Cicli in JS

Programmazione

In JavaScript i cicli (o iterazioni) sono costrutti fondamentali che permettono di eseguire ripetutamente un blocco di istruzioni fino al verificarsi di una condizione.

L’idea alla base è quella di automatizzare operazioni ripetitive senza dover scrivere più volte lo stesso codice.

I cicli si basano sempre su tre componenti logici: inizializzazione (un valore di partenza, spesso un contatore), condizione (che viene valutata a ogni iterazione e decide se il ciclo prosegue) e incremento o aggiornamento (che modifica la variabile di controllo per avvicinarsi alla condizione di uscita).

Questo meccanismo consente di processare strutture dati come array, liste o oggetti, oppure di realizzare flussi logici ripetitivi con grande efficienza.

JavaScript mette a disposizione diverse tipologie di cicli, ciascuna adatta a scenari specifici. Il ciclo for è utile quando si conosce a priori il numero di ripetizioni; il while si utilizza quando si vuole iterare fino a che una condizione rimane vera, senza sapere in anticipo quante volte; il do...while garantisce almeno una esecuzione del blocco prima del controllo della condizione.

Oltre a questi, esistono cicli più moderni come for...of (ideale per scorrere collezioni come array e stringhe) e for...in (usato per iterare sulle proprietà enumerabili di un oggetto).

La scelta del ciclo dipende quindi dalla natura del problema: conoscere bene le differenze aiuta a scrivere codice più leggibile, performante e sicuro.

- **Controllo della condizione**

Ogni ciclo verifica una condizione logica: se è vera il blocco viene eseguito, se è falsa il ciclo termina.
Questo meccanismo impedisce iterazioni infinite, purché la condizione sia aggiornata correttamente.

- **Variabile di controllo**

Nei cicli come for, la variabile di controllo (es. un contatore numerico) viene inizializzata, modificata a ogni iterazione e usata per verificare la condizione.
È fondamentale aggiornarla correttamente per garantire l'uscita dal ciclo.

- **Ripetizione automatica**

I cicli permettono di eseguire più volte lo stesso blocco di codice senza duplicarlo, rendendo il programma più compatto e meno soggetto a errori di scrittura o manutenzione.

- **Versatilità delle strutture**

JavaScript offre diversi tipi di cicli (for, while, do...while, for...of, for...in), ognuno adatto a contesti differenti: numerici, collezioni di dati o proprietà di oggetti.
Saper scegliere quello giusto rende il codice più leggibile e performante.

- **Ciclo for**

```
1. for (let i = 0; i < 5; i++) {
2.   console.log("Iterazione numero: " + i);
3. }
```

In questo esempio il ciclo parte da i = 0, ripete finché i < 5 e incrementa i di 1 a ogni iterazione.

È usato quando si conosce il numero esatto di ripetizioni.

- **Ciclo while**

```
1. let n = 0;
2. while (n < 3) {
3.   console.log("Valore di n: " + n);
4.   n++;
5. }
```

Qui il ciclo continua finché la condizione n < 3 è vera.

È utile quando non si conosce a priori quante volte ripetere il blocco.

- **Ciclo do...while**

```
1. let x = 0;
2. do {
3.   console.log("Eseguito almeno una volta, x = " + x);
4.   x++;
5. } while (x < 2);
```

Questo ciclo esegue almeno una volta il blocco di codice, perché la condizione viene verificata dopo l'esecuzione.

- **Ciclo for...of**

```
1. let numeri = [10, 20, 30];
2.
3. for (let num of numeri) {
4.   console.log("Numero: " + num);
5. }
```

for...of serve per scorrere direttamente gli elementi di array, stringhe o collezioni, senza gestire un contatore manuale.

- **Ciclo for...in**

```
let persona = { nome: "Luca", età: 25, città: "Roma" };

for (let chiave in persona) {
  console.log(chiave + ": " + persona[chiave]);
}
```

for...in itera sulle proprietà enumerabili di un oggetto, utile per lavorare con strutture tipo dizionari.

- **Errori comuni nei cicli**

Uno degli errori più frequenti nell'uso dei cicli in JavaScript è dimenticare di aggiornare correttamente la variabile di controllo, causando un ciclo infinito che può bloccare il programma.

Un altro problema ricorrente è impostare male la condizione logica, ad esempio con un limite sbagliato (<= invece di <) ottenendo una iterazione in più o in meno.

Inoltre, quando si lavora con array, è comune confondere l'indice con il valore, oppure superare la lunghezza dell'array e accedere a elementi undefined.

Infine, bisogna evitare di usare for...in sugli array, perché restituisce le chiavi (indici come stringhe) e non gli elementi, creando confusione e bug.

```
// Lista di studenti con i loro voti

let studenti = [
  { nome: "Anna", voto: 28 },
  { nome: "Marco", voto: 30 },
  { nome: "Luca", voto: 24 }
];

// 1) Ciclo for: stampiamo i nomi degli studenti
for (let i = 0; i < studenti.length; i++) {
  console.log("Studente: " + studenti[i].nome);
}

// 2) Ciclo while: calcoliamo la media dei voti
let somma = 0;
let index = 0;
while (index < studenti.length) {
  somma += studenti[index].voto;
  index++;
}
let media = somma / studenti.length;
console.log("Media voti: " + media);

// 3) Ciclo for...of: stampiamo i voti superiori a 25
for (let studente of studenti) {
  if (studente.voto > 25) {
    console.log(studente.nome + " ha un voto alto: " +
studente.voto);
  }}

// 4) Ciclo for...in: stampiamo le proprietà del primo studente
for (let chiave in studenti[0]) {
  console.log("Proprietà: " + chiave + " = " + studenti[0][chiave]);
}
```

Spiegazione

- Con il ciclo for scorriamo l'array studenti usando un contatore (i) e stampiamo i nomi.
- Con il ciclo while sommiamo i voti di tutti gli studenti e calcoliamo la media. Usiamo un indice separato che si incrementa fino alla fine della lista.
- Con il ciclo for...of percorriamo direttamente gli oggetti dell'array e selezioniamo solo quelli con un voto maggiore di 25, stampando un messaggio personalizzato.
- Con il ciclo for...in esploriamo le proprietà del primo studente (nome e voto), utile per ispezionare dinamicamente gli oggetti.


Buon Davante a tutti