Логические операторы

|| (ИЛИ):

```
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**:

```
if (1 || 0) { // сработает как if( true || false )
        alert( 'верно' );
}
```

|| (ИЛИ):

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

```
let hour = 12,
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
        alert( 'Офис до 10 или после 18 или в выходной закрыт' );
}
```

|| (ИЛИ). Короткий цикл вычислений:

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:

```
let x;

true || (x = 1);

alert(x); // undefined, x не присвоен
```

|| (ИЛИ). Короткий цикл вычислений:

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

```
let x;

false || (x = 1);

alert(x); // 1
```

При этом оператор ИЛИ возвращает то значение, на котором остановились вычисления. Причём, не преобразованное к логическому типу.

```
alert( 1 || 0 ); // 1
alert( true || 'неважно что' ); // true

alert( null || 1 ); // 1
alert( undefined || 0 ); // 0
```

&& (И):

```
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
```

```
let hour = 12,
let minute = 30;

if (hour == 12 && minute == 30) {
        alert( 'Время 12:30' );
}
```

&& (И):

Как и в ИЛИ, в И допустимы любые значения:

```
if (1 && 0) { // вычислится как true && false
    alert( 'не сработает, т.к. условие ложно' );
}
```

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

```
// 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
```

Приоритет у && больше, чем у ||:

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

Поэтому в следующем коде сначала будет вычислено правое И: **1 && 0 = 0**, а уже потом – ИЛИ.

```
alert( 5 || 1 && 0 ); // 5
```

! (НЕ):

```
let result = !value;
```

Действия **!**:

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

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

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

```
alert( !!"строка" ); // true
alert( !!null ); // false
```