Ing. Fabrizio Fioravanti, Ph.D.
Ing. Marius Spinu, Ph.D.

Javascript

Settembre 2001

1. Javascript: le origini

Il World Wide Web si è sviluppato grazie alla possibilità di poter visualizzare la grafica e la multimedialità in rete. Mosaic, il primo browser, venne rilasciato nel 1992 e permetteva di visualizzare la grafica oltre al testo; nel 1994 parte degli sviluppatori di Mosaic fondarono la Netscape Communications Corporation e il loro browser si rivelò ben presto di qualità superiore, tanto che svilupparono **Javascript** che venne implementato per la prima volta sulla versione beta di Netscape Navigator 2.0 nel giugno 1995. Tale linguaggio apportò un notevole cambiamento alle pagine HTML, perciò alcuni effetti, che erano realizzabili soltanto con l'interfaccia CGI, diventarono più facili da effettuarsi e la stessa dinamicità non restò più limitata alle sole gif animate.

Il 1995, inoltre, resta una pietra miliare nello sviluppo di Internet perché accanto a Netscape anche un'altra società saliva alla ribalta, grazie al previdente investimento sulle potenzialità del Web: la Sun Microsystems Inc., che aveva presentato qualche mese prima Java, il linguaggio evoluto che si proponeva di diventare uno standard nella comunicazione in rete.

JavaScript è stato creato grazie ad una stretta collaborazione tra Netscape e Sun Microsystem, ed è ovvio, quindi, che la comprensione di un tale linguaggio non possa prescindere da Java. E' bene precisare che JavaScript è cosa molto diversa da Java. Entrambi i linguaggi sono orientati agli oggetti, ma mentre Java è usato per creare applicazioni autonome o applet, JavaScript viene interpretato insieme al codice HTML (del quale è parte integrante, e senza il quale non può esistere), senza necessità di macchine virtuali o conoscenze approfondite di modelli orientati agli oggetti. I due linguaggi hanno in comune parte della sintassi e della struttura ma sarebbe impossibile, solo per fare un esempio, creare un programma troppo complesso in JavaScript. JavaScript, quindi, si rivolge agli sviluppatori di siti web che intendono ottenere risultati apprezzabili senza necessità di imparare complessi linguaggi di programmazione.

Come tutti i linguaggi di programmazione orientati agli oggetti, JavaScript stabilisce una gerarchia di oggetti che consente di definire con esattezza proprietà altrimenti indefinibili. Gli oggetti principali in Nestscape sono: window, document, history, location e navigator. Tali oggetti sono collegati tra loro da relazioni strutturali del tipo:

Oggetto1.oggetto2.proprietà2

Oggetto1 è al livello più alto, oggetto2 è una proprietà di oggetto1 e proprietà2 è una proprietà di oggetto2

JavaScript si compone di elementi di programmazione quali: argomenti, gestori di eventi, funzioni, letterali, espressioni, metodi, oggetti, operatori, proprietà, istruzioni, valori e variabili.

Nella pratica JavaScript consente di arricchire documenti HTML con script più o meno complessi e più o meno utili. L'apertura di finestre indipendenti da quella principale del browser è una delle peculiarità di JavaScript, che oltre a dare esempi pratici consente di definire approfondire elementi concettuali.

Qualcuno si potrà chiedere cosa faceva la Microsoft? Ebbene, in quel periodo veniva distribuito Internet Explorer 2.0, che si presentava carente sotto diversi punti di vista, e rivelava come quella società fosse molto scettica in questo campo.

Tra Javascript e Java si pensa che ci siano vari aspetti in comune, oltre ad un nome molto simile, ma le differenze sono tante o poche, secondo i punti di vista. Innanzitutto il primo mito da sfatare è proprio nel nome, in quanto Javascript, alla sua prima apparizione si chiamava LiveScript, per il parallelismo con LiveWire, un linguaggio che la stessa Netscape aveva messo a punto per la gestione della programmazione dal lato server, ma i due linguaggi, affermatisi contemporaneamente, non potevano che avere 'vite parallele', ed infatti, nel dicembre del 1995 la Netscape e la Sun annunciarono di collaborare allo sviluppo di LiveScript, che prese il nome attuale di Javascript.

Javascript e Jscript

Nel 1996, però, la Microsoft iniziò a mostrare un grande interessamento per il Web quindi si avanzò l'ipotesi che per Netscape i giorni fossero ormai contati, tuttavia la lotta, benché impari, si presentò più dura del solito in quanto Netscape cresceva, anche se lentamente, su basi solide, e su un browser che nasceva già potente, mentre Explorer rivelava tutti i difetti di un browser nato in fretta e con strategie

spesso contrastate dall'evidenza dei fatti. In quest'ultimo caso è emblematico il tentativo della Microsoft di contrapporre a Javascript una versione ridotta del Visual Basic che prese il nome di VBScript, ma le sue capacità si presentarono limitate da diversi bug. La Microsoft con Internet Explorer 3.0 dovette ripiegare verso l'adozione di un linguaggio che di fatto era molto simile a Javascript, ma che per esigenza di copyright, non poteva avere lo stesso nome, per cui fu definito JScript.

