



# Funzioni in JS

*Programmazione*

**Le funzioni in JavaScript rappresentano uno dei costrutti fondamentali del linguaggio, poiché permettono di racchiudere blocchi di codice in unità riutilizzabili e organizzate.**

**Una funzione può essere richiamata più volte in punti diversi del programma, evitando ripetizioni e favorendo la modularità.**

**Inoltre, le funzioni consentono di astrarre operazioni complesse dietro un nome semplice e significativo, migliorando la leggibilità e la manutenzione del codice.**

**In JavaScript, le funzioni sono considerate “first-class citizens”, ovvero entità che possono essere assegnate a variabili, passate come parametri ad altre funzioni o restituite come risultato, caratteristica che rende il linguaggio estremamente flessibile e potente.**



**Dal punto di vista concettuale, una funzione è composta da un nome (opzionale), una lista di parametri e un corpo che racchiude le istruzioni da eseguire.**

**Quando la funzione viene invocata, i parametri ricevono valori specifici e il codice al suo interno viene eseguito, restituendo opzionalmente un risultato tramite la parola chiave return.**

**In JavaScript esistono diverse modalità di definizione delle funzioni (tradizionali, anonime, arrow functions), ciascuna con peculiarità specifiche relative al contesto di utilizzo e alla gestione dello scope.**



## **Caratteristiche principali delle funzioni in JavaScript:**

- **Modularità:** permettono di suddividere il codice in blocchi più piccoli e indipendenti, rendendo il programma più leggibile e gestibile.
- **Riutilizzabilità:** una funzione scritta una volta può essere richiamata più volte, evitando duplicazioni e riducendo il rischio di errori.
- **Parametri e argomenti:** possono ricevere input tramite parametri e lavorare su valori diversi in base a ciò che viene passato durante la chiamata.
- **Valore di ritorno:** possono restituire un risultato al termine della loro esecuzione, utile per proseguire il flusso del programma.



## Caratteristiche principali delle funzioni in JavaScript:

- **First-class citizens:** possono essere trattate come qualsiasi altro valore: assegnate a variabili, passate come argomenti o restituite da altre funzioni.
- **Scope e contesto:** il comportamento delle variabili all'interno della funzione è influenzato dallo scope (locale o globale) e dal contesto di esecuzione, che in JavaScript può essere gestito tramite la parola chiave `this`.
- **Tipologie diverse:** JavaScript supporta funzioni dichiarate, funzioni anonime e arrow functions, ognuna con differenze sintattiche e semantiche che incidono soprattutto sulla gestione dello scope.



**Ecco una serie di esempi pratici di funzioni in JavaScript, ciascuno con commento esplicativo:**

## **Funzione dichiarata (classica)**

```
1.// Dichiarazione di una funzione che somma due numeri
2.
3.function somma(a, b) {
4.    return a + b; // Restituisce la somma dei parametri
5.}
6.
7.// Invocazione della funzione
8.console.log(somma(3, 5)); // Output: 8
```

**Qui la funzione è definita con un nome (somma), accetta due parametri e restituisce un risultato.**



## Funzione anonima assegnata a variabile

1. **// Una funzione senza nome salvata in una variabile**
2. **let moltiplica = function(x, y) {**
3. **return x \* y;**
4. **};**
- 5.
6. **console.log(moltiplica(4, 6)); // Output: 24**

**In questo caso la funzione non ha nome e viene gestita come valore assegnato a una variabile.**



## Arrow Function (sintassi compatta)

1. **// Sintassi breve per una funzione che divide due numeri**
2. **let dividi = (a, b) => a / b;**
- 3.
4. **console.log(dividi(10, 2)); // Output: 5**
- 5.

**Le arrow function sono utili per funzioni semplici e hanno regole particolari sullo scope di this.**





## Funzione senza parametri

1. **// Una funzione che non riceve parametri**
2. **function saluto() {**
3. **console.log("Ciao dal mondo delle funzioni!");**
4. **}**
- 5.
6. **saluto(); // Output: Ciao dal mondo delle funzioni!**

**Può esistere una funzione che non ha input ma esegue comunque un compito.**



## Funzione che restituisce un'altra funzione

```
1.// Una funzione che genera funzioni
2.function creaMoltiplicatore(fattore) {
3.    return function(numero) {
4.        return numero * fattore;
5.    };
6.}
7.
8.let perDue = creaMoltiplicatore(2);
9.console.log(perDue(5)); // Output: 10
```

Le funzioni possono restituire altre funzioni: qui `creaMoltiplicatore` genera una nuova funzione personalizzata.



**Molti sviluppatori alle prime armi con JavaScript incontrano difficoltà legate alla gestione delle funzioni, soprattutto a causa della flessibilità del linguaggio.**

**Un errore frequente è legato alla dichiarazione e all'invocazione: spesso si confonde la definizione di una funzione con la sua esecuzione.**

**Scrivere il nome della funzione senza parentesi significa riferirsi all'oggetto funzione stesso, mentre aggiungere le parentesi tonde ne provoca l'esecuzione.**

**Questo porta facilmente a risultati inattesi se non si ha chiara la differenza.**



**Un altro ambito critico è lo scope delle variabili e il contesto di esecuzione.**

**In JavaScript le variabili possono avere ambito locale o globale a seconda di come sono dichiarate e l'uso di `this` cambia significato in base a dove e come la funzione viene richiamata.**

**Inoltre, errori comuni includono la mancata gestione dei valori di ritorno, l'uso improprio dei parametri (ad esempio passare più o meno valori del necessario) e la confusione tra funzioni dichiarate e funzioni espresse tramite arrow functions, che non hanno il proprio `this` interno.**



## Errori comuni con le funzioni in JavaScript:

- **Dimenticare le parentesi all'invocazione: scrivere funzione; invece di funzione(); non esegue la funzione, ma restituisce il riferimento all'oggetto.**
- **Confondere dichiarazione ed espressione di funzione: una funzione dichiarata può essere usata prima della sua definizione (hoisting), mentre una espressione no.**
- **Uso scorretto di this: nelle arrow functions this non si riferisce all'oggetto chiamante, ma allo scope esterno, causando confusione.**



## **Errori comuni con le funzioni in JavaScript:**

- **Ignorare il valore di ritorno:** definire un return ma dimenticare di usarne il risultato, o dimenticare di scrivere return quando necessario.
- **Passaggio errato dei parametri:** chiamare la funzione con meno argomenti del previsto (che diventano undefined) o con più, senza gestirli correttamente.
- **Variabili globali non volute:** non dichiarare correttamente le variabili dentro la funzione (mancanza di let o const), rischiando di modificarne altre nello scope globale.



**Buon Davante a tutti**

