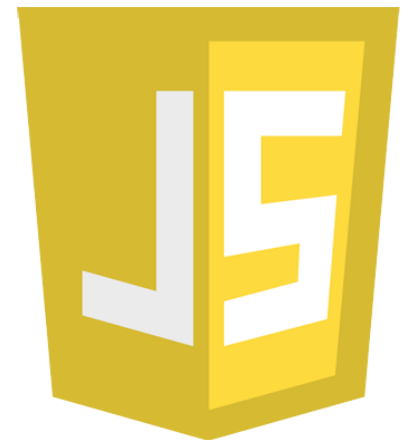


JAVASCRIPT

Riccardo Cattaneo

Lezione 2



Le Variabili

Possiamo immaginare le variabili come delle “scatole” all’interno delle quali immagazzinare dati. In JavaScript per creare delle variabili è sufficiente assegnare un contenuto ed anche se non è obbligatorio, aggiungere **sempre** la keyword **var** (quando scegliamo un nome alla variabile ricordiamoci di non inserire spazi, usare la convenzione camelCase. Può iniziare con _ \$ a-z). Ad esempio :

JS esempio.js > ...

```
1  console.log('hello world');  
2  
3  var nome  = "Corso Javascript";  
4  var _tipo = "Linguaggio di programmazione";  
5  var $lato = "Lato client e lato server"  
6  
7  console.log(nome);  
8  console.log(_tipo);  
9  console.log($lato);
```

Dichiarazione implicita delle variabili

JavaScript non prevede la dichiarazione obbligatoria delle variabili. Il semplice utilizzo di un identificatore indica all'engine di creare implicitamente la variabile, se non esiste già. Tuttavia questa pratica può favorire errori ed ambiguità.

variabile = 3;

Una variabile non dichiarata viene automaticamente creata al primo utilizzo ed è accessibile da qualsiasi punto di uno script. Questo significa che in script complessi si può correre il rischio di utilizzare involontariamente variabili con lo stesso nome generando comportamenti non previsti. È opportuno pertanto **dichiarare sempre una variabile** tramite la parola chiave **var**:

var miaVariabile = 3;

Commenti, punto e virgola e maiuscole

Il codice JavaScript è composto da una sequenza di istruzioni che viene eseguita dal browser. In questa sequenza, ciascuna istruzione è delimitata da un punto e virgola, come nel seguente esempio:

```
var x = 5;  
x = x + 1;
```

Terminare un'istruzione con **il punto e virgola non è obbligatorio** in JavaScript, possiamo scrivere la sequenza precedente anche così:

```
var x = 5  
x = x + 1
```

In base alle specifiche del linguaggio, il parser effettua un inserimento automatico del punto e virgola. Tuttavia **è buona norma inserire sempre la punteggiatura di terminazione** per evitare ambiguità e risultati inattesi.

Come ogni linguaggio, JavaScript prevede delle sequenze di caratteri per **inserire commenti nel codice**. Tutto ciò che viene marcato come commento non viene preso in considerazione dall'interprete JavaScript. Possiamo inserire due tipi di commenti nel codice:

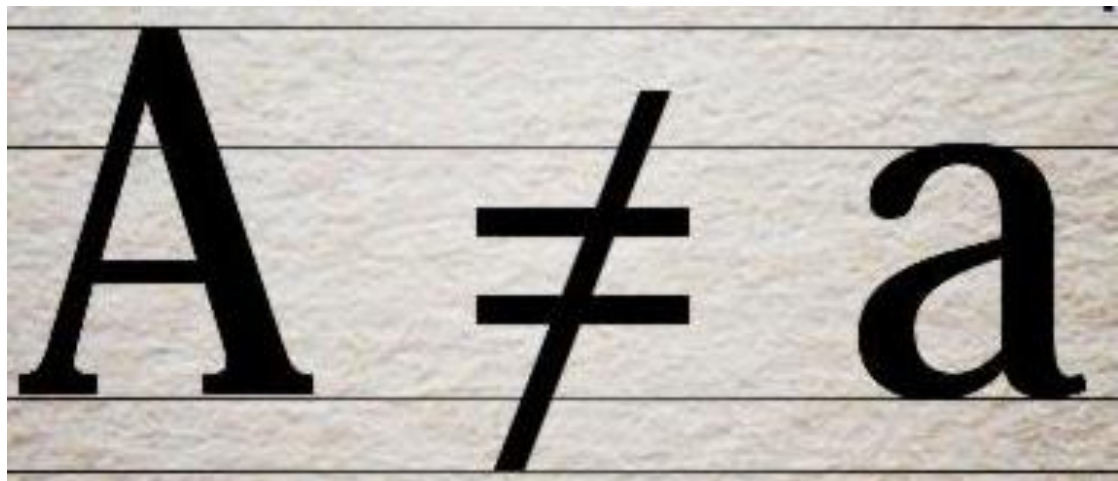
- commento per singola riga;
- commento multiriga.

