Formazione Orientata al Lavoro

JAVASCRIPT

Introduzione al linguaggio Javascript

JavaScript è un linguaggio di scripting, ossia un linguaggio di programmazione interpretato e non compilato (come ad esempio Java). Javascript ha aggiunto alle pagine HTML la possibilità di essere modificate in modo dinamico, in base all'interazione dell'utente con il browser (lato client).

Come per qualsiasi linguaggio di programmazione, per iniziare a lavorare con JavaScript abbiamo bisogno di almeno tre strumenti:

- Un **editor**;
- Un **interprete** o **compilatore**;
- Un **debugger**;

Esistono 3 modi per inserire codice **js** (Javascript) in una pagina html:

- Inserendo codice Inline, direttamente nel codice all'interno di tag html;
- Inserendo blocchi di codice tramite il tag <script>, sia nel tag <head> sia nel <body>;
- Collegando file esterni con il seguente:

```
<script type="text/javascript" src="nomeFile.js"></script>
```

Caratteristiche del linguaggio

A differenza di altri linguaggi di programmazione, in **JavaScript** terminare un'istruzione con il punto e virgola non è obbligatorio, a meno che non si vogliano scrivere più istruzioni sulla stessa riga.

JavaScript è *case sensitive*, cioè fa distinzione tra maiuscole e minuscole nei nomi di istruzioni, variabili e costanti.

I commenti sono parti di codice che non vengono **interpretati/compilati**, utili al programmatore per inserire nel programma annotazioni. Il commento a singola riga inizia con i caratteri // , o usiamo /* … */ per commenti su più righe.

JavaScript prevede come tipi di dato primitivi: **numeri**, **stringhe** (‘’ ‘’), **boolean** (true/false), **null**, **undefined** (valore che non esiste).
Tutti gli altri elementi previsti dal linguaggio, come gli **array**, le **espressioni regolari**, le **funzioni**, sono in realtà oggetti.
Anche i tipi di dato primitivi hanno oggetti corrispondenti con relative proprietà e metodi. JavaScript **converte automaticamente** un tipo primitivo nel corrispondente oggetto quando utilizziamo un suo metodo o una sua proprietà.

JavaScript ha un **unico tipo di dato numerico**, non fa distinzione tra **intero** e **decimale**.

Funzioni javascript : funzioni predefinite

Con le funzioni, è possibile dare un nome ad un intero blocco di codice, che potrà poi essere utilizzato in qualsiasi parte del programma. JavaScript è dotato di diverse funzioni predefinite.
Ecco tre tra le funzioni più usate:

- alert("messaggio") stampa un messaggio a video.
- confirm("messaggio") apre una finestra di conferma.
- prompt("messaggio") apre una finestra di dialogo.

```
<html>
<head>
<script type="text/javascript">
alert("Ciao");
</script>
</head>
<body>
</body>
</html>
```

Le funzioni

Il codice Javascript prevede l'uso di funzioni *function* per eseguire un determinato algoritmo.

```
function nomeFunzione(argomenti){
istruzioni;
return var;  // opzionale , nel caso debba restituire un valore
}
```

La parola chiave *function* serve a ***definire*** una funzione che potrà poi essere ***invocata*** a seguito di un evento scatenato tramite i tag html.

Come tutti i linguaggi di programmazione, se una variabile è dichiarata e inizializzata in una funzione, sarà locale e visibile solo quella funzione, altrimenti, se dichiarata fuori da ogni funzione, è globale, e visibile a tutte le funzioni nel file in cui definita.

