**Java Script Numbers**

JavaScript Numbers

[❮ Previous](https://www.w3schools.com/js/js_string_templates.asp)[Next ❯](https://www.w3schools.com/js/js_number_methods.asp)

JavaScript has only one type of number. Numbers can be written with or without decimals.

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers1)

Extra large or extra small numbers can be written with scientific (exponent) notation:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers2)

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.  
  
This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

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

Integer Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

Example

let x = 999999999999999;   // x will be 999999999999999  
let y = 9999999999999999;  // y will be 10000000000000000

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate1)

The maximum number of decimals is 17.

Floating Precision

Floating point arithmetic is not always 100% accurate:

let x = 0.2 + 0.1;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate2)

To solve the problem above, it helps to multiply and divide:

let x = (0.2 \* 10 + 0.1 \* 10) / 10;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate3)

ADVERTISEMENT

Adding Numbers and Strings

WARNING !!

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

let x = 10;  
let y = 20;  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add)

If you add two strings, the result will be a string concatenation:

Example

let x = "10";  
let y = "20";  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings1)

If you add a number and a string, the result will be a string concatenation:

Example

let x = 10;  
let y = "20";  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings2)

If you add a string and a number, the result will be a string concatenation:

Example

let x = "10";  
let y = 20;  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings5)

A common mistake is to expect this result to be 30:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings3)

A common mistake is to expect this result to be 102030:

Example

let x = 10;  
let y = 20;  
let z = "30";  
let result = x + y + z;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings4)

The JavaScript interpreter works from left to right.

First 10 + 20 is added because x and y are both numbers.

Then 30 + "30" is concatenated because z is a string.

Numeric Strings

JavaScript strings can have numeric content:

let x = 100;         // x is a number  
  
let y = "100";       // y is a string

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

let x = "100";  
let y = "10";  
let z = x / y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string1)

This will also work:

let x = "100";  
let y = "10";  
let z = x \* y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string2)

And this will work:

let x = "100";  
let y = "10";  
let z = x - y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string3)

But this will not work:

let x = "100";  
let y = "10";  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string4)

In the last example JavaScript uses the + operator to concatenate the strings.

NaN - Not a Number

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

Example

let x = 100 / "Apple";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_string)

However, if the string contains a numeric value , the result will be a number:

Example

let x = 100 / "10";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_number)

You can use the global JavaScript function isNaN() to find out if a value is a not a number:

Example

let x = 100 / "Apple";  
isNaN(x);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_isnan_true)

Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN:

Example

let x = NaN;  
let y = 5;  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_math)

Or the result might be a concatenation like NaN5:

Example

let x = NaN;  
let y = "5";  
let z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_concat)

NaN is a number: typeof NaN returns number:

Example

typeof NaN;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_typeof)

Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

Example

let myNumber = 2;  
// Execute until Infinity  
while (myNumber != Infinity) {  
  myNumber = myNumber \* myNumber;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity)

Division by 0 (zero) also generates Infinity:

Example

let x =  2 / 0;  
let y = -2 / 0;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity_zero)

Infinity is a number: typeof Infinity returns number.

Example

typeof Infinity;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity_typeof)

Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

Example

let x = 0xFF;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_hex)

Never write a number with a leading zero (like 07).  
Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

By default, JavaScript displays numbers as **base 10** decimals.

But you can use the toString() method to output numbers from **base 2** to **base 36**.

Hexadecimal is **base 16**. Decimal is **base 10**. Octal is **base 8**. Binary is **base 2**.

Example

let myNumber = 32;  
myNumber.toString(32);  
myNumber.toString(16);  
myNumber.toString(12);  
myNumber.toString(10);  
myNumber.toString(8);  
myNumber.toString(2);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_tostring)

JavaScript Numbers as Objects

Normally JavaScript numbers are primitive values created from literals:

let x = 123;

But numbers can also be defined as objects with the keyword new:

let y = new Number(123);

Example

let x = 123;  
let y = new Number(123);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object)

Do not create Number objects.

The new keyword complicates the code and slows down execution speed.

Number Objects can produce unexpected results:

When using the == operator, x and y are **equal**:

let x = 500;  
let y = new Number(500);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object1)

When using the === operator, x and y are **not equal**.

let x = 500;  
let y = new Number(500);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object2)

Note the difference between (x==y) and (x===y).

(x == y) true or false?

let x = new Number(500);  
let y = new Number(500);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object3)

(x === y) true or false?

let x = new Number(500);  
let y = new Number(500);

# JavaScript Number Methods

[❮ Previous](https://www.w3schools.com/js/js_numbers.asp)[Next ❯](https://www.w3schools.com/js/js_arrays.asp)

Number methods help you work with numbers.

## Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

## The toString() Method

The toString() method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

### Example

let x = 123;  
x.toString();  
(123).toString();  
(100 + 23).toString();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tostring)

## The toExponential() Method

toExponential() returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

### Example

let x = 9.656;  
x.toExponential(2);  
x.toExponential(4);  
x.toExponential(6);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toexponential)

The parameter is optional. If you don't specify it, JavaScript will not round the number.

ADVERTISEMENT

## The toFixed() Method

toFixed() returns a string, with the number written with a specified number of decimals:

### Example

let x = 9.656;  
x.toFixed(0);  
x.toFixed(2);  
x.toFixed(4);  
x.toFixed(6);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tofixed)

toFixed(2) is perfect for working with money.

