Java Script

Rev. 4.4 del 24/04/2021

Java Script

Con il termine **script** si intende un **insieme di comandi interpretati da un certo applicativo** che, nel caso di Java Script, è costituito dal browser. Java Script è un linguaggio che consente di scrivere, all' interno di una pagina HTML, delle sezioni di codice che verranno interpretate dal browser al momento della visualizzazione della pag

Un tag HTML può richiamare uno script js mediante l'utilizzo dei cosiddetti **ATTRIBUTI DI EVENTO** che, in risposta ad un certo evento come ad esempio un click, consentono di associare delle righe di codice da eseguire nel momento in cui si verificherà l'evento. La potenzialità di java script consiste principalmente nel fatto che, indipendentemente dal tag che ha scatenato l'evento, **le istruzioni possono accedere in modo diretto a qualunque tag della pagina e modificarne sia l'aspetto grafico sia il contenuto**.

Le istruzioni Java Script possono essere scritte:

- Direttamente all'interno del tag (se le azioni da compiere sono poche)
```
<input type="button" value = "Esegui" onClick="alert('salve')">
```
- All'interno della sezione di HEAD della pagina all'interno del seguente TAG
```
<script type="application/javascript" >
    istruzioni js
</script>
```
- In un file esterno con estensione .js richiamato nel modo seguente:
```
<script type="application/javascript" src="index.js"> </script>
```

Sintassi base

- Le **istruzioni base** sono le stesse dell'ANSI C e sono **case sensitive** (distinzione maiuscole / minuscole)
- Il **punto e virgola** finale è facoltativo. Diventa obbligatorio quando si scrivono più istruzioni sulla stessa riga.
- Lo switch accetta come parametro anche le stringhe (come in C#, a differenza del C ANSI)
- Java Script non distingue tra **virgolette doppie** e **virgolette semplici.** Ogni coppia deve essere dello stesso tipo
  Per eseguire un terzo livello di annidamento si può usare \" oppure `&quot;` : alert ("salve \"Mondo\" ");
- Le variabili dichiarate fuori dalle funzioni hanno visibilità globale ma **non vengono inizializzate.**

Dichiarazione delle variabili

```
let A, B;
```

- **La dichiarazione delle variabili non è obbligatoria**. Una variabile non dichiarata viene automaticamente creata in corrispondenza del suo primo utilizzo **come variabile globale**, e dunque sarà visibile anche fuori dalla funzione. Facile causa di errori. E' consigliato impostare all'inizio "`use strict`"; che rende obbligatoria la dichiarazione delle variabili. Oltre a `let` è possibile utilizzare `const` per le costanti.
- **Le variabili non sono tipizzate**, cioè nella dichiarazione non deve essere specificato il tipo. I tipi sono comunque **Number, String, Boolean, Object**, e il cast al tipo viene eseguito automaticamente in corrispondenza di ogni assegnazione. Il tipo **Float** non esiste (usare Number) ma esiste parseFloat( ).
- E' comunque possibile tipizzare una variabile creando una istanza esplicita:
```
let n=new Number();
let s=new String();
let b=new Boolean();   // I valori true e false sono minuscoli.
```
  Le parentesi tonde sono quelle del costruttore e possono indifferentemente essere inserite o omesse.
- Al momento della dichiarazione ogni variabile viene inizializzata al valore `undefined` che non equivale a \0 e nemmeno a stringa vuota, ma equivale a "non inizializzato". `if (A == undefined)` ……
- Per i riferimenti si utilizza normalmente il valore `null` (che numericamente è uguale ad undefined).
- L'operatore **typeof** restituisce come stringa il tipo corrente di una variabile. ES: if (typeof A == "Number")
- In javascript non è consentito passare i parametri scalari per riferimento. Eventualmente devono essere racchiusi all'interno di un Object

Funzioni Utente

Occorre omettere **sia** il tipo del parametro **sia** il tipo del valore restituito.

```
function eseguiCalcoli(n) {
     istruzioni;
     return risultato;
}
```

L'accesso agli elementi della pagina

Il metodo `document.getElementById` consente di accedere al TAG avente l'ID indicato come parametro. Restituisce un riferimento all'elemento indicato. [In caso di più elementi con lo stesso ID ritorna il primo che trova].

```
let ref = document.getElementById("txtNumero");
let n = parseInt(ref.value);
```

Il metodo `getElementsByName`

Notare il plurale (**Elements**) necessario in quanto il **`name`** non è univoco e quindi gli elementi individuati sono normalmente più d uno. Restituisce un oggetto **NodeList** contenente tutti gli elementi aventi il **`name`** indicato, nell'ordine in cui sono posizionati all'interno della pagina. Un NodeList è una collection simile ad un **vettore enumerativo** accessibile tramite indice numerico e con la proprietà .length. Non riconosce però tutti i metodi tipici di un vettore enumerativo (ad esempio non riconosce il metodo indexOf()).
Anche nel caso in cui sia presente un solo elemento viene comunque sempre restituito un vettore (lungo 1).

```
let opts = document.getElementsByName("optHobbies");
  for (let i=0;i< opts.length;i++)
    console.log(opts[i].value);
```

I meotdi `getElementsByTagName` e `getElementsByClassName`

La prima restituisce sotto forma di **NodeList** tutti i tag di un certo tipo (es DIV).
La seconda restituisce sotto forma di **NodeList** tutti i tag che implementano una certa classe.
Entrambe accedono a figli e nipoti. Restituiscono un **NodeList** anche nel caso in cui sia presente un solo elemento:

```
let body= document.getElementsByTagName("body")[0];
body.style.backgroundColor="blue";
let carte = document.getElementsByClassName("carta");     // senza il puntino
```

Ricerche annidate

```
let wrapper = document.getElementById("wrapper")
let vet = wrapper.getElementsByTagName("p");  // tag <p> interni a wrapper
```

Le proprietà `.innerHTML` `.outerHTML` `.textContent`

Consentono entrambe di accedere al contenuto di qualsiasi tag HTML, dove per contenuto si intende ciò che viene scritto fra l'**apertura del tag** e la **chiusura del tag** stesso. Consentono di inserire al loro interno altri tag HTML

innerHTML accede al contenuto del tag, tag radice escluso
outerHTML restituisce anche il tag radice (cioè il tag al quale si applica la property). Comoda per cambiare tag.
textContent accede/imposta il solo contenuto testuale del tag (definita all'interno delle specifiche XML)
innerText simile alla precedente, standardizzata più tardi nel DOM model. Leggermente più pesante. Differenze:

- Restituisce anche eventuali testi hidden
- Riconosce il \n
- E' definita solo nel caso degli HTMLElement (che sono un sottoinsieme dei NodeElement)

La gestione degli eventi

Gli attributi di evento sono quelli che consentono di richiamare le funzioni utente scritte nella sezione Java Script.
`<input type="button" value = "Esegui" onClick="esegui()">`
All'interno di un attributo di evento si possono richiamare più funzioni separate da **;**.

L'evento onLoad
Nel file JS è possibile scrivere codice diretto "esterno" a qualsiasi funzione. Questo codice viene però eseguito **prima** che sia terminato il caricamento della pagina per cui **non** può fare riferimento agli oggetti della pagina che verranno istanziati soltanto più tardi. In tal caso i riferimenti saranno tutti NULL ed il programma va in errore.
Per eventuali inizializzazioni occorre utilizzare l'evento `<body onLoad="init()">`

L'evento onClick nel tag <a>
Al posto del pulsante si può utilizzare un **collegamento ipertestuale fittizio** che, anziché aprire una nuova pagina, esegua una funzione Java Script. Esistono 2 sintassi equivalenti. Omettendo href sparirebbe la sottolineatura.

```
<a href='#'  onClick='esegui()'> Esegui  </a>
<a href='javascript:esegui()'>   Esegui  </a>
```

Gli oggetti base

L'oggetto Math

Proprietà Statiche

Math.PI	PI graco	3,1416
Math.E	Base logaritmi naturali	2,718

Math.LN2	Logaritmo naturale di 2
Math.LN10	Logaritmo naturale di 10
Math.LOG2E	Logaritmo in base 2 di e
Math.LOG10E	Logaritmo in base 10 di e

Metodi Statici

Math.sqrt(N)	Radice Quadrata
Math.abs(N)	Valore assoluto di N (modulo)
Math.max(N1, N2)	Restituisce il maggiore fra i due numeri
Math.min(N1,N2)	Restituisce il minore fra i due numeri
Math.pow(10,N)	Elevamento a potenza : 10^N // oppure 10 ** N
Math.round(N)	Arrotondamento all'intero più vicino
Math.ceil(N)	Arrotondamento all'intero superiore
Math.floor(N)	Tronca all'intero inferiore. **L'operatore di divisione / restituisce un double anche nel caso di divisione fra interi, per cui il risultato deve eventualmente essere troncato**.
%	restituisce il resto di una divisione fra interi
Math.random()	Restituisce un double 0 <= x < 1 **esattamente come in ANSI C**. *Randomize è automatico* Per generare un num tra A e B : `Math.floor((B-A+1)*Math.random())+ A`

```
let n = 12.34567;
alert(n.toFixed(2))     // 12.34
```

