


[🏠](#) → [Язык JavaScript](#) → [Основы JavaScript](#)
 7 мая 2022 г.

Логические операторы

В JavaScript есть четыре логических оператора: `||` (ИЛИ), `&&` (И) и `!` (НЕ), `??` (Оператор нулевого слияния). Здесь мы рассмотрим первые три, оператор `??` будет в следующей статье.

Несмотря на своё название, данные операторы могут применяться к значениям любых типов. Полученные результаты также могут иметь различный тип.

Давайте рассмотрим их подробнее.

|| (или)

Оператор «ИЛИ» выглядит как двойной символ вертикальной черты:

```
1 result = a || b;
```

Традиционно в программировании ИЛИ предназначено только для манипулирования булевыми значениями: в случае, если какой-либо из аргументов `true`, он вернёт `true`, в противоположной ситуации возвращается `false`.

В JavaScript, как мы увидим далее, этот оператор работает несколько иным образом. Но давайте сперва посмотрим, что происходит с булевыми значениями.

Существует всего четыре возможные логические комбинации:

```
1 alert( true || true ); // true
2 alert( false || true ); // true
3 alert( true || false ); // true
4 alert( false || false ); // false
```



Как мы можем наблюдать, результат операций всегда равен `true`, за исключением случая, когда оба аргумента `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`:

```
1 if (1 || 0) { // работает как if( true || false )
2   alert( 'truthy!' );
3 }
```



Обычно оператор `||` используется в `if` для проверки истинности любого из заданных условий.

К примеру:



```
1 let hour = 9;
2
3 if (hour < 10 || hour > 18) {
4   alert( 'Офис закрыт.' );
5 }
```

Можно передать и больше условий:



```
1 let hour = 12;
2 let isWeekend = true;
3
4 if (hour < 10 || hour > 18 || isWeekend) {
5   alert( 'Офис закрыт.' ); // это выходной
6 }
```

ИЛИ "||" находит первое истинное значение

Описанная выше логика соответствует традиционной. Теперь давайте поработаем с «дополнительными» возможностями JavaScript.

Расширенный алгоритм работает следующим образом.

При выполнении ИЛИ `||` с несколькими значениями:

```
1 result = value1 || value2 || value3;
```

Оператор `||` выполняет следующие действия:

- Вычисляет операнды слева направо.
- Каждый операнд конвертирует в логическое значение. Если результат `true`, останавливается и возвращает исходное значение этого операнда.
- Если все операнды являются ложными (`false`), возвращает последний из них.

Значение возвращается в исходном виде, без преобразования.

Другими словами, цепочка ИЛИ `||` возвращает первое истинное значение или последнее, если такое значение не найдено.

Например:



```
1 alert( 1 || 0 ); // 1
2 alert( true || 'no matter what' ); // true
3
4 alert( null || 1 ); // 1 (первое истинное значение)
5 alert( null || 0 || 1 ); // 1 (первое истинное значение)
6 alert( undefined || null || 0 ); // 0 (поскольку все ложно, возвращается последнее)
```

Это делает возможным более интересное применение оператора по сравнению с «чистым, традиционным, только булевым ИЛИ».

1. Получение первого истинного значения из списка переменных или выражений.

Представим, что у нас имеется ряд переменных, которые могут содержать данные или быть `null/undefined`. Как мы можем найти первую переменную с данными?

С помощью `||`:

```
1 let currentUser = null;
2 let defaultUser = "John";
3
4 let name = currentUser || defaultUser || "unnamed";
5
6 alert( name ); // выбирается "John" – первое истинное значение
```



Если бы и `currentUser`, и `defaultUser` были ложными, в качестве результата мы бы наблюдали `"unnamed"`.

2. Сокращённое вычисление.

Операндами могут быть как отдельные значения, так и произвольные выражения. ИЛИ `||` вычисляет их слева направо. Вычисление останавливается при достижении первого истинного значения. Этот процесс называется «сокращённым вычислением», поскольку второй операнд вычисляется только в том случае, если первого недостаточно для вычисления всего выражения.

Это хорошо заметно, когда выражение, указанное в качестве второго аргумента, имеет побочный эффект, например, изменение переменной.

В приведённом ниже примере `x` не изменяется:

```
1 let x;
2
3 true || (x = 1);
4
5 alert(x); // undefined, потому что (x = 1) не вычисляется
```



Если бы первый аргумент имел значение `false`, то `||` приступил бы к вычислению второго и выполнил операцию присваивания:

```
1 let x;
2
3 false || (x = 1);
4
5 alert(x); // 1
```



Присваивание – лишь один пример. Конечно, могут быть и другие побочные эффекты, которые не проявятся, если вычисление до них не дойдёт.

Как мы видим, этот вариант использования `||` является "аналогом `if`". Первый операнд преобразуется в логический. Если он оказывается ложным, начинается вычисление второго.

