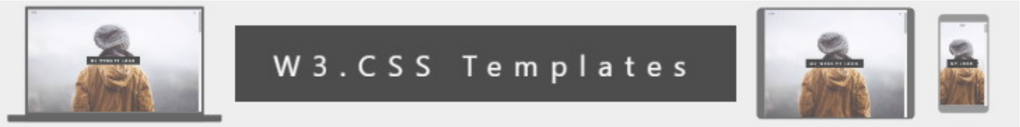
w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE



# JavaScript Numbers

❮ Previous Next ❯

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

### Example

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

Try it yourself »

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

### Example

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

Try it yourself »

## 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:

| | | |
|---|---|---|
| 52 bits (0 - 51) | 11 bits (52 - 62) | 1 bit (63) |

## Precision

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

### Example

```
var x = 999999999999999;   // x will be 999999999999999
var y = 9999999999999999;  // y will be 10000000000000000
```

Try it Yourself »

The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

### Example

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

Try it yourself »

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

### Example

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

Try it Yourself »

## Adding Numbers and Strings

WARNING !!

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

### Example

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

Try it Yourself »

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

### Example

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

Try it Yourself »

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

### Example

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

Try it Yourself »

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

### Example

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

Try it Yourself »

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

### Example

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

Try it Yourself »

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

### Example

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

Try it Yourself »

The JavaScript compiler 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:

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

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

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

This will work:

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

Try it Yourself »

This will also work:

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

Try it Yourself »

And this will work:

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

Try it Yourself »

But this will not work:

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

Try it Yourself »

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

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

Try it Yourself »

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

### Example

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

You can use the global JavaScript function isNaN() to find out if a value is a number:

## Example

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

Try it Yourself »

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

## Example

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

Try it Yourself »

Or the result might be a concatenation:

## Example

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

Try it Yourself »

NaN is a number: typeof NaN returns number:

## Example

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

Try it Yourself »

# Infinity

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

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

Try it yourself »

Division by 0 (zero) also generates Infinity:

### Example

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

Try it Yourself »

Infinity is a number: typeof Infinity returns number.

### Example

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

Try it Yourself »

## Hexadecimal

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

### Example

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

Try it Yourself »

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 as base 16 (hex), base 8 (octal), or base 2 (binary).

```
var myNumber = 128;
myNumber.toString(16);  // returns 80
myNumber.toString(8);   // returns 200
myNumber.toString(2);   // returns 10000000
```

Try it Yourself »

## Numbers Can be Objects

Normally JavaScript numbers are primitive values created from literals:

**var x = 123;**

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

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

### Example

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

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

Try it yourself »

Do not create Number objects. It slows down execution speed.
The **new** keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal numbers are equal:

### Example

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

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

Try it Yourself »

When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

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

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

Try it Yourself »

Or even worse. Objects cannot be compared:

### Example

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

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

Try it Yourself »

Note the difference between (x==y) and (x===y).
Comparing two JavaScript objects will always return false.

## Test Yourself with Exercises!

Exercise 1 » Exercise 2 » Exercise 3 » Exercise 4 »

❮ Previous Next ❯