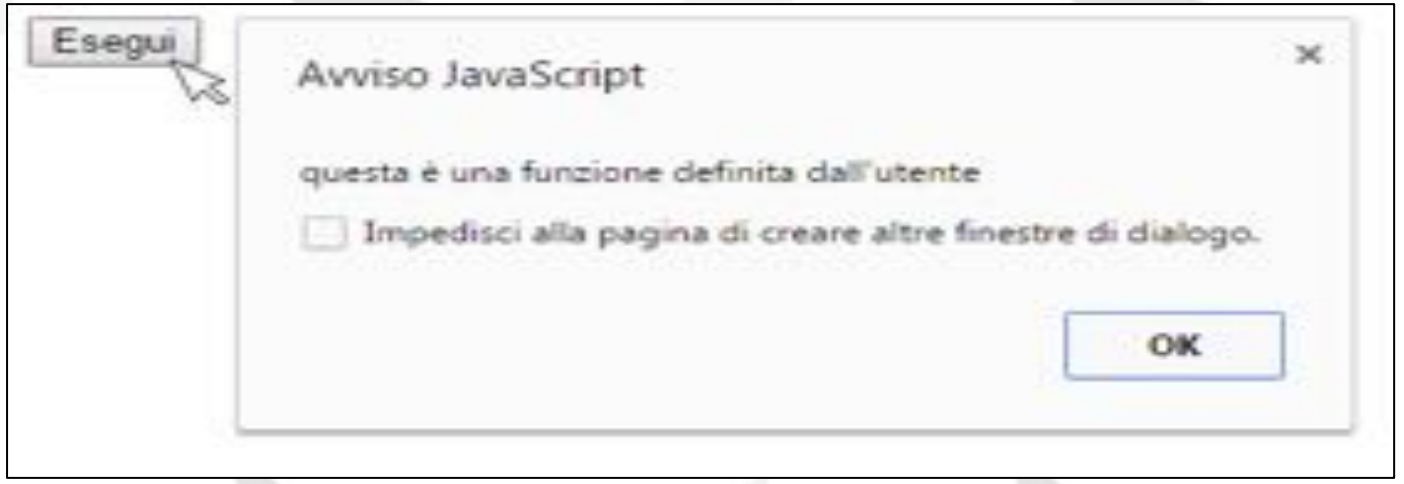
Mentre le parole chiavi sono riservate e non devono essere utilizzate per dare un nome ad una variabile, i nomi delle funzioni predefinite possono essere riutilizzate. Naturalmente l'implementazione della nostra function sovrascriverà quella predefinita.

Funzioni javascript : funzioni create

Con le funzioni definite dall'utente è possibile denominare un blocco di codice e chiamarlo quando se ne ha bisogno. Una funzione si definisce nella sezione Head di una pagina Html, ed è definita con la parola fuction, seguita dal nome della funzione e dall'argomento.

Esempio:

```
<head>
<script type="text/javascript">
function mostralert(){
alert("questa è una funzione
definita dall'utente");
}
</script>
</head>
<body>
<input type="button" value="Esegui"
onclick="mostralert();"/>
</body>
```



Le variabili

Le variabili sono contenitori in memoria in cui conservare dati, identificate da un nome.

JavaScript non prevede una dichiarazione obbligatoria delle variabili. L'utilizzo di un identificatore permette di creare implicitamente la variabile, se non esiste già. È comunque preferibile dichiararla, attraverso la parola chiave ***var.***
Il nome della variabile non deve coincidere con una parola chiave, non può iniziare con un numero e non può contenere caratteri speciali.

Una volta dichiarate le variabili possono contenere valori di ogni tipo e può cambiare il tipo di dati associati in qualsiasi momento, la conversione tra i diversi tipi di dati avviene in modo automatico.

```
var miaVariabile;

miaVariabile = 3;
miaVariabile = "testo"
```

Distinguiamo tra **variabili locali** e **globali**: le locali sono **vincolate all'interno di una funzione o di un metodo** nelle quali vengono definite e non sono accessibili dalle altre funzioni; le globali sono create fuori da metodi e funzioni e **disponibili in tutto il documento**.

Gli array

Se una variabile è un contenitore di un UNICO valore, il contenitore di un insieme di variabili (anche di diverso tipo) è detto ARRAY. Concepiamolo come un insieme di celle, ognuna delle quali contiene un valore, ed è identificata da un indice (la prima cella ha indice 0), tramite il quale possiamo recuperare il valore all'interno. È utile quando dobbiamo dichiarare più variabili a cui poi dovremmo accedere e su cui operare tramite cicli.

