# Number

**`Number`** is a primitive wrapper object used to represent and manipulate numbers like `37` or `-9.25`.

The `Number` constructor contains constants and methods for working with numbers. Values of other types can be converted to numbers using the `Number()` function.

The JavaScript `Number` type is a double-precision 64-bit binary format IEEE 754 value, like `double` in Java or C#. This means it can represent fractional values, but there are some limits to what it can store. A `Number` only keeps about 17 decimal places of precision; arithmetic is subject to rounding. The largest value a `Number` can hold is about 1.8E308. Values higher than that are replaced with the special `Number` constant `Infinity`.

A number literal like `37` in JavaScript code is a floating-point value, not an integer. There is no separate integer type in common everyday use. (JavaScript now has a `BigInt` type, but it was not designed to replace Number for everyday uses. `37` is still a `Number`, not a BigInt.)

`Number` may also be expressed in literal forms like `0b101`, `0o13`, `0x0A`. Learn more on numeric lexical grammar here.

## Description

When used as a function, `Number(value)` converts a string or other value to the Number type. If the value can't be converted, it returns `NaN`.

### Literal syntax

```
123    // one-hundred twenty-three
123.0  // same
123 === 123.0  // true
```

### Function syntax

Function syntax

```
Number('123')  // returns the number 123
Number('123') === 123  // true


Number("unicorn")  // NaN
Number(undefined)  // NaN
```

## Constructor

`Number()`

Creates a new `Number` value.

## Static properties

`Number.EPSILON`

The smallest interval between two representable numbers.

`Number.MAX_SAFE_INTEGER`

The maximum safe integer in JavaScript (`2^53 - 1`).

`Number.MAX_VALUE`

The largest positive representable number.

`Number.MIN_SAFE_INTEGER`

The minimum safe integer in JavaScript (`-(2^53 - 1)`).

`Number.MIN_VALUE`

The smallest positive representable number—that is, the positive number closest to zero (without actually being zero).

`Number.NaN`

Special "**N**ot **a** **N**umber" value.

`Number.NEGATIVE_INFINITY`

Special value representing negative infinity. Returned on overflow.

`Number.POSITIVE_INFINITY`

Special value representing infinity. Returned on overflow.

**`Number.prototype`**

Allows the addition of properties to the `Number` object.

## Static methods

**`Number.isNaN()`**

Determine whether the passed value is `NaN`.

**`Number.isFinite()`**

Determine whether the passed value is a finite number.

**`Number.isInteger()`**

Determine whether the passed value is an integer.

**`Number.isSafeInteger()`**

Determine whether the passed value is a safe integer (number between `-(2^53 - 1)` and `2^53 - 1`).

**`Number.parseFloat(string)`**

This is the same as the global `parseFloat()` function.

**`Number.parseInt(string, [radix])`**

This is the same as the global `parseInt()` function.

## Instance methods

**`Number.prototype.toExponential(fractionDigits)`**

Returns a string representing the number in exponential notation.

**`Number.prototype.toFixed(digits)`**

Returns a string representing the number in fixed-point notation.

**`Number.prototype.toLocaleString([locales [, options]])`**

Returns a string with a language sensitive representation of this number. Overrides the `Object.prototype.toLocaleString()` method.

**`Number.prototype.toPrecision(*precision*)`**

Returns a string representing the number to a specified precision in fixed-point or exponential notation.

**`Number.prototype.toString([*radix*])`**

Returns a string representing the specified object in the specified *radix* ("base"). Overrides the `Object.prototype.toString()` method.

**`Number.prototype.valueOf()`**

Returns the primitive value of the specified object. Overrides the `Object.prototype.valueOf()` method.

## Examples

### Using the Number object to assign values to numeric variables

The following example uses the `Number` object's properties to assign values to several numeric variables:

```
const biggestNum     = Number.MAX_VALUE
const smallestNum    = Number.MIN_VALUE
const infiniteNum    = Number.POSITIVE_INFINITY
const negInfiniteNum = Number.NEGATIVE_INFINITY
const notANum        = Number.NaN
```

### Integer range for Number

The following example shows the minimum and maximum integer values that can be represented as `Number` object. (More details on this are described in the ECMAScript standard, chapter *6.1.6 The Number Type* .)

```
const biggestInt  = Number.MAX_SAFE_INTEGER  //  (2**53 - 1) =>  900719925
const smallestInt = Number.MIN_SAFE_INTEGER  // -(2**53 - 1) => -900719925
```

When parsing data that has been serialized to JSON, integer values falling outside of this range can be expected to become corrupted when JSON parser coerces them to `Number` type.

A possible workaround is to use `String` instead.

A possible workaround is to use `BigInt` instead.

Larger numbers can be represented using the `BigInt` type.

### Using Number to convert a Date object

The following example converts the `Date` object to a numerical value using `Number` as a function:

```
let d = new Date('December 17, 1995 03:24:00')
console.log(Number(d))
```

This logs `819199440000`.

### Convert numeric strings and null to numbers

```
Number('123')     // 123
Number('123') === 123 // true
Number('12.3')    // 12.3
Number('12.00')   // 12
Number('123e-1')  // 12.3
Number('')        // 0
Number(null)      // 0
Number('0x11')    // 17
Number('0b11')    // 3
Number('0o11')    // 9
Number('foo')     // NaN
Number('100a')    // NaN
Number('-Infinity') // -Infinity
```

## Specifications

| Specification |
| --- |
| ECMAScript Language Specification (ECMAScript)<br># sec-number-objects |

## Browser compatibility

Report problems with this compatibility data on GitHub