Le Stringhe

String è un oggetto che deve pertanto essere istanziato**.** Le stringhe sono **immutabili** come in C#
Per dichiarare una stringa sono disponibili le due seguenti sintassi (l'istanza statica **`String s1`** non è supportata):

```
var s1;                  // riferimento non tipizzato. Contiene undefined
var s2 = new String();   // riferimento tipizzato ma che contiene sempre undefined
```

Inizializzazione di una stringa in fase di dichiarazione

```
var s3 = "salve";               // oggetto stringa inizializzato a "salve"
var s4 = new String("salve");   // oggetto stringa inizializzato a "salve"
var len = s1.length         // lunghezza della stringa
```

s1 = s1.**toUpperCase**() // Restituisce la stringa convertita in maiscuolo
s2 = s1.**toLowerCase**() // Restituisce la stringa convertita in minuscolo
s2 = s1.**substr**(posIniziale, [qta]) // Estrae i caratteri da posIniziale per una lunghezza pari a qta.
s2 = s1.**substring**(posIniziale, posFinale) // Estrae i caratteri da posIniziale a posFinale, posFinale escluso.
s1 = "Salve a tutti" s2=s1.substring(0, 5) => "Salve". Se posFinale > length si ferma a fine stringa
ris = s1.**includes**(s2) // Restituisce **true** se s1 include s2. **false** in caso contrario. **Disponibile anche sui vettori**
N = s1.**indexOf**(s2, [pos]) // Ricerca s2 dentro s1 a partire dalla posizione pos. (Il primo carattere ha indice 0).
Restituisce la posizione della prima ricorrenza. Se non ci sono occorrenze restituisce –1
Se s2 = "" restituisce il carattere alla posizione pos. **Disponibile anche sui vettori**
N = s1.**lastIndexOf**(s2, [pos]) // Ricerca s2 dentro s1 a partire dalla posizione pos andando all'indietro.
Se pos non è specificato parte dalla fine della stringa (length – 1).
N = s1.**search**(s2) // Simile a indexOf ma: non accetta il 2° parametro pos, **accetta le Regular Expr,** non è disponibile sui vettori
s2 = s1.**replace**("x", "y") // Sostituisce SOLO la prima occorrenza.
replaceAll("x", "y") // Sostituisce tutte le occorrenze

C = s1.**charAt**(pos) // Restituisce come **stringa** il carattere alla posizione pos (partendo a 0)
N = s1.**charCodeAt**(pos) // Restituisce il **codice Ascii** del carattere alla posizione pos (o del primo se manca pos)
s2 = **String.fromCharCode**(97,98,99) Viene istanziata una nuova stringa contenente "abc"
s2 = n.**toString**() // Restituisce la conversione in stringa. **n.toString(16)** restituisce una stringa esadecimale
n.toString(2) restituisce una stringa binaria
vect = s.**split**("separatore"); // Esegue lo split rispetto al carattere indicato.
s = vect.**join**("separatore"); // Restituisce in un'unica str tutti gli elementi del vett separati dal chr indicato

n.**toFixed**(3) // indica il numero di cifre dopo la virgola da visualizzare

- Proprietà e metodi possono essere applicati anche in forma diretta: "Mario Rossi".length
- A differenza di Ansi C in javascript le stringhe possono essere **confrontate** direttamente utilizzando gli operatori < >

Una semplice funzione per aggiungere uno 0 davanti ad un numero <10

```
function pad(number) {
     return (number < 10 ? '0' : '') + number
}
```

Altri metodi relativi alla formattazione grafica

.big()	restituisce una stringa in testo grande
.blink()	restituisce una stringa con testo lampeggiante
.bold()	restituisce una stringa in grassetto
.fontsize()	restituisce una stringa avente il fontsize specificato
.italics ()	restituisce una stringa in corsivo
.small()	restituisce una stringa in testo piccolo
.strike()	restituisce una stringa barrata
.sup()	restituisce una stringa in formato apice
.sub()	restituisce una stringa in formato pedice

Vettori Enumerativi

I vettori sono oggetti a indice 0 e possono essere dichiarati anche SENZA specificare la dimensione.
Il vettore crescerà dinamicamente man mano che si aggiungeranno elementi al suo interno.
Per creare un vettore sono disponibili due sintassi equivalenti, una breve ed una più prolissa:

var vect = [ ]; // dichiaro un vettore enumerativo al momento avente lunghezza 0
var vect = **new Array**() // Simile alla precedente ma tipizzata

L'assegnazione della dimensione in fase dichiarativa è comunque talvolta necessaria e può essere realizzata soltanto mediante la seconda delle precedenti sintassi:
var vect = **new Array**(30) // Viene istanziato un array di 30 elementi

Attenzione che invece la seguente sintassi dichiara un vettore lungo 1 contenente il valore 30:
var vect = [30]; // Viene istanziato un array contenente un solo elemento, con valore 30.

Inizializzazione di un vettore in fase di dichiarazione

var vect = **new Array**(30,31) // Viene istanziato un array di **2** elementi, contenente i valori 30 e 31
var vect = [30, 31]; // Viene istanziato un array di **2** elementi, contenente i valori 30 e 31

var vect = **new Array** ('pippo', 'pluto', 'minnie');
var vect = ['pippo', 'pluto', 'minnie'];

var vect = **new Array (**'pippo'**)**; // Viene istanziato un array di **1** elemento, contenente 'pippo'
var vect = [titolo, autore, categoria, prezzo]; // Viene caricato nel vettore il contenuto delle variabili indicate

Gli Array possono essere **eterogenei**, cioè contenere dati differenti: numeri, stringhe, oggetti, etc
Per accedere all'i-esimo elemento si usano le **parentesi quadre** vectStudenti[3] = "Mario Rossi"

```
vect.length          // lunghezza del vettore
vect.push("a", "b", "c");   // Tutti i valori indicati vengono aggiunti in coda al vettore
vect.pop( );         // Estrae l'ultimo elemento in coda al vettore
vect[vect.length] = "value";   // Equivalente al push(). E' anche consentita una istruzione del tipo:
vect[100] = "value";   // Se il vettore fosse lungo 10, verrebbe creata la cella 99,
                          con le celle 10-98 undefined.  Però length==100
pos = vect.indexOf(item)      // Restituisce la posizione dell'elemento indicato.
ris = vect.includes(item)     // Restituisce true se vect include l'elemento indicato.
                          // Entrambe funzionano correttamente anche con i vettori di json

vect.splice(pos, n);  //  Consente di eliminare n oggetti a partire dalla posizione pos.
                      //  Se pos == -1 elimina l'ultimo elemento !
                      //  Restituisce un vettore contenente gli elementi eliminati.
```

Dopo i primi due parametri è possibile passare a splice altri elementi che vengono aggiunti nella posizione indicata
`vect.splice(pos,3,"A","B")` Vengono rimossi 3 elementi alla posizione pos e vengono aggiunti "A" e "B"
Se come secondo parametro viene passato 0 **`splice()`** aggiunge gli elementi indicati senza rimuoverne altri

```
vect.sort();          // Metodo di ordinamento del vettore dall'elemento più piccolo al più grande
vect.reverse();       // Inverte gli elementi di un vettore. Il prima diventa l'ultimo e viceversa.
var vect2= vect.slice(1,3);  // Restituisce un nuovo vettore contenente gli elementi 1 e 2 (3 escluso) del vettore
                             originale. Numeri negativi consentono di selezionare a partire dal fondo.
                             slice(-2) restituisce gli ultimi 2 elementi. Disponibile anche sulle stringhe
```

I Cicli `for of` e `for in`

Presenta due sintassi diverse a seconda che si voglia scorrere un vettore enumerativo o associativo:

vettore enumerativo : `var vet=["a", "b", "c"];`

```
for(var item of vet) alert(item);    //  item rappresenta il contenuto della cella

for(var i in vet) {       //  i è una stringa che rappresenta l'indice (chiave) della cella
   i = parseInt(i);       //  prima di qualunque altra operazione occorre convertire in intero
   alert(vet[i])
}
```

Attenzione che, nel ciclo **FOR OF**, la variabile di ciclo **`item`** (detta **cursore**) è una **copia** del contenuto della cella, per cui eventuali modifiche apportate al cursore vanno perse al termine del ciclo.

vettore associativo : `var vet={a:"a1", b:"b1"};`

```
for(var key in vet) alert(vet[key]) //  key è una stringa che rappresenta la chiave
```

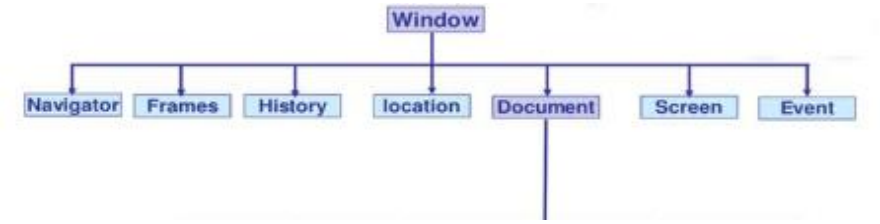
Le Matrici

In Java Script non esiste il concetto canonico di matrice. E' però possibile, dopo aver dichiarato un vettore, trasformare ogni cella del vettore in un puntatore ad un nuovo vettore. Esempio di creazione di una **matrice 10 x 3** :

```
var mat = new Array (10);
// Le seguenti istruzioni sono obbligatorie Devo far capire a JS che si tratta di una matrice e non di un vettore
for (i=0; i<mat.length; i++)
    mat[i] = new Array (3);     // meglio: mat[i]=[]
mat[0][0] = "Marsha";   mat[0][1] = "Carol";   mat[0][2] = "Greg";
```

Il DOM di una pagina HTML

Il **DOM** (*Document Object Model*) è una **API** (*Application Programming Interface*), **cioè un insieme di oggetti e funzioni che consentono di accedere tramite javascript a tutti gli elementi della pagina HTML**.
Il DOM è definito dal W3C. L'oggetto base è l'oggetto **`window`** che rappresenta la scheda di navigazione corrente e rimane allocato finché la finestra non viene chiusa.