DICHIARAZIONE DI UN ARRAY IN JS:

```
var mesi = ["Gennaio", "Febbraio", "Marzo", ...]; //stesso tipo
var tipi = ["testo", null, 768];  //tipi diversi
```

ACCESSO ALLA PRIMA VARIABILE DELL'ARRAY:

```
var primoElemento = mesi[0];
```

SCORRERE UN ARRAY CON UN CICLO FOR:

```
for(var i=0;i<mesi.length;i++){
fai qualcosa con mesi[i];
}
```

Length è la proprietà dell'array che contiene la sua dimensione, cioè il numero di variabili che contiene. La sua capacità totale è dichiarata all'inizio e non è più modificabile.

Gli operatori (1)

Ci sono inoltre alcune operazioni matematiche, oltre le classiche (+,-,*,/,%) utili nei linguaggi di programmazione:

numero++ // con questa sintassi si incrementa in automatico la variabile 'numero' di 1. Sarebbe equivalente scrivere 'numero = numero +1'.

numero-- // così invece si decrementa in automatico una variabile di 1. Anche qui sarebbe equivalente scrivere 'numero = numero – 1'.

In Javascript è possibile unire due tipi di dato (uguali o diversi) attraverso un processo chiamato CONCATENAZIONE (tramite l'operatore '+').

Abbiamo due valori "Skill" e "Factory" concatenandoli e inserendoli nella variabile messaggio:

```
messaggio = "Skill"+"Factory";
```

Adesso la variabile messaggio conterrà il valore di stringa "SkillFactory".

Invece se volessimo concatenare una stringa ed un intero contenuto nella variabile i = 7:

```
messaggio = "Skill"+i;
```

messaggio conterrà " Skill7 ".

Gli operatori (2)

Come negli altri linguaggi di programmazione anche in javascript esistono gli operatori di confronto, e il risultato di queste operazioni restituiscono un valore booleano (true/false):

Var1 == **Var2** // Restituisce true se i valori delle due variabili sono uguali

Var1 === **Var2** // Restituisce true se i valori delle due variabili sono uguali e dello stesso tipo

Var1 != **Var2** // Restituisce true se le due variabili hanno un valore diverso

Var1 < **Var2** // Restituisce true se il valore della Var1 è minore del valore contenuto nella Var2

Var1 > **Var2** // Restituisce true se il valore della Var1 è maggiore del valore contenuto nella Var2

Var1 <= **Var2** // Restituisce true se il valore contenuto nella Var1 è minore oppure uguale rispetto al valore contenuto nella Var2

Var2 >= **Var2** // Restituisce true se il valore contenuto nella Var1 è maggiore oppure uguale rispetto al valore contenuto nella Var2

Gli operatori (3)

Gli operatori logici solitamente vengono utilizzati nei confronti, quando c'è necessità che due confronti siano verificati contemporaneamente.

(Var1 == Var2) && (Var3 == Var4) // AND : restituisce true se sono vere entrambe le condizioni. In questo caso la condizione sarà vero solo se entrambi i confronti saranno veri ovvero se il valore contenuto nella variabile Var1 è uguale al valore Var2 e se il valore contenuto nella variabile Var3 è uguale al valore contenuto nella variabile Var4.

(Var1 == Var2) || (Var3 <= Var4) // OR : restituisce true se è vera almeno una delle due condizioni. In questo caso per essere vera sarà necessario che sia vero il confronto tra il valore di eguaglianza tra Var1 e Var2 oppure che Var3 sia minore o uguale di Var4 ma non è necessario che lo siano entrambe.

! (Var1 == Var2) // NOT : restituisce true se la condizione all'interno non si verifica (in questo caso restituisce true se Var1 e Var2 sono diverse. È equivalente a verificare che (Var1 != Var2).

