JavaScript

JavaScript Date Objects: -

JavaScript Date Objects let us work with dates.

const d = new Date(); // Fri Feb 21 2025 13:59:36 GMT+0530 (India Standard Time)

const d = new Date("2025-02-20");

// Fri Feb 20 2025 05:30:00 GMT+0530 (India Standard Time)

Note: -

Date objects are static. The "clock" is not "running".

The computer clock is ticking, date objects are not.

JavaScript Date Output: -

By default, JavaScript will use the browser's time zone and display a date as a full text string.

9 ways to create a new date object.

new Date(): -

const d = new Date(); // Fri Feb 21 2025 14:06:17 GMT+0530 (India Standard Time)

new Date(date string): -

const d = new Date("October 13, 2014 11:13:00");

// Mon Oct 13 2014 11:13:00 GMT+0530 (India Standard Time)

const d = new Date("2022-03-25");

// Fri Mar 25 2022 05:30:00 GMT+0530 (India Standard Time)

new Date(year,month): -

new Date(year,month,day): -

new Date(year,month,day,hours): -

new Date(year,month,day,hours,minutes): -

new Date(year,month,day,hours,minutes,seconds): -

new Date(year,month,day,hours,minutes,seconds,ms): -

const d = new Date(2018, 11, 24, 10, 33, 30, 0);

// Mon Dec 24 2018 10:33:30 GMT+0530 (India Standard Time)

JavaScript counts months from 0 to 11:

January = 0, December = 11

Specifying a month higher than 11, will not result in an error but add the overflow to the next year.

Specifying a day higher than max, will not result in an error but add the overflow to the next month.

Using 6, 4, 3, or 2 Numbers: -

const d = new Date(2018, 11, 24, 10, 33, 30);

// Mon Dec 24 2018 10:33:30 GMT+0530 (India Standard Time)

const d = new Date(2018, 11, 24, 10, 33);

// Mon Dec 24 2018 10:33:00 GMT+0530 (India Standard Time)

const d = new Date(2018, 11, 24, 10);

// Mon Dec 24 2018 10:00:00 GMT+0530 (India Standard Time)

const d = new Date(2018, 11, 24);

Mon Dec 24 2018 00:00:00 GMT+0530 (India Standard Time)

const d = new Date(2018, 11);

// Sat Dec 01 2018 00:00:00 GMT+0530 (India Standard Time)

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

const d = new Date(2018);

// Thu Jan 01 1970 05:30:02 GMT+0530 (India Standard Time)

One and two digit years will be interpreted as 19xx

const d = new Date(99, 11, 24);

// Fri Dec 24 1999 00:00:00 GMT+0530 (India Standard Time)

JavaScript Stores Dates as Milliseconds

Zero time is January 01, 1970 00:00:00 UTC

One day (24 hours) is 86 400 000 milliseconds.

const d = new Date(100000000000);

// Sat Mar 03 1973 15:16:40 GMT+0530 (India Standard Time)

const d = new Date(-100000000000);

// Mon Oct 31 1966 19:43:20 GMT+0530 (India Standard Time)

Date Methods: -

Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of date objects, using either local time or UTC (universal, or GMT) time.

The **toDateString()** method converts a date to a more readable format.

```
const d = new Date();
d.toDateString(); // Fri Feb 21 2025
```

The **toUTCString()** method converts a date to a string using the UTC standard.

```
d.toUTCString(); // Fri, 21 Feb 2025 08:55:20 GMT
```

The **toISOString()** method converts a date to a string using the ISO standard.

```
d.toISOString(); // 2025-02-21T08:56:02.627Z
```

JavaScript Date Input: -

ISO Date - "2015-03-25" (The International Standard)

Short Date - "03/25/2015"

Long Date - "Mar 25 2015" or "25 Mar 2015"

The ISO format follows a strict standard in JavaScript.

The other formats are not so well defined and might be browser specific.

The ISO 8601 syntax (YYYY-MM-DD) is also the preferred JavaScript date format

```
const d = new Date("2015-03-25");
```

Wed Mar 25 2015 05:30:00 GMT+0530 (India Standard Time)

ISO dates can be written without specifying the day

```
const d = new Date("2015-03");
```

Sun Mar 01 2015 05:30:00 GMT+0530 (India Standard Time)

ISO dates can be written without month and day (YYYY)

```
const d = new Date("2015");
// Thu Jan 01 2015 05:30:00 GMT+0530 (India Standard Time)
```

ISO Dates (Date-Time): -

ISO dates can be written with added hours, minutes, and seconds.

