# Sommario



## 1. typeof

`typeof` è un operatore unario in JavaScript che viene utilizzato per ottenere il tipo di una variabile o espressione. Restituisce una **stringa** che rappresenta il tipo del valore.

Esempi:

```
console.log(typeof 42);          // "number"
console.log(typeof 'Hello');     // "string"
console.log(typeof true);        // "boolean"
console.log(typeof undefined);   // "undefined"
console.log(typeof null);        // "object" (questo è un bug storico di
JavaScript, vedremo più sotto)
console.log(typeof {});          // "object"
console.log(typeof []);          // "object" (gli array sono considerati
oggetti)
console.log(typeof function(){}); // "function"
```

## 2. null

`null` è un valore primitivo in JavaScript che rappresenta **l'assenza intenzionale di un valore**. Quando una variabile è assegnata a `null`, significa che la variabile è stata definita, ma non ha un valore.

Esempi:

```
let a = null;
console.log(a);  // null
console.log(typeof a);  // "object" (un comportamento storico di JavaScript)
```

**Nota importante**: Sebbene `null` sia un tipo di valore primitivo, `typeof null` restituisce erroneamente "object". Questo è un bug che è rimasto nel linguaggio per motivi di compatibilità retroattiva.

## 3. `NaN` (Not a Number)

`NaN` è una costante speciale in JavaScript che rappresenta **un valore che non è un numero valido**. Può verificarsi quando si esegue un'operazione che non può produrre un risultato numerico valido.

Esempi:

```
console.log(0 / 0);          // NaN
console.log(Math.sqrt(-1)); // NaN
console.log('hello' / 2);   // NaN
```

**Tratto distintivo di `NaN`**:

- `NaN` **non è uguale a se stesso**. Per esempio:

  ```
  console.log(NaN === NaN);  // false
  ```

- Se vuoi verificare se un valore è `NaN`, devi usare la funzione `isNaN()`:

  ```
  console.log(isNaN(NaN));   // true
  console.log(isNaN(42));    // false
  ```

## 4. push()

Il metodo `push()` è un **metodo degli array** in JavaScript. Viene utilizzato per aggiungere uno o più elementi **alla fine di un array** e restituisce la nuova lunghezza dell'array.

Esempi:

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr);  // [1, 2, 3, 4]

arr.push(5, 6);
console.log(arr);  // [1, 2, 3, 4, 5, 6]

console.log(arr.push(7)); // 7, la nuova lunghezza dell'array
console.log(arr);         // [1, 2, 3, 4, 5, 6, 7]
```

Il metodo `push()` modifica l'array originale e restituisce il **nuovo numero di elementi** nell'array dopo l'aggiunta.

Gestire le date in JavaScript può sembrare complicato inizialmente, ma JavaScript fornisce strumenti potenti per lavorare con il tempo e le date. La classe principale che si utilizza per gestire le date in JavaScript è **`Date`**. Principali operazioni che si possono fare con le date in JavaScript.

## 5. Creare un Oggetto `Date`

Puoi creare un oggetto `Date` in vari modi.

### a. Creazione di una data corrente

Se vuoi ottenere la data e l'ora correnti, basta creare un nuovo oggetto `Date` senza parametri:

```
const now = new Date();
console.log(now);  // Es. "2024-12-02T13:45:30.000Z"
```

### b. Creazione di una data con una stringa

Puoi creare un oggetto `Date` passando una stringa che rappresenta una data, nel formato `YYYY-MM-DD` o altri formati ISO 8601:

```
const dateString = new Date('2024-12-01');
console.log(dateString);  // Es. "2024-12-01T00:00:00.000Z"
```

### c. Creazione di una data con parametri specifici

Puoi anche specificare anno, mese, giorno, ora, minuto, secondo e millisecondo passando questi valori nel costruttore `Date(year, month, day, hours, minutes, seconds, milliseconds)`:

```
const specificDate = new Date(2024, 11, 25, 12, 30, 0);  // 2024-12-25 12:30:00
console.log(specificDate);  // Es. "2024-12-25T11:30:00.000Z"
```

**Nota**: In JavaScript, i mesi vanno da `0` (gennaio) a `11` (dicembre), quindi `11` rappresenta dicembre.

## 6. Ottenere Parti della Data

Una volta che hai un oggetto `Date`, puoi ottenere diverse informazioni sulla data, come l'anno, il mese, il giorno, l'ora, ecc.

