# Number, Math & Date

# Number

# Numbers In JavaScript

In JavaScript, numbers represent both integer and floating-point values.

Unlike many other languages, JavaScript has only one number type:
Number. No separate types for integers or floats.

Number is technically a constructor function (also called a built-in object)
in JavaScript

```
let age = 25;           // Integer
let price = 19.99;      // Floating-point
let negative = -7;      // Negative number
```

# Literal Method v/s Constructor Method For Creating a Number:

```
let num = 131245;
let num1 = Number("21234");
let num2 = Number(12352);
```

Literal Method

```
let num2 = new Number(12352354);
```

Constructor Method

new Number() creates an object, not a primitive number.

This can lead to confusing bugs and unexpected behavior.

Always use number literals or Number(value) without new.

# Number Properties & Methods :

| Property | Description |
|---|---|
| Number.MAX_VALUE | Largest possible number |
| Number.MIN_VALUE | Smallest possible number (positive) |
| Number.POSITIVE_INFINITY | Infinity |
| Number.NEGATIVE_INFINITY | -Infinity |
| Number.NaN | "Not-a-Number" |
| Number.EPSILON | Smallest difference between numbers |
| Number.isNaN() | Checks if value is NaN |

| Method | Description |
|---|---|
| parseInt() | Parses string to integer |
| parseFloat() | Parses string to float |
| isNaN() | Checks if value is NaN |
| Number.isFinite() | Checks if number is finite |

# Number Instance Methods (Used on Number Primitives)

| Method | Description |
| --- | --- |
| .toFixed(n) | Formats number to n decimal places |
| .toExponential(n) | Converts to exponential notation |
| .toPrecision(n) | Formats to n total digits |
| .toString() | Converts number to string |
| .valueOf() | Returns primitive value of Number object |

# Math

# Math in JavaScript

JavaScript provides a built-in Math object for mathematical operations.

| Method | Description |
| --- | --- |
| Math.round(x) | Rounds to nearest integer |
| Math.floor(x) | Rounds down |
| Math.ceil(x) | Rounds up |
| Math.abs(x) | Absolute value |
| Math.sqrt(x) | Square root |
| Math.pow(x, y) | x to the power of y |
| Math.min(...args) | Smallest number |
| Math.max(...args) | Largest number |
| Math.random() | Random number between 0 (inclusive) and 1 (exclusive) |

# Generating Random Numbers In A Given Range

```
// Random number between 0 and 1
let rand = Math.random();

// Random integer between min and max (inclusive)
function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

console.log(getRandomInt(1, 10)); // Random integer 1 to 10
```

Note:

- Math.ceil() gives 0 only if Math.random() returns exactly 0 — but that's extremely rare.
- Math.floor(... + 1) is slightly more consistent in intent and avoids the edge case of 0.

# Date

# Dates In JavaScript

Fundamentals:

1. GMT (Greenwich Mean Time) Time Zone
2. UTC (Coordinated Universal Time)
3. ISO Standard (International Organization for Standardization)
4. Unix Time And Unix Epoch
5. How computers keep track of time?
6. Time Zones

## GMT (Greenwich Mean Time)

GMT stands for Greenwich Mean Time.

It was the first global standard for time.

## UTC (Coordinated Universal Time)

UTC is the modern standard for time measurement.
Similar to GMT, but more accurate (uses atomic clocks).

It’s the reference time zone used by computers and systems.

Time zones are usually written as offsets from UTC, like:UTC+1, UTC-4, etc.

# ISO Standard (International Organization for Standardization)

ISO 8601 is the international standard for date and time formats.

It helps avoid confusion caused by different formats (like dd/mm/yyyy vs mm/dd/yyyy).

The ISO format is:

```
YYYY-MM-DDTHH:mm:ss.sssZ
Example: 2025-05-19T14:30:00Z
```

- T separates date and time
- Z means UTC time

# Unix Time And Unix Epoch

Unix Epoch is the zero point for time in most computer systems, especially those based on Unix (including Linux, macOS, and modern operating systems).

So, The Unix Epoch is the starting point:

```
January 1, 1970, 00:00:00 UTC
```

Unix Time (or Epoch Time) = Number of seconds (or milliseconds) since that moment.

```
Date.now(); // gives milliseconds since epoch
```

Used in almost every programming language (including JavaScript).

# How computers keep track of time?

Computers don’t store dates in human-readable format.

They count time as numbers — specifically:

- Number of milliseconds or seconds since a fixed point in time.

This makes it easier to do calculations (like adding days or comparing dates).

# Time Zones

The Earth is divided into 24 time zones, based on the rotation of the planet.

Each time zone represents a region where the local time is the same.