В большинстве случаев лучше использовать «обычный» `if`, чтобы облегчить понимание кода, но иногда это может быть удобно.

&& (И)

Оператор И пишется как два амперсанда `&&`:

```
1 result = a && b;
```

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

```
1 alert( true && true );    // true
2 alert( false && true );   // false
3 alert( true && false );   // false
4 alert( false && false );  // false
```

Пример с `if`:

```
1 let hour = 12;
2 let minute = 30;
3
4 if (hour == 12 && minute == 30) {
5     alert( 'The time is 12:30' );
6 }
```

Как и в случае с ИЛИ, любое значение допускается в качестве операнда И:

```
1 if (1 && 0) { // вычисляется как true && false
2     alert( "не сработает, так как результат ложный" );
3 }
```

И «&&» находит первое ложное значение

При нескольких подряд операторах И:

```
1 result = value1 && value2 && value3;
```

Оператор `&&` выполняет следующие действия:

- Вычисляет операнды слева направо.
- Каждый операнд преобразует в логическое значение. Если результат `false`, останавливается и возвращает исходное значение этого операнда.
- Если все операнды были истинными, возвращается последний.

Другими словами, И возвращает первое ложное значение. Или последнее, если ничего не найдено.

Вышеуказанные правила схожи с поведением ИЛИ. Разница в том, что И возвращает первое *ложное* значение, а ИЛИ – первое *истинное*.

Примеры:



```
1 // Если первый операнд истинный,  
2 // И возвращает второй:  
3 alert( 1 && 0 ); // 0  
4 alert( 1 && 5 ); // 5  
5  
6 // Если первый операнд ложный,  
7 // И возвращает его. Второй операнд игнорируется  
8 alert( null && 5 ); // null  
9 alert( 0 && "no matter what" ); // 0
```

Можно передать несколько значений подряд. В таком случае возвратится первое «ложное» значение, на котором остановились вычисления.



```
1 alert( 1 && 2 && null && 3 ); // null
```

Когда все значения верны, возвращается последнее



```
1 alert( 1 && 2 && 3 ); // 3
```

i Приоритет оператора && больше, чем у ||

Приоритет оператора И && больше, чем ИЛИ ||, так что он выполняется раньше.

Таким образом, код `a && b || c && d` по существу такой же, как если бы выражения && были в круглых скобках: `(a && b) || (c && d)`.

Как и оператор ИЛИ ||, И && иногда может заменять if.

К примеру:



```
1 let x = 1;  
2  
3 (x > 0) && alert( 'Greater than zero!' );
```

Действие в правой части && выполнится только в том случае, если до него дойдут вычисления. То есть, alert сработает, если в левой части (x > 0) будет true.

Получился аналог:



```
1 let x = 1;  
2  
3 if (x > 0) {  
4   alert( 'Greater than zero!' );  
5 }
```

Однако, как правило, вариант с if лучше читается и воспринимается.

Он более очевиден, поэтому лучше использовать его.

! (НЕ)

Оператор НЕ представлен восклицательным знаком `!`.

Синтаксис довольно прост:

```
1 result = !value;
```

Оператор принимает один аргумент и выполняет следующие действия:

1. Сначала приводит аргумент к логическому типу `true/false`.
2. Затем возвращает противоположное значение.

Например:

```
1 alert( !true ); // false
2 alert( !0 ); // true
```



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

```
1 alert( !!"non-empty string" ); // true
2 alert( !!null ); // false
```



То есть первое НЕ преобразует значение в логическое значение и возвращает обратное, а второе НЕ снова инвертирует его. В конце мы имеем простое преобразование значения в логическое.

Есть немного более подробный способ сделать то же самое – встроенная функция `Boolean` :

```
1 alert( Boolean("non-empty string") ); // true
2 alert( Boolean(null) ); // false
```



Приоритет НЕ `!` является наивысшим из всех логических операторов, поэтому он всегда выполняется первым, перед `&&` или `||`.

✓ Задачи

Что выведет alert (ИЛИ)?

важность: 5

Что выведет код ниже?

```
1 alert( null || 2 || undefined );
```

решение

Что выведет alert (ИЛИ)?

важность: 3

Что выведет код ниже?

```
1 alert( alert(1) || 2 || alert(3) );
```

решение

Что выведет alert (И)?

важность: 5

Что выведет код ниже?

```
1 alert( 1 && null && 2 );
```

решение

Что выведет alert (И)?

важность: 3

Что выведет код ниже?

```
1 alert( alert(1) && alert(2) );
```

решение

Что выведет этот код?

важность: 5

Что выведет код ниже?

```
1 alert( null || 2 && 3 || 4 );
```

решение

Проверка значения из диапазона

важность: 3

Напишите условие `if` для проверки, что переменная `age` находится в диапазоне между 14 и 90 включительно.

«Включительно» означает, что значение переменной `age` может быть равно 14 или 90 .

