

2.2. Логические операторы

2.2.1. Операторы сравнения

Операторы сравнения используются для проверки условия. Операторы сравнения сравнивают два значения и возвращают значение **true** или **false**.

Оператор	Описание	Пример
<code>==</code>	Оператор равенства сравнивает два значения, и если они равны, возвращает true , иначе возвращает false	<pre>alert(2 == 1); // false, неверно alert('' == false); // true alert('01' == 1); // true, сравнивается как 1 == 1 alert(false == 0); // true, false становится числом 0 alert(true == 1); // true, так как true становится числом 1 alert(null == 0); // false alert(undefined == 0); // false</pre>
<code>===</code>	Оператор тождественности также сравнивает два значения и их тип, и если они равны, возвращает true , иначе возвращает false	<pre>alert(0 === false); // false, т.к. типы различн</pre>
<code>!=</code>	Сравнивает два значения, и если они не равны, возвращает true , иначе возвращает false	<pre>alert(2 != 1); // true</pre>
<code>!==</code>	Сравнивает два значения и их типы, и если они не равны, возвращает true , иначе возвращает false	<pre>alert('1' !== 1); // true</pre>

>	Сравнивает два значения, и если первое больше второго, то возвращает true , иначе возвращает false	<pre> alert(2 > 1); // true, верно alert('2' > 1); // true, сравнивается как 2 > 1 alert(null > 0); // false alert('Б' > 'А'); // true alert(+ "2" > + "14"); // false </pre>
<	Сравнивает два значения, и если первое меньше второго, то возвращает true , иначе возвращает false	<pre> alert(4 < 3); // false </pre>
>=	Сравнивает два значения, и если первое больше или равно второму, то возвращает true , иначе возвращает false	<pre> alert(2 >= 1); // true alert (null >= 0) ; // true </pre>
<=	Сравнивает два значения, и если первое меньше или равно второму, то возвращает true , иначе возвращает false	<pre> alert(2 >= 1); // false </pre>

Логические значения можно использовать и напрямую, присваивать переменным, работать с ними как с любыми другими:

```

var a = true; // присваивать явно
var b = 3 > 4; // или как результат сравнения
alert( b ); // false
alert( a == b ); // (true == false) неверно, выведет false

```

Сравнение операторов равенства "==" и тождественности "===".

Оператор равенства "=="	Оператор тождественности "==="
<pre>var in = 100; var str = "100"; var result = in == str; alert (result); //true</pre>	<pre>var in = 100; var str = "100"; var result = in === str; alert (result); // false</pre>

Аналогично работают операторы неравенства != и !==

Примеры работы операторов **undefined** и **null**

Несравнимый undefined	сравнения null с 0
<p>Значение undefined вообще нельзя сравнивать:</p> <pre>alert(undefined > 0); // false (1) alert(undefined < 0); // false (2) alert(undefined == 0); // false (3)</pre>	<p>Сравним null с нулём:</p> <pre>alert(null > 0); // false alert(null == 0); // false alert(null >= 0); // true</pre>

При проверке равенства значения **null** и **undefined** обрабатываются особым образом: они равны друг другу, но не равны чему-то ещё.

2.2.2. Операторы связи

Логические операторы используются для связки нескольких операторов сравнения.

В таблице ниже приведены логические операторы доступные в JavaScript (предположим, что $x=2$, а $y=9$):

Оператор	Значение	Описание	Пример	Результат
&&	И	Возвращает true , если обе операции сравнения возвращают true, иначе возвращает false	$(x==2 \ \&\& \ y==9)$	true
			$(x==3 \ \&\& \ y==9)$	false
	ИЛИ	Возвращает true, если хотя бы одна операция сравнения возвращают true, иначе возвращает false	$(x==2 \ \ y==8)$	true
			$(x==3 \ \ y==9)$	true
			$(x==5 \ \ y==6)$	false
!	НЕ	Возвращает true, если операция сравнения возвращает $!(x==3)$ false	true	

Оператор &&

Возвращает **true**, если обе операции сравнения возвращают **true**, иначе возвращает **false**.

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

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

Можно сказать, что "&& запинается на лжи".

Приоритет оператора **&&** больше, чем **||**, 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
```

Пример,

```
var income = 100;
var percent = 10;
var result = income > 50 && percent < 12;
alert (result); //true
```

Оператор ||

Возвращает **true**, если хотя бы одна операция сравнения возвращают **true**, иначе возвращает **false**.

Пример,

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

Оператор **||** используют, в частности, чтобы выбрать первое «истинное» значение из списка

```
var undef; // переменная не присвоена, т.е. равна undefined
var zero = 0;
var emptyStr = "";
var msg = "Привет!";
var result = undef || zero || emptyStr || msg || 0;
alert( result ); // выведет "Привет!" - первое значение, которое является true
```

Если все значения «ложные», то **||** возвратит последнее из них:

```
alert( undefined || '' || false || 0 ); // 0
```

Оператор !

Возвращает **true**, если операция сравнения возвращает **false**.

