Преобразование типов

Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.

Например, ***alert*** автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам.

Есть также случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

Строковое преобразование

Строковое преобразование происходит, когда требуется представление чего-либо в виде строки.

Например, alert(value) преобразует значение к строке.

Также мы можем использовать функцию String(value), чтобы преобразовать любое значение к строке.

```
let value = true;
alert(typeof value); // Выведет: boolean

value = String(value); // теперь value это строка "true"
alert(typeof value); // Выведет: string
```

Численное преобразование

Численное преобразование происходит в математических функциях и выражениях.

Например, когда операция деления / применяется не к числу:

```
alert( "6" / "2" ); // 3, строки преобразуются в числа
```

Мы можем использовать функцию Number(value), чтобы явно преобразовать value к числу:

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // становится числом 123

alert(typeof num); // number
```

Явное преобразование часто применяется, когда мы ожидаем получить число из строкового контекста, например из текстовых полей форм.

Если строка не может быть явно приведена к числу, то результатом преобразования будет NaN. Например:

```
let age = Number("Любая строка вместо числа");

alert(age); // NaN, преобразование не удалось
```

Правила численного преобразования:

Значение	Преобразуется в…
undefined	NaN
null	0
true / false	1 / 0
string	Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то получаем 0, иначе из непустой строки «считывается» число. При ошибке результат NaN.

Логическое преобразование

Логическое преобразование самое простое.

Происходит в логических операциях, но также может быть выполнено явно с помощью функции Boolean(value).

Правило преобразования:

- ✔ Значения, которые интуитивно «пустые», вроде 0, пустой строки, null, undefined и NaN, становятся false.
- ✔ Все остальные значения становятся true.

Базовые операторы, математика

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 вычитает значения
```

Математика

Поддерживаются следующие математические операторы:

- ✔ Сложение **+**
- ✔ Вычитание **-**
- ✔ Умножение ***
- ✔ Деление **/**
- ✔ Взятие остатка от деления **%**
- ✔ Возведение в степень ****

Взятие остатка %

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

Результат `a % b` – это остаток от целочисленного деления `a` на `b`.

Например:

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

Возведение в степень **

В выражении a ** b оператор возведения в степень умножает a на само себя b раз.

Например:

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

Математически, оператор работает и для нецелых чисел. Например, квадратный корень является возведением в степень 1/2:

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

Сложение строк при помощи бинарного +

Давайте рассмотрим специальные возможности операторов JavaScript, которые выходят за рамки школьной арифметики.

Обычно при помощи плюса '+' складывают числа.

Но если бинарный оператор '+' применить к строкам, то он их объединяет в одну:

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

Обратите внимание, если хотя бы один операнд является строкой, то второй будет также преобразован в строку.

Например:

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

Как видите, не важно, первый или второй операнд является строкой.

Приведение к числу, унарный +

Плюс + существует в двух формах: бинарной, которую мы использовали выше, и унарной.

Унарный, то есть применённый к одному значению, плюс + ничего не делает с числами. Но если операнд не число, унарный плюс преобразует его в число.

На самом деле это то же самое, что и Number(...), только короче.

```
// Не влияет на числа
let x = 1;
alert( +x ); // 1

let y = -2;
alert( +y ); // -2

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

Сокращённая арифметика с присваиванием

Часто нужно применить оператор к переменной и сохранить результат в ней же.

Например:

```
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.

Инкремент/декремент

Одной из наиболее частых числовых операций является увеличение или уменьшение на единицу. Для этого существуют специальные операторы:

Инкремент **++** увеличивает переменную на 1:

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

Декремент **--** уменьшает переменную на 1:

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

Операторы ++ и -- могут быть расположены не только после, но и до переменной.

- ✔ Когда оператор идёт после переменной — это «постфиксная форма»: counter++.
- ✔ «Префиксная форма» — это когда оператор идёт перед переменной: ++counter.

Обе эти инструкции делают одно и то же: увеличивают counter на 1.

Чтобы увидеть разницу, вот два небольших примера:

```
let counterA = 1;
let a = ++counterA;
alert(a); // 2

let counterB = 1;
let b = counterB++;
alert(b); // 1
```

Операторы сравнения

В JavaScript они записываются так:

- ✔ Больше/меньше: a > b, a < b.
- ✔ Больше/меньше или равно: a >= b, a <= b.
- ✔ Равно: a == b. Обратите внимание, для сравнения используется двойной знак равенства ==. Один знак равенства a = b означал бы присваивание.
- ✔ Не равно. В математике обозначается символом ≠, но в JavaScript записывается как a != b.

Результат сравнения имеет логический тип

Все операторы сравнения возвращают значение логического типа:

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

Например:

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

Сравнение разных типов

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

Например:

```
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.

Строгое сравнение

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

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

Та же проблема с пустой строкой:

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

Это происходит из-за того, что операнды разных типов преобразуются оператором == к числу. В итоге, и пустая строка, и false становятся нулём.

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

Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования.

Давайте проверим:

```
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 места для ошибок.