Il **commento a singola riga** (inline) inizia con i caratteri **//** (doppio slash)

Il **commento multiriga** (multiline) prevede la sequenza iniziale **/*** (slash e asterisco) e si conclude con ***/** (asterisco e slash)

JavaScript è case sensitive

Un altro aspetto sintattico da tenere in considerazione è il fatto che Javascript è *case sensitive*, cioè fa **distinzione tra maiuscole e minuscole** nei nomi di istruzioni, variabili e costanti.



Parole riservate

Un'altra regola che devono rispettare le variabili è che non possono essere delle **parole riservate** del linguaggio. Per sapere l'elenco di tutte le parole è possibile cercarlo su internet.

Consiglio come sito di riferimento il w3schools (https://www.w3schools.com/js/js_reserved.asp) che non è il sito ufficiale del linguaggio, ma è ben fatto e completo.

JavaScript Reserved Words

[< Previous](#)[Next >](#)

In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Words marked with* are new in ECMAScript 5 and 6.

Stringhe, numeri e altri tipi di dati

Javascript prevede cinque **tipi di dato primitivi**, stringhe, numeri, booleani, null e undefined, e un tipo di dato complesso, **gli oggetti**.

Javascript ci consente di trattare i tipi primitivi (stringhe, numeri, etc.) **come se fossero oggetti**. Mette a disposizione diversi metodi per farlo, che molto presto studieremo, prima però dobbiamo capire come funzionano, perché ovviamente **i tipi primitivi non sono oggetti** (cercheremo quindi di fare chiarezza).

Stringhe in JavaScript

Una **stringa in JavaScript** è una sequenza di caratteri delimitata da **doppi o singoli apici**. Le seguenti sono esempi di stringhe:

- "stringa numero uno"
- 'stringa numero due'

Non c'è una regola per stabilire quale delimitatore utilizzare. L'unica regola è che il delimitatore finale deve essere uguale al delimitatore iniziale. Questo ci consente di scrivere stringhe come le seguenti senza incorrere in errori:

- "l'altro ieri"
- "questa è una 'stringa'"

Un tipo speciale di stringa è la **stringa vuota**, cioè una stringa senza caratteri. Essa può essere rappresentata indifferentemente come "" oppure "".

Escape di un carattere

Per poter utilizzare un carattere speciale come se fosse uno normale, è sufficiente farlo precedere da un backslash: \". Abbiamo appena fatto l'escaping di un carattere.

```
console.log(" questo è un \"corso speciale\" ");
```

Caratteri Speciali

All'interno delle stringhe è possibile indicare alcuni caratteri speciali di JavaScript ovvero delle sequenze che costituiscono un mezzo per formattare il testo:

<code>\&</code>	visualizza la & commerciale
<code>\n</code>	nuova riga
<code>\r</code>	ritorno a capo
<code>\t</code>	tab orizzontale
<code>\\</code>	visualizza il backslash

Concatenazione di stringhe

L'unico **operatore su stringhe** specifico è l'operatore di concatenazione. Esso consente di creare una nuova stringa come risultato della concatenazione di due stringhe ed è rappresentato dal simbolo del “più” (+):

"piano" + "forte" // "pianoforte"

Il suo utilizzo è abbastanza immediato, ma con le variabili si possono ottenere risultati non previsti dal momento che il simbolo utilizzato è lo stesso dell'addizione. Ad esempio, nella seguente espressione:

x + y

come fa JavaScript a stabilire se deve sommare o concatenare? Vedremo più avanti la risposta a questa domanda e i possibili effetti indesiderati che si possono generare.

Come detto in precedenza le stringhe in javascript sono oggetti, in quanto tali ha proprietà e metodi. L'unica proprietà è **length** che restituisce il numero di caratteri presenti all'interno di una stringa. Mentre i metodi più importanti sono :

toUpperCase() : trasforma la stringa tutta in maiuscolo.

toLowerCase() : trasforma la stringa tutta in minuscolo.

split() : divide la stringa in array prendendo come delimitatore il carattere passato all'ingresso del metodo.

replace() : sostituisce una parte della stringa prendendo la stringa passata per prima e la sostituisce con la stringa passata per secondo parametro.