I principali oggetti figli dell'oggetto windows sono:

- **document** = documento html caricato nella finestra (analogo di xmlDoc nei documenti XML)
- **location** = informazioni sulla url corrente
- **navigator** = oggetto di navigazione. Utile ad esempio per eseguire dei redirect.

Principali Metodi dell'oggetto window

window è l'oggetto predefinito del DOM, per cui i suoi metodi sono utilizzabili anche omettendo window stesso.

alert("Messaggio") Visualizza una finestra di messaggio con un unico pulsante OK non modificabile.

prompt("Messaggio") Rappresenta il tipico Input Box con i pulsanti OK e ANNULLA. Un secondo parametro opzionale indica un valore iniziale da assegnare al campo di immissione. Clickando su OK restituisce come stringa il valore inserito. Clickando su ANNULLA restituisce **null**

confirm("Messaggio") Finestra di conferma contenente i due pulsanti OK e ANNULLA. Restituisce **true** se l'utente clicka su OK oppure **false** se l'utente clicka su ANNULLA. Il test sul boolean è in genere diretto: if (confirm("Vuoi veramente chiudere?")) { …….. }

parseInt(s) Converte una stringa o un float in numero intero.
In caso di stringa **si ferma automaticamente al primo carattere non numerico**.
In caso di float **il numero viene troncato all'intero inferiore** (come `Math.floor()`)

parseFloat(s) Converte una stringa in float

.toString( ); Converte qualsiasi variabile in stringa

NaN Not a Number. Quando si esegue una operazione matematica su una variabile non inizializzata (o non contenente numeri) Java Script restituisce come risultato dell'espressione il valore NaN.

isNaN(s) Consente di testare se la variabile s contiene il valore NaN, nel qual caso restituisce true.

escape(s) Codifica gli spazi e tutti i caratteri particolari (come & e %) nei rispettivi codici ascii in formato esadecimale. Lo spazio diventa %20, dove il % è il carattere utilizzato in C per inserire caratteri speciali. Esempio: s1 = **escape("salve mondo")** diventa "salve%20moindo".

unescape(s) Opposta rispetto alla precedente. Sostituisce i codici % con il carattere corrispondente.
Es: var s = "RIS: " + **unescape("%B1")**; dove B1 è la codifica esadec di ± (177 dec \xB1 esa)

focus() / blur() Porta la finestra in primo piano / sotto le altre finestre

scroll(x,y) Fa scorrere la finestra di x colonne orizzontalmente e y righe verticalmente.

open`("file.html", [target], ["Opzioni separate da virgola"])`

target: `"_self"` apre la nuova pagina nella stessa scheda
`"_blank"` apre la nuova pagina in una nuova scheda
A differenza di `<a href>` in cui il default di target era `"_self"`, questa volta il default è `"_blank"`
Come target si può passare anche il nome di un **`iframe`**.

close( ) Chiude la finestra corrente se è stata aperta in locale oppure tramite lo script medesimo.
Se la pagina proviene da un server la finestra **non** viene chiusa: Funziona soltanto per finestre aperte in locale oppure tramite il metodo .open()

Gli eventi

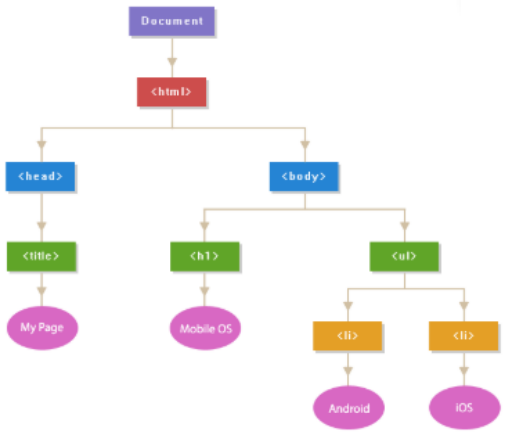
`window.onload = function(){ ..... }`

richiamato all'avvio dell'applicazione al termine del caricamento del DOM.
Rappresenta una alternativa all'evento onLoad del tag BODY.
Viene però generato **SOLO** se il body non presenta l'evento onLoad che è prioritario.

onUnload Richiamato quando si abbandona un documento (sostituito da un nuovo documento)
onResize Richiamato quando la finestra attiva viene ridimensionata

L'oggetto *`document`* e gli elementi della pagina

Rappresenta il documento HTML che si sta visualizzando all'interno dell'oggetto window.
Sono figli di **`document`** tutti i tag della pagina html, rappresentabili mediante la seguente struttura **ad albero**:



I cerchi fucsia rappresentano le foglie dell'albero

`document.documentElement`	**tag <HTML>** (root dell'albero HTML)
`document.body`	tag <BODY>
`document.forms["formName"]`	form avente il name indicato all'interno della collezione delle forms

Proprietà comuni a tutti i tag

textContent	rappresenta il solo contenuto testuale di un tag: `"testo"`
innerHTML	rappresenta l'intero contenuto html di un tag `"<p> testo </p>"`
tagName	contiene il nome stesso del tag : `h1, div, input, button`, etc

Proprietà definite SOLO per i controlli

disabled true / false. In caso di **disabled=true** il controllo non risponde più agli eventi.
Questo però NON vale ad esempio per i tag DIV e P che non sono dei controlli.

value esiste **SOLO** nel caso dei tag input e rappresenta il contenuto del tag input.
Il tag ***button*** non dispone della proprietà ***value*** ma potrò usare ***innerHTML*** oppure ***textContent***

eventi comuni

onClick( ) // return False abbatte l'evento.
onDblClick() // poco utilizzato
onMousedown( ) // Inizio click, **onMouseup( )** // Fine click
onMouseover( ) // Ingresso del mouse sull'elemento
onMouseout( ) // Uscita del mouse dall'elemento
onMousemove( ) // Spostamento del mouse all'interno dell'elemento
onKeypress( ) // Se al gestore di evento viene restituito **false**, il gestore abbatte il tasto. Non riconosce backspace
onKeydown( ) // Tramite event viene passato il codice fisico del tasto
onKeyup( ) // Tramite event viene passato il codice fisico del tasto

focus() metodo che assegna il fuoco all'oggetto
onFocus() evento generato dopo che l'oggetto ha ricevuto il fuoco
blur() metodo che toglie il fuoco all'oggetto passandolo al controllo con tabIndex successivo
onBlur() evento generato dopo che l'oggetto ha perso il fuoco

Text Box e **TextBox multiline** (TextArea)

value rappresenta il contenuto del campo, anche nel caso delle TEXT AREA.

onInput() generato dopo ogni singola variazione all'interno del textbox. Vale solo per textbox e textarea
onChange() generato dopo che è cambiato il contenuto del campo, ma soltanto quando si abbandona il campo.
select( ) metodo che seleziona il contenuto del textbox
onSelect() generato dopo che il contenuto del campo è stato (anche solo parzialmente) selezionato.

Input[type=button]

value testo del pulsante
onClick() Click sul pulsante. Il metodo **`.click( )`** consente di forzare un click sul pulsante

Radio Button

Sono mutualmente esclusivi soltanto gli option button con lo **stesso name**, che possono pertanto essere gestiti come un vettore di controlli.

.length numero di pulsanti appartenenti al gruppo
[i].checked true / false indica se il radio button è selezionato. L'indice parte da 0
[i].value Restituisce il contenuto dell'attributo value (Valore restituito in corrispondenza del submit se il radio button è selezionato)
onClick() Deve essere associato ad ogni singolo radio button e viene generato in corrispondenza del **click** sul radio button (*dopo che il cambio di stato è avvenuto*)
onChange() Nel caso dei radio button è IDENTICO a onClick. Viene generato in corrispondenza della selezione del radio button. La selezione di un elemento da codice **NON provoca la generazione dell'evento onChange**, che deve essere invocato manualmente tramite il metodo .click().
[i].click() Forza un click sul pulsante che cambia pertanto di stato. Genera sia l'evento onClick sia onChange.