Le istruzioni condizionali (1)

```
if (condizione1) {
        istruzione1;
        }
        else{
        istruzione2;
        }
```

Tradotto letteralmente un if è un "SE" la condizione1 tra parentesi tonde si verifica si esegue il blocco di istruzione1, se la condizione1 non si verifica si esegue il blocco else, quindi l'istruzione2.
Possiamo creare anche if-else annidati, così da verificare più condizioni in sequenza:

```
if (condizione1) {
        istruzione1;
        }
        else if(condizione2){
        istruzione2;
        }
        else{
        istruzione3;
        }
```

Le istruzioni condizionali (2)

```
switch (var) {
                    case 1 : {
                    istruzioni1;
                    break;
                    }

                    case ''a'' : {
                    istruzioni2;
                    break;
                    }
                    ...
                    default : {
                    istruzioniD;
                    break;
                    }
        }
        istruzioni;
```

Quando ci sono più valori da confrontare alla stessa variabile possiamo far uso della struttura SWITCH – CASE.
var è la variabile di cui vogliamo confrontare il valore: nel caso var = 1, se eseguono le istruzioni1 e con la parola chiave 'break' si esce dal ciclo e si eseguono le successive istruzioni. Nel caso il primo case non si sia verificato, si procede con il case 2: se var = ''a'' si eseguono le istruzioni2. Uno switch può avere un numero indefinito di case, ma potrebbe non verificarsene nessuno: in questo caso si eseguono le istruzioniD del blocco di default.

Le istruzioni iterative (1)

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

Il ciclo WHILE permette di eseguire il blocco di istruzioni tra graffe finchè la condizione nel while è vera.

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

Il ciclo DO-WHILE permette di eseguire le istruzioni tra do e while finché la condizione nel while si verifica. La differenza con il while è che le istruzioni vengono eseguite sicuramente almeno una volta, poi viene verificata la condizione e in base al confronto si eseguono ancora o no.

Le istruzioni iterative (2)

```
for(inizializzazione; condizione; incremento/decremento/operazione){
istruzioni; }
```

Il for permette di eseguire il blocco di istruzioni un determinato numero di volte, finchè la variabile inizializzata nel primo campo del for non raggiunge la condizione indicata nel secondo campo. Questa variabile ad ogni ciclo viene modificata, a seconda delle esigenze, nel terzo campo del for.

Per prima cosa si inizializza una variabile 'contatore' del ciclo con un valore di partenza:

```
for(var i=0;....; ...){...}
```

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

```
for(var i=0; i<5;...){...}
```

Al termine di ciascuna iterazione viene eseguita l'operazione del terzo campo. Il ciclo poi ricomincia con la valutazione della condizione.

```
for(var i=0; i<5; i++){...}
```

Interrompere un ciclo

Il comando **break** interrompe l'esecuzione di un blocco di istruzioni ed esce dal ciclo in corso. L'esecuzione del programma riprende poi dalla prima istruzione che segue il blocco.

```
//stampa x fino a 4
while(true){
stampa x;
if(x == 5)
   break; //condizione d'uscita: x=5 e il ciclo s'interrompe
x++;
}
```

Il comando **continue** invece interrompe un'unica interazione del ciclo, senza provocarne l'uscita. Salta l'esecuzione delle istruzioni che seguono per riprendere dall'inizio il blocco di istruzioni, all'iterazione successiva.

```
//stampa x fino a 9, saltando la stampa del 5
while(x < 10){
x++;
7 if(x == 5)
    continue; //condizione d'interruzione: se x=5 salta le istruzioni successive nel ciclo e torna all'inizio del ciclo.
stampa x;
}
```

L'oggetto stringa (1)

Come per i valori numerici, anche le stringhe hanno un corrispondente oggetto predefinito che prevede metodi per la loro manipolazione.

```
var stringa = "Ecco una stringa";
```

Un oggetto String dispone della proprietà length che indica il numero di caratteri di cui è composta una stringa:

```
var num = stringa.length;
```

Per estrarre un carattere da una stringa possiamo utilizzare il metodo charAt() che prende come argomento la posizione del carattere all'interno della stringa partendo da zero:

```
var car = stringa.charAt(2);
```

in 'car' ci sarà 'c'

Per sostituire uno o più caratteri all'interno di una stringa ricorriamo al metodo replace(), che prende due argomenti: la sottostringa da cercare e la nuova stringa con cui sostituire la sottostringa. Ad esempio, la seguente istruzione sostituisce la prima occorrenza di "s" con il corrispondente carattere maiuscolo "S":

```
var stringa = "Ecco una stringa".replace("s", "S");
```

