

Цели занятия

- научиться явно преобразовывать типы данных в JavaScript;
- познакомиться с базовыми операторами и методами примитивов;
- подробно разобрать работу с числами;

Полезные материалы

- [Современный учебник JavaScript](#)
- [Встроенный объект Math для работы с математикой](#)
- [Подробная статья про все нюансы чисел в JavaScript](#)

Преобразование типов

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

Например, `alert` автоматически преобразует любое значение к строке, а математические операторы преобразуют значения к числам. Но также есть случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

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 значение к строке:

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

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

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
```

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
```

Логическое преобразование — самое простое, происходит в логических операциях (позже мы познакомимся с условными проверками и подобными конструкциями), 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`

```
alert(Boolean(1)); // true  
alert(Boolean(0)); // false  
alert(Boolean("Привет!")); // true  
alert(Boolean("")); // false
```

Базовые операторы

Многие операторы знакомы нам ещё со школы: сложение `+`, умножение `*`, вычитание `-` и так далее. Также в JavaScript есть несколько других, менее очевидных операторов, но прежде чем изучить их давайте разберёмся с терминологией.

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;
let 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 `**`.

Первые четыре оператора очевидны, а про `%` и `**` стоит сказать несколько слов.

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); // 2, остаток от деления 8 на 3
```

Возведение в степень `**` — в выражении `a ** b` оператор возведения в степень умножает `a` на само себя `b` раз. Например:

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

Сложение строк при помощи бинарного `+` — обычно при помощи плюса `+` складывают числа. Но если бинарный оператор `+` применить к строкам, то он их объединяет в одну; это называется конкатенацией строк.

```
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"
```

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.

```
// 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
```

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

Операторы сравнения

- 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 `= b` означал бы присваивание.
- Не равно. В математике обозначается символом $\neq$, но в JavaScript записывается как `a != b`.

Все операторы сравнения возвращают значение логического типа: `true` — означает «да», «верно», «истина», а `false` — означает «нет», «неверно», «ложь».

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

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
```

Поэтому использование обычного сравнения `==` может вызывать проблемы. Например, оно не отличает 0 от false. Поэтому для сравнений лучше использовать оператор строгого равенства `===`, который сравнивает значения без преобразования типов.

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

Другими словами, если `a` и `b` имеют разные типы, то проверка `a === b` немедленно возвращает false без попытки их преобразования. Ещё есть оператор строгого неравенства `!==`, аналогичный `!=`.

Другие математические операции

В JavaScript встроен объект `Math`, который содержит различные математические функции и константы. Несколько примеров:

Math.random() — возвращает случайное число в диапазоне от 0 (включительно) до 1 (но не включая 1)

```
alert( Math.random() ); // 0.1234567894322
alert( Math.random() ); // 0.5435252343232
alert( Math.random() ); // ... (любое количество случайных чисел)
```

Math.max(a, b, c...) — возвращает наибольшее число из перечисленных аргументов.

```
alert( Math.max(3, 5, -10, 0, 1) ); // 5
```

Math.min(a, b, c...) — возвращает наименьшее число из перечисленных аргументов.

```
alert( Math.min(1, 2) ); // 1
```

Числа

В современном JavaScript существует два типа чисел:

1. Обычные числа в JavaScript хранятся в 64-битном формате, который также называют «числа с плавающей точкой двойной точности». Это числа, которые мы будем использовать чаще всего. Мы поговорим о них на этом занятии.
2. BigInt числа дают возможность работать с целыми числами произвольной длины. Они нужны достаточно редко, поэтому пока что разбирать их не будем.

Способы записи числа

Представьте, что нам надо записать число 1 миллиард. Самый очевидный путь:

```
let billion = 1000000000;
```

Но в реальной жизни мы обычно опускаем запись множества нулей, так как можно легко ошибиться. Укороченная запись может выглядеть как "1млрд" или "7.3млрд" для 7 миллиардов 300 миллионов.

В JavaScript можно использовать букву "e", чтобы укоротить запись числа. Она добавляется к числу и заменяет указанное количество нулей:

```
let billion = 1e9; // 1 миллиард, буквально: 1 и 9 нулей  
alert(7.3e9); // 7.3 миллиардов (7,300,000,000)
```

Сейчас давайте запишем что-нибудь очень маленькое. К примеру, 1 микросекунду (одна миллионная секунды):

```
let ms = 0.000001;
```

Записать микросекунду в укороченном виде нам снова поможет "e".

```
let ms = 1e-6; // шесть нулей, слева от 1
```

Округление чисел

Одна из часто используемых операций при работе с числами – это округление. В JavaScript есть несколько встроенных функций для работы с округлением:

- **Math.floor(n)** — округление в меньшую сторону: 3.1 становится 3, а -1.1 — -2.
- **Math.ceil(n)** — округление в большую сторону: 3.1 становится 4, а -1.1 — -1.
- **Math.round(n)** — округление до ближайшего целого: 3.1 становится 3, 3.6 — 4, а -1.1 — -1.
- **Math.trunc(n)** — производит удаление дробной части без округления: 3.1 становится 3, а -1.1 — -1 (не поддерживается в Internet Explorer)
- **Метод toFixed(n)** — округляет число до n знаков после запятой и возвращает строковое представление результата.

```
let num = 12.34;  
alert(num.toFixed(1)); // "12.3"
```

Обратите внимание, что результатом `toFixed` является строка. Если десятичная часть короче, чем необходима, будут добавлены нули в конец строки:

```
let num = 12.34;
```

```
alert(num.toFixed(5)); // "12.34000", добавлены нули,  
чтобы получить 5 знаков после запятой
```

parseInt и parseFloat

Для явного преобразования к числу можно использовать `+` или `Number()`. Если строка не является в точности числом, то результат будет `NaN`:

```
alert(+"100px"); // NaN
```

В реальной жизни мы часто сталкиваемся со значениями у которых есть единица измерения, например `"100px"` или `"12em"` в CSS. Также во множестве стран символ валюты записывается после номинала `"19€"`. Так как нам получить числовое значение из таких строк? Для этого есть `parseInt` и `parseFloat`.

Они «читают» число из строки. Если в процессе чтения возникает ошибка, они возвращают полученное до ошибки число. Функция `parseInt` возвращает целое число, а `parseFloat` возвращает число с плавающей точкой:

```
alert( parseInt("100px") ); // 100  
alert( parseFloat("12.5em") ); // 12.5  
  
alert( parseInt("12.3") ); // 12, вернётся только целая часть  
alert( parseFloat("12.3.4") ); // 12.3, произойдёт остановка  
чтения на второй точке
```

Функции `parseInt/parseFloat` вернут `NaN`, если не смогли прочесть ни одну цифру:

```
alert( parseInt("a123") ); // NaN, на первом символе  
происходит остановка чтения
```