## The toPrecision() Method

toPrecision() returns a string, with a number written with a specified length:

### Example

let x = 9.656;  
x.toPrecision();  
x.toPrecision(2);  
x.toPrecision(4);  
x.toPrecision(6);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toprecision)

## The valueOf() Method

valueOf() returns a number as a number.

### Example

let x = 123;  
x.valueOf();  
(123).valueOf();  
(100 + 23).valueOf();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_valueof)

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a valueOf() and a toString() method.

## Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

* The Number() method
* The parseInt() method
* The parseFloat() method

These methods are not **number** methods, but **global** JavaScript methods.

## Global JavaScript Methods

JavaScript global methods can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Number() | Returns a number, converted from its argument. |
| parseFloat() | Parses its argument and returns a floating point number |
| parseInt() | Parses its argument and returns an integer |

## The Number() Method

Number() can be used to convert JavaScript variables to numbers:

### Example

Number(true);  
Number(false);  
Number("10");  
Number("  10");  
Number("10  ");  
Number(" 10  ");  
Number("10.33");  
Number("10,33");  
Number("10 33");  
Number("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number)

If the number cannot be converted, NaN (Not a Number) is returned.

## The Number() Method Used on Dates

Number() can also convert a date to a number.

### Example

Number(new Date("1970-01-01"))

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number_date_0)

The Number() method returns the number of milliseconds since 1.1.1970.

The number of milliseconds between 1970-01-02 and 1970-01-01 is 86400000:

### Example

Number(new Date("1970-01-02"))

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number_date_1)

### Example

Number(new Date("2017-09-30"))

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number_date)

## The parseInt() Method

parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

### Example

parseInt("-10");  
parseInt("-10.33");  
parseInt("10");  
parseInt("10.33");  
parseInt("10 20 30");  
parseInt("10 years");  
parseInt("years 10");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parseint)

If the number cannot be converted, NaN (Not a Number) is returned.

## The parseFloat() Method

parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:

### Example

parseFloat("10");  
parseFloat("10.33");  
parseFloat("10 20 30");  
parseFloat("10 years");  
parseFloat("years 10");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parsefloat)

If the number cannot be converted, NaN (Not a Number) is returned.

## Number Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| MAX\_VALUE | Returns the largest number possible in JavaScript |
| MIN\_VALUE | Returns the smallest number possible in JavaScript |
| POSITIVE\_INFINITY | Represents infinity (returned on overflow) |
| NEGATIVE\_INFINITY | Represents negative infinity (returned on overflow) |
| NaN | Represents a "Not-a-Number" value |

## JavaScript MIN\_VALUE and MAX\_VALUE

MAX\_VALUE returns the largest possible number in JavaScript.

### Example

let x = Number.MAX\_VALUE;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_max)

MIN\_VALUE returns the lowest possible number in JavaScript.

### Example

let x = Number.MIN\_VALUE;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_min)

## JavaScript POSITIVE\_INFINITY

### Example

let x = Number.POSITIVE\_INFINITY;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_pos_infinity)

POSITIVE\_INFINITY is returned on overflow:

### Example

let x = 1 / 0;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_pos_infinity2)

## JavaScript NEGATIVE\_INFINITY

### Example

let x = Number.NEGATIVE\_INFINITY;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_neg_infinity)

NEGATIVE\_INFINITY is returned on overflow:

### Example

let x = -1 / 0;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_neg_infinity2)

## JavaScript NaN - Not a Number

### Example

let x = Number.NaN;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_nan)

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

### Example

let x = 100 / "Apple";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_string)

## Number Properties Cannot be Used on Variables

Number properties belongs to the JavaScript's number object wrapper called **Number**.

These properties can only be accessed as Number.MAX\_VALUE.

Using myNumber.MAX\_VALUE, where myNumber is a variable, expression, or value, will return undefined:

### Example

let x = 6;  
x.MAX\_VALUE

# JavaScript Random

[❮ Previous](https://www.w3schools.com/js/js_math.asp)[Next ❯](https://www.w3schools.com/js/js_booleans.asp)

## Math.random()

Math.random() returns a random number between 0 (inclusive),  and 1 (exclusive):

### Example

// Returns a random number:  
Math.random();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random)

Math.random() always returns a number lower than 1.

## JavaScript Random Integers

Math.random() used with Math.floor() can be used to return random integers.

There is no such thing as JavaScript integers.

We are talking about numbers with no decimals here.

### Example

// Returns a random integer from 0 to 9:  
Math.floor(Math.random() \* 10);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_0_9)

### Example

// Returns a random integer from 0 to 10:  
Math.floor(Math.random() \* 11);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_0_10)

### Example

// Returns a random integer from 0 to 99:  
Math.floor(Math.random() \* 100);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_0_99)

### Example

// Returns a random integer from 0 to 100:  
Math.floor(Math.random() \* 101);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_0_100)

### Example

// Returns a random integer from 1 to 10:  
Math.floor(Math.random() \* 10) + 1;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_1_10)

### Example

// Returns a random integer from 1 to 100:  
Math.floor(Math.random() \* 100) + 1;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_1_100)

ADVERTISEMENT

## A Proper Random Function

As you can see from the examples above, it might be a good idea to create a proper random function to use for all random integer purposes.

This JavaScript function always returns a random number between min (included) and max (excluded):

### Example

function getRndInteger(min, max) {  
  return Math.floor(Math.random() \* (max - min) ) + min;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_function)

This JavaScript function always returns a random number between min and max (both included):

### Example

function getRndInteger(min, max) {  
  return Math.floor(Math.random() \* (max - min + 1) ) + min;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_random_function2)