JAVASCRIPT

Riccardo Cattaneo

Lezione 3

Definire Array in Javascript

Gli **array** consentono di associare più valori ad un unico nome di variabile (o identificatore). In genere i valori contenuti in un array hanno una qualche affinità (ad esempio l'elenco dei giorni della settimana). L'uso degli array evita di definire più variabili e semplifica lo svolgimento di operazioni cicliche su tutti i valori. Ad es:

```
var giorniDellaSettimana = [
   "lunedì",
   "martedì",
   "mercoledì",
   "giovedì",
   "venerdì",
   "sabato",
   "domenica"];
```

Come si vede in questa dichiarazione i valori dell'array sono delimitati da **parentesi quadre** e separati da virgole. Questo è un modo di definire un **array basato sulla rappresentazione letterale (literal)**.

La sintassi per scrivere un array è composta da parentesi quadre con all'interno gli elementi separati da una virgola, ad esempio :

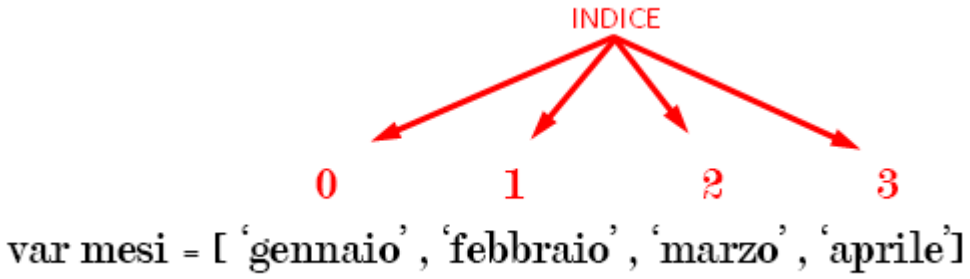
var nomi = ['Mario' , 'Anna' , 'Luca'] ;

Gli array in Javascript sono dinamici: crescono o si riducono in base alla necessità e non è necessario dichiarare una dimensione fissa per l'array al momento della creazione.

nomi = ['Mario' , 'Anna' , 'Luca' , 'Francesca'] ;

indice

Ogni valore all'interno dell'array è detto **elemento** ed ogni elemento ha una posizione numerica chiamata **indice.**



Questo vuol dire che se voglio stampare o utilizzare il valore marzo dovrò fare :

```
console.log( mesi[2] );
var meseScelto = mesi[2];
```

Una volta definito un array possiamo accedere ai singoli elementi facendo riferimento al nome della variabile e all'indice corrispondente all'elemento, ricordandoci che **la numerazione degli indici parte da zero**. Così, ad esempio, se vogliamo accedere al primo elemento dell'array dei giorni della settimana scriveremo:

```
var primoGiorno = giorniDellaSettimana[0];
```

Gli elementi di un array possono essere di **qualsiasi tipo di dato** previsto da JavaScript, quindi possiamo avere array con elementi di diverso tipo:

```
var mioArray = [123, "stringa", true, null];
```

Gli array in Javascript possono essere sparsi: gli elementi non sono obbligati ad avere indici contigui

nomi[9] = ‘Carlo’; // ok

Fare attenzione che in questo caso la posizione con indice 4, 5, 6, 7 e 8 esistono e rimangono vuote, quindi se provo a farmi stampare la lunghezza dell’array il risultato sarà 10 ;

nomi[0] = ‘Mario’;
nomi[1] = ‘Anna’;
nomi[2] = ‘Luca’;
nomi[3] = ‘Francesca’;
nomi[4] = empty;
nomi[5] = empty;
nomi[6] = empty;
nomi[7] = empty;
nomi[8] = empty;
nomi[9] = ‘Carlo’;

Attributo length

Per sapere quanti elementi sono presenti all'interno di un array è possibile utilizzare l'attributo **length** dell'oggetto array. Per fare una prova possiamo stampare a video la lunghezza dell'array dell'esempio precedente in questo modo :

```
console.log(giorniDellaSettimana.length);
```

ATTENZIONE

Fare attenzione nel distinguere **l'indice dell'array** con la **lunghezza dell'array**. Ricordiamo che l'indice parte da ZERO mentre il conteggio della lunghezza parte da UNO.

Questo vuol dire che, tornando all'esempio dei giorni della settimana, l'array ha una lunghezza di 7 elementi con indice che va da 0 a 6.

