LE FUNZIONI

Le **funzioni in**[**JavaScript**](https://www.html.it/guide/guida-javascript-di-base/) hanno un ruolo importantissimo, *una funzione è un insieme di istruzioni racchiuse in un blocco di codice, che può essere contraddistinto da un nome, può accettare argomenti o parametri di ingresso e restituire valori in uscita.*

Una cosa fondamentale da dire è che le funzioni sono riutilizzate lungo tutto il codice e basta semplicemente cambiare gli argomenti.

A questo punto, secondo questa prima definizione, l’utilizzo di una funzione all’interno di uno script prevede due fasi:

* una fase di **definizione** o dichiarazione della funzione in cui si assegna un nome ad un blocco di codice;
* una fase di **invocazione** o chiamata in cui il blocco di codice viene eseguito.

**Definire una funzione JavaScript**

Lo schema sintattico per la definizione di una funzione in JavaScript è il seguente:

function nome(argomenti) {

// istruzioni

}

La parola chiave function evidenzia che stiamo dichiarando una funzione il cui nome segue le stesse regole dei nomi di variabili. Gli argomenti sono una lista opzionale di variabili separati da virgole che verranno utilizzate all’interno del corpo della funzione, cioè il blocco istruzioni.

Una volta dichiarata, una funzione non viene eseguita subito. Stiamo semplicemente dicendo all’engine JavaScript che al blocco di codice indicato viene assegnato un nome.

**JS: Invocare una funzione**

L’esecuzione vera e propria avviene con l’invocazione o chiamata, il cui schema sintattico è:

nome(valori);

dove nome è il nome della funzione e valori è una lista opzionale di espressioni separata da virgole.

**L’istruzione return**

Nel corpo della funzione può essere presente l’istruzione return che consente di terminare e restituire un valore al codice che l’ha chiamata. Questo ci consente di assegnare ad una variabile il valore restituito da una funzione o utilizzare una funzione all’interno di una espressione.

**Esempi di funzioni JavaScript**

Vediamo qualche esempio di definizione e chiamata di funzione:

function somma() {

var z = 11 + 5;

return z;

}

var risultato = somma();

In questo esempio abbiamo definito una funzione senza argomenti che somma due interi e restituisce il risultato. L’invocazione della funzione fa sì che venga eseguita la somma ed il risultato venga assegnato alla variabile risultato.

La funzione somma() che abbiamo definito può essere utilizzata in qualsiasi espressione e verrà trattata da JavaScript come un valore numerico:

var risultato = 5 + somma() \* 2;

In questo esempio l’invocazione somma() sarà equivalente al valore 16, per cui la variabile risultato avrà valore 37.

**Variabili locali e globali**

Le variabili sono dichiarate all’interno del corpo di una funzione e sono accessibili soltanto all’interno della funzione e non vengono viste fuori di essa o, in termini tecnici, hanno uno scope locale.

Lo **scope** o *ambito di visibilità* di una variabile è la parte di uno script all’interno del quale si può fare riferimento ad essa. Le variabili dichiarate all’interno di una funzione sono dette *locali* alla funzione dal momento che sono accessibili soltanto all’interno del suo corpo.

Le variabili dichiarate fuori da qualsiasi funzione sono dette *globali* e sono accessibili da qualsiasi punto dello script, anche all’interno di funzioni.

Chiariamo il concetto con un esempio:

var x = 10;

var y;

function incrementa(){

 var z = 5;

 x = x + z;

}

incrementa();

y = x + 1;

Nell’esempio abbiamo dichiarato due variabili globali x e y. Entrambe sono accessibili anche all’interno della funzione incrementa() e infatti la funzione accede a x per incrementarla del valore della variabile locale z.

Il valore della variabile x dopo l’invocazione della funzione incrementa() sarà pertanto 15 ed il valore finale assegnato a y sarà 16.

**Variabili locali e globali** possono avere lo stesso nome, ma in questo caso si vengono a creare delle ambiguità la cui risoluzione dipende dalla regola secondo cui si fa riferimento all’ambito di visibilità più vicino all’utilizzo della variabile. Quindi, se consideriamo il seguente esempio:

var x = 10;

var y;

function incrementa() {

 var x = 5;

 x = x + 1;

 return x;

}

incrementa();

y = x + 1;

Copy

vediamo che all’interno del corpo della funzione incrementa() viene dichiarata una variabile locale con lo stesso nome della variabile globale x. In questo caso il riferimento a x all’interno della funzione è inteso come il riferimento alla variabile locale, in quanto lo scope locale è il più vicino all’uso della variabile. Spesso si dice che la variabile locale nasconde la variabile globale.