In java script NON è possibile associare gli eventi onClick e onChange alla collezione dei radio button (come in jQuery) ma occorre assegnare la procedura di evento per ogni singolo pulsante

Per vedere qual è la voce selezionata all'interno di un gruppo di radio buttons occorre pertanto fare un **ciclo** sui singoli attributi **checked** e vedere per ognuno se è selezionato oppure no. Nelle ultime versioni è riconosciuta anche una proprietà riassuntiva **.value** (come per i ListBox) che però non è standard.

CheckBox

Presentano lo stesso identico funzionamento dei Radio Button, con l'unica differenza che non sono esclusivi.
Gli eventi **onClick**() e onChange devono essere applicati sul singolo CheckBox

List Box (Non riconosce placeholder)

selectedIndex	indice della voce selezionata. Inizialmente vale 0. Impostando -1 viene automaticamente visualizzata una riga iniziale vuota che scomparirà in corrispondenza della prima selezione.
length	numero di `options` presenti nella lista. Identico a **options.length**
onChange	Evento richiamato in corrispondenza della selezione di una nuova opzione.
onClick	Evento richiamato ad ogni singolo click sulla Lista (abbastanza inutile).

options	Vettore delle opzioni presenti nella lista (a base 0**). Non consente di aggiungere nuove opzioni.**
.options.length	Numero di opzioni
.options[i].selected	true / false a seconda che l'opzione sia selezionata o meno
.options[i].value	Valore Nascosto interno all'opzione
.options[i].innerHTML	testo visualizzato all'interno dell'opzione.

L'oggetto <Select> dispone di un comodissimo attributo **value** che, **in corrispondenza della selezione di una voce, contiene il value del tag <options> attualmente selezionato.** Inizialmente contiene il **value** della prima voce. Se il **value** è vuoto assume automaticamente il valore interno della OPTION (innerHTML)

Viceversa **option button e check box**, essendo costituiti da un vettore di oggetti **privo di un contenitore esterno** come <select>, non dispongono di una proprietà **value** riferita all'intero vettore, per cui per vedere quale option button è selezionato occorre fare un **ciclo** sui vari elementi. Non è disponibile nemmeno **selectedIndex**.

Il metodo **`getElementById("myList").options`** restituisce un vettore enumerativo di oggetti **`options`**.
Il contenuto di ciascuna **`option`** può essere letto con .innerHTML.
Per aggiungere una nuova opzione alla lista si può utilizzare la seguente sintassi:

```
myList.options[myList.options.length] = new Option('Text', 'value');
```

Il push in questo caso NON è accettato !

Nota

Firefox, quando si esegue il refresh di una pagina, mantiene in cache lo stato di tutti i controlli, visualizzando gli stessi valori anche dopo il refresh. Lo scopo è quello di facilitare l'autocomplete e anche quello di mantenere i valori già inseriti in una form di registrazione. A tal fine utilizza il valore **`on`** come default dell'attributo html5 autocomplete. Per disabilitare questo effetto è sufficiente impostare **`autocomplete="off"`** su tutti i tag input.

Accesso e gestione degli attributi HTML

E' possibile accedere agli attributi HTML di un elemento in due modi, che però **NON** sono del tutto equivalenti:

1. **accesso diretto** tramite il **`puntino`** che, partendo dal puntatore all'elemento, consente di accedere a qualsiasi attributo html di quell'elemento
```
var ref = document.getElementById("myImg");
ref.id="id1";
ref.className = "myClass"
```
2. metodi **`getAttribute( )`** **`setAttribute( )`** e **`removeAttribute( ).`**
```
ref.setAttribute ("id", "id1");
ref.setAttribute ("class", "myClass");
```

Default state e Current State

- Gli attributi definiti staticamente all'interno del file HTML rappresentano il cosiddetto ***defaultState*** del controllo, che rimane immutato anche se l'utente a run time modifica il contenuto del controllo attraverso l'interfaccia grafica, scrivendo all'interno di un TextBox o selezionando un CheckBox.
- I valori che l'utente scrive a run time attraverso l'interfaccia grafica vengono salvati all'interno del cosiddetto ***currentState***, il cui valore inizialmente **coincide** con il defaultState, ma poi viene modificato nel momento in cui l'utente scrive qualcosa all'interno dell'interfaccia grafica.

Lo scopo di questa doppia informazione è quella di mantenere memorizzato il valore scritto staticamente all'interno del file html in modo da poterlo ripristinare in qualsiasi momento.

getAttribute e setAttribute vanno a leggere/scrivere il ***defaultState*** del controllo. Se il valore del campo viene modificato attraverso l'interfaccia grafica, getAttribute() non se ne accorge. Allo stesso modo se si usa setAttribute() per modificare il defaultState del campo **dopo** che l'utente ne ha modificato il contenuto tramite l'interfaccia grafica, la modifica NON verrà visualizzata sulla pagina.

L'accesso diretto tramite puntino viceversa accede in lettura / scrittura al ***currentState*.**
Anch'esso però presenta un suo limite: In lettura può essere utilizzato soltanto per gli attributi considerati "validi" nelle specifiche del DOM. Ad esempio l'attributo "**name**" è considerato valido soltanto per alcuni controlli (<a>, <applet>, <button>, <form>, <frame>, <iframe>, <img>, <input>, <map>, <meta>, <object>, <param>, <select>, and <textarea>).
Sugli altri controlli restituisce inizialmente `undefined`. In scrittura è invece utilizzabile su qualsiasi attributo.

Un nuovo attributo impostato tramite setAttribute() può essere letto SOLO con getAttribute() e non con accesso diretto e viceversa. O si usa una sintassi oppure si usa l'altra.

Alla luce di quanto sopra è raccomandato utilizzare sempre l'accesso diretto, accesso diretto che diventa indispensabile per quegli attributi che possono essere modificati dinamicamente all'interfaccia grafica, cioè:

- l'attributo **value** dei textBox sia singoli che multiline
- l'attributo **checked** di checkbox e radiobutton
- l'attributo **selectedIndex** di un tag select non dispone di un *defaultState* html e dunque è accessibile soltanto tramite accesso diretto con il puntino (così come anche il **value** riassuntivo).
  La stessa cosa vale per il value di un textBox multiline, che non dispone di un *defaultState* html.

*Notare che se l'utente non apporta variazioni all'interfaccia grafica (ad esempio se si utilizzano dei checkbox soltanto in visualizzazione per indicare degli stati), setAttribute() può essere utilizzato anche nei casi precedenti. Però, ad esempio nel caso dei **radio button**, in corrispondenze del setAttribute("checked"), occorre eseguire manualmente un removeAttribute("checked") sul radio button precedentemente selezionato, altrimenti dopo un po' rimangono tutti selezionati e vince l'ultimo.*

Gli attributi booleani nel defaultState e nel currentState

Il **dafaultState** riflette il valore statico presente nel file **HTML** che gestisce gli attributi booleani semplicemente tramite presenza / assenza dell'attributo. La presenza significa **`true`**. L'assenza significa **`false`**.
Qualunque valore venga assegnato al defaultState, questo valore che viene sempre convertito in **`true`** indipendentemente dal valore medesimo.

Scrivendo ad esempio `<button disabled="false">` è come se si settasse a **`true`** l'attributo disabled ed il pulsante risulterà disabilitato. L'unico modo per abilitare il pulsante è quello di "rimuovere" l'attributo disabled.
Se si desidera assegnare un valore ad un attributo booleano il consiglio di HTML5 è quello di ripetere il nome dell'attributo medesimo: **`<button disabled="disabled">`**

La corrispondente sintassi javascript corretta è la seguente:

```
btn.setAttribute("disabled", "disabled")
btn.removeAttribute("disabled")
```

Viceversa il **currentState** è un booleano vero e proprio, per cui lato javascript si può scrivere:

```
btn.disabled=true
btn.disabled=false
```

Creazione di nuovi attributi

In qualunque momento è possibile creare nuovi attributi HTML mediante una delle seguenti sintassi:

```
ref.nuovoAttributo = "valore";
ref.setAttribute ("nuovoAttributo", "valore");
```

Accesso e gestione delle proprietà CSS

Riguardo alle proprietà di stile, anche in questo caso c'è una distinzione fra le **proprietà di stile impostati staticamente** tramite HTML/CSS e le **proprietà di stile impostate dinamicamente** tramite javaScript.
Quando tramite javaScript si assegna un nuovo valore ad una proprietà di stile, questo valore non sovrascrive completamente il valore statico impostato tramite HTML/CSS, il quale, pur non essendo più visibile, rimane memorizzato come valore statico della proprietà.