решение

Проверка значения вне диапазона

важность: 3

Напишите условие `if` для проверки, что значение переменной `age` НЕ находится в диапазоне 14 и 90 включительно.

Напишите два варианта: первый с использованием оператора НЕ `!` , второй – без этого оператора.

решение

Вопрос о "if"

важность: 5

Какие из перечисленных ниже `alert` выполнятся?

Какие конкретно значения будут результатами выражений в условиях `if(...)` ?

```
1 if (-1 || 0) alert( 'first' );
2 if (-1 && 0) alert( 'second' );
3 if (null || -1 && 1) alert( 'third' );
```

решение

Проверка логина

важность: 3

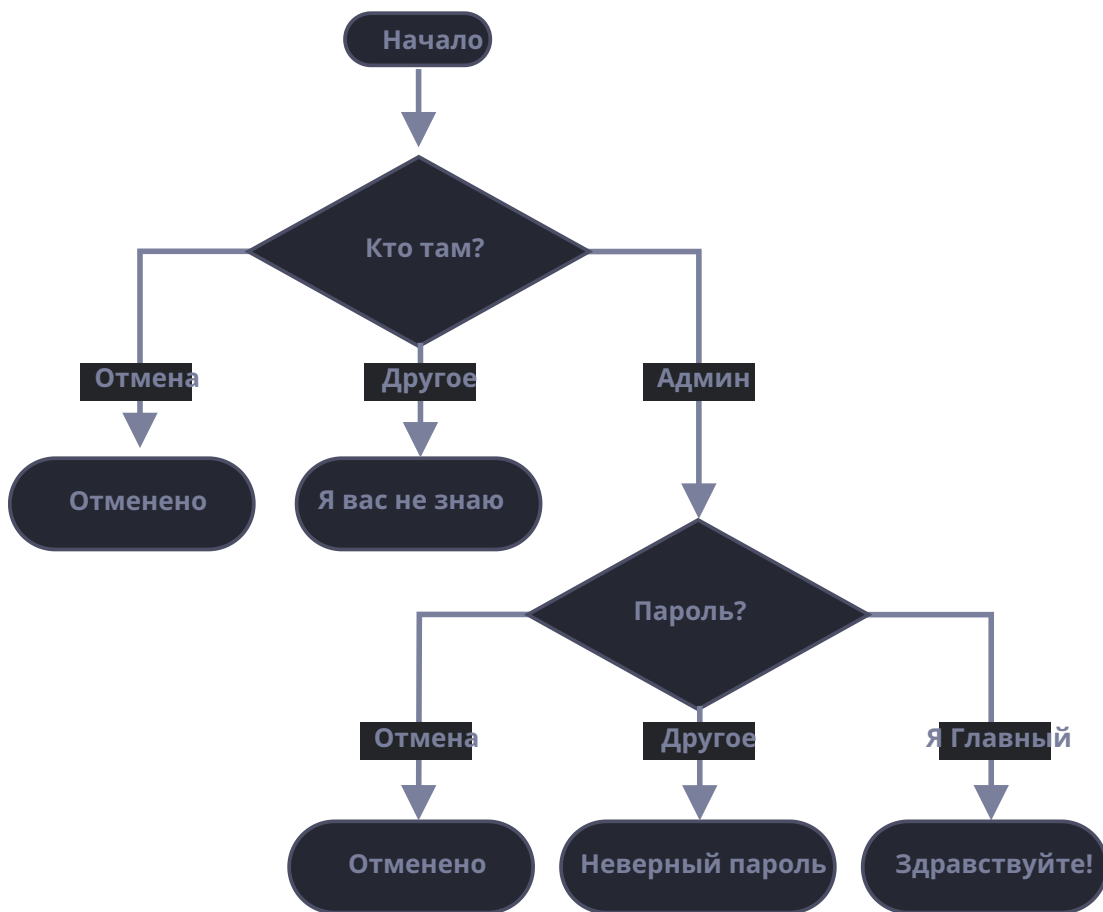
Напишите код, который будет спрашивать логин с помощью `prompt` .

Если посетитель вводит «Админ», то `prompt` запрашивает пароль, если ничего не введено или нажата клавиша `Esc` – показать «Отменено», в противном случае отобразить «Я вас не знаю».

Пароль проверять так:

- Если введён пароль «Я главный», то выводить «Здравствуйте!»,
- Иначе – «Неверный пароль»,
- При отмене – «Отменено».

Блок-схема:



Для решения используйте вложенные блоки `if`. Обращайте внимание на стиль и читаемость кода.

Подсказка: передача пустого ввода в приглашение `prompt` возвращает пустую строку `' '`. Нажатие клавиши `Esc` во время запроса возвращает `null`.

Запустить демо

решение



Предыдущий урок

Следующий урок



Поделиться



Карта учебника

Проводим курсы по JavaScript и фреймворкам.



Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))

- Если что-то непонятно в статье — пишите, что именно и с какого места.