Operatori aritmetici

Per quel che riguarda il tipo di dato numerico, abbiamo gli **operatori aritmetici**, che consentono la combinazione di valori numerici. Si tratta di operatori binari corrispondenti ai classici operatori matematici:

Operatore	Nome
+	addizione
-	sottrazione
/	divisione
*	moltiplicazione
%	modulo o resto

Gli operatori aritmetici seguono le regole di precedenza matematiche e, come in matematica, possiamo utilizzare le parentesi (ma soltanto quelle tonde) per modificare il loro ordine di valutazione:

```
4 + 5 * 6 + 7
(4 + 5) * (6 + 7)
```

Il valore della prima espressione è 41 mentre la seconda espressione vale 117.

```
var num1 = 10;

console.log(num1);          // 10
console.log(num1/2);        // 5
console.log(num1*3);        // 30
console.log(num1*'ciao');   // NaN
console.log(num1+'ciao');   // 10ciao
console.log(num1+10);       // 20
```

```
C:\Users\Rick\Desktop\js>node esempio.js
10
5
30
NaN
10ciao
20
```

Operatori Compatti

Oltre all'operatore **uguale (=)**, esistono altri operatori di assegnamento derivanti dalla combinazione degli operatori aritmetici e dell' operatore =. Essi rappresentano una abbreviazione di espressioni di assegnamento la cui espressione da assegnare utilizza lo specifico operatore aritmetico.

Forma Compatta	Scrittura Equivalente
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Operatori Unari

Gli operatori aritmetici prevedono anche quattro operatori unari:

Operatore	Nome
+	Numero Positivo o cast a numero
-	Negazione
++	Incremento
--	decremento

La negazione consente di ottenere un valore negativo di un numero. Ad esempio, -x è il valore negativo del valore numerico rappresentato dalla variabile x. Gli operatori di incremento e decremento sono applicabili soltanto a variabili e consentono di ottenere un valore rispettivamente aumentato o diminuito di uno. È molto importante la posizione dell'operatore.

Infatti, se l'operatore si trova a sinistra dell'operando, l'operazione di incremento o decremento viene eseguita prima della valutazione dell'espressione, mentre se si trova a destra l'operazione viene eseguita dopo. Chiariamo il concetto con un esempio:

```
var x = 3;
var y = ++x;
```

La variabile x verrà prima incrementata di uno ed il risultato verrà assegnato alla variabile y. Quindi la situazione finale vedrà x e y con il valore di 4. Nel seguente caso, invece:

```
var x = 3;
var y = x++;
```

il valore della variabile x verrà assegnato alla variabile y e soltanto successivamente x verrà incrementata di uno. Come risultato avremo x con il valore di 4 e y con il valore di 3.

```
var numero = '20';
console.log(numero + ' è una ' + typeof(numero));

numero = +numero;
console.log(numero + ' è una ' + typeof(numero));
```

```
C:\Users\Rick\Desktop\js>node esempio.js
20 è una string
20 è una number
```

Operatori relazionali

Gli **operatori relazionali** sono quegli operatori utilizzati per il confronto tra valori (e stabilire una "relazione d'ordine"):

Operatore	Nome
<	minore
<=	Minore o uguale
>	maggiore
>=	Maggiore o uguale
==	Valore Uguale
!=	Valore diverso
===	Strettamente uguale (tipo e valore)
!==	Strettamente diverso (tipo e valore)

Le espressioni che utilizzano gli operatori relazionali restituiscono un **valore booleano** in base all'esito del confronto. Ad esempio:

```
4 > 2     // true
5 != 4 + 1   //false
```

Il significato della maggior parte degli operatori dovrebbe essere abbastanza intuitivo

```
var numero = 10 < 15;
console.log(numero + ' è di tipo ' + typeof(numero));   // true

var numero2 = 10==10;
console.log(numero2 + ' è di tipo ' + typeof(numero2)); // true

numero  = 30;
numero2 = 30;
var numero3 = numero == numero2;
console.log(numero3 + ' è di tipo ' + typeof(numero3)); // true

numero  = 30;
numero2 = '30';
var numero4 = numero == numero2;
console.log(numero4 + ' è di tipo ' + typeof(numero4)); // true
var numero5 = numero === numero2;
console.log(numero5 + ' è di tipo ' + typeof(numero5)); // false
```

Attenzione ai boolean

Fare attenzione se dobbiamo effettuare un confronto con i boolean. Javascript considera true = 1 e false = 0.

```
numero = 1;
numero2 = true;

var numero6 = numero==numero2;
var numero7 = numero===numero2;
console.log(numero6 + ' è di tipo ' + typeof(numero6)); // true
console.log(numero7 + ' è di tipo ' + typeof(numero7)); // false
```

JavaScript

Operatori logici

Gli **operatori logici** consentono la combinazione di espressioni booleane. JavaScript prevede due operatori binari e un operatore unario:

