполняется после тела на каждой итерации, но перед проверкой условия.
5. Идти на шаг 2.

Иными словами, поток выполнения: начало → (если условие → тело → шаг) → (если условие → тело → шаг) → … и так далее, пока верно условие.

**Пропуск частей for**

Любая часть for может быть пропущена.

Например, можно убрать начало. Цикл в примере ниже полностью идентичен приведённому выше:

|  |
| --- |
| var i = 0;  for (; i < 3; i++) {  alert( i ); // 0, 1, 2  } |

Можно убрать и шаг:

|  |
| --- |
| var i = 0;  for (; i < 3;) {  alert( i );  // цикл превратился в аналог 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**

Выйти из цикла можно не только при проверке условия но и, вообще, в любой момент. Эту возможность обеспечивает директива break.

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

|  |
| --- |
| var sum = 0;  while (true) {  var value = +prompt("Введите число", '');  if (!value) break; // (\*)  sum += value;  }  alert( 'Сумма: ' + sum ); |

Директива break в строке (\*), если посетитель ничего не ввёл, 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-конце цикла, а посередине.

**Следующая итерация: continue**

Директива continue прекращает выполнение текущей итерации цикла.

Она – в некотором роде «младшая сестра» директивы break: прерывает не весь цикл, а только текущее выполнение его тела, как будто оно закончилось.

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

Например, цикл ниже использует continue, чтобы не выводить чётные значения:

|  |
| --- |
| for (var i = 0; i < 10; i++) {  if (i % 2 == 0) continue;  alert(i);  } |

Для чётных i срабатывает continue, выполнение тела прекращается и управление передаётся на следующий проход for.

# Домашнее задание

* Владея навыками написания циклов и условных операторов, выполните задание со \* из предыдущего урока
* Напишите цикл, выводит такой треугольник:

|  |
| --- |
| #  ##  ###  ####  #####  ######  ####### |

* \* Напишите программу, создающую строку, содержащую решётку 8х8, в которой линии разделяются символами новой строки. На каждой позиции либо пробел, либо #. В результате должна получиться шахматная доска.

# Практика

Наша адресная книга будет представлять собой клиентское веб-приложение. Все хранение данных будет производиться в обыкновенном javascript-объекте. Для каждого элемента адресной книги (пользователя) предлагается хранить следующие единицы информации: имя, фамилия, дата рождения и номера телефонов. Например, организовать объект хранилища можно так:

// User storage

**var** users = [

{

firstName: 'Dmitry',

lastName: 'Kozlov',

birthDay: '16.07.1990',

phones: [

'89001234567',

'83431234567'

]

},

{

firstName: 'Ivan',

lastName: 'Petrov',

birthDay: '08.06.1956',

phones: [

'89001234567',

'83431234567'

]

}

];

Далее попробуем создать еще элементов адресной книги, запросив соответствующие данные у пользователя. Сделать это можно, например, с помощью оператора prompt. Созданного пользователя добавим в общее хранилище (массив) с помощью оператора push, а затем выведем всех пользователей, которые у нас есть с помощью массива:

**var** user = {};

user.firstName = **prompt**('Введите имя:');

user.lastName = **prompt**('Введите фамилию:');

user.birthDay = **prompt**('Дату рождения:');

user.phones = [];

user.phones[0] = **prompt**('Номер телефона:');

users.push(user);

**for** (**var** i = 0; i < users.**length**; ++i) {

console.**log**('Имя: ' + users[i].firstName, ' Фамилия: ' + users[i].lastName, ' Дата рождения: ' + users[i].birthDay, ' Телефоны: ' + users[i].phones.**join**(', '));

}

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)