```
var income = 100;  
var result1 = !(income > 50);  
alert (result1); // false, так как income > 50 возвращает true
```

```
var isDeposit = false;  
var result2 = !isDeposit;  
alert (result2); // true
```

Двойное НЕ используют для преобразования значений к логическому типу:

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

Внимание: строка "0" становится true

В отличие от многих языков программирования, "0" в JavaScript является true, как и строка из пробелов:

```
alert( !!"0" ); // true  
alert( !!" " ); // любые непустые строки, даже из пробелов - true!
```

Условные операторы

Операторы ветвления предназначены для того, чтобы программа могла запускать тот или иной блок кода, в зависимости от верности **true** или не верности **false** условия.

Существует много **видов операторов ветвления**, в этом уроке мы рассмотрим два из них:

- оператор ветвления **if**
- оператор ветвления **if else**

Оператор ветвления **if**

Оператор ветвления **if** запускает код, если условие возвращает **true**.

В качестве условия, в операторах ветвления, могут выступать операции сравнения или логические операции.

Схема оператора ветвления **if**, выглядит следующим образом:

```
if (условие) {  
    оператор_если_истина (код);  
}
```


Пример,

```
// создадим две переменные
var One, Two;

// присвоим переменным значения
One = 5;
Two = 3;

if (One > Two) {
  alert("Условие возвратило true");
}
```

В этом примере сначала проверяется значение выражения "условие" (`One > Two`). Если "условие" истинно (`true`), выполняется оператор "оператор_если_истина" { `alert("Условие возвратило true");` }. В противном случае не выполняется ничего.

Оператор ветвления if else

Оператор ветвления **if else**, предназначен для запуска того или иного блока кода, в зависимости от значения которое вернёт условие: **true** или **false**

Схема оператора ветвления **if else**, выглядит следующим образом:

```
if (условие) {  
    оператор_если_истина (код);  
} else {  
    оператор_если_ложь (код);  
}
```

Пример,

```
// создадим две переменные  
var One, Two;  
// присвоим переменным значения  
One = 5;  
Two = 3;  
if (One > Two) {  
    alert("Условие возвратило true");  
} else {  
    alert("Условие возвратило false");  
}
```

В этом примере сначала проверяется значение выражения "условие". Если "условие" истинно (true), выполняется оператор "оператор_если_истина". В противном случае выполняется оператор "оператор_если_ложь".

Преобразование к логическому типу

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

В логическом контексте:

- Число **0**, пустая строка **""**, **null**, **undefined** и **NaN** являются **false**,
- Остальные значения – **true**.

Пример,

Такое условие никогда не

выполнится

```
if (0) { // 0 преобразуется к false
```

```
...
```

```
}
```

Такое условие выполнится

всегда:

```
if (1) { // 1 преобразуется к true
```

```
...
```

```
}
```

В конструкциях ветвления встречаются случаи, когда необходимо сделать выбор из нескольких альтернатив. Например, переменная может принимать несколько фиксированных значений ("1", "2", "3" и т.п.) в зависимости от которых необходимо направлять программу по разным маршрутам. В таких случаях можно скомбинировать несколько конструкций ветвления следующим образом:

```
if(логическое_выражение_1)
{
    оператор_1;
}
else if(логическое_выражение_2)
{
    оператор_2;
}
else if(логическое_выражение_3)
{
    оператор_3;
}
```

Пример,

```
var h = parseInt(prompt("Введите текущее время
(количество часов):", 9));
var время_сутки = "";
if (h>6 && h<=11) {
    время_сутки = "утро";
} else if (h>11 && h<=18) {
    время_сутки = "день";
} else if (h>18 && h<=21) {
    время_сутки = "вечер";
} else if ((h>21 && h<=23) || (h>=0 && h<=6) ) {
    время_сутки = "ночь";
} else {
    время_сутки = "неопределенное время суток";
}
alert ("Сейчас " + время_сутки + ".");
```

Вложенная конструкция if

В следующем примере, если в первый раз находится нужный диапазон, то мы выходим из программы и не делаем остальные ненужные проверки, если же нет, то пробуем найти его во второй раз (делаем вторую проверку) и т.д.

Это увеличивает скорость работы нашей программы по сравнению с проверкой всех семи условий.

```
var t = prompt("Введите температуру:", 36);
t = parseFloat(t);
if (t < 20 || t >= 46) {
    alert("Вы труп");
} else{
    if (t >= 20 && t < 27) {
        alert("У Вас кома");
    } else {
        if ( (t >= 27 && t < 36) || (t >= 37 && t < 42)) {
            alert("Вы больны");
        } else {
            if (t >= 36 && t < 37) {
                alert("Вы здоровы");
            } else {
                alert("У вас потеря сознания");
            }
        }
    }
}
}
```