Operatore	Nome
&&	and
||	or
!	not

Le seguenti sono espressioni che utilizzano operatori logici:

```
5 > 2 && 3 != 4  // true
true || 4 >= 6   // true
!5==5            // false
```

La combinazione di operatori logici con altre espressioni può essere insidiosa, nel senso che le regole di precedenza nella valutazione possono generare valori diversi da quelli attesi. È opportuno pertanto utilizzare le parentesi per indicare esplicitamente l'ordine di valutazione e per renderle più leggibili. Le espressioni precedenti diventano pertanto:

```
(5 > 2) && (3 != 4)  // true
 true || (4 >= 6)    // true
!(5==5)              // false
```

Conversioni tra tipi di variabili

Le variabili JavaScript possono contenere valori di qualsiasi tipo e cambiare senza problemi il tipo di dato contenuto. Questa flessibilità libera lo sviluppatore dal dover dichiarare esplicitamente il tipo di dato che può contenere una variabile, scaricando però sull'*engine* JavaScript il compito di dover prendere delle decisioni in determinate situazioni ed effettuare delle conversioni implicite. Ad esempio :

```
var x = "12";
var y = "13";
var z = x * y;
```

Il valore che ci aspettiamo nella variabile z è il risultato della moltiplicazione dei valori numerici risultanti dalla conversione delle stringhe contenute in x e y. Ed in effetti questo è il comportamento di JavaScript. Ma vediamo un altro esempio:

```
var x = "12";
var y = x + 13;
```

Che valore avrà la variabile y? Sarà la stringa "1213" o l'intero 25?

Per complicare ancora un po' le cose, consideriamo un'altra situazione:

```
var x;
var y = x + 13;
```

Quale sarà il valore di y in questo caso?

In assenza di indicazioni da parte dello sviluppatore, JavaScript dovrà prendere delle decisioni e convertire l'espressione in un tipo piuttosto che in un altro. Conoscere i criteri in base ai quali JavaScript converte implicitamente le espressioni è fondamentale per evitare bug spesso molto insidiosi. Partiamo dal principio in base al quale in JavaScript ogni tipo di dato primitivo può essere convertito in un altro tipo di dato primitivo: numeri, stringhe, booleani, undefined e null.

```javascript
var val2 = "10";
var val3 = "14";
var val4 = val2*val3;    // moltiplica => 140
console.log(val4);

var val5 = val2+val3;    // concatena => 1014
console.log(val5);

var val6 = 30 + val2;
console.log(val6);       // concatena => 3010

var val7;
console.log(val7 + 50); // Nan
```

JavaScript

prompt()

Il metodo prompt() è una funzione «globale» di javascript che consente di far apparire una finestra di dialogo con un campo di testo all'interno del quale l'utente può inserire del contenuto. Opzionalmente in ingresso alla funzione si può passare una stringa che verrà visualizzata prima del campo di testo. Se l'utente clicca su OK il metodo prompt restituisce il valore inserito come una **stringa**, in caso contrario restituisce null.

```
var nome = prompt(‘’inserisci il tuo nome’’);
console.log(nome);
```

Esercizio 3.1

Scrivere un programma che dichiara e valorizza 5 numeri a proprio piacimento e restituisca in output la somma e la media.

Esercizio 3.2

Scrivere un programma che dichiara e valorizza una variabile con l'anno corrente e un’altra variabile con il proprio anno di nascita e determini:

- l’età della persona
- quanti anni sono necessari per raggiungere i 100

Restituisca in output entrambi i risultati.

Esercizio 3.3

Scrivi un programma che dato un numero di secondi, calcoli la quantità di ore, minuti e secondi corrispondenti e poi stampi il risultato. L'output avrà solo numeri interi.

Esempio:

Input: 12560

Output: 3 ore, 29 minuti e 20 secondi

Esercizio 3.4

Questo codice chiede all'utente di inserire due numeri, quindi ne mostra la somma. Funziona in maniera errata. L'output nell'esempio qui sotto è 12. Perché? Prova a correggerlo, il risultato deve essere 3.

```
var a = prompt("Inserisci il primo numero");   // 1
var b = prompt("Inserisci il secondo numero");  // 2

console.log(a + b); // 12
```

Esercizio 3.5

Quale sarà il valore finale delle variabili a, b, c e d dopo l'esecuzione del codice sotto?

```
let a = 1, b = 1;

let c = ++a; // ?
let d = b++; // ?
```

Esercizio 3.6

Quale sarà il risultato di a e x dopo l'esecuzione del codice sotto?

```
let a = 2;
let x = 1 + (a *= 2);
```