- Nel momento in cui viene assegnato un nuovo valore tramite javascript, questo valore "maschera" il valore statico, e qualunque funzione di lettura (compresa getComputedStyle) restituisce sempre il valore dinamico
- Nel momento in cui il valore assegnato tramite javascript viene rimosso (tramite assegnazione di **stringa vuota** oppure **none**), automaticamente viene riassegnato all'elemento il valore statico memorizzato all'interno del file HTML/CSS che non può in alcun modo essere rimosso / modificato

Sintassi di accesso:

1. In modo diretto tramite la proprietà **.style**: `ref.style.backgroundColor = "blue";`
   Questa sintassi rappresenta il modo migliore per modificare singolarmente i vari attributi di stile. L'eventuale trattino nel nome dell'attributo (es `background-color`) deve essere sostituito dalla notazione camelCasing.
   style è un object che può accedere ai vari campi sia tramite **puntino** sia tramite **parentesi quadre**
   In lettura restituisce ***soltanto*** le proprietà impostate dinamicamente tramite java script
   E' chiaramente possibile utilizzare gli attributi composti impostando più valori in una sola riga:
   `ref.style.border = "2px solid black";`

2. Tramite i metodi `.getAttribute("style") / .setAttribute("style")`
   Simili ai precedenti, però leggono / scrivono l'intero attributo **style** in un sol colpo, per cui hanno senso SOLTANTO quando effettivamente interessa leggere / scrivere TUTTE le proprietà di stile insieme
   `.getAttribute`("style") restituisce tutte le proprietà di stile dell'oggetto corrente, sotto forma di ***stringa serializzata***, però **SOLTANTO** quelle impostate dinamicamente tramite javaScript e **NON** quelle impostate staticamente in HTML/CSS, e nemmeno quelle impostate tramite setAttribute ("class", "className")
   `.setAttribute("style", "font-size:100px; font-style:italic; color:#ff0000");`
   consente di definire contemporaneamente più CSS property (con un'unica stringa CSS), però **sovrascrive** tutte le proprietà di stile eventualmente già impostate dinamicamente tramite javascript.
   `.removeAttribute`("style") rimuove tutte le proprietà di stile impostate tramite java script

3. `ref.style.cssText += "font-size:100px; font-style:italic; color:#ff0000";`
   Utilizzabile sia in lettura che in scrittura, va bene nel caso in cui si desideri impostare più proprietà insieme.
   In lettura esegue in pratica una serializzazione della proprietà style esattamente come getAttribute("style")
   In scrittura **consente il concatenamento** per cui diventa possibile aggiungere nuove proprietà CSS all'elemento corrente senza rimuovere quelle precedentemente impostate.
   Anche questa proprietà 'vede' soltanto le proprietà di stile impostate dinamicamente in javaScript.

Accesso in lettura agli attributi di stile impostati tramite CSS statico

La funzione **getComputedStyle(refPointer)** consente di accedere in **lettura** a TUTTE le proprietà CSS dell'elemento corrente, sia quelle create dinamicamente in JavaScript, sia quelle create staticamente in HTML.

```
if(getComputedStyle(ref).visibility=="hidden")
            ref.style.visibility="visible";
```

- Se java script modifica dinamicamente una certa proprietà, getComputedStyle restituisce il nuovo valore.
- Se javascript non sovrascrive il valore dei CSS (oppure rimuove eventuali nuovi valori aggiunti), getComputedStyle restituisce i valori statici memorizzati all'interno del file CSS.

Note:

1) I **colori** impostati in javascript tramite nel formato #ESA, in lettura vengono convertiti in formato RGB (mentre quelli impostati tramite nome rimangono come sono). Il metodo getComputedStyle restituisce i colori SEMPRE SOLO in formato RGB separando i valori tramite VIRGOLA SPAZIO

```
if(ref.style.backgroundColor != "rgb(234, 234, 234)")
```

2) Il test sul **backgroundImage** è piuttosto problematico in quanto, a differenza di **img.src**, **backgroundImage** restituisce una url completa. es **url("img/img1.gif")** Per ottenere il nome dell'immagine si può utilizzare **.substring(9, len-6)** oppure, se possibile, fare il test su stringa vuota.

```
var len = this.style.backgroundImage.length;
var img = this.style.backgroundImage.substring(9, len-6);
```

Aggiunta / Rimozione di una classe

```
_ref.classList.add("className");
_ref.classList.remove("className");
_ref.classList.toggle("className", true);
_ref.classList.toggle("className", false);
```

Per vedere se un certo elemento contiene una classe oppure no si usa la seguente sintassi:

```
if (_ref.classList.contains("className ")) ……...
```

Accesso diretto agli elementi del DOM attraverso i selettori CSS

I metodi **querySelector()** e **querySelectorAll()** accettano come parametro qualunque selettore / pseudoselettore CSS scritto utilizzando la sintassi dei CSS.

- Il metodo **querySelector()** ritorna il primo Elemento **discendente** (cioè compresi figli e nipoti) che corrisponde al selettore specificato
- Il metodo **querySelectorAll()** ritorna un vettore enumerativo di tutti i nodi corrispondenti al selettore indicato

Esempio querySelector()

Se si specifica un **ID** diventa sostanzialmente equivalente a `document.getElementById`.
Però attenzione che, a livello sintattico, a differenza di `document.getElementById`, nel caso di `querySelector` occorre anteporre un #, in quanto quertselector si aspetta un selettore CSS scritto così come lo si scriverebbe all'interno di un file CSS

```
let _wrapper = document.querySelector("#wrapper");
```

Esempio querySelector()

L'esempio restituisce tutti i tag input di tipo **text** (non fattibile con getElementsByTagsName)

```
let vet=document.querySelectorAll("input[type=text]")
for (let item of vet)
   item.style.backgroundColor="red"
```

Invece di partire da document si può partire da un **parentNode**:

```
let _wrapper = document.querySelector("#wrapper");
let vet=_wrapper.querySelectorAll("input[type=text]")
```

Accesso diretto ai controlli di una form

Per accedere direttamente ai controlli di una form si può utilizzare il **name** della form seguito dal **name** del controllo

```
var nomeUtente = document.form1.txtUser.value
```

Il collegamento degli Eventi tramite codice

Per l'associazione di una procedura di evento in javascript si può usare :

- l'assegnazione diretta di una stringa all'evento desiderato preceduto dal prefisso **on**

```
ref.onclick = "esegui()";
ref.onclick = null;
```

- **setAttribute**() identico al precedente sempre con utilizzo del prefisso **on** e l'assegnazione di una stringa Entrambe queste sintassi accettano soltanto variabili primitive (ed eventualmente this), ma **non oggetti.**

```
ref.setAttribute("onclick", "esegui(a, b)");
ref.removeAttribute("onclick");
```

- **addEventListener** ha come parametro l'evento vero e proprio (scritto **senza prefisso on**) e come secondo parametro un puntatore a funzione PRIVO DI PARAMETRI (o funzione anonima scritta in loco)

```
ref.addEventListener("click", esegui);  // senza parametri
ref.addEventListener("click",  function(){ esegui(n1, n2) });
```

  Accetta come parametri anche Object. Ha un 3° parametro booleano in cui il true indica priorità più alta. Se si associano più procedure ad un medesimo evento, queste vengono eseguite nell'ordine in cui sono state associate. Assegnando ad una di esse il valore true sul 3° par, questa varrà eseguita prima delle altre.

- **removeEventListener**

```
ref.removeEventListener("click", esegui);
```

  - NON consente di rimuovere listener creati nella pagina html, ma solo listener creati con addEventListener
  - Il secondo parametro (puntatore alla funzione da rimuovere) è obbligatorio e **NON può essere omesso**
  - NON è possibile rimuovere dei listener definiti tramite **funzione anonima**, ma a tale scopo occorre definire sempre una named function del tipo: `var myFunction = function() {  }`
  - Per disabilitare l'event Handler è comunque sufficiente impostare **disabled=true**

Il puntatore this

Se una procedura di evento viene associata tramite il metodo **addEventListener**, la procedura medesima diventa un metodo di evento dell'oggetto che ha eseguito l'associazione, per cui al suo interno **è possibile utilizzare l'oggetto this** che rappresenta un puntatore all'oggetto del DOM che ha scatenato l'evento (sender).

Nel caso invece dell'utilizzo di onclick o setAttribute(), se la funzione di evento ha necessità di accedere all'elemento corrente, occorre passare manualmente il **this** come parametro alla funzione di evento:

- Nelle funzioni richiamate tramite **html**

```
<input type="button"  onClick="visualizza(this)">
```

- In caso di **setAttribute()**

```
ref.setAttribute("onclick", "visualizza(this)");
```

Mancata associazione del this

Prestare MOLTA attenzione al fatto che, se la funzione di evento richiama un'altra sottofunzione, all'interno della sottofunzione l'associazione del **`this` NON è più valida.** Infatti la sottofunzione non può sapere chi è che l'ha richiamata e quindi a che cosa si riferisce il **`this`**, All'interno delle sottofunzioni, **`this`** NON rappresenta l'oggetto che ha richiamato la funzione, ma un generico "spazio" delle funzioni.

In caso di necessità occorre eventualmente passare il this in modo manuale ed esplicito:

```
ref.addEventListener("click",  function(){ visualizza(this) })

function visualizza (_this)  {
     alert (_this.value);
}
```

Nota 1: Il passaggio dei parametri in java script

I numeri e i valori booleani vengono **copiati**, **passati** e **confrontati** per valore e, come in java, NON è possibile passarli per riferimento.
Vettori, Matrici e Oggetti in generale sono **copiati**, **passati** e **confrontati** per riferimento e, come in java, NON è possibile passarli per valore. Il confronto fra due oggetti identici restituisce false a meno che i puntatori non stiano puntando allo stesso oggetto.
Le stringhe vengono **copiate** e **passate** per riferimento, ma vengono **confrontate** per valore.
Due oggetti **String** creati con **new** String("something") vengono confrontati per riferimento.
Se uno o entrambi i valori è un valore stringa (senza il new), allora il confronto viene eseguito per valore.

