



IT-ACADEMY

# JavaScript числа

В JavaScript есть только один тип числа. Числа можно писать с десятичными знаками или без них.

```
var x = 3.14;    // A number with decimals
var y = 3;       // A number without decimals
```

Сверхбольшие или сверхмалые числа можно записывать в экспоненциальной нотации:

```
var x = 123e5;   // 12300000
var y = 123e-5; // 0.00123
```

**Числа JavaScript всегда являются 64-битными числами с плавающей запятой**

# Числа

В отличие от многих других языков программирования, JavaScript не определяет различные типы чисел, такие как целые числа, короткие, длинные, с плавающей точкой и т.д.

Числа на JavaScript всегда хранятся в виде чисел с плавающей точкой двойной точности, в соответствии с международным стандартом IEEE 754.

В этом формате числа хранятся в 64 битах, где число (дробь) хранится в битах от 0 до 51, экспонента в битах от 52 до 62 и знака в бите 63:

| Value (aka Fraction/Mantissa) | Exponent          | Sign       |
|-------------------------------|-------------------|------------|
| 52 bits (0 - 51)              | 11 bits (52 - 62) | 1 bit (63) |

# Точность

Целые числа (числа без точки или показателя степени) имеют точность до 15 цифр.

```
var x = 999999999999999; // x will be 999999999999999
var y = 999999999999999; // y will be 10000000000000000
```

Максимальное количество десятичных знаков - 17, но арифметика с плавающей запятой не всегда точна на 100%:

```
var x = 0.2 + 0.1; // x will be 0.30000000000000004
```

Чтобы решить указанную выше проблему, это помогает умножать и делить

```
var x = (0.2 * 10 + 0.1 * 10) / 10; // x will be 0.3
```

Adding Numbers and Strings

# Добавление чисел и строк

## ВНИМАНИЕ!!!

JavaScript использует оператор + как для сложения, так и для конкатенирования.

Числа добавляются. Строки конкатенированы.

Если добавить два числа, то в результате получится число:

```
var x = 10;  
var y = 20;  
var z = x + y;           // z  
will be 30 (a number)
```

Если добавить две строки, то в результате получится конкатенация строк:

```
var x = "10";  
var y = "20";  
var z = x + y;           // z  
will be 1020 (a string)
```

# Если вы добавите число и строку, результатом будет конкатенация строк:

```
var x = 10;  
var y = "20";  
var z = x + y;           // z will be 1020 (a  
string)
```

Если вы добавите строку и число, то в результате получится конкатенация строк:

```
var x = "10";  
var y = 20;  
var z = x + y;           // z will be 1020 (a  
string)
```

Распространенная ошибка - ожидать, что результат будет 30:

```
var x = 10;  
var y = 20;  
var z = "The result is: " + x  
+ y;
```

# Распространенной ошибкой является ожидание такого результата на уровне 102030:

```
var x = 10;  
var y = 20;  
var z = "30";  
var result = x + y + z;
```

Интерпретатор JavaScript работает слева направо.

Первые  $10 + 20$  добавляются потому, что  $x$  и  $y$  - это оба числа.

Затем  $30 + "30"$  складывается, потому что  $z$  - строка.

# Числовые строки

Строки JavaScript могут иметь числовое содержание:

```
var x = 100;           // x is a number
```

```
var y = "100";        // y is a string
```

JavaScript попытается преобразовать строки в числа во всех числовых операциях:

Это сработает:

```
var x = "100";  
var y = "10";  
var z = x / y;      // z will be 10
```

This will also work:

```
var x = "100";  
var y = "10";  
var z = x * y;      // z will be 1000
```

# Числа

И это работает:

```
var x = "100";  
var y = "10";  
var z = x - y;           // z will be 90
```

But this will not work:

```
var x = "100";  
var y = "10";  
var z = x + y;           // z will not be 110 (It  
will be 10010)
```

В последнем примере JavaScript использует оператор + для конкатенирования строк.

# NaN - Не Номер

NaN - это зарезервированное слово на JavaScript, означающее, что номер не является законным номером.

Попытка сделать арифметику с нецифровой строкой приведет к NaN (Not a Number):

```
var x = 100 / "Apple"; // x will be NaN (Not a Number)
```

Однако если строка содержит числовое значение, то результатом будет число:

```
var x = 100 / "10"; // x will be 10
```

**Вы можете использовать глобальную JavaScript-функцию `isNaN()`, чтобы узнать, является ли значение числом:**

```
var x = 100 / "Apple";  
isNaN(x); // returns true  
because x is Not a Number
```

**Осторожнее с Наном.**

Если вы используете NaN в математической операции, результатом будет также NaN:

```
var x = NaN;  
var y = 5;  
var z = x + y; // z will be NaN
```

# Или результатом может быть конкатенация:

```
var x = NaN;  
var y = "5";  
var z = x + y;           // z will be NaN5
```

**NaN** - это номер: **тип NaN** возвращает номер:

```
typeof NaN;           // returns "number"
```

## Бесконечность

Бесконечность (или `-Infinity`) - это значение, которое JavaScript вернет, если вы вычислить число за пределами максимально возможного числа.

