Преобразование типов. Операторы сравнения. Сравнение с null и undefined. Логические операторы

Преобразование типов. Примитивные типы

Строковое – происходит, когда нам нужно что-то вывести. Может быть вызвано с помощью **String(value)**. Для примитивных значений работает очевидным образом. **Строка +** что-нибудь приводится к строке.
`alert` автоматически преобразует любое значение к строке.

Численное – происходит в **математических операциях** (за исключением строка + что-нибудь). При **сравнении** каждый операнд приводится к числу. Может быть вызвано с помощью **Number(value).** **Унарный плюс** преобразует операнд в число.
Преобразование подчиняется правилам:

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

Логическое – происходит в логических операциях (н-р, проверка условия if). Может быть вызвано с помощью **Boolean(value).** Двойное **!!** используют для преобразования значений к логическому типу
Подчиняется правилам:

Значение	Становится...
`0`, `null`, `undefined`, `NaN`, `""`	`false`
любое другое значение	`true`

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`.
- 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.

Все операторы сравнения возвращают значение **логического типа**.

Чтобы определить, что ***одна строка больше другой***, JavaScript использует «алфавитный» или «лексикографический» порядок. Другими словами, строки сравниваются посимвольно. Используется кодировка Unicode, а не настоящий алфавит.
Алгоритм сравнения двух строк довольно прост:

- Сначала сравниваются первые символы строк.
- Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
- Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
- Сравнение продолжается, пока не закончится одна из строк.
- Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

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

Сравнение с null и undefined

- При строгом равенстве **===**
  Эти значения различны, так как различны их типы.
  alert(null === undefined); // false

- При нестрогом равенстве **==**
  Эти значения равны друг другу и не равны никаким другим значениям. Это специальное правило языка.
  alert(null == undefined); // true

  При применении оператора == к null или undefined **преобразования в число не производится**. Значение **null равно только null или undefined** и не равно ничему больше. Аналогично для **undefined**

  ```
  null == 0            // false, null не преобразуется в 0
  null == null         // true
  undefined == undefined  // true
  null == undefined      // true
  ```

- При использовании математических операторов и других операторов сравнения < > <= >=
  Значения null/undefined преобразуются к **числам**: null становится 0, а undefined – NaN.
  Сравним `null` с нулём:

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

- Значение **undefined несравнимо с другими значениями:**

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

Сравнение с NAN

NaN является неравным (посредством сравнения через ==, !=, ===, and !==) любому другому значению, включая **другое значение NaN**.

Логические операторы

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.

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.

При выполнении **И &&** с несколькими значениями:

result = value1 && value2 && value3;

Оператор && выполняет следующие действия:

- Вычисляет операнды слева направо.

- Каждый операнд конвертирует в логическое значение. Если результат `false`, останавливается и возвращает исходное значение этого операнда.
- Если все операнды являются истинными, возвращает **последний** из них.

! Приоритет оператора && больше, чем у ||.

! (НЕ)

Оператор принимает один аргумент и выполняет следующие действия:

- Сначала приводит аргумент к логическому типу true/false.
- Затем возвращает противоположное значение.

Оператор объединения с null '??'

Эта возможность была добавлена в язык недавно. В старых браузерах может понадобиться полифил.

В этой статье мы будем говорить, что значение выражения «определено», если оно отличается от **null** или **undefined**. Результат выражения **a ?? b** будет следующим:

- **a,** если значение a определено,
- **b**, если значение a не определено.

То есть оператор ?? возвращает первый аргумент, если он не null/undefined, иначе второй.

Как правило, оператор ?? нужен для того, чтобы задать значение по умолчанию для потенциально неопределённой переменной. Кроме этого, можно записать последовательность из операторов ??, чтобы получить первое значение из списка, которое не является null/undefined.

```
let firstName = null;
let lastName = null;
let nickName = "Суперкодер";
// показывает первое определённое значение:
alert(firstName ?? lastName ?? nickName ?? "Аноним"); // Суперкодер
```

Сравнение с ||

Эта возможность была добавлена в язык недавно. В старых браузерах может понадобиться полифил.

Важное **различие** между ними заключается в том, что:

|| возвращает первое истинное значение.

?? возвращает первое определённое значение.

Проще говоря, оператор || не различает false, 0, пустую строку "" и null/undefined. Для него они все одинаковые, т.е. являются ложными значениями. Если первым аргументом для оператора || будет любое из перечисленных значений, то в качестве результата мы получим второй аргумент.

Однако на практике часто требуется использовать значение по умолчанию только тогда, когда переменная является null/undefined. Ведь именно тогда значение действительно неизвестно/не определено.

Например, рассмотрим следующий пример:

```
let height = 0;
alert(height || 100); // 100
alert(height ?? 100); // 0
```

Преобразование типов для объектов

При работе с объектами, напомним, также существует всего три направления преобразований: в число, в строку, и в логическое значение.

Самое простое — это преобразование в логическое значение: любое значение, не являющееся примитивом, всегда неявно конвертируется в true, это справедливо и для пустых объектов и массивов.

Объекты преобразуются в примитивные значения с использованием внутреннего метода **[[ToPrimitive]],** который ответственен и за преобразование в числовой тип, и за преобразование в строку.

И при конверсии в число, и при конверсии в строку используются два метода объекта, передаваемого [[ToPrimitive]]: это **valueOf** и **toString**. Оба метода объявлены в Object.prototype, и, таким образом, доступны для любого типа, основанного на Object, например — это Date, Array, и так далее.

В целом, **работа алгоритма** выглядит следующим образом:

- Если входное значение является примитивом — не делать ничего и вернуть его.
- Вызвать input.toString(), если результат является значением примитивного типа — вернуть его.
- Вызвать input.valueOf(), если результат является значением примитивного типа — вернуть его.
- Если ни input.toString(), ни input.valueOf() не дают примитивное значение — выдать ошибку TypeError.

При преобразовании в число сначала вызывается valueOf (3), если результат получить не удаётся — вызывается toString (2). При преобразовании в строку используется обратная последовательность действий — сначала вызывается toString (2), а в случае неудачи вызывается valueOf (3).

Большинство встроенных типов не имеют метода valueOf, или имеют valueOf, который возвращает сам объект, для которого он вызван (this), поэтому такое значение игнорируется, так как примитивом оно не является. Именно поэтому преобразование в число и в строку может работать одинаково — и то и другое сводится к вызову toString().

Метод valueOf для типов **Object** и **Array** возвращают сами эти объекты, поэтому это значение игнорируется.

Различные операторы могут вызывать либо преобразование в число, либо преобразование в строку с помощью параметра preferredType. Но есть два исключения: оператор нестрогого равенства == и оператор + с двумя операндами вызывают конверсию по умолчанию (preferredType не указывается или устанавливается в значение default). В этом случае большинство встроенных типов рассматривают, как стандартный вариант поведения, конверсию в число, за исключением типа Date, который выполняет преобразование объекта в строку.

Стандартные методы toString() и valueOf() можно переопределить для того, чтобы вмешаться в логику преобразования объекта в примитивные значения.