Numeri in JavaScript

JavaScript ha un unico tipo di dato numerico, cioè non c'è distinzione formale, ad esempio, tra intero e decimale. Internamente tutti i valori numerici sono rappresentati come numeri in virgola mobile, ma se non è specificata la parte decimale il numero viene trattato come intero (sono a 64 bit i numeri da $-9.223.372.036.854.775.808$ a $+9.223.372.036.854.775.807$).

- `var interoNegativo = -10;`
- `var zero = 0;`
- `var interoPositivo = 123;`
- `var numeroDecimale = 0.52;`
- `var altroNumeroDecimale = 12.34;`
- `var decimaleNegativo = -1.2;`

Javascript per venire in aiuto al programmatore mette a disposizione delle funzioni globali che lo aiutano nella gestione dei numeri, le funzioni più conosciute e più usate sono :

- `toFixed()` (visualizza solo N cifre decimali)
- `parseInt()` (cast da stringa a intero)
- `parseFloat()` (cast da stringa a decimale)

toFixed()

Una funzione globale messo a disposizione dal linguaggio è **toFixed()**. E' una funzione che prende in ingresso un numero che sta ad indicare quante cifre decimali visualizzare, arrotondando per eccesso o per difetto. Ad esempio :

```
1  
2   var importo = 12.6789;  
3   console.log(importo.toFixed(2)); // 12.68  
4
```

ATTENZIONE

Fare però attenzione ad una regola molto importante della funzione `toFixed()` : si applica su un numero (la funzione `toFixed()` è una funzione che «appartiene» **solo** ai numeri) ma che «ritorna» una stringa.

```
1
2  var importo = 12.6789;
3  console.log(importo.toFixed(2)); // '12.68' è una stringa
4  console.log(importo.toFixed(2) + 4); // '12.684' concatena
5
```

parseInt()

La funzione `parseInt` in Javascript viene generalmente utilizzata per convertire il suo primo argomento in un numero intero. La sintassi è la seguente: `parseInt(stringa)`. Dove `stringa` rappresenta la stringa da convertire.

```
1
2  var importo = 12.6789;
3  console.log(importo.toFixed(2)); // '12.68' è una stringa
4  console.log(importo.toFixed(2) + 4); // '12.684' concatena
5
6  var nuovoImporto = importo.toFixed(2); // '12.68' è una stringa
7  console.log(parseInt(nuovoImporto) + 4); // '16' somma
8
```

parseFloat()

La funzione `parseFloat()` fa la stessa cosa del `parseInt()` con l'unica differenza che torna un numero decimale invece di un numero intero.

```
1
2  var importo = 12.6789;
3  console.log(importo.toFixed(2)); // '12.68' è una stringa
4  console.log(importo.toFixed(2) + 4); // '12.684' concatena
5
6  var nuovoImporto = importo.toFixed(2); // '12.68' è una stringa
7  console.log(parseInt(nuovoImporto) + 4); // '16' somma
8  console.log(parseFloat(nuovoImporto) + 4); // '16.68' somma
9
```

NaN

NaN (Not a Number) è una costante globale che viene invocata quando si esegue una funzione matematica con i numeri e questa operazione torna qualcosa che non è un numero.

La prima cosa da dire è che **NaN non è uguale a se stesso**. Questo vuol dire che se ad esempio scrivo :

```
console.log( 1 === 1 );           // true  
console.log( NaN === NaN )      // false
```

Questo vuol dire che se noi vogliamo verificare se un risultato è uguale a NaN non posso confrontarlo con NaN. Ad esempio se scrivo

```
var risultato = 10 * 'prova';  
console.log(risultato);
```

Mi restituirà in console NaN. Ma se noi facciamo un confronto come ad esempio (risultato === NaN) non sarà mai true... anche se risultato in questo caso vale NaN.

IsNaN e Number.IsNaN()

Quindi come possiamo verificare se un risultato di un'operazione «Non è un numero»? Esistono due modi per saperlo : **isNaN()** (è una funzione globale) e **Number.IsNaN()** (è un oggetto globale). Quindi se provo a fare :

- `isNaN(risultato); // true`
- `Number.IsNaN(risultato); // true`

La differenza è che **Number.IsNaN** ritornerà **true solo se gli viene passato NaN**. Se gli passiamo qualunque altra cosa ritornerà false. Se ad esempio gli passiamo :

```
Console.log(Number.IsNaN(10));      // false
```

```
Console.log(Number.IsNaN('ciao'));  // false
```

Se invece usiamo la funzione globale **isNaN()** **ritornerà true se gli viene passato qualsiasi valore diverso da un numero**. In ogni modo cerca di fare un cast ad un numero (ad esempio stringa vuota e false la converte a 0). Se riprendiamo l'esempio di prima e scrivo :

```
console.log(isNaN('ciao')); // true  
console.log(isNaN(""));    // false  
console.log(isNaN(false))  // false
```

Consiglio...