(YYYY-MM-DDTHH:MM:SSZ)

```
const d = new Date("2015-03-25T12:00:00Z");
// Wed Mar 25 2015 17:30:00 GMT+0530 (India Standard Time)
```

Date and time is separated with a capital T.

UTC time is defined with a capital letter Z.

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead.

const d = new Date("2015-03-25T12:00:00-06:30");

// Thu Mar 26 2015 00:00:00 GMT+0530 (India Standard Time)

UTC (Universal Time Coordinated) is the same as GMT (Greenwich Mean Time)

Omitting T or Z in a date-time string can give different results in different browsers.

Time Zones: -

When setting a date, without specifying the time zone, JavaScript will use the browser's time zone.

When getting a date, without specifying the time zone, the result is converted to the browser's time zone.

JavaScript Short Dates: -

Short dates are written with an "MM/DD/YYYY" syntax like this

const d = new Date("03/25/2015");

// Wed Mar 25 2015 00:00:00 GMT+0530 (India Standard Time)

In some browsers, months or days with no leading zeroes may produce an error.

The behavior of "YYYY/MM/DD" is undefined.
Some browsers will try to guess the format. Some will return NaN.

The behavior of "DD-MM-YYYY" is also undefined.
Some browsers will try to guess the format. Some will return NaN.

JavaScript Long Dates: -

Long dates are most often written with a "MMM DD YYYY" syntax like this.

const d = new Date("Mar 25 2015");

const d = new Date("25 Mar 2015");

Month and day can be in any order. And, month can be written in full (January), or abbreviated (Jan)

Date Input - Parsing Dates: -

If you have a valid date string, you can use the Date.parse() method to convert it to milliseconds.

Date.parse() returns the number of milliseconds between the date and January 1, 1970

let msec = Date.parse("March 21, 2012"); // 1332268200000

then,

const d = new Date(msec); // Wed Mar 21 2012 00:00:00 GMT+0530 (India Standard Time)

JavaScript Get Date Methods: -

new Date() returns a date object with the current date and time.

The getFullYear() Method: -

The getFullYear() method returns the year of a date as a four digit number.

```
const d = new Date();
d.getFullYear(); // 2025
```

The getMonth() Method: -

The getMonth() method returns the month of a date as a number (0-11)

```
const d = new Date();
d.getMonth(); // add 1 to get correct result.
```

Also,

```
const months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"];

const d = new Date();

let month = months[d.getMonth()];
```

The getDate() Method: -

The getDate() method returns the day of a date as a number (1-31)

```
const d = new Date();
d.getDate(); // 21
```

The getHours() Method: -

The getHours() method returns the hours of a date as a number (0-23)

```
const d = new Date();
d.getHours(); //15
```

The getMinutes() Method: -

The getMinutes() method returns the minutes of a date as a number (0-59)

```
const d = new Date();
d.getMinutes(); // 14
```

The getSeconds() Method: -

The getSeconds() method returns the seconds of a date as a number (0-59)

```
const d = new Date();
d.getSeconds(); // 6
```

The getMilliseconds() Method: -

The getMilliseconds() method returns the milliseconds of a date as a number (0-999)

```
const d = new Date();
d.getMilliseconds(); // 527
```

The getDay() Method: -

The getDay() method returns the weekday of a date as a number (0-6)

In JavaScript, the first day of the week (day 0) is Sunday.

Some countries in the world consider the first day of the week to be Monday.

```
const d = new Date();
d.getDay(); // 5
```

You can use an array of names, and getDay() to return weekday as a name.

```
const days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"];

const d = new Date();

let day = days[d.getDay()]; // Friday
```

The getTime() Method: -

The getTime() method returns the number of milliseconds since January 1, 1970

```
const d = new Date("1970-01-01");
d.getTime(); // 0

const d = new Date();
d.getTime(); // 1740131350159
```

The Date.now() Method: -

Date.now() returns the number of milliseconds since January 1, 1970.

```
let ms = Date.now(); // 1740131395961
```

Calculate the number of years since 1970/01/01

```
const minute = 1000 * 60;

const hour = minute * 60;

const day = hour * 24;
```

const year = day * 365;

let years = Math.round(Date.now() / year); //55

The get methods above return Local time. The get methods return information from existing date objects.

