> Эта страница была переведена с английского языка силами сообщества. Вы тоже можете внести свой вклад, присоединившись к русскоязычному сообществу MDN Web Docs.

Числа и даты

Данный раздел представляет собой введение в работу с числами и датами в JavaScript.

Числа

Ли

двоичным форматом IEEE 754 (т.е. числами, принадлежащими диапазону между -(253 -1) и 253 -1). Целые числа не рассматриваются как отдельный тип чисел. В дополнение к числам с плавающей запятой, к числовому типу данных относятся также три символьные величины: `+бесконечность`, `-бесконечность`, и `NaN` (не-число). В разделе типы и структуры данных в JavaScript числовой тип данных описан в контексте с другими примитивными типами в JavaScript.

Вы можете использовать четыре типа числовых литералов: десятичный, двоичный, восьмеричный и шестнадцатеричный.

Десятичные числа

```
JS
1234567890;
42;

// Будьте внимательны при использование нулей в начале чисел:
```

```
0888; // 888 обрабатывается как десятичное
0777; // обрабатывается как восьмеричное в нестрогой форме (511 в десятичной)
```

Обратите внимание, что десятичные литералы могут начинаться с нуля (`0`) за которым следует другая десятичная цифра, но если следующая за нулём цифра меньше 8, то число обрабатывается как восьмеричное.

Двоичные числа

Синтаксис двоичных чисел использует ведущий 0 за которым следует латинская буква "B" в верхнем или нижнем регистре (`0b` or `0B`). Если цифры после `0b` не являются 0 или 1, то будет сгенерированно `SyntaxError` с сообщением: "Missing binary digits after 0b".

JS

```
var FLT_SIGNBIT = 0b10000000000000000000000000000000; // 2147483648
var FLT_EXPONENT = 0b01111111100000000000000000000000; // 2139095040
var FLT_MANTISSA = 0b00000000000111111111111111111111; // 8388607
```

Восьмеричные числа

Синтаксис восьмеричных чисел использует ноль в начале. Если цифры после 0 не входят в диапазон от 0 до 7, число будет интерпретировано как десятичное.

JS

```
var n = 0755; // 493
var m = 0644; // 420
```

Шестнадцатеричные числа

Синтаксис шестнадцатеричных чисел использует ведущий 0 за которым следует латинская буква "X" в верхнем или нижнем регистре (`0x` or `0X`). Если цифры после 0x не входят в диапазон (0123456789ABCDEF), то будет сгенерированно `SyntaxError` с сообщением: "Identifier starts immediately after numeric literal".

JS

```
0xffffffffffffffff; // 295147905179352830000
0x123456789abcdef; // 81985529216486900
```

```
0xa; // 10
```

Объект Number

У этого встроенного объекта `Number` есть свойства для целочисленных констант, таких как максимальное число, не-число и бесконечность. Вы не можете изменить значения этих свойств, и вы должны использовать их следующим образом:

```
JS
var biggestNum = Number.MAX_VALUE;
var smallestNum = Number.MIN_VALUE;
var infiniteNum = Number.POSITIVE_INFINITY;
var negInfiniteNum = Number.NEGATIVE_INFINITY;
var notANum = Number.NaN;
```

Как видно из примера выше, для получения перечисленных значений, нужно обращаться к свойствам предопределённого глобального объекта `Number`. Тогда как у экземпляра этого объекта, созданного вами при работе программы, этих свойств не будет.

В следующей таблице приведён список свойств объекта `Number`.

Свойство	Описание
`Number.MAX_VALUE`	Наибольшее число из возможных для представления
`Number.MIN_VALUE`	Наименьшее число из возможных для представления
`Number.NaN`	Специальное "Не числовое" ("not a number") значение
`Number.NEGATIVE_INFINITY`	Специальное значение "Минус бесконечность"; возвращается при переполнении
`Number.POSITIVE_INFINITY`	Специальное значение "Плюс бесконечность"; возвращается при переполнении
`Number.EPSILON`	Разница между единицей и наименьшим значением, большим единицы, которое может быть представлено типом `Number`.

Свойство	Описание
Number.MIN_SAFE_INTEGER	Минимальное целое, безопасное число в JavaScript.
Number.MAX_SAFE_INTEGER	Максимальное целое, безопасное число в JavaScript.

Метод	Описание
Number.parseFloat()	Принимает строку как аргумент, и возвращает числовое значение с плавающей точкой, которое удалось распознать. Тоже самое что и глобальная функция parseFloat() .
Number.parseInt()	Принимает строку как аргумент, и возвращает целочисленное значение в заданной системе исчисления, которое удалось распознать. Тоже самое что и глобальная функция parseInt() .
Number.isFinite()	Определяет, является ли число, переданное в качестве аргумента, конечным.
Number.isInteger()	Определяет, является ли число, переданное в качестве аргумента, целым.
Number.isNaN()	Определяет, является ли число, переданное в качестве аргумента, NaN (не числом). Аналогичный, но более надёжный метод чем глобальная функция isNaN() .
Number.isSafeInteger()	Определяет, является ли .число, переданное в качестве аргумента, целым и безопасным.

