Операторы

Термины: «унарный», «бинарный», «операнд»:

Операнд – то, к чему применяется оператор. Например, в умножении 5 * 2 есть два операнда: левый операнд равен 5, а правый операнд равен 2. Иногда их называют «аргументами» вместо «операндов».

Унарным называется оператор, который применяется к одному операнду. Например, оператор унарный минус "-" меняет знак числа на противоположный:

```
let x = 1;
x = -x;
alert( x ); // -1, применили унарный минус
```

Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

```
let x = 1, y = 3;
alert( y - x ); // 2, бинарный минус
```

Сложение строк, бинарный +:

```
let s = "моя" + "строка";
alert(s); // моястрока

alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"

alert(2 + 2 + '1' ); // будет "41", а не "221"

alert( 2 - '1' ); // 1
alert( '6' / '2' ); // 3

// Преобразует нечисла в числа
alert( +true ); // 1
alert( +"" );   // 0

let apples = "2";
let oranges = "3";
alert( apples + oranges ); // "23", так как бинарный плюс складывает строки
alert( +apples + +oranges ); // 5
```

Остаток от деления %:

Оператор взятия остатка %, несмотря на обозначение, никакого отношения к процентам **не имеет**.

Его результат **a % b** – это *остаток от деления* **a** на **b**.

```
alert( 5 % 2 ); // 1, остаток от деления 5 на 2
alert( 8 % 3 ); // 2, остаток от деления 8 на 3
alert( 6 % 3 ); // 0, остаток от деления 6 на 3
```

Возведение в степень **:

Для натурального числа **b** результат **a** ** **b** равен **a**, *умноженному на само себя* **b** раз.

```
alert( 2 ** 2 ); // 4  (2 * 2)
alert( 2 ** 3 ); // 8  (2 * 2 * 2)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2)
```

Оператор работает и для нецелых чисел.

```
alert( 4 ** (1/2) ); // 2 (степень 1/2 эквивалентна взятию квадратного корня)
alert( 8 ** (1/3) ); // 2 (степень 1/3 эквивалентна взятию кубического корня)
```

Инкремент/декремент:

Одной из наиболее частых операций является увеличение или уменьшение переменной на единицу.

```
let counter = 2;
counter++;          // работает как counter = counter + 1, но запись короче
alert( counter ); // 3
```

```
let counter = 2;
counter--;          // работает как counter = counter - 1,  но запись короче
alert( counter ); // 1
```

Постфиксная форма: counter**++** - сначала вернет значение, потом добавит единицу

Префиксная форма: **++**counter. - сначала добавит единицу, потом вернет значение

Операторы ++/-- могут также использоваться внутри выражений. Их приоритет выше, чем у арифметических операций.

```
let counter = 1;
alert( 2 * ++counter ); // 4
```

Но лучше использовать стиль «*одна строка – одно действие*»

Сокращённая арифметика с присваиванием:

Существуют операторы, которые позволяют выполнить арифметическое действие с переменной, после чего сразу присвоить результат вычислений этой же переменной.

```
let n = 2;
n = n + 5;
n = n * 2;
```

Эту запись можно укоротить при помощи совмещённых операторов **+=** и ***=**:

```
let n = 2;
n += 5; // теперь n=7 (работает как n = n + 5)
n *= 2; // теперь n=14 (работает как n = n * 2)

alert( n ); // 14
```

Подобные краткие формы записи существуют для всех арифметических и побитовых операторов: **/=, -=** и так далее.

Операторы сравнения:

- Больше/меньше: **a > b, a < b.**
- Больше/меньше или равно: **a >= b, a <= b.**
- Равно: **a == b**.
- Не равно: **a != b.**

Операторы сравнения, как и другие операторы, возвращают значение. Это значение имеет ***логический тип***:

- **true** – означает *«да», «верно», «истина»*.
- **false** – означает *«нет», «неверно», «ложь»*.

```
alert( 2 > 1 );  // true (верно)
alert( 2 == 1 ); // false (неверно)
alert( 2 != 1 ); // true (верно)
```

Результат сравнения можно присвоить переменной, как и любое значение

```
let result = 5 > 4; // результат сравнения присваивается переменной result
alert( result ); // true
```

Сравнение строк:

Чтобы определить, что одна строка больше другой, JavaScript использует *«алфавитный»* или *«лексикографический»* порядок. Другими словами, строки сравниваются ***посимвольно***.

```
alert( 'Я' > 'А' ); // true
alert( 'Кот' > 'Код' ); // true
alert( 'Сонный' > 'Сон' ); // true
```

Сравнение ***'Я' > 'А'*** завершится на первом шаге, тогда как строки "***Кот"*** и ***"Код"*** будут сравниваться посимвольно:

К *равна* **К**.
о *равна* **о**.
т *больше чем* **д**. На этом сравнение заканчивается. Первая строка больше.

Сравнение разных типов:

При сравнении значений разных типов JavaScript приводит каждое из них к числу.

```
alert( '2' > 1 ); // true, строка '2' становится числом 2
alert( '01' == 1 ); // true, строка '01' становится числом 1
```

Логическое значение ***true*** становится **1**, а ***false*** – **0**.

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

Будьте бдительны! Возможна следующая ситуация:

- Два значения равны.
- Одно из них **true** как логическое значение, другое – **false**.

```
let a = 0;
alert( Boolean(a) ); // false
let b = "0";
alert( Boolean(b) ); // true
alert(a == b); // true!
```

Строгое сравнение:

Использование обычного сравнения **==** может вызывать проблемы. Например, оно не отличает **0** от **false**

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

операнды разных типов преобразуются оператором == к числу

Оператор *строгого равенства* === проверяет равенство **без** приведения типов.

Если **a** и **b** имеют разные типы, то проверка **a === b** немедленно возвращает ***false*** без попытки их преобразования.

```
alert( 0 === false ); // false, так как сравниваются разные типы
```

Ещё есть **оператор *строгого неравенства* !==**, аналогичный **!=**.

Оператор строгого равенства дольше писать, но он делает код более очевидным и оставляет меньше мест для ошибок.

Сравнение с null и undefined:

При строгом равенстве **===**

```
alert( null === undefined ); // false
```

При нестрогом равенстве **==**

```
alert( null == undefined ); // true
```

При использовании математических операторов и других операторов сравнения < > <= >=

Значения **null/undefined** преобразуются к числам: **null** становится **0**, а **undefined** – **NaN**.

```
alert( null > 0 );  // (1) false
alert( null == 0 ); // (2) false
alert( null >= 0 ); // (3) true
```

Причина в том, что нестрогое равенство и сравнения **> < >= <=** работают по-разному. Сравнения преобразуют **null** в число, рассматривая его как **0**. Поэтому выражение (3) **null >= 0** истинно, а **null > 0** ложно.

Несравнимое значение undefined:

Значение **undefined** несравнимо с другими значениями

```
alert( undefined > 0 ); // false (1)
alert( undefined < 0 ); // false (2)
alert( undefined == 0 ); // false (3)
```

- Сравнения *(1)* и *(2)* возвращают ***false***, потому что **undefined** преобразуется в **NaN**, а **NaN** – это специальное числовое значение, которое возвращает ***false*** при любых сравнениях.
- Нестрогое равенство (3) возвращает ***false***, потому что **undefined** равно только **null** и ничему больше.