Facciamo notare che, indipendentemente dal punto in cui viene dichiarata una variabile, essa esiste in tutto lo scope a cui appartiene. Consideriamo ad esempio il seguente codice:

var x = 10;

var y;

function incrementa() {

 var s = x; // x ha valore undefined

 var x = 5; // x ha valore 5

 x = x + s; // x ha valore NaN

 return x;

}

incrementa();

y = x + 1; // x ha valore 10

Copy

All’interno della funzione incrementa() abbiamo utilizzato la variabile x prima della sua dichiarazione. Nonostante ciò x non fa riferimento alla variabile globale, ma a quella locale dichiarata subito dopo e il suo valore è ancora undefined.

Un’altra osservazione, legata in qualche modo a questa appena accennata, è che a differenza di quanto accade in altri linguaggi come ad esempio C e C#: dichiarare una variabile all’interno di un blocco di codice non crea un nuovo scope per la variabile!

## Let

Se invece abbiamo bisogno di **creare uno scope specifico** per una o più variabili possiamo ricorrere all’istruzione **let**. Questa istruzione, consente di dichiarare una o più variabili in modo analogo a var, ma a differenza di quest’ultima limita lo scope della variabile al blocco di codice, all’istruzione o all’espressione in cui viene utilizzata. Quindi, ad esempio, nel seguente codice:

var x = 10;

var y;

{

 let x = 20;

 y = x + 1;

}

y = x + y;

avremo uno **scope a livello di blocco** di codice in cui la variabile x dichiarata tramite let nasconde quella esterna dichiarata con var. Il risultato finale combina i valori delle due variabili omonime.

Un ambito in cui è senz’altro utile utilizzare l’istruzione let è nelle iterazioni, come ad esempio nel for:

E’ possibile definire una gerarchia di scope all’interno di ciascuno dei quali definire l’accessibilità delle variabili.

## parseInt e parseFloat

La funzione **parseInt**() converte una stringa in un valore intero. La funzione prevede due parametri: il primo è la stringa da convertire, mentre il secondo è opzionale e indica la base del sistema di rappresentazione numerica utilizzato. I seguenti sono esempi di utilizzo della funzione:

parseInt("12") // 12

parseInt("12abc") // 12

parseInt("a12bc") // NaN

parseInt("12.5") // 12

parseInt("12", 8) // il valore di 12 nel sistema di numerazione ottale (base 8), cioè 10

Da notare come la presenza di caratteri non numerici in coda alla stringa venga ignorata, mentre impedisce di fatto la conversione se questi si trovano all’inizio.

La funzione **parseFloat**() restituisce un valore numerico considerando l’eventuale virgola:

parseFloat("12") //12

parseFloat("12.5") //12.5

È possibile effettuare conversioni esplicite tra gli altri tipi di dato ricorrendo agli oggetti, come vedremo più avanti.

Abbiamo già avuto modo di vedere le funzioni parseInt() e parseFloat() per convertire una stringa rispettivamente in un valone numerico intero e decimale.

Ricordiamo brevemente che la funzione parseInt() prevede due parametri: il primo è la stringa da convertire, mentre il secondo è opzionale e indica la base del sistema di rappresentazione numerica utilizzato.

La funzione parseFloat() invece prevede un solo argomento e restituisce un valore numerico intero o decimale in base alla presenza del separatore decimale.

## isNaN e isFinite

Rimanendo sempre nell’ambito dei numeri abbiamo le funzioni isNaN() e isFinite().

La funzione isNaN() prende un argomento e restituisce true se il suo valore è NaN, cioè non è un valore numerico valido, false altrimenti.

Analogamente, isFinite() restituisce true se il valore del suo argomento è diverso da Infinity e da NaN.

## escape e unescape

Date le sue origini strettamemte legate al Web, JavaScript mette a disposizione una serie di funzioni predefinite per la codifica e decodifica di stringhe destinate a viaggiare sulla rete.

In particolare la funzione escape() restituisce la codifica una stringa lasciando inalterate cifre, lettere e i caratteri +-\*/.\_@ e rimpiazzando tutti gli altri caratteri con la codifica esadecimale preceduta da un carattere percentuale (%). Il seguente è un esempio di uso della funzione escape():

var stringa = escape("Questa è una stringa!");

La funzione unescape() esegue il procedimento contrario convertendo le codifiche esadecimali nei corrispondenti caratteri ASCII.