Прототип Number предоставляет ряд методов, для получения значения числа в различных форматах. В следующей таблице перечислены методы, доступные через Number.prototype .

Метод	Описание
toExponential()	Возвращает строку, представляющую число в экспоненциальном представлении.

Метод	Описание
toFixed()	Возвращает строку, представляющую число с заданным количеством разрядов после запятой.
toPrecision()	Возвращает строку, представляющую число с указанной точностью.

Объект Math

Встроенный глобальный объект `Math` содержит свойства и методы для математических констант и функций. Например, свойство объекта `Math` `PI` содержит значение математической константы "Пи" (3.141...), которые вы можете использовать в программе как

```
JS
Math.PI;
```

Подобным образом, математические функции являются методами объекта `Math` . Они включают тригонометрические, логорифмические, экспоненциальные и другие функции. Например, если вы хотите использовать тригонометрическую функцию синуса, вы напишете следующий код

```
JS
Math.sin(1.56);
```

Заметьте, что все тригонометрические методы объекта `Math` принимают аргументы в радианах.

В следующей таблице перечислены методы объекта `Math` .

Метод	Описание
abs()	Возвращает абсолютное значение (модуль) аргумента
sin() , cos() , tan()	Стандартные тригонометрические функции; принимают аргументы в радианах

Метод	Описание
asin() , acos() , atan() , atan2()	Обратные тригонометрические функции; возвращают значения в радианах
sinh() , cosh() , tanh()	Гиперболические тригонометрические функции; принимают аргументы в гиперболических углах
asinh() , acosh() , atanh()	Обратные гиперболические тригонометрические функции; возвращают значения в гиперболических углах
pow() , exp() , expm1() , log10() , log1p() , log2()	Экспоненциальные и логорифмические функции
floor() , ceil()	Возвращают наибольшее/наименьшее целое, которое меньше/больше или равно входному значению
min() , max()	Возвращают наибольшее или наименьшее (соответственно) из входных числовых значений, перечисленных через запятую
random()	Возвращает случайное число от 0 до 1
round() , fround() , trunc() ,	Функции округления и отсечения дробной части
sqrt() , cbrt() , hypot()	Корень квадратный, корень кубический, корень квадратный из суммы квадратов аргументов
sign()	Знак числа, показывает является ли входное число позитивным, негативным или равным нулю
clz32() , imul()	Количество первых нулевых бит в 32-битном двоичном представлении. Возвращает результат Си-подобного 32-битного целочисленного умножения двух аргументов.

В отличии от большинства других объектов, вам не нужно создавать свои экземпляры объекта `Math`. Всегда следует использовать глобальный объект `Math` непосредственно.

Объект Date

JavaScript не имеет отдельного типа данных для хранения дат. Однако, вы можете использовать объект `Date` и его методы для работы с датами и временном в вашем приложении. Объект `Date` имеет большое количество методов для записи, чтения и оперирования датой и временем, а свойств не имеет ни одного.

JavaScript оперирует датами во многом аналогично языку Java. Для работы с датой эти два языка имеют множество одинаковых методов, и оба они хранят даты как количество миллисекунд, прошедших с 00:00:00 1 Января 1970 года.

Период значений, которые может принимать Data, простирается от -100 000 000 до100 000 000 дней, относительно 1 Января 1970 года.

Чтобы создать свой экземпляр объекта `Date` используйте такой код :

```
JS
var dateObjectName = new Date([parameters]);
```

где `dateObjectName` это имя переменной, которой будет присвоено созданное значение с типом `Date` ; вместо этой переменной, присвоить созданное значение можно как свойство, любому существующему объекту.

Вызов `Date` как функции, без ключевого слова `new`, возвращает текущую дату и время в виде текстовой строки.

Содержимым блока `parameters` в коде представленном выше, могут выступать любая из следующих конструкций:

- Пусто: создаёт текущую дату и время. Пример: `today = new Date();` .
- Текстовая строка, содержащая дату и время в следующем формате: "Месяц день, год часы:минуты:секунды". Например: `var Xmas95 = new Date("December 25, 1995 13:30:00")` . Если не указать секунды, минуты или часы, то их значение будет принято за 0.
- Набор целочисленных значений для года, месяца и дня. Например: `var Xmas95 = new Date(1995, 11, 25)` .

- Набор целочисленных значений для года, месяца, дня, часов, минут и секунд. Например: `var Xmas95 = new Date(1995, 11, 25, 9, 30, 0);`.