Trasformare i caratteri di una stringa da maiuscolo a minuscolo e viceversa:

```
var minuscolo = stringa.toLowerCase();
var maiuscolo = stringa.toUpperCase();
```

L'oggetto stringa (2)

Possiamo individuare la posizione di una sottostringa all'interno di una stringa utilizzando il metodo indexOf():

```
var stringa = "Ecco una stringa";
var sottoStringa = "Ecco";
var result = stringa.indexOf(sottoStringa); // in result ci sarà 0
```

Il metodo substr() ci consente di estrarre una sottostringa di un certo numero di caratteri a partire da una posizione:

```
var result = stringa.substr(posizione da cui partire, numero di caratteri da estrarre);
```

Possiamo creare un array a partire da una stringa, grazie al metodo split(). Come argomento possiamo passare il carattere che diventerà il separatore degli elementi dell'array:

```
var result = "Ecco una nuova stringa".split("n");
//result == ["Ecco u", "a ", "uova stri", "ga"]
```

Per ottenere l'array di tutti i caratteri di una stringa possiamo passare a split() la stringa vuota:

```
stringa.split("");
```

Per eliminare spazi iniziali o finali alla stringa si usa il metodo:

```
stringa.trim();
```

Le espressioni regolari

JavaScript ha un supporto per la gestione delle espressioni regolari basato sull'oggetto **RegExp**. Un'espressione regolare in JavaScript quindi è un oggetto, con delle proprietà e metodi che consentono di gestire testi, individuare e sostituire stringhe all'interno di altre stringhe.

Ci sono due modi per creare un'espressione regolare:

```
var exp = new RegExp("espressione");
var exp = /espressione/;
```

'Un' espressione regolare è uno schema di stringa composto da una sequenza di caratteri alfanumerici e di eventuali caratteri speciali'.

Le parentesi quadre permettono di specificare un insieme di caratteri alfanumerici. Ad esempio, la seguente espressione regolare indica l'insieme delle stringhe che iniziano per vocale e finiscono con bc:

```
var exp = /[aeiou]bc/;
```

Abbiamo anche la possibilità di specificare un intervallo di caratteri indicano l'elemento iniziale e quello finale, come ad esempio [a-z] o [0-9].
Per fare un altro esempio concreto di espressione regolare, quella che segue individua lo schema di un codice fiscale:

```
var codiceFiscale = /^[a-z]{6}\d{2}[abcdehlmprst]\d{2}[a-z]\d{3}[a-z]$/i;
```

/i permette di non far distinziona tra maiuscole e minuscole

Le espressioni regolari

Il modo più semplice per definire una regular expression è settarla direttamente nell'input utilizzando l'attributo pattern es:

```
<input type="text" pattern="[A-Za-z]{3}">
```

Un secondo modo per la creazione di espressioni regolari, consiste nell'utilizzo dell'oggetto RegExp (definendolo in una function). Questo viene istanziato in una variabile un po' come accade per l'oggetto Date().

```
var espressione = new RegExp('pattern');
```

```
var espressione = new RegExp('^[a-z]+$','i');
```

Come vedete il pattern è il primo attributo atteso dal metodo mentre l'eventuale modificatore è il secondo.

Le espressioni regolari

Metodo Test()

Per impostare il controllo di un'espressione regolare si utilizza il metodo **test()** dell'oggetto **RegExp** il quale restituirà true in caso di successo o false in caso contrario.
Vediamo la sintasi:

```
if (espressione.test(stringa)){  //"stringa" è conforme all'espressione}
else{   // "stringa" NON è conforme all'espressione}
```

Naturalmente possiamo impostare il controllo in modo inverso attraverso l'operatore logico di negazione. Se vogliamo che il nostro programma faccia qualcosa solo se l'espressione regolare non è verificata scriveremo così:

```
if (!espressione.test(stringa))
{  // avviso dell'errore... }
```

il che equivale a:

```
if (espressione.test(stringa) == false){  // avviso dell'errore... }
```

Le espressioni regolari

Metodo Test()