UTC Date Get Methods: -

Method	Same As	Description
getUTCDate()	getDate()	Returns the UTC date
getUTCFullYear()	getFullYear()	Returns the UTC year
getUTCMonth()	getMonth()	Returns the UTC month
getUTCDay()	getDay()	Returns the UTC day
getUTCHours()	getHours()	Returns the UTC hour
getUTCMinutes()	getMinutes()	Returns the UTC minutes
getUTCSeconds()	getSeconds()	Returns the UTC seconds
getUTCMilliseconds()	getMilliseconds()	Returns the UTC milliseconds

The difference between Local time and UTC time can be up to 24 hours.

The getTimezoneOffset() Method: -

The getTimezoneOffset() method returns the difference (in minutes) between local time an UTC time.

let diff = d.getTimezoneOffset(); // -330

JavaScript Set Date Methods: -

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

Set Date Methods: -

If u use innerHTML property, the result will be a string and full date format. If not, you get the result in milliseconds.

The setFullYear() Method: -

```
const d = new Date("January 01, 2025");
d.setFullYear(2020);
```

// Wed Jan 01 2020 00:00:00 GMT+0530 (India Standard Time)

The setFullYear() method can optionally set month and day.

```
const d = new Date("January 01, 2025");
d.setFullYear(2020, 11, 3);
```

// Thu Dec 03 2020 00:00:00 GMT+0530 (India Standard Time)

The setMonth() Method: -

```
const d = new Date("January 01, 2025");
d.setMonth(11); // Mon Dec 01 2025 00:00:00 GMT+0530 (India Standard Time)
```

The setDate() Method: -

```
const d = new Date("January 01, 2025");
d.setDate(15); // Wed Jan 15 2025 00:00:00 GMT+0530 (India Standard Time)
```

The setDate() method can also be used to add days to a date.

```
d.setDate(d.getDate() + 50); // Thu Feb 20 2025 00:00:00 GMT+0530 (India Standard Time)
```

The setHours() Method: -

```
const d = new Date("January 01, 2025");
d.setHours(22); // Wed Jan 01 2025 22:00:00 GMT+0530 (India Standard Time)
```

The setHours() method can also be used to set minutes and seconds.

```
const d = new Date("January 01, 2025");
d.setHours(22, 10, 20); / / Wed Jan 01 2025 22:10:20 GMT+0530 (India Standard Time)
```

The setMinutes() Method: -

```
const d = new Date("January 01, 2025");
d.setMinutes(30); // Wed Jan 01 2025 00:30:00 GMT+0530 (India Standard Time)
```

The setSeconds() Method: -

```
const d = new Date("January 01, 2025");
d.setSeconds(30); // Wed Jan 01 2025 00:00:30 GMT+0530 (India Standard Time)
```

The setMilliseconds() Method: -

```
const d = new Date("January 01, 2025");

d.setMilliseconds(545); // 1735669800545
```

The setTime() Method: -

```
const d = new Date();

d.setTime(1609459200000); // 1609459200000
```

Compare Dates: -

```
let text = "";

const today = new Date();

const someday = new Date();

someday.setFullYear(2100, 0, 14);

if (someday > today) {
```

```
  text = "Today is before January 14, 2100.";
} else {
  text = "Today is after January 14, 2100.";
}
// Today is before January 14, 2100.
```

JS Numbers and BigInt: -

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

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

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.

Integer Precision: -

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

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

The maximum number of decimals is 17.

Floating Precision: -

Floating point arithmetic is not always 100% accurate.

```
let x = 0.2 + 0.1; // 0.30000000000000004
```

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

```
let x = (0.2 * 10 + 0.1 * 10) / 10; // 0.3
```

Adding Numbers and Strings: -

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

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

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

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

The JavaScript interpreter works from left to right.

Numeric Strings: -

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

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

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)

```
let x = 100 / "Apple"; // NaN
```

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

```
isNaN(x); // true or false
```

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

typeof NaN returns number

Infinity: -

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

Division by 0 (zero) also generates Infinity.

typeof Infinity returns number.

Hexadecimal: -

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

```
let x = 0xFF; //255
```

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.

```
let myNumber = 32;
myNumber.toString(16); //20
myNumber.toString(10); //32
```

myNumber.toString(8); //40
myNumber.toString(2); //100000

numbers can also be defined as objects with the keyword new.

let y = new Number(123);

Comparing two JavaScript objects always returns false.

BigInt: -

JavaScript can only safely represent integers Up to 9007199254740991 [+(2^{53}-1)] and down to -9007199254740991 [-(2^{53}-1)]. Integer values outside this range lose precision.

To create a BigInt, append n to the end of an integer or call BigInt()

let y = 9999999999999999n; // 9999999999999999

let x = 1234567890123456789012345n;
let y = BigInt(1234567890123456789012345)

typeof a BigInt is "bigint"

Operators that can be used on a JavaScript Number can also be used on a BigInt

Arithmetic between a BigInt and a Number is not allowed (type conversion lose information).