Деление на 0 (ноль) также генерирует бесконечность:

```
var x = 2 / 0;        // x will be Infinity  
var y = -2 / 0;       // y will be -Infinity
```

```
var myNumber = 2;  
while (myNumber !== Infinity) { // Execute  
  until Infinity  
  myNumber = myNumber * myNumber;  
}
```

**Бесконечность - это число: тип бесконечности возвращает число.**

```
typeof Infinity; // returns "number"
```

## Шестнадцатеричный

JavaScript интерпретирует числовые константы как шестнадцатеричные, если им предшествует 0x.

```
var x = 0xFF; // x will be 255
```

Никогда не пишите число с ведущим нулем (например, 07).

Некоторые версии JavaScript интерпретируют числа как восьмеричные, если они пишутся с ведущим нулем. Вы можете использовать метод toString() для вывода чисел из базы 2 в базу 36.

По умолчанию JavaScript отображает числа в виде базовых 10 десятичных знаков.

```
var myNumber = 32;
myNumber.toString(10); // returns 32
myNumber.toString(32); // returns 10
myNumber.toString(16); // returns 20
myNumber.toString(8); // returns 40
myNumber.toString(2); // returns 100000
```

**Цифры могут быть Объектами**  
Обычно JavaScript-номера являются примитивными значениями, созданными из литералов:

```
var x = 123;
```

**Но числа также могут быть определены как объекты с новым ключевым словом:**

```
var y = new Number(123);
```

```
var x = 123;
var y = new Number(123);

// typeof x returns number
// typeof y returns object
```

Не создавайте объекты Number. Это замедляет скорость выполнения.  
Новое ключевое слово усложняет код. Это может привести к неожиданным результатам:

При использовании оператора == равные числа равны:

```
var x = 500;  
var y = new Number(500);
```

```
// (x == y) is true because x and y have  
equal values
```

При использовании оператора === равные числа не равны, так как оператор === ожидает равенства как по типу, так и по значению.

Или еще хуже. Объекты нельзя сравнивать:

```
var x = new Number(500);  
var y = new Number(500);
```

```
// (x == y) is false because objects cannot be  
compared
```

```
var x = 500;  
var y = new Number(500);
```

```
// (x === y) is false because x and y have  
different types
```

**Обратите внимание на разницу между (x==y) и (x===y).**

**Сравнение двух объектов JavaScript всегда возвращает false.**

# Числовые методы

**Метод числа помогает работать с числами.**

Числовые методы и свойства

Примитивные значения (например, 3.14 или 2014), не могут иметь свойств и методов (потому что они не являются объектами).

Но в JavaScript методы и свойства также доступны для примитивных значений, потому что JavaScript рассматривает примитивные значения как объекты при выполнении методов и свойств.

## Метод ToString()

Метод toString() возвращает число в виде строки.

Все методы number могут использоваться для любого типа чисел (литералов, переменных или выражений):

```
var x = 123;
x.toString();           //
returns 123 from variable x
(123).toString();      //
returns 123 from literal 123
(100 + 23).toString(); //
returns 123 from expression 100 +
23
```

# Метод toExponential()

функция toExponential() возвращает строку с числом, округленным и записанным с помощью экспоненциальной нотации.

Параметр определяет количество символов за десятичной точкой:

```
var x = 9.656;  
x.toExponential(2); // returns 9.66e+0  
x.toExponential(4); // returns 9.6560e+0  
x.toExponential(6); // returns  
9.656000e+0
```

# Метод toFixed()

toFixed() возвращает строку с числом, записанным с указанным количеством десятичных знаков:

```
var x = 9.656;  
x.toFixed(0);           // returns 10  
x.toFixed(2);           // returns 9.66  
x.toFixed(4);           // returns 9.6560  
x.toFixed(6);           // returns 9.656000
```

toFixed(2) идеально подходит для работы с деньгами.

## Метод toPrecision()

toPrecision() возвращает строку с числом, записанным с заданной длиной:

```
var x = 9.656;  
x.toPrecision();       // returns  
9.656  
x.toPrecision(2);     // returns 9.7  
x.toPrecision(4);     // returns  
9.656  
x.toPrecision(6);     // returns  
9.65600
```

# Метод `valueOf()`

`valueOf()` возвращает число как число.

```
var x = 123;
x.valueOf();           // returns 123 from
variable x
(123).valueOf();      // returns 123 from
literal 123
(100 + 23).valueOf(); // returns 123 from
expression 100 + 23
```

В JavaScript число может быть примитивным значением (`typeof = число`) или объектом (`typeof = объект`).

Метод `valueOf()` используется внутри JavaScript для преобразования объектов `Number` в примитивные значения.

Нет смысла использовать его в своем коде.

***Все типы данных JavaScript имеют метод `valueOf()` и метод `toString()`.***

# Преобразование переменных в числа

Существует 3 метода на JavaScript, которые могут быть использованы для преобразования переменных в числа:

Метод `Number()`

Метод `parseInt()`

Метод `parseFloat()`

Эти методы являются не количественными, а глобальными методами JavaScript.

## Глобальные методы JavaScript

Глобальные методы JavaScript могут быть использованы на всех типах данных JavaScript.

Это наиболее актуальные методы при работе с числами:









**Спасибо**