Vediamo un esempio concreto: supponiamo di voler verificare che l'username inserito dall'utente sia composto solo da lettere e numeri avvisandolo (con un alert) in caso di errore:

```
// definisco in una variabile un username in modo arbitrario
// nella realtà recupereremo questo valore da un campo input di un form
var username = 'mario.rossi';
// imposto un modello
var re = /^[a-z0-9]+$/i;
// faccio la verifica
if (!re.test(username)) {  alert('Puoi inserire solo lettere e numeri!');    }
```

Le espressioni regolari

Metodo exec()

Il metodo exec() restuisce la stringa corrispondente al pattern (se trovata) oppure null (in caso di match non riuscito).

```
// definisco una stringavar
str = 'Mr.Webmaster';
// definisco un semplice pattern per verificare
// che la stringa inizi per "Se"
var re = /^Se/;
var res = re.exec(str);
document.write(res);
```

Le espressioni regolari

Costruzione del pattern.
La sintassi delle espressioni regolari

Marcatori di inizio e termine stringa:

- ^ - verifica l'inizio di una stringa (attenzione! se utilizzato all'interno di una parentesi quadra indica una negazione);
- $ - verifica la fine di una stringa;

Metacaratteri:

- . - qualsiasi carattere ad esclusione del ritorno a capo;
- \w - verifica che una stringa sia alfanumerica, minuscola
- \W - verifica che una stringa sia alfanumerica, maiuscola
- \d - indica qualsiasi cifra numerica;
- \D - indica qualsiasi carattere che non sia una cifra numerica;
- \s - indica uno spazio vuoto;
- \S - indica qualsiasi carattere che non sia uno spazio vuoto;
- \t - indica una tabulazione;
- \r - indica un *carriage return*;
- \n - indica un *linefeed*;

Parentesi quadre:

- [...] - le parentesi quadre identificano *range* e classi di caratteri;
- [abc] - identifica qualsiasi carattere tra quelli tra parentesi quadra;
- [^abc] - identifica qualsiasi carattere ad esclusione di quelli tra quelli tra parentesi quadra (^ funge da negazione);
- [a-z] o [0-9] - il trattino all'interno della parentesi quadra identifica un *range* di caratteri;

Le espressioni regolari

Costruzione del pattern.
La sintassi delle espressioni regolari

Parentesi tonde:

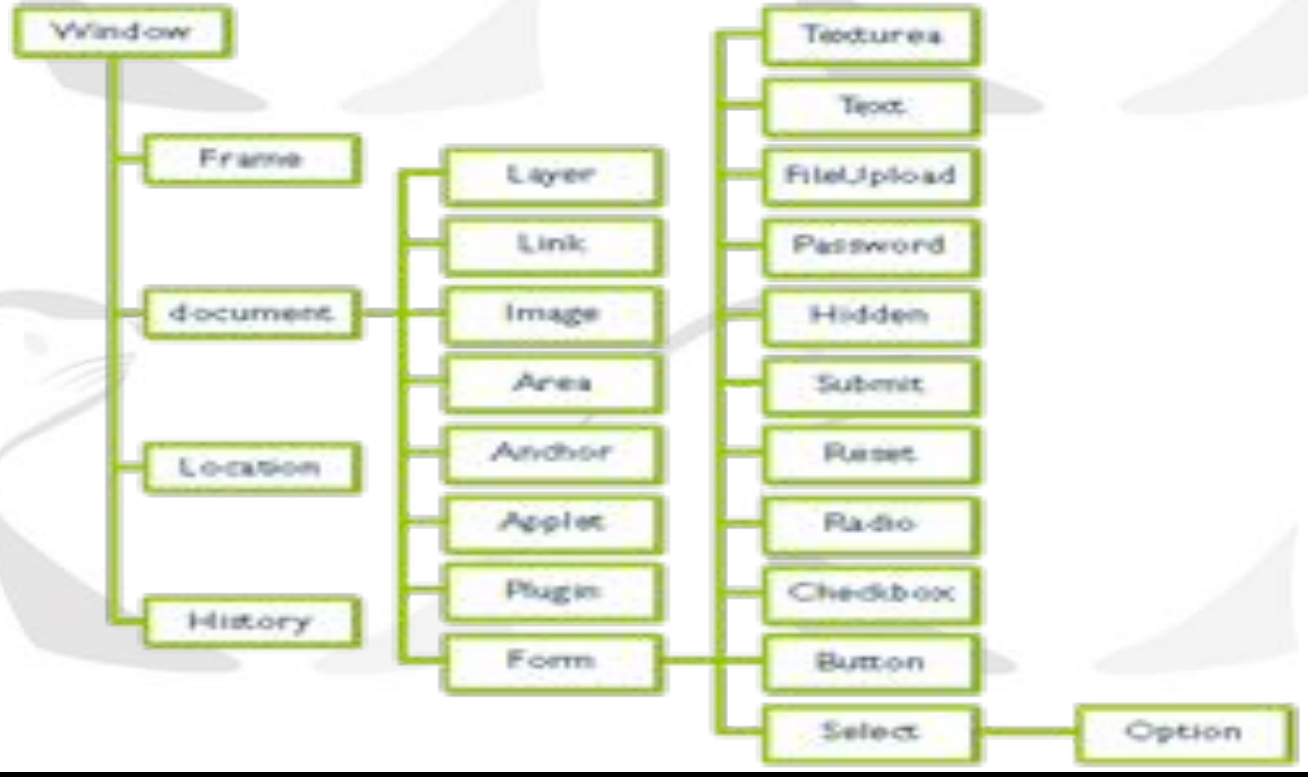
- (...) - le parentesi tonde identificano dei gruppi di caratteri (i gruppi diventamo fondamentali se si deve effettuare una sostituzione mediante espressioni regolari);
- (a|b) - il simobolo | identifica un alternanza (o "a" o "b") all'interno del gruppo;

