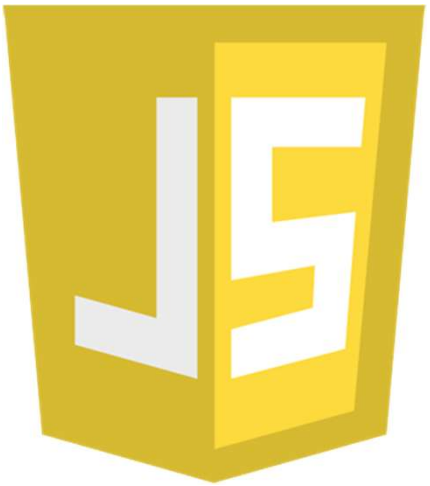
JAVASCRIPT

Riccardo Cattaneo

Lezione 4



Espressioni e operatori

Un'**espressione** è una combinazione di valori, variabili ed operatori che rappresentano un nuovo valore. Ad esempio, la seguente è un'espressione:

x + 1

Nel caso specifico si tratta di un'espressione che combina una variabile x ed il valore numerico 1 tramite l'operatore + (più) per ottenere l'incremento di un'unità del valore di x.

È evidente che un ruolo fondamentale nelle espressioni è assunto dagli operatori, dal momento che determinano il valore risultante dell'espressione.

if - istruzioni condizionali

Le istruzioni condizionali sono una categoria di istruzioni che consentono di eseguire blocchi di codice alternativi in base ad una condizione, consentendo ad uno script di prendere, in un certo senso, delle decisioni. Javascript prevede due istruzioni condizionali: **if** e **switch**.

Possiamo trovare l'istruzione if in tre forme:

if semplice;

if con alternativa (if..else);

if a cascata (if..else if…else).

La forma if pura esegue un blocco di codice solo se una condizione è vera. Il suo schema sintattico è:

```
if (condizione) {
        // istruzioni
}
```

dove condizione è un'espressione **booleana** e istruzioni rappresenta appunto la sequenza di istruzioni da eseguire nel caso in cui la condizione sia vera.

La seconda forma è **if...else**. In questa forma viene eseguito un blocco di codice oppure un altro in base al valore della condizione. Lo schema sintattico si presenta così:

```
if (condizione) {
        // istruzioni1
} else {
        //istruzioni2
}
```

Se condizione è vera vengono eseguite le istruzioni del blocco istruzioni1 altrimenti viene eseguito il blocco istruzioni2.

La terza forma di if è **if..else if...else** o if a cascata, che mette a disposizione più alternative di esecuzione:

```
if (condizione1) {
        istruzioni1
} else if (condizione2) {
        istruzioni2
} else {
        istruzioni3
}
```

In questo caso, se condizione1 è vera viene eseguito il blocco istruzioni1. Se condizione1 non è soddisfatta viene verificata condizione2 e in base al suo valore verrà eseguito il blocco istruzioni2 o istruzioni3.

```javascript
var val8 = 10;

if(val8 > 20){
    console.log('val8 è maggiore di 20');
}

if(val8 > 20){
    console.log('val8 è maggiore di 20');
}else{
    console.log('val8 è minore o uguale di 20');    // OK
}

if(val8 > 20){
    console.log('val8 è maggiore di 20');
}else if(val8 == 20){
    console.log('val8 è uguale a 20');
}else{
    console.log('val8 è minore di 20');             // OK
}
```

JavaScript

Switch...case

L'istruzione switch può essere una valida forma sostitutiva dell'istruzione if in tutti i casi in cui si deve testare il valore assegnato a una variabile, ed è da preferire quando le condizioni da esaminare sono piuttosto numerose.

L'istruzione switch si fa preferire per la sua particolare sinteticità e la facilità di lettura rispetto a if. È inoltre piú rapida da eseguire perché a differenza di quanto accade con if, l'interprete non deve esaminare ogni opzione in attesa di trovare, qualora esista, quella corrispondente al caso verificatosi, ma va subito a cercare il valore restituito da un'espressione all'interno dei vari casi di switch.

```
switch (espressione) {
        case espressione1:
                istruzioni1;
        break;
        case espressione2:
                istruzioni2;
        break;
        default:
                istruzioni4;
        break;
}
```

Talvolta possiamo decidere di non mettere l'istruzione **break** in corrispondenza di uno o più **case** perché vogliamo eseguire uno stesso blocco di codice per un gruppo di valori.

Il seguente è un esempio del genere:

```
switch (voto) {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
                giudizio = "insufficiente";
                break;
        case 6:
                giudizio = "sufficiente";
                break;
        case 7:
                giudizio = "discreto";
                break;
        case 8:
                giudizio = "buono";
                break;
        default:
                messaggio = "non classificato";
                break;
}
```

While e do-while, le iterazioni base

Un'altra categoria di istruzioni comunemente usata nei linguaggi di programmazione è rappresentata dalle iterazioni o cicli. Javascript prevede le classiche istruzioni di iterazione come **while** , **do while** e **for**.

Di seguito lo schema sintattico del while:

```
while (condizione) {
        // istruzioni
}
```

Finché la condizione sarà **vera** (true) verranno eseguite le istruzioni contenute nel blocco di codice.