Nota 2: Passaggio di un parametro ad addEventListener : var e let

Si supponga di avere un elenco di `<button>` all'interno della pagina html e si consideri il seguente codice:

```
var btns = document.getElementsByTagName("button");
for (var i=0; i<btns.length; i++) {
  (1)   btns[i].setAttribute("onclick", "esegui(" + i + ")");
  (2)   btns[i].addEventListener("click", function() { esegui(i) });
}
```

Nel primo caso, ad ogni iterazione del ciclo, viene creata una stringa statica che viene assegnata alla proprietà di evento "**onclick**". All'interno di questa stringa viene inserito tramite concatenamento il valore corrente di i. Per cui al momento del click sul primo pulsante verrà richiamata la procedura esegui(0) e così via per gli altri pulsanti.

Nel secondo caso invece viene creata una associazione tra un evento ("**click**") ed una funzione (in questo caso anonima, ma se anche la funzione fosse scritta esternamente con un nome sarebbe esattamente la stessa cosa). Come parametro viene passato un riferimento alla variabile i. Nel senso che, al momento del click su un pulsante, verrà richiamata la funzione esegui(i) alla quale verrà passato per valore il valore **corrente** della variabile i.

Per cui all'interno di esegui() la variabile i assumerà il valore attuale al momento del click, cioè il valore raggiunto al termine del ciclo, cioè `btns.length`, indipendentemente da quale pulsante sia stato premuto.
Dunque la seconda soluzione così com'è **non va bene.** Si può comunque ovviare utilizzando this.

L' istruzione let

L'istruzione **var** dichiara una variabile **allocata globalmente** ma visibile SOLTANTO all'interno della procedura in cui viene dichiarata. Questo è il motivo per cui, nell'esempio (2) precedente la variabile i continua a vivere ed essere utilizzabile anche dopo che la procedura è terminata. Nel momento in cui si scrive

```
btns[i].addEventListener("click", function() { esegui(i) });
```

ad `esegui()` viene passato un riferimento alla variabile globale i per cui quando l'utente farà click sul pulsante, la procedura visualizzare il valore corrente di i (cioè `btns.length`)

Inoltre se l'istruzione **var** viene utilizzata all'interno di un ciclo, la variabile sarà visibile da quel momento in poi per l'intera procedura, anche fuori dal ciclo in cui è stata dichiarata. Se al termine del ciclo precedente si eseguisse un **`alert(i);`** questa produrrebbe come risultato il valore corrente della i al termine del ciclo for, cioè `btns.length` senza causare errore.

L'istruzione **let** dichiara invece una variabile **allocata localmente** nella sezione di codice in cui viene utilizzata. Se ad esempio si utilizza l'istruzione **let** all'interno di un ciclo for, la variabile dichiarata con let non sarà accessibile al di fuori del ciclo medesimo.

Se la variabile i del ciclo precedente venisse dichiarata tramite **let** nel modo seguente:

```
for (let i=0; i<btns.length; i++)
        btns[i].addEventListener("click", function() { esegui(i) });
```

alla funzione esegui(i) verrebbe passata una COPIA del valore corrente della variabile i, cioè al primo btn verrebbe passato 0, al secondo btn verrebbe passato 1 e così via, **per cui il parametro i verrebbe passato correttamente**.

Il pre-Caricamento delle immagini in memoria

Per scaricare una immagine da un server web occorrono mediamente alcuni secondi. Improponibile se questa immagine deve essere utilizzata per eseguire un rollover. A tal fine è possibile, durante il caricamento della pagina, scaricare le immagini necessarie salvandole in memoria. Queste immagini verranno eventualmente visualizzate in corrispondenza di un dato evento successivo (un click o un mouseOver).

Per creare una singola immagine in memoria occorre utilizzare il costruttore dell'oggetto Image:

```
var myImg = new Image( );       //come parametri opzionali si possono passare Width e Height
myImg.src = "img/immagine1.jpg"
```

Dentro ***.src*** viene salvato il percorso dell'immagine, che potrà poi essere utilizzato per copiare l'immagine all'interno di un generico **_imgBox** di tipo img presente all'interno della pagina HTML:

```
_imgBox.src = myImg.src
```

Effetto RollOver

Il metodo più comune per realizzare un pulsante grafico è quello di includere un tag IMG all'interno di un tag <a>, avente HREF che punta all'indirizzo desiderato. Sul pulsante grafico si può poi applicare un effetto di RollOver al passaggio del mouse.

Per ottenere questo effetto occorre, al momento dell'onLoad, caricare le immagini in due variabili globali:

```
imgOn.src  = "img/immagine1.jpg";
imgOff.src = "img/immagine2,jpg";
```

caricando staticamente l'immagineOff anche dentro il pulsante grafico imgBox.
Dopo di che :

```
onMouseOver="this.src = imgOn.src"
onMouseOut= "this.src = imgOff.src"
```

La creazione dinamica degli oggetti

In Java Script è possibile creare tag dinamicamente ed aggiungerli all'interno di altri tag utilizzando un tipico modello ad albero. I metodi da utilizzare sono:

- `document.createElement("tagName")` `// per creare un nuovo tag`
- `parent.appendChild("tagName")` `// per appenderlo ad un tag esistente`

Esempio di creazione di una nuove righe e celle all'interno di una tabella :

```
var tabella = document.getElementById("mainTable");
for (var i=0; i<DIM; i++){
    var tr = document.createElement("tr")
    tabella.appendChild(tr)
    for (var j=0; j<DIM; j++){
        td = document.createElement("td")
        tr.appendChild(td)
        var img = document.createElement("img")
        td.appendChild(img);
        img.id=`img-${i}-${j}`
        img.addEventListener("click", cambiaImmagine)
        var span = document.createElement("span")
        span.innerHTML+="immagine"+j;
        td.appendChild(span);
    }
}
var btn = document.createElement("input");
btn.type = "button"
btn.value = "Elabora"
btn.style.width="100px";
btn.style.height="40px";
btn.addEventListener("click", elabora);
```

Note

1) Nel caso di `createElement("input")` si può specificare un secondo parametro che indica il tipo di input che si intende creare : `createElement("input", "text")`
2) Per alcuni elementi è disponibile l'operatore `new`
   `var opt = new Option(text, value) // Opzione da aggiungere ad un select`
3) Nel metodo `parent.appendChild(elem)`, se l'elemento ricevuto come parametro è già appeso al DOM, viene automaticamente 'tagliato' ed appeso nella nuova posizione.

Per accedere ad un nodo figlio del nodo corrente si usa .`childNodes[i]` a base 0

```
let div = wrapper.childNodes[2]  // terzo figlio
```

Per vedere se il nodo ha figli si usa .`hasChildNodes[i]`

```
if(wrapper.hasChildNodes) .....
```

Tecniche per la creazione di una matrice di oggetti interni ad un contenitore

Supponiamo di avere un tag statico `<div id="wrapper">` e di volerlo riempire con 10x10 = 100 tag DIV da creare dinamicamente ed appendere a wrapper.

A tale scopo si possono implementare diverse soluzioni:

1) Utilizzare una tabella come nel caso precedente. A questo punto il wrapper potrebbe essere di tipo <table> al quale si aggiungono poi i <TR> e infine i <TD>. Soluzione perfetta per visualizzare tabelle di dati ma poco adatta ai giochi (spaziature indesiderate fra celle e anche fra righe)

2) Utilizzare un contenitore di tipo <DIV> e impostare sugli elementi interni **`float:left`** oppure ancora meglio **`display:inline-block`**. Gli elementi interni vengono disposti uno a fianco dell'altro fino al raggiungimento del margine destro del contenitore, in corrispondenza del quale vanno automaticamente a capo. A tale scopo diventa fondamentale la larghezza del contenitore che deve essere esattamente 10 volte la larghezza degli elementi interni. A volte però (gioco della roulette) il contenitore necessita di uno spazio vuoto a destra. In tal caso si potrebbe agire sul padding-right del contenitore

3) Utilizzare un contenitore di tipo <DIV> e con **`position:relative`** ed assegnare **`position:absolute`** agli elementi interni. In questo caso ci svincoliamo dalla larghezza del contenitore

4) Anziché impostare **`float:left`** o **`display:inline-block`** sugli elementi interni, si possono lasciare gli elementi interni così come sono (display:block) e costruire il corpo per righe creando prima un tag <DIV> con **`display:flex`**, e poi aggiungendo al suo interno i 10 tag <DIV> che andranno a costituire la riga, simulando in pratica una tabella. Anche in questo caso ci svincoliamo completamente dalle dimensioni del contenitore.