Ripetizioni di caratteri:

- {x} - indica il carattere precedente per x volte
- {x,} - indica il carattere precedente per x o più volte
- {x,y} - il carattere precedente si ripetere x volte ma non più di y
- ? - indica il carattere precedente per 0 o 1 occorrenza
- * - equivale a {0,}
- + - equivale a {1,}

Il DOM

Il **DOM** (*Document Object Model*) fornisce una rappresentazione del documento HTML come un albero in cui esiste una gerarchia di oggetti, che possono essere facilmente manipolabili e recuperabili tramite Javascript.
La radice di tale albero è l'oggetto ***document*** a cui sono collegati i diversi nodi corrispondenti agli elementi presenti nella pagina (body,h1,div…). Ogni elemento del DOM ha un attributo (align, type, src..) ed eventualmente un testo da contenere.



Il DOM: getElementById

RECUPERARE ELEMENTI DEL DOM:
Se un elemento possiede l'attributo id:

```
<p id="paragrafo">Questo è un paragrafo</p>
```

possiamo utilizzare il metodo getElementById() dell'oggetto document:

```
var p = document.getElementById("paragrafo");
```

Se un elemento possiede l'attributo name:

```
<p name="paragrafo">Questo è un paragrafo</p>
```

possiamo utilizzare il metodo getElementByName() dell'oggetto document:

```
var p = document.getElementsByName("paragrafo")[0];
```

Se utilizziamo il metodo getElementByClassName() possiamo ottenere l'elenco dei nodi a cui è stato assegnato un determinato valore come attributo class.

MODIFICARE GLI ELEMENTI DEL DOM:
La proprietà *innerHTML* rappresenta il contenuto HTML di un elemento ed è accessibile sia in lettura che in scrittura.

```
var p = document.getElementById("paragrafo");
p.innerHTML = "Testo del paragrafo";
```

Eventi nel browser

Gli eventi permettono di gestire il comportamento delle applicazioni al verificarsi di una interazione dell'utente, da altre applicazioni o dal sistema stesso.

Per intercettare gli eventi che scatenati, utilizziamo degli handler (o listener), funzioni che vengono associate ad un certo evento.
Si possono specificare eventi e handler all'interno dei tag HTML, sfruttando speciali attributi e direttamente nel codice JavaScript, tramite function nei tag <script> </script> associando specifiche proprietà degli elementi del DOM;

Gli eventi possibili sono generati da: mouse, tastiera(*onkeypress(), onkeydown(), onkeyup()),* elementi html (onfocus(), onchange(), onselect()), o dalla finestra (onload(), onresize()).
Tra gli eventi più gestiti in un'applicazione web ci sono quelli generati dal mouse.

onclick()
onmousedown()
onmouseover()
Ed altri..
I NOMI DEGLI EVENTI NON SONO CASE SENSITIVE (ONCLICK, onclick, onClick produrranno lo stesso evento).