Requisito fondamentale nell'uso del while è che le istruzioni contenute nel blocco di codice modifichino la condizione, altrimenti si rischia di incorrere in un ciclo infinito.

```
var valoreMassimo = 10;
var contatore = 1;

while (contatore <= valoreMassimo){
    console.log(contatore);
    contatore++;
}
```

```
1
2
3
4
5
6
7
8
9
10

C:\Users\Rick\Desktop\js>
```

Passiamo ad un esempio più concreto, e cioè utilizzare il ciclo while per scorrere un array.

```
var giorni = ['lunedi','martedi','mercoledi','giovedi','venerdi','sabato','domenica'];
var contatore = 0;

while(contatore < giorni.length){
    console.log(giorni[contatore]);
    contatore++;
}
```

do-while

Una variante del while è il do…while:

```
do {
        // istruzioni
}
while (condizione)
```

La differenza sostanziale rispetto al while classico consiste nel fatto che la condizione viene valutata dopo aver eseguito le istruzioni. Questo garantisce che il blocco di codice verrà eseguito almeno una volta.

```
<body>
    <script>

        do{
            var a = prompt("Inserisci un numero");

        }while(a < 0)

        alert('ok numero inserito correttamente');

    </script>
</body>
```

Il ciclo for

Una alternativa al while è l'istruzione for. Questa istruzione è generalmente intesa come l'esecuzione di un blocco di codice per **un numero determinato di volte.**

Il ciclo for in Javascript è molto simile al while. Vediamo il suo schema sintattico e cerchiamo di capire le differenze :

```
for (inizializzazione; condizione; modifica) {
        // istruzioni
}
```

inizializzazione: Javascript esegue l'istruzione specificata in inizializzazione prima di avviare le iterazioni (il ciclo).

condizione: è l'espressione booleana che viene valutata prima di eseguire ciascuna iterazione. Se è falsa non viene eseguito il blocco di istruzioni associato al for. Se invece la condizione è vera viene eseguito il blocco di codice.

modifica: al termine di ciascuna iterazione viene eseguita l'istruzione modifica (la più classica è l'incremento di un contatore). Il ciclo poi ricomincia con la valutazione della condizione.

```
var giorni = ['lunedi','martedi','mercoledi','giovedi','venerdi','sabato','domenica'];
var contatore = 0;

for(var cont = 0; cont < giorni.length ; cont++){
    console.log(giorni[cont]);
}
```

In questo esempio abbiamo stampato i valori di un array con la stessa logica del ciclo while con l'unica «**sostanziale**» differenza che nel ciclo for ho tutte le informazioni, quindi so subito quante volte cicla, mentre nel while, avendo solo la condizione non posso sapere a priori quante ciclerà.

While vs For

Ciclo while e ciclo for fanno la stessa cosa ma in modo diverso, ricordiamo… **il risultato è lo stesso**. Vediamoli a confronto e proviamoli :

```
for(var cont = 1; cont <= 10 ; cont++){
    console.log(cont);
}
```

```
var cont2 = 1;

while(cont2 <=10){
    console.log(cont2);
    cont2++;
}
```

for-in e for-of

Per lavorare più comodamente con gli array Javascript prevede due varianti del for: il for...in e il for...of. Vediamo come scrivere le istruzioni precedenti facendo uso del **for...in**:

```
var quantita = [12, 24, 42, 11, 29];
var totale = 0;
for (var indice in quantita) {
        totale = totale +  quantita[indice];
}
```

Sfruttando questa variante del for non abbiamo bisogno di specificare la lunghezza dell'array né l'istruzione di modifica della condizione. Javascript rileva che la variabile quantita è un array ed assegna ad ogni iterazione alla variabile indice il valore dell'indice corrente.

Utilizzando il **for..of** invece possiamo scrivere il ciclo in questo modo:

```
var quantita = [12, 24, 42, 11, 29];
var totale = 0;
for (var valore of quantita) {
        totale = totale +  valore;
}
```

Ad ogni iterazione Javascript assegna alla variabile valore il contenuto di ciascun elemento dell'array. Queste varianti del for ci consentono di scrivere meno codice quando lavoriamo con gli array.

Esercizio 4.1

Dichiarare e valorizzare un numero intero scegliendone uno da 1 a 10 e mostrare a video la relativa tabellina, ad esempio se scelgo il numero 5 visualizzerò a video (eseguire l'esercizio sia con il ciclo for che con il ciclo while):

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

Ecc…

Esercizio 4.2

Scrivere un programma che stampa i numeri da 0 a 10 sia con il ciclo for che con il ciclo while;

Esercizio 4.3

Modificare l'esercizio 2 in modo che stampa i numeri da 5 a 15 sia con il ciclo for che con il ciclo while;

Esercizio 4.4

Scrivere un programma che conta da 0 a 20 con passo 2 e stampa i numeri ottenuti (0,2,...,20) sia con il ciclo for che con il ciclo while ;

Esercizio 4.5

Modificare l'esercizio 4 in modo che stampi il doppio dei numeri ottenuti (0,4,...,40) sia con il ciclo for che con il ciclo while;

Esercizio 4.6

Scrivere un programma che dichiara e valorizza un array di 5 numeri a proprio piacimento e restituisca in output la somma e la media utilizzando il ciclo for.

Esercizio 4.7

Scrivere un programma che dichiara e valorizza un array di 7 numeri a proprio piacimento e restituisca in output solamente il numero più grande (utilizzare il ciclo for).

Esercizio 4.8

Scrivere un programma che dichiara e valorizza un array di 10 numeri a proprio piacimento e restituisca in output il totale di quanti numeri pari ci sono e quanti numeri dispari (utilizzare il ciclo for).

Ad es.

Nell’ array ci sono 3 numeri pari e 7 numeri dispari