Example:

- India (IST) = UTC +5:30
- London (GMT/UTC) = UTC +0
- New York (EST) = UTC -5

| Term | Meaning |
| --- | --- |
| **Time Zone** | Local time difference from UTC |
| **GMT** | Historical time standard |
| **UTC** | Global, accurate time reference |
| **ISO Format** | Standard format for date-time |
| **Unix Time** | Time counted from Jan 1, 1970 |
| **Epoch** | Starting reference time for systems |

# Date In JavaScript

## 1. Creating a Date

```
// ✅ 1. No Arguments (Current Date & Time)
let now = new Date(); // Current date and time

// ✅ 2. Date String (ISO or other formats)
let date1 = new Date("2025-05-19T12:00:00"); // ISO format (recommended)
let date2 = new Date("May 19, 2025 12:00:00"); // Long format
let date3 = new Date("2025/05/19 12:00:00");   // Slash format (less reliable)

// ✅ 3. Numbers: new Date(year, monthIndex, day, hours, minutes, seconds, ms)
let date4 = new Date(2025, 4, 19, 12, 0, 0); // May 19, 2025, 12:00:00

// ✅ 4. Milliseconds Since Epoch (Unix timestamp)
let date5 = new Date(1747632000000); // Milliseconds since Jan 1, 1970

// ✅ 5. Copy Another Date
let original = new Date("2025-05-19");
let copy = new Date(original);
```

## 2. Getting Parts of the Date

```
date.getFullYear(); // 2025
date.getMonth(); // 0-11 (0 = January)
date.getDate(); // 1-31
date.getDay(); // 0-6 (0 = Sunday)
date.getHours(); // 0-23
date.getMinutes(); // 0-59
date.getSeconds(); // 0-59
date.getTimezoneOffset() // UTC - local time
date.getMilliseconds() // 0-999 (milliseconds portion of the current second)
```

## 3. Setting Parts of the Date

```
now.setFullYear(2024);
now.setMonth(0); // January
now.setDate(1);
```

## 4. Working with Timestamps

```
Date.now(); // milliseconds since Jan 1, 1970
let timestamp = new Date().getTime();
```

## 5. Formatting Dates

```
date.toISOString();
date.toString();
date.toDateString();
date.toTimeString();
date.toLocaleString();
date.toLocaleDateString();
date.toLocaleTimeString();
```

Custom Formatting (like dd/mm/yyyy) :

```
let d = new Date();
let formatted = `${d.getDate()}/${d.getMonth()+1}/${d.getFullYear()}`;
```

| Method | What it returns |
|---|---|
| toISOString() | Standard UTC format (ISO 8601) |
| toString() | Full date & time string |
| toDateString() | Date only (weekday, month, year) |
| toTimeString() | Time only with time zone |
| toLocaleDateString() | Local date format (e.g., DD/MM/YYYY) |
| toLocaleTimeString() | Local time format (e.g., 12-hour) |

Notes:

- Months are zero-indexed in JS: 0 = January, 11 = December.
- JS Date is based on UTC internally, but displays in local time.
- Date calculations can be tricky—consider libraries like Luxon, Day.js, or date-fns for advanced use.

# Calculations On Time Stamps

## 1. What Is a Timestamp?

- A timestamp is the number of milliseconds since the Unix Epoch (Jan 1, 1970 UTC).
- In JavaScript, Date objects internally store time as timestamps

## 2. Getting the Current Timestamp

- Use Date.now() to get the current timestamp:
- const now = Date.now(); // milliseconds since Jan 1, 1970

## 3. Convert Date to Timestamp

Use getTime() on a Date object:

```
const date = new Date();
const timestamp = date.getTime();
```

## 4. Convert Timestamp to Date

Pass the timestamp to the Date constructor:

```
const date = new Date(17162126678533);
console.log(date)
console.log(date.toLocaleString());
```

## 5. Adding or Subtracting Time

Timestamps are in milliseconds, so you can add or subtract directly:

- 1 second = 1000 ms
- 1 minute = 60 * 1000 ms
- 1 hour = 60 * 60 * 1000 ms

Example:

```
const oneHourLater = Date.now() + (60 * 60 * 1000);
```

## 6. Calculating Differences Between Dates

Subtract timestamps to find duration:

```
const start = new Date('2025-05-20T10:00:00Z');
const end = new Date('2025-05-20T12:00:00Z');
const diffMs = end - start; // in milliseconds
const diffHours = diffMs / (1000 * 60 * 60);
```

## 7. Converting Milliseconds to Human Units

Convert milliseconds into readable time:

```
const ms = 5405000;
const minutes = Math.floor(ms / 1000 / 60);
const seconds = ((ms % 60000) / 1000);
console.log(`${minutes} min ${seconds} sec`)
```

# Locale & Options

```
const date = new Date();

date.toLocaleString(locale, options) // Full date + time
date.toLocaleDateString(locale, options) // Date only
date.toLocaleTimeString(locale, options) // Time only
// locale --> en-IN, en-GB, fr-Fr, hi-En
```

| Option | Possible Values |
|---|---|
| weekday | "long", "short", "narrow" |
| year | "numeric", "2-digit" |
| month | "numeric", "2-digit", "long", "short" |
| day | "numeric", "2-digit" |
| hour | "numeric", "2-digit" |
| minute | "numeric", "2-digit" |
| second | "numeric", "2-digit" |
| hour12 | true or false |
| timeZone | "UTC", "Asia/Kolkata", etc. |
| timeZoneName | "short", "long" |