```
const now = new Date();

// Anno
console.log(now.getFullYear());  // 2024

// Mese (da 0 a 11, quindi aggiungere 1 per il mese reale)
console.log(now.getMonth());     // 11 (Dicembre) se oggi è dicembre

// Giorno del mese
console.log(now.getDate());      // 2 (Se oggi è il 2 dicembre)

// Giorno della settimana (0 = Domenica, 1 = Lunedì, ...)
console.log(now.getDay());       // 1 (Lunedì)

// Ore
console.log(now.getHours());     // 13 (ora corrente)
```

```
// Minuti
console.log(now.getMinutes());   // 45 (minuti correnti)

// Secondi
console.log(now.getSeconds());   // 30 (secondi correnti)
```

## 7. Modificare una Data

Puoi anche modificare le parti di una data, come anno, mese, giorno, ecc., utilizzando i metodi `set` corrispondenti.

```
const date = new Date();

// Impostare il nuovo anno
date.setFullYear(2025);
console.log(date);  // Es. "2025-12-02T13:45:30.000Z"

// Impostare il nuovo mese
date.setMonth(5);  // 5 rappresenta giugno (mese 6)
console.log(date);  // Es. "2025-06-02T13:45:30.000Z"

// Impostare il giorno
date.setDate(15);
console.log(date);  // Es. "2025-06-15T13:45:30.000Z"
```

## 8. Confrontare le Date

Le date in JavaScript sono rappresentate come oggetti di tipo `Date`, ma possono essere confrontate direttamente, perché vengono automaticamente convertite in numeri (in millisecondi dal 1 gennaio 1970):

```
const date1 = new Date(2024, 11, 2);
const date2 = new Date(2024, 11, 2);

console.log(date1 === date2);   // false (perché sono oggetti diversi)
console.log(date1.getTime() === date2.getTime());  // true (confronto in
millisecondi)
```

Puoi anche confrontare due date usando `getTime()` o confrontando direttamente gli oggetti `Date` in base ai loro millisecondi.

## 9. Formattare la Data

JavaScript non ha un metodo nativo potente per formattare le date (come in altri linguaggi), ma puoi usare il metodo `toLocaleDateString()` per formattare le date in modo leggibile:

```
const now = new Date();

// Formattare la data secondo le impostazioni locali
console.log(now.toLocaleDateString());  // "12/2/2024" (o formato simile in base
alla località)

// Personalizzare il formato
console.log(now.toLocaleDateString('it-IT'));  // "02/12/2024" (formato
italiano)
```

Puoi anche usare `toLocaleString()` per ottenere la data e l'ora:

```
console.log(now.toLocaleString());  // "02/12/2024, 13:45:30" (formato
dipendente dalla località)
```

Per formattazioni più complesse, puoi usare librerie come **`date-fns`** o **`moment.js`** (sebbene `moment.js` sia stato deprecato, è ancora molto usato).

## 10. Operazioni con le Date

Puoi fare operazioni sulle date come sommare o sottrarre giorni, mesi o anni. Tuttavia, JavaScript non fornisce metodi diretti per farlo, quindi dovrai manipolare manualmente i valori. Ecco un esempio di come sommare giorni a una data:

```
const now = new Date();
now.setDate(now.getDate() + 5);  // Aggiungi 5 giorni alla data corrente
console.log(now);
```

## 11. Timestamp e Millisecondi

Le date in JavaScript sono rappresentate come **timestamp**, che è il numero di millisecondi trascorsi dal 1 gennaio 1970 (Epoch Time).

- **Ottenere il timestamp corrente**:

  ```
  const timestamp = Date.now();
  console.log(timestamp);  // Es. 1638495092830
  ```

- **Creare una data da un timestamp**:

  ```
  const timestamp = 1638495092830;
  const dateFromTimestamp = new Date(timestamp);
  console.log(dateFromTimestamp);
  ```

## 12. Librerie per una gestione avanzata delle date

Se vuoi lavorare con date in modo più robusto, puoi usare librerie come:

- **`date-fns`**: una libreria moderna per la gestione delle date.
- **`luxon`**: una libreria potente per lavorare con date e orari.
- **`moment.js`**: sebbene deprecata, è ancora molto utilizzata.

## 13. Riepilogo:

- **Creazione di una data**: Usa `new Date()`.
- **Ottenere valori da una data**: Usa metodi come `getFullYear()`, `getMonth()`, `getDate()`.
- **Modificare una data**: Usa metodi come `setFullYear()`, `setMonth()`, `setDate()`.
- **Formattare le date**: Usa `toLocaleDateString()`, `toLocaleString()`.
- **Operazioni con le date**: Aggiungi o sottrai giorni o mesi manualmente.