# Логические операторы

В JavaScript есть три логических оператора: || (ИЛИ), && (И) и ! (НЕ).

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:

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:

alert( true || true ); // true

alert( false || true ); // true

alert( true || false ); // true

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:

if (1 || 0) { // работает как if( true || false )

alert( 'truthy!' );

}

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

К примеру:

let hour = 9;

if (hour < 10 || hour > 18) {

alert( 'Офис закрыт.' );

}

Можно передать и больше условий:

let hour = 12;

let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {

alert( 'Офис закрыт.' ); // это выходной

}

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:

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:

alert( 1 || 0 ); // 1

alert( true || 'no matter what' ); // true

alert( null || 1 ); // 1 (первое истинное значение)

alert( null || 0 || 1 ); // 1 (первое истинное значение)

alert( undefined || null || 0 ); // 0 (поскольку все ложно, возвращается последнее значение)

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

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 ||:

let currentUser = null;

let defaultUser = "John";

let name = currentUser || defaultUser || "unnamed";

alert( name ); // выбирается "John" – первое истинное значение

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

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:

let x;

true || (x = 1);

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:

let x;

false || (x = 1);

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 &&:

result = a && b;

В традиционном 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

Пример с if:

let hour = 12;

let minute = 30;

if (hour == 12 && minute == 30) {

alert( 'The time is 12:30' );

}

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

if (1 && 0) { // вычисляется как true && false

alert( "не сработает, так как результат ложный" );

}

И «&&» находит первое ложное значение

При нескольких подряд операторах И:

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:

// Если первый операнд истинный,

// И возвращает второй:

alert( 1 && 0 ); // 0

alert( 1 && 5 ); // 5

// Если первый операнд ложный,

// И возвращает его. Второй операнд игнорируется

alert( null && 5 ); // null

alert( 0 && "no matter what" ); // 0

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

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

Когда все значения верны, возвращается последнее

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

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:

let x = 1;

(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:

let x = 1;

if (x > 0) {

alert( 'Greater than zero!' );

}

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:

result = !value;

Оператор принимает один аргумент и выполняет следующие 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( !!"non-empty string" ); // true

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:

alert( Boolean("non-empty string") ); // true

alert( Boolean(null) ); // false

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