Normalmente, quando si svolgono operazioni matematiche si preferisce usare **Number.IsNaN** in quanto javascript, quando effettuo delle operazioni e per qualche motivo il risultato non torna un numero (come nel precedente esempio 'prova'*5) ritorna sempre il valore Nan.

Quindi se usiamo Number.IsNaN torna true solo se è NaN.

I tipi di dato null

JavaScript prevede due tipi di dato speciali per rappresentare valori nulli e non definiti.

Il tipo di dato **null** prevede il solo valore null, che rappresenta un valore che non rientra tra i tipi di dato del linguaggio, cioè non è un valore numerico valido, né una stringa, né un oggetto.

È possibile assegnare il valore null ad una variabile come nel seguente esempio:

```
var x = null;
```

I tipi di dato undefined e booleano

Il tipo di dato **undefined** rappresenta un valore che non esiste. Anche questo tipo di dato contiene un solo valore: undefined.

Questo è il valore di una variabile non inizializzata, a cui non è stato assegnato nessun valore, nemmeno null.

Il tipo di dato booleano prevede due soli valori: true (vero) e false (falso).

Tipizzazione debole o dinamica

Vediamo ora come javascript è un linguaggio a tipizzazione debole. Questo vuol dire che una variabile può assumere valori di tipi diverso, tipizzazione modificata in base al valore assegnato.

Se ad esempio dichiaro una variabile **var test = 'buongiorno a tutti'**; questa variabile javascript la vede come una stringa perché dopo l'uguale ho messo l'apice. Se invece alla riga successiva scrivo **test = 1;** la stessa variabile test per javascript adesso è in numero.

typeof

L'operatore **typeof** è un modo semplice per controllare il tipo di una variabile nel codice. Questo è importante perché JavaScript è un linguaggio tipizzato dinamicamente ed è utile per la convalida dei dati e il controllo del tipo.

La sintassi è la seguente : far seguire dall'operatore `typeof` il nome della variabile in modo da ricevere il tipo di dato in quel momento.

```
var test;  
console.log("test" + " => " + typeof test);  
  
var test = "buongiorno a tutti";  
console.log("test" + " => " + typeof test);  
  
var test = 10;  
console.log("test" + " => " + typeof test);  
  
var test = true;  
console.log("test" + " => " + typeof test);
```

```
test => undefined  
test => string  
test => number  
test => boolean
```


Esercizio 1.1

Si scriva un programma in un file javascript con il nome `esercizio1_1.js` JavaScript che definisce una variabile `nome` che ha come valore il proprio nome (es. Mario).

Il programma deve stampare in output la stringa `Ciao Mario`, utilizzando la variabile definita e l'operatore di concatenazione. Eseguire l'esercizio nelle due modalità : **lato client e lato server**.

Esercizio 1.2

Lato client : creare una pagina html contenente uno script che dichiara una variabile numerica, assegnandole il valore iniziale di 100. Stampare a video il valore della variabile all'interno di un paragrafo della pagina. Successivamente cambiare valore alla variabile (valore = 70) e stampare nuovamente all'interno di un paragrafo il nuovo valore. **Lato server** : eseguire l'esercizio sulla console.

Primo esercizio JavaScript

Valore iniziale: 100

Valore finale: 70

Esercizio 1.3

Creare un progetto Javascript che dichiari e contestualmente inizializzi una variabile $x = 10$; che dichiari la variabile y e successivamente assegni ad y il valore di x ; che dichiari e contestualmente inizializzi la variabile z e gli assegna un nome di persona a proprio piacimento e tramite output stampa il valore delle variabili ed il tipo.

Eseguire l'esercizio nelle due modalità : **lato client e lato server**.


Esercizio 1.4

Creare un progetto Javascript che dichiari una variabile di tipo stringa con valore '50', stampare a video il valore ed il tipo. Verificare tramite una funzione globale se il valore è di tipo numerico oppure no e stampare a video l'esito (true o false).

Successivamente convertire la stringa in un numero e stampare a video il valore con il tipo (che ora è cambiato).

Esercizio 1.5

Creare un progetto Javascript che memorizza all'interno di una variabile i giorni della settimana separati dalla virgola e stampare a video tramite `console.log()` il risultato. Scrivere il codice in modo che, facendo la modifica ad **un solo valore**, i giorni della settimana vengono visualizzati uno sotto l'altro senza la virgola.

Lunedì, Martedì, Mercoledì, Giovedì, Venerdì, Sabato, Domenica 

Lunedì
Martedì
Mercoledì
Giovedì
Venerdì
Sabato
Domenica