Come si struttura un programma Js (1)

```
<html>
<head>

<script type="text/javascript">
//Codice javascript sempre dentro il tag script

function  mioconta(n){
var i = 0;
for(i=0; i<n; i++)
    document.writeln("numero:"+(i+1)+"<br>");
}

</script>

</head>
<body>
<input type="button"value="conta"
onclick="mioconta(10);">
</body>
</html>
```

Tutte le variabili js sono **variant**, vi si assegna il tipo solo quando vi si assegna l'informazione all'interno
(var i = 0 ed i diventa di tipo int).

document.writeln() fà riferimento ad un oggetto, già istanziato, che è il documento html. Write è il metodo da eseguire sull'oggetto document che permette di stampare. write scrive e si ferma il cursore, con writeln il cursore si sposta su una nuova linea.

La parola chiave function introduce il concetto di FUNZIONE: un blocco di istruzioni a cui si può far riferimento attraverso un nome, e che si può eseguire quando è necessario. È dichiarata e implementata nel tag <script> che specifica che segue codice di linguaggio js.
function NomeFunzione(eventuali argomenti){
blocco di istruzioni
}

Quando si esegue questa funzione?
Nel caso del nostro esempio, in eseguito ad un evento, onclick ="mioconta(10)" inserito come attributo di un tag html.
Dentro il codice js può essere inserito html tramite " ":
...)+"
");

Come si struttura un programma JS (2)

```
<html>
<head>
<script type="text/javascript">

function premitasto(n){
if(n==1){
    alert('Hai premuto il tasto 1');
} else if(n==2) {
    alert('Hai premuto il tasto 2');
} }

</script>
</head>
<body>
<form>
<input type="button" value="tasto1" onclick="premitasto(1)" />
<input type="button" value="tasto2" onclick="premitasto(2)" />
</form>
</body>
</html>
```

La funzione alert permette di visualizzare un messaggio di avvertimento che blocca qualsiasi azione dell'utente finché non clicca sul pulsante 'ok':

alert('Hai generato un alert');

Altre funzioni che permettono di generare finestre di dialogo sono confirm() e prompt().

ATTENZIONE: nel caso una struttura condizionale (if) o iterativa (for) comprenda SOLO un'unica istruzione da eseguire, le parentesi graffe non sono obbligatorie. Nel caso di più istruzioni, si inseriscono le graffe così da raggrupparle come fossero un'unica istruzione.

Come si struttura un programma JS (3)

```
<html>
<body>
<script type="text/javascript"">
function premitasto(n){
if(n==1){
document.scheda.testo.value="hai premuto il tasto 1";
}else if(n==2){
document.scheda.testo.value="hai premuto il tasto 2";
} }
</script>
<form name="scheda">
<input type="button" value="tasto1" onclick="premitasto(1);" />
<input type="button" value="tasto2" onclick="premitasto(2);" />
<input type="button" value="tasto3" onclick="premitasto(3);" />
<br>
<input type="text" name="testo" />
</form>
</body>
</html>
```

document.scheda.testo.value='hai premuto il bottone 1';
è la riga di codice che ci permette di accedere al documento corrente (document), precisamente alla struttura di nome scheda (form name="scheda"), in cui è presente una casella di testo dal nome testo(text name="testo"), e setta il VALORE del punto del documento raggiunto (.value='hai premuto il bottone 1');

HTML riesce a far riferimento agli oggetti della pagina come fossero vettori.
La form ad esempio può essere richiamata, oltre per nome, anche per ordine di creazione, attraverso l'indice.
Scheda = forms[0] e testo = elements[3].

document.forms[0].elements[3].value = " ";

setta il valore dell'elemento 4 della prima form del documento.

Formazione Orientata al Lavoro

Licenza d'uso