Nota Importante sul concatenamento di stringhe all'interno di `innerHTML`

Supponiamo di avere all'interno della pagina html un tag DIV con **`id=wrapper`** al quale andiamo ad appendere tramite java script altri tag creati dinamicamente. Se ad un certo punto si utilizza una istruzione dei tipo

```
_wrapper.innerHTML += "Messaggio da concatenare all'interno del wrapper"
```

prestare attenzione al fatto che il **concatenamento** all'interno della proprietà **`innerHTML`** forza nel browser una **rigenerazione dell'intero contenuto del tag wrapper**, per cui eventuali puntatori javascript agli oggetti creati dinamicamente all'interno di wrapper andrebbero tutti persi. Ad esempio tutti i listener di evento, ma anche la possibilità di intercettare il **checked** su un checkbox, perché il puntatore usato in fase di creazione NON punterebbe più al checkbox ridisegnato all'interno di wrapper.

Per cui, se si lavora con gli oggetti, occorre abbandonare definitivamente il concatenamento di stringhe.

Accesso alle righe di una Tabella

Il puntatore a tabella presenta una interessante proprietà **rows** che rappresenta un vettore enumerativo contenente i puntatori alle varie righe che costituiscono la tabella. A sua volta la riga contiene una collezione di **cells**

```
var _table = document.getElementById("table")
if(_table.rows.length > 0) {
   var tr = _table.rows[i];
   if(tr.cells.length > 0)
        var cella = tr.cells[j]
```

Cancellazione dei dati di una tabella

```
tabella.innerHTML="";  // oppure
while (tabella.childNodes.length > 2)
       tabella.removeChild(tabella.childNodes[2]);
```

Il primo **childNode** rappresenta un sottocontenitore della tabella
Il secondo **childNode** è rappresentato dall'eventuale cella TH

L'oggetto event

All'interno di una qualunque procedura di evento è possibile utilizzare un oggetto **event** che contiene diverse informazioni sull'evento e sull'oggetto che ha scatenato l'evento**. Lo standard prevede che questo oggetto venga automaticamente passato come ultimo parametro alla procedura di evento**. Il passaggio del parametro **event** da parte de chiamante non è obbligatorio. Il chiamato, se lo vuole leggere, lo deve dichiarare esplicitamente:

```
function esegui(event){   }
```

Principali Proprietà

event.target puntatore all'elemento che ha scatenato l'evento (**mozilla**)
event.srcElement puntatore all'elemento che ha scatenato l'evento (**chrome**)
event.type contiene il nome dell'evento (es "click")
event.keyCode contiene il keyCode (codice fisico) del tasto premuto. Non distingue ad esempio tra maiuscole e minuscole, però intercetta tasti come le freccine o lo shift. Per le lettere il keyCode coincide con il codice ASCII della lettera maiuscola (65 – 90), mentre per i tasti numerici coincide con il codice ASCII del carattere numerico premuto
event.clientX coordinate X del mouse rispetto alla window corrente
event.clientY coordinate Y del mouse rispetto alla window corrente

L'oggetto window: proprietà, metodi ed eventi

name E' il nome assegnato ad una finestra aperta da codice.

closed Se true significa che la finestra è stata chiusa. Dalla finestra attuale è possibile creare una nuova finestra mediante il metodo open ricevendo un puntatore alla finestra. Con il puntatore è possibile analizzare il closed della nuova finestra per vedere se è stata chiusa (o se non è ancora stata creata)

status Contenuto della barra di stato inferiore. Passando il mouse su un collegamento ipertestuale, il browser visualizza automaticamente nella barra di stato inferiore l'URL completo del link.
Java Script può modificare questo msg mediante l'evento **onMouseOver.** Però se si vuole sostituire l'azione di default con una azione utente, occorre restituire al gestore onMouseOver il valore **true**. Altrimenti l'azione di default maschera l'azione utente.
<a href="pagina3.htm" onMouseOver="window.status='Caratteristiche Tecniche'; **return true**">

defaultStatus Messaggio iniziale visualizzato nella barra di stato dopo il caricamento di una nuova pagina

setInterval

Sia `setTimeout` che `setInterval` sono asincrone, cioè avviano la procedura all'interno di un thread separato, per cui eventuali istruzioni successive a setInterval / setTimeout vengono eseguite subito dopo.

setInterval( ) richiede due parametri:

- Un puntatore a funzione
- Un tempo espresso in millisecondi

Esempio:

```
var timerID=setInterval(visualizza, 1000);
```

La procedura **`visualizza`** verrà richiamata ciclicamente a intervalli regolari di 1000 msec, cioè 1 sec in modo analogo all'oggetto timer di C#.

setInterval() restituisce un ID che può essere utilizzato per arrestare il temporizzatore :

```
if(timerID) clearInterval(timerID)
```

Per riavviarlo occorre riscrivere l'intera istruzione setInterval()

Esempio di visualizzazione dell'ora corrente

```
function visualizzaOraCorrente() {
     var d=new Date();
     _div.innerHTML = d.toLocaleTimeString();
}
```

In alternativa si può incrementare una variabile globale `seconds` ogni 1000 msec.
Quando (`seconds%60==0`) `minutes++`;

setTimeout

setTimeout( ) è analogo a setInterval() ma la funzione indicata **viene eseguita una sola volta**

var timer =**`setTimeout(visualizza, 1000)`**

`visualizza` viene avviata dopo 1 sec dal richiamo di setTimeout() e **viene eseguita una sola volta** a meno che, al termine della procedura medesima, venga di nuovo richiamato setTimeout() che la fa ripartire un'altra volta.

Esattamente come setInterval() restituisce un **ID diverso da 0** che consente di disabilitare il timer prima dello scadere del tempo indicato

```
if(timerID) clearTimeout(timerID
```

window.open

open("file.htm", ["target"], ["Opzioni separate da virgola"])

Il **primo parametro** indica il file da caricare. Se si specifica "", verrà aperta una nuova scheda vuota.

Il **secondo parametro target** rappresenta la scheda di apertura della pagina e può assumere i valori "_blank", "_self", etc. oppure un nome alfanumerico (**TARGET**) nel qual caso il file verrà aperto in una nuova scheda a cui verrà assegnato il target indicato.

```
<a href="#" onClick='window.open("pag2.htm", "Finestra2");'> apri </a>
<a href='TerzaPagina.html'  target="Finestra2"> vai</a>
```

TerzaPagina.html verrà aperta all'interno della scheda Finestra2 creata da window.open()

Il terzo parametro consente di aprire la pagina **in una nuova finestra** e consente di esprimere le caratteristiche della nuova finestra. In tal caso come secondo parametro si può impostare stringa vuota oppure un target identificativo. I vari parametri devono essere scritti **senza spaziature.**
window.open('pagina2.htm', '', 'resizable=no, width=300, height=300, left=320, top=230, fullsceen=no, menubar, toolbar=no, scrollbars=yes, status=no');
Il valore yes può anche essere omesso scrivendo soltanto il nome dell'opzione.

Opzioni terzo parametro:

Nome	Valore	Spiegazione
width height	Numerico pixel	Larghezza – Altezza della finestra
left	Numerico pixel	Distanza dalla sinistra del monitor
top	Numerico pixel	Distanza dal lato superiore del monitor
fullscreen	yes / no	Apertura a tutto schermo
menubar	yes / no	Presenza del menù
toolbar	yes / no	Presenza della toolbar
scrollbars	yes / no	Presenza delle scroll bar
status	yes / no	Presenza della status bar in basso
location	***yes / no***	***Presenza della barra degli indirizzi***
resizable	***yes / no***	***Ridimensionabile***

location e resizable sembrano deprecati. Al loro posto si può usare il widget dialog di jQueryUi che è basato non su windows.open ma sulle inline dialogs, cioè la visualizzazione di un tag DIV in primo piano con oscuramento della parte sottostante.

Il metodo open viene spesso sfruttato per aprire banner pubblicitari

```
<a href="pagina2.htm" onClick='window.open("banner.htm", " NuovaFinestra ");'> Vai a pagina2 </a>
<body onLoad='window.open("banner.htm", " NuovaFinestra ");'>      oppure body onUnload
```

Gestione del riferimento alla finestra aperta da open

Il metodo open restituisce un puntatore alla nuova finestra appena aperta. Esempio:

```
var ref = window.open("pagina2.htm", "NuovaFinestra");
ref .close()      // Chiude la nuova finestra
ref  = null       // Dopo la chiusura di una finestra è bene rilasciare il puntatore
```

La proprietà opener

Ogni finestra (window) ha una interessante proprietà **opener** che è un puntatore alla finestra o frame che ha generato la sottofinestra mediante window.open(). Per la finestra principale opener = null. Esempio:

```
<input type="text" onChange = "opener.document.getElementById().value="x">
```

Altre Proprietà e metodi dell'oggetto document

Proprietà

title E' il titolo della pagina impostato nella head dal tag title
lastModified Data e ora dell'ultima modifica della pagina

Il metodo document.write (s) Consente di scrivere dinamicamente il contenuto di una **nuova** pagina.
All'interno della stringa s può essere inserito qualunque tag html.
Se il metodo viene eseguito verso una pagina già caricata, write troverà il documento chiuso e provvederà a rimuoverlo sostituendolo con un documento vuoto in cui andrà a scrivere il contenuto di s.