Получение даты в русскоязычном формате

```
new Date().toLocaleString('ru',
  {
    day: 'numeric',
    month: 'long',
    year: 'numeric'
  });
// текущая дата в формате: '10 января 2019 г.'
```

Подробнее про метод toLocalString

Методы объекта Date

Методы объекта `Date` для работы с датой и временем делятся на следующие категории:

- "set" методы, служат для установки параметров объекта `Date`.
- "get" методы, служат для получения параметров объекта `Date`.
- "to" методы, служат для получения значения объекта `Date` в текстовом виде.
- "parse" и UTC методы, служат для распознавания дат и времени из текстового формата.

При помощи методов групп "get" и"set", вы можете получить и установить значения секунд, минут, часов, дня месяца, дня недели, месяца и года по отдельности. Отдельно выделим метод `getDay`, который возвращает день недели, однако соответствующего ему метода `setDay` не существует, потому-что день недели высчитывается автоматически. Все эти методы используют в своей работе целочисленные значения по следующим правилам:

- Секунды и минуты: от 0 до 59
- Часы: от 0 до 23
- Дни недели: от 0 (Воскресенье) до 6 (Суббота)

- Дни месяца: от 1 до 31
- Месяцы: от 0 (Январь) до 11 (Декабрь)
- Год: год относительно 1900 года.

Например, предположим, что вы определили дату следующим образом:

```
JS
var Xmas95 = new Date("December 25, 1995");
```

Тогда `Xmas95.getMonth()` вернёт 11, а `Xmas95.getFullYear()` вернёт 1995.

Методы `getTime` и `setTime` могут быть полезны при сравнении дат. Метод `getTime` возвращает количество миллисекунд, прошедших с 00:00:00 1 Января, 1970 года.

Для примера рассмотрим следующий код, показывающий количество дней оставшихся до конца текущего года:

```
JS
var today = new Date(); // Получаем текущую дату
var endYear = new Date(1995, 11, 31, 23, 59, 59, 999); // Устанавливаем месяц и день на конец года
endYear.setFullYear(today.getFullYear()); // Устанавливаем текущий год
var msPerDay = 24 * 60 * 60 * 1000; // Количество миллисекунд в одних сутках
var daysLeft = (endYear.getTime() - today.getTime()) / msPerDay;
var daysLeft = Math.round(daysLeft); // возвращает количество дней, оставшихся до конца года
```

Этот пример создаёт объект `Date` названный `today, который содержит текущую дату и время` . Затем, создаётся вспомогательный `Date` объект, названный `endYear`, которому устанавливаются значения, указывающие на конец года. После этого, при помощи рассчитанного количества миллисекунд в сутках, вычисляется количество дней между `today` и `endYear` , При этом используются метод `getTime` и округление, для получения количества полных дней.

Метод `parse` полезен для присвоения значений существующим объектам `Date` из текстового формата. Например, следующий код использует методы `parse` и `setTime` чтобы установить значение переменной `IPOdate` :

JS

```
var IPOdate = new Date();
IPOdate.setTime(Date.parse("Aug 9, 1995"));
```

Пример

В следующем примере приведена функция `JSClock()`, которая возвращает время в формате электронных часов:

JS

```
function JSClock() {
  var time = new Date();
  var hour = time.getHours();
  var minute = time.getMinutes();
  var second = time.getSeconds();
  var temp = "" + (hour > 12 ? hour - 12 : hour);
  if (hour == 0) temp = "12";
  temp += (minute < 10 ? ":0" : ":") + minute;
  temp += (second < 10 ? ":0" : ":") + second;
  temp += hour >= 12 ? " P.M." : " A.M.";
  return temp;
}
```

Первым делом, функция JSClock создаёт новый объект `Date`, названный `time`; так как объект создаётся баз параметров, переменная `time` содержит текущую дату и время. Затем вызываются методы `getHours`, `getMinutes` и `getSeconds`, чтобы установить значения переменным `hour`, `minute` и `second`.

Следующие четыре выражения строят текстовую строку на основе time. Первое из них создаёт переменную `temp`, присваивая ей значение при помощи условия; Если `hour` больше чем 12, то (`hour - 12`), иначе просто `hour`. В случае же, когда `hour` равно 0, берётся фиксированное значение "12".

Следующее выражение приращивает минуты к переменной `temp`. Если количество минут меньше 10, то условное выражение добавляет строку с двоеточием и заполняющим нулём, иначе, просто строку с двоеточием. Секунды приращиваются к переменной `temp` тем же путём.

В завершение всего, последнее условное выражение добавляет строку "P.M." к переменной `temp` если `hour` равно или больше 12, иначе, добавляется строка "A.M.".