# Costrutti

| Size | Small topic |
| --- | --- |

Controllo del flusso

---

## Comandi condizionali:

Molto spesso, quando si scrive codice, si desidera eseguire azioni diverse per decisioni diverse.  È possibile utilizzare le istruzioni condizionali nel codice per eseguire questa operazione.
In JavaScript abbiamo le seguenti affermazioni condizionali:

- Utilizzare `if` per specificare un blocco di codice da eseguire, se una condizione specificata è vera
- Utilizzare `else` per specificare un blocco di codice da eseguire, se la stessa condizione è falsa
- Utilizzare `else if` per specificare una nuova condizione da verificare, se la prima condizione è falsa
- Utilizzare `switch` per specificare molti blocchi di codice alternativi da eseguire

Si utilizza l'istruzione `if` per specificare un blocco di codice JavaScript da eseguire se una condizione è vera.

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Si utilizza l'istruzione `else` per specificare un blocco di codice che vogliamo eseguire se un la condizione è falsa

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Si utilizza l'istruzione `else if` per specificare una nuova condizione se la prima condizione è falsa.

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

L'istruzione `switch` viene utilizzata per eseguire diverse azioni in base a condizioni diverse e selezionare uno dei tanti blocchi da eseguire.

```
switch(expression) {
  case x:
    // code block
```

```
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Ecco come funziona:

- L'espressione `switch` viene valutata una volta.
- Il valore dell'espressione viene confrontato con i valori di ciascun caso.
- Se c'è una corrispondenza, viene eseguito il blocco di codice associato.
- Se non c'è corrispondenza, viene eseguito il blocco di codice predefinito (`default`).

Quando JavaScript raggiunge una `break` parola chiave, esce dal blocco switch, ciò interromperà l'esecuzione all'interno del blocco switch.
Non è necessario interrompere l'ultimo caso in un blocco interruttori. Il blocco si rompe (finisce) comunque lì.

> *Se non viene inserito il `break` , lo `switch` eseguirà anche la restante parte di codice senza rispettare i blocchi che ci avevamo predisposto*

Il `default` serve per far uscire fuori dallo `switch`, sia che venga eseguito un blocco, sia che non venga eseguito

## Comandi iterativi:

Spesso ci troviamo difronte ad un seguente problema che possiamo risolvere mediante un loop, javaScript supporta diversi tipi di loop:

- `for` scorre un blocco di codice un certo numero di volte

- `for/in` scorre le proprietà di un oggetto
- `for/of` scorre i valori di un oggetto iterabile
- `while` scorre un blocco di codice mentre una condizione specificata è vera
- `do/while` scorre anche un blocco di codice mentre una condizione specificata è vera

L'istruzione `for` crea un ciclo con 3 espressioni opzionali:

Permette di ripetere un insieme di comandi per un dato numero di volte di solito usato per iterazione determinata, ovvero quando si conosce a priori il numero di iterazioni da eseguire

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}



for (let i = 0; i < 5; i++) {
  text += "The number is " + i;
}
```

- L'espressione 1 imposta una variabile prima dell'inizio del ciclo (sia `i = 0`).
- L'espressione 2 definisce la condizione per l'esecuzione del ciclo (`i < 5`).
- L'espressione 3 aumenta un valore (`i++`) ogni volta che il blocco di codice nel ciclo è stato eseguito.

Esempio:
*Programma che calcola la somma dei primi "n" numeri naturali con "n" letto da tastiera*

```
let n = Number(prompt("n?"))
let somma = 0
let i

for (i=1; i<=n; i=i+1) {
  somma += i
}
```

```
console.log(somma)
```

L’istruzione `for in` può  scorrere le proprietà di un array:

```
const obj = { a: 1, b: 2, c: 3 };

for (let key in obj) {
  console.log(key);
}
// Output: "a", "b", "c"
```

L'istruzione `for of` scorre i valori di un oggetto iterabile, consente di scorrere strutture di dati iterabili come array, stringhe, mappe, elenchi di nodi e altro:

```
const arr = [1, 2, 3, 4];

for (let element of arr) {
  console.log(element);
}
// Output: 1, 2, 3, 4
```

Il ciclo `while` scorre un blocco di codice fintanto che una condizione specificata è vera.

```
while (condition) {
  // code block to be executed
}



while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Esempio:
Leggere e sommare numeri fino a che no viene inserito -1, stampare poi la somma ottenuta

```
let somma = 0

let n = Number(promt("n?"))
while (n != -1) {

}
```

Il ciclo `do while` è una variante del ciclo while. Questo ciclo eseguirà il blocco di codice una volta, prima di verificare se la condizione è vera, quindi ripeterà il ciclo finché la condizione è vera.

```
do {
  // code block to be executed
}
while (condition);



do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

# Funzioni

Una funzione  è un blocco di codice progettato per eseguire un'attività particolare. Una funzione  viene eseguita quando "qualcosa" la invoca (la chiama).

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Una funzione JavaScript è definita con la `function` parola chiave, seguita da un **nome** , seguito da parentesi `()`.
I nomi delle funzioni possono contenere lettere, cifre, trattini bassi e segni di dollaro (stesse regole delle variabili). Le parentesi possono includere i nomi dei parametri separati da virgole: **( *parametro1, parametro2, ...* )** Il codice da eseguire, dalla funzione, è racchiuso tra parentesi graffe: `{}`

**I parametri** della funzione sono elencati tra parentesi `()` nella definizione della funzione. **Gli argomenti** della funzione sono i **valori** ricevuti dalla funzione quando viene richiamata.
All'interno della funzione, gli argomenti (i parametri) si comportano come variabili locali.

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Il codice all'interno della funzione verrà eseguito quando "qualcosa" **invoca** (chiama) la funzione:

- Quando si verifica un evento (quando un utente fa clic su un pulsante)
- Quando viene invocato (chiamato) dal codice JavaScript
- Automaticamente (auto-invocato)

Le variabili dichiarate all'interno di una funzione JavaScript diventano **LOCALI** per la funzione. È possibile accedere alle variabili locali solo dall'interno della funzione.

Poiché le variabili locali vengono riconosciute solo all'interno delle loro funzioni, le variabili con lo stesso nome possono essere utilizzate in funzioni diverse.

Quando JavaScript raggiunge `return` un'istruzione, la funzione interrompe l'esecuzione. Se la funzione è stata invocata da un'istruzione, JavaScript "ritornerà" per eseguire il codice dopo l'istruzione di richiamo.
Le funzioni spesso calcolano un **valore restituito** . Il valore restituito viene "restituito" al "chiamante":

**Esempio:**

```
let x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

### Ma perché vengono utilizzate le funzioni?

Puoi riutilizzare il codice, ovvero definisci il codice una volta e usalo più volte. È possibile utilizzare lo stesso codice più volte con argomenti diversi, per produrre risultati diversi.

### Come si richiama un funzione?

L'operatore () richiama la funzione ad esempio, una funzione `toCelsius` fa riferimento all'oggetto funzione e `toCelsius()` fa riferimento al risultato della funzione.
L'accesso a una funzione senza () restituirà l'oggetto funzione invece del risultato della funzione.

Le funzioni possono essere utilizzate allo stesso modo delle variabili, in tutti i tipi di formule, assegnazioni e calcoli. Ad esempio:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```