Creazione dinamica di un documento tramite document.write()

Dopo aver creato una nuovo finestra vuota tramite window.open()

```
var w=window.open("","_blank");
```

è possibile andare a scrivere dentro utilizzando il metodo window.document.write():

```
w.document.write("<h1 align='center'>Titolo della nuova pagina</h1>");
```

Metodi dell'oggetto document per la scrittura dinamica:

open() Apre un documento in scrittura. Opzionale. Se il documento è chiuso write lo apre automaticamente
write (s) Se usato all'interno di una pagina vuota consente di creare dinamicamente il contenuto della pagina. Il contenuto di s viene scritto alla posizione attuale del cursore. **All'interno della stringa s può essere inserito qualunque tag html** compreso \n Il flusso di output viene però automaticamente chiuso al termine del caricamento della pagina. Dunque se il metodo viene eseguito verso una pagina già caricata, write troverà il documento chiuso e provvederà a rimuoverlo sostituendolo con un documento vuoto in cui andrà a scrivere il contenuto di s.
writeln (s) Come write() con in più il ritorno a capo aggiunto automaticamente al fondo di s
close () Serve per chiudere il flusso al termine delle write. Sebbene il flusso venga chiuso automaticamente al termine del caricamento della pagina, i manuali consigliano di eseguire sempre il close() subito dopo l'ultimo write. Altrimenti potrebbero esserci problemi nel caricamento di immagini e moduli

OGGETTO window.location

Contiene tutte le informazioni sulla URL corrente

Proprietà e Metodi

href URL attuale completa http://indirizzo. **Proprietà predefinita di location, per cui può anche essere omessa**. Modificare la proprietà href dell'oggetto location è il modo più semplice per caricare una nuova pagina mediante uno script: `location.href="pagina3.htm"` oppure anche da HTML:

```
onclick="window.location.href='home.html'"
```

Accetta come parametro anche un'ancora interna alla pagina corrente
`window.location.href='#ancora'`

Notare che l'impostazione della proprietà **`href`** **NON termina l'elaborazione dello script** che prosegue eseguendo eventuali istruzioni successive. Per terminare lo script si può utilizzare:

- `return false`; termina la funzione in corso
- **`window.stop()`**; termina l'intero script

Nota: All'interno di href, come in html, si può specificare un indirizzo di posta elettronica, preceduto da mailto: in tal caso verrà aperto il client di posta predefinito. All'indirizzo di posta possono essere concatenati anche dei parametri riguardanti ad esempio il body da preimpostare.

reload( ) Ricarica l'intero documento (come il tasto Reload del browser)
Ha un parametro facoltativo che ha un valore di default pari a **false** nel qual caso il reload viene fatto dalla cache se possibile). Impostando **true** viene forzato il reload dal server.

replace("URL") Carica una nuova pagina nella finestra corrente. Rispetto alla precedente elimina la pagina attuale dalla cronologia. Facendo INDIETRO l'utente non vedrà più la pagina corrente ma ritornerà alla pagina antecedente. Utile per eliminare dalla cronologia pagine intermedie utilizzate in una certa fase.

protocol protocollo di accesso alla risorsa. Es **http**, **file**, **ftp**.
hostname nome del dominio richiesto
port porta di comunicazione. 80 nel caso di http, stringa vuota nel caso del procollo *file*
host hostname : port

pathname risorsa richiesta
search restituisce la ***queryString*** della url comprensiva del ?
hash = "Capitolo2" Consente di navigare verso un nuovo ancoraggio presente nella pagina

OGGETTO window.history

Contiene tutte le informazioni relative alle URL visitate prima e dopo rispetto alla URL attuale.
Consente la navigazione avanti e indietro attraverso la storia della finestra corrente.

Proprietà

length Numero di pagine visitate precedentemente rispetto alla pagina attale
current URL della pagina attualmente caricata
previous URL della pagina precedente nella cronologia
next URL della pagina successiva nella cronologia (ha senso solo se si è usato il pulsante back)

back() Ritorna alla pagina precedente. La pagina **viene ricaricata**, però vengono automaticamente ripassati al server eventuali parametri get e post (esattamente come avviene con il pulsante BACK del browser).
forward() Va avanti di una pagina (se esiste)
go(-1) Va avanti / indietro ad una posizione ben definita. go(-2) torna indietro di 2 pagine.
La pagina **viene ricaricata** con il passaggio automatico dei parametri get e post, esattamente come avviene per il metodo back() e per il pulsante BACK del browser.

OGGETTO navigator

Al momento dell'apertura del browser, viene allocato un oggetto NAVIGATOR, fratello dell'oggetto WINDOW, contenente tutte le informazioni sul browser che si sta utilizzando. Questo oggetto rimane allocato in *unica* istanza fino alla chiusura del browser.

appName Nome del browser. Es Microsoft Internet Explorer
appVersion Versione del browser Es versione 4.0 (compatible; MSIE 5.5; Windows 98)
appCodeName Nome in codice del browser. Es "Mozilla"
userAgent E' la stringa di intestazione inviata all'host quando gli si richiede una pagina web. Contiene informazioni sul browser, sul sistema operativo e sulle rispettive versioni

Approfondimenti

```
"use strict";
```

Inserito sulla prima riga di un file js obbliga l'utente a dichiarare le variabili.
Comodo per evitare di utilizzare variabili inesistenti in seguito ad un errore di battitura.

L'operatore ===

Confronta non solo il valore ma anche il tipo

```
var a = 1;
var b = "1";
if (a==b)   // true
if (a===b)  // false
```

try and catch

```
try {
      alert("Welcome guest!");
}
catch(err) {
      document.getElementById("demo").innerHTML = err.message;
}
```

Accesso alle variabili attraverso il DOM

```
var a = 15;
window["a"]++;
alert(a);       // 16
```

Consente anche di accedere direttamente agli elementi del DOM attraverso il loro ID

```
   window["btnIndietro"].disabled=true;
```

Inserimento di una variabile all'interno di una stringa

Il **backtick** (apice singolo rovesciato = **ALT 96)** è uno speciale delimitatore di stringa che:

- consente di inserire delle variabili direttamente all'interno della stringa tramite **${varName}**
- consente di **andare a capo** all'interno della stringa, che può essere scritta su più righe di testo

```
`Sono l'elemento ${i}
 Il mio valore è ${vet[i]}`
```

Assegnazione di una booelana tramite condizione diretta

```
var ok = (a > 0)
```

Se `(a > 0)` allora ad ok viene assegnato il valore `true`, altrimenti viene assegnato il valore `false`

The ternary conditional operator

E' una tecnica disponibile in tutti i linguaggi per compattare al massimo il costrutto if.

Si supponga di dover eseguire le seguenti assegnazioni:

```
if(ok)  msg = 'yes';
else    msg = 'no';
```

Questo costrutto può essere riscritto in modo molto più conciso nel modo seguente:

```
msg = ok ? 'yes' : 'no';
```

Cioè se la variabile **ok** è vera, alla variabile **msg** viene assegnato yes, altrimenti viene assegnato no.

Note:

1) Attenzione al fatto che, dopo il ?, occorre necessariamente utilizzare o dei valori diretti (come nell'esempio) oppure delle funzioni che restituiscono un valore. Non è consentito utilizzare delle procedure perché non potrebbero assegnane nessun valore a msg
2) **msg potrebbe anche essere omesso**, nel qual caso il costrutto si limita ad eseguire una delle due funzioni di destra a seconda del valore di ok. Anche in questo caso però a destra non sono ammesse procedure ma sempre soltanto funzioni.

Utilizzo dell'operatore || sulle stringhe

In java script è possibile eseguire una OR fra due o più stringhe.
Il risultato è pari al contenuto della prima stringa che presenta una valore diverso da undefined.

```
var ris  = stringa1 || stringa2 || stringa3
```

```
var a;
var b = "hi"
console.log(a)      // undefined
console.log(a||b)   // "hi"
```

Parametri opzionali

```
function ricerca(param1 = false) {
}
```

Se il chiamante non passa nessun parametro, **param1** viene automaticamente settato a false

Funzioni con numero arbitrario di parametri

E' anche possibile definire una funzione con **firma priva di parametri** e poi passare alla funzione un numero arbitrario di parametri. Esempio:

```
visualizza("pippo", "pluto", "minnie");
function visualizza(){
    var result = '';
    for (var i = 0; i < arguments.length; i++)
       result += arguments[i] + "\n";
    alert(result);
```

Lettura dei parametri GET

```
function leggiParametriGet(){
    var json = {};
    var parametri = [];
    var s = window.location.search;
    // estraggo dal punto interrogativo in avanti
    s = s.substr(s.indexOf("?") + 1);
    // sostiuisco %20 con " "
    var exp = new RegExp("%20", "g");
    s = s.replace(exp, " ");
    parametri = s.split("&");

    var parametro = [];
    for (var i = 0; i < parametri.length; i++)
    {
      parametro = parametri[i].split("=");
      var key = parametro[0];
      var value = parametro[1];
      // Se il nome del parametro termina con [], compatto i valori in una stringa
      if(key.substr(key.length-6, 6)=="%5B%5D"){
            key=key.substr(0,key.length-6);
            if (!(key in json))
                  json[key] = value;
            else
                  json[key]+=", " + value;
      }
      else
            json[key] = value;
    }
    return json;
```

Un sito di rapido test del codice

www.webtoolkitonline.com