Unsigned right shift (>>>) can not be done on a BigInt (it does not have a fixed width).

A BigInt can not have decimals.

let x = 5n;
let y = x / 2;
// Error: Cannot mix BigInt and other types, use explicit conversion.

let x = 5n;
let y = Number(x) / 2; //2.5

BigInt can also be written in hexadecimal, octal, or binary notation.

let hex = 0x20000000000003n;
let oct = 0o400000000000000003n;
let bin = 0b100000000000000000000000000000000000000000000000000011n;

Minimum and Maximum Safe Integers

let x = Number.MAX_SAFE_INTEGER; // 9007199254740991

let x = Number.MIN_SAFE_INTEGER; // -9007199254740991

The Number.isInteger() Method

The Number.isInteger() method returns true if the argument is an integer.

```
Number.isInteger(10); //true
Number.isInteger(10.5); //false
```

The Number.isSafeInteger() Method: -

A safe integer is an integer that can be exactly represented as a double precision number.

The Number.isSafeInteger() method returns true if the argument is a safe integer.

```
Number.isSafeInteger(10); //true
Number.isSafeInteger(12345678901234567890); //false
```

Safe integers are all integers from $-(2^{53} - 1)$ to $+(2^{53} - 1)$.
This is safe: 9007199254740991. This is not safe: 9007199254740992.

JavaScript Number Methods: -

toString() Method: -

The toString() method returns a number as a string

```
let x = 123;
x.toString();
(123).toString();
(100 + 23).toString(); // all gives 123
```

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.

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

```
let x = 9.656;

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

toFixed() Method: -

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

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

toPrecision() Method: -

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

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

valueOf() Method: -

valueOf() returns a number as a number.

```
let x = 123;
x.valueOf();
(123).valueOf();
(100 + 23).valueOf(); // all give 123
```

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

Converting Variables to Numbers: -

The Number() Method: -

The Number() method can be used to convert JavaScript variables to numbers.

```
Number(true); //1

Number(false); //0

Number("10"); //10

Number("  10"); //10

Number("10  "); //10

Number(" 10  "); //10

Number("10.33"); //10.33

Number("10,33"); //NaN

Number("10 33"); //NaN

Number("John"); //NaN

Number(new Date("1970-01-01")) // 0 milliseconds
```

The parseInt() Method: -

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

```
parseInt("-10"); //-10
parseInt("-10.33"); //-10
parseInt("10"); //10
```

```
parseInt("10.33");   //10
parseInt("10 20 30"); //10
parseInt("10 years"); //10
parseInt("years 10"); //NaN
```

The parseFloat() Method: -

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

```
parseFloat("10"); //10
parseFloat("10.33"); //10.33
parseFloat("10 20 30"); //10
parseFloat("10 years"); //10
parseFloat("years 10"); // NaN
```

Number Object Methods: -

Method	Description
Number.isInteger()	Returns true if the argument is an integer
Number.isSafeInteger()	Returns true if the argument is a safe integer
Number.parseFloat()	Converts a string to a number
Number.parseInt()	Converts a string to a whole number

Number Methods Cannot be Used on Variables.

The number methods above belong to the JavaScript Number Object.

In the arguments, put input and get the result.

Number Properties: -

Number properties belong to the JavaScript Number Object.

Property	Description
EPSILON	The difference between 1 and the smallest number > 1.
MAX_VALUE	The largest number possible in JavaScript
MIN_VALUE	The smallest number possible in JavaScript
MAX_SAFE_INTEGER	The maximum safe integer (2^{53} - 1)
MIN_SAFE_INTEGER	The minimum safe integer -(2^{53} - 1)
POSITIVE_INFINITY	Infinity (returned on overflow)
NEGATIVE_INFINITY	Negative infinity (returned on overflow)
NaN	A "Not-a-Number" value

```
let x = Number.EPSILON; //2.220446049250313e-16

let x = Number.MAX_VALUE; //1.7976931348623157e+308

let x = Number.MIN_VALUE; // 5e-324
```

```
let x = Number.MAX_SAFE_INTEGER; //9007199254740991

let x = Number.MIN_SAFE_INTEGER; //-9007199254740991

let x = Number.POSITIVE_INFINITY; //Infinity

POSITIVE_INFINITY is returned on overflow //let x = 1 / 0;

let x = Number.NEGATIVE_INFINITY; //-Infinity

let x = Number.NaN; //NaN
```