L'introduzione a Javascript è sembrata importante perché in questo settore, anche se apparentemente le due società dichiarano di seguire gli standard della ECMA-262, la guerra continua a giocarsi a colpi bassi e, se non si rievoca un poco di storia, difficilmente si riesce a comprenderne le motivazioni intrinseche, e difficilmente si riesce a comprendere anche perché in Italia il 70% dei navigatori utilizza Explorer, mentre negli USA, dove nel 1995 il Web era in piena esplosione, questa cifra scende a poco più della metà. Per questo motivo si capisce come Netscape attiri poco l'attenzione di coloro che si sono avvicinati negli ultimi momenti ad Internet e addirittura lo possono giudicare inferiore ad Explorer, ma bisogna comprendere come questa società, effettivamente in difficoltà, stia difendendo ciò che finora ha fatto di buono, tra cui lo stesso Javascript, e l'utilizzo di versioni precedenti di quel browser si rivelano ancora efficienti e stabili, mentre la Microsoft ha dovuto eclissare le sue.

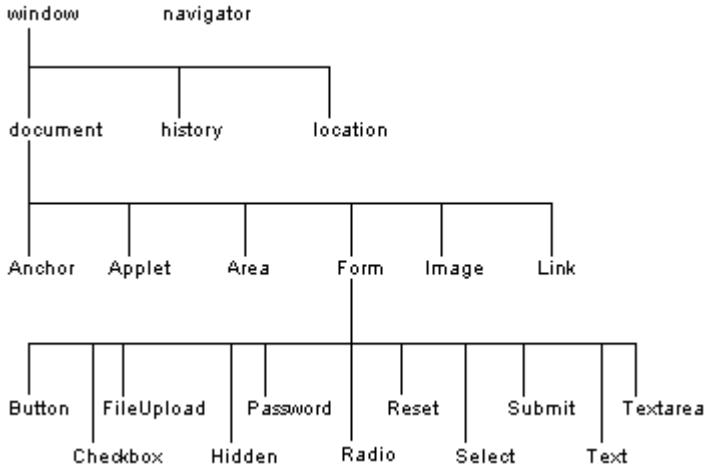
Le varie novità introdotte da Microsoft e da Netscape creano continui disorientamenti e spesso si perde di vista la vera forza di Javascript: **la compatibilità con i browser anche più datati**. Un sito programmato in HTML e in Javascript (nella sua versione 1.1 ma anche 1.2) sicuramente sarà visibile da quasi il 90% dei navigatori (e la cifra è destinata a salire).

2. Aspetti e caratteristiche generali

Javascript è molto semplice da imparare per chi già conosce linguaggi simili come il C++ o Java, ma non è neanche difficile per chi si avvicina per la prima volta a questo linguaggio, data la sua semplicità sintattica e maneggevolezza. Tuttavia ciò può rappresentare un'arma a doppio taglio perché la semplicità si gioca anche su una disponibilità limitata di oggetti quindi alcuni procedimenti, all'apparenza molto semplici, richiedono *script* abbastanza complessi.

Quando, con il nostro navigatore, leggiamo un documento HTML gli oggetti presenti nella pagina (ed individuabili *grosso modo* dai vari TAG) seguono una certa **gerarchia** di interpretazione. Cioè, il browser, li legge con scrupoloso ordine predefinito e ci offre il risultato finale alla nostra vista. E', proprio seguendo questa gerarchia che noi possiamo, con JavaScript, intervenire su ognuno di questi oggetti HTML, cambiando, a piacimento del programmatore, le caratteristiche.

Gli oggetti e la loro gerarchia di interpretazione sono riassunti nella seguente tabella.



La gerarchia presentata non si deve intendere come definitiva poiché ogni versione di browser potrebbe aggiungere o eliminare elementi. Anche in questo documento potrebbero essere citati oggetti non presenti nella gerarchia sopra presentata.

La caratteristica principale di Javascript, infatti, è quella di essere un *linguaggio di scripting*, ma soprattutto è il *linguaggio di scripting per eccellenza* e certamente quello più usato. Questa particolarità comporta una notevole serie di vantaggi e svantaggi secondo l'uso che se ne deve fare e tenendo in considerazione il rapporto che si instaura nel meccanismo client-server. Spiegando in parole molto semplici quest'ultimo rapporto, possiamo dire che il server invia i dati al client e questi dati possono arrivare in due diversi formati: in formato testo (o ASCII) o in formato binario (o codice macchina). Il client sa comprendere solo il formato binario (cioè la sequenza di 1 e 0), per cui se i dati arrivano in questo formato diventano immediatamente eseguibili (e purtroppo senza la possibilità di effettuare controlli), mentre se il formato è diverso devono essere interpretati e tradotti in formato binario, e quindi il client ha bisogno di un filtro o meglio di un interprete che sappia leggere questi dati e li possa tradurre in binario. I dati in formato testo sono visibili all'utente come semplici combinazioni di caratteri e di parole, quindi di facile manipolazione, ma richiedono più tempo per la loro interpretazione a causa dei passaggi e delle trasformazioni che devono subire per essere compresi dal client; i dati in formato binario, invece, sono di difficile (impossibile) comprensione da parte dell'utente, ma immediatamente eseguibili dal client, senza richiedere passaggi intermedi.

Effettuata questa premessa i linguaggi di solito utilizzati per il Web si possono suddividere in quattro tipologie:

1. *HTML: è in formato testo e non è un linguaggio nel senso tradizionale, ma un impaginatore per consente di posizionare degli oggetti nella pagina con le caratteristiche indicate, naturalmente per la sua peculiarità risulta essere statico e non interagisce con l'utente e non può prendere decisioni se non per i formulari, mentre per la sua interpretazione ha bisogno di un browser;*

2. *linguaggi compilati: sono quei linguaggi abbastanza complessi in cui il sorgente (un file di testo con le operazioni da eseguire) viene* ***compilato*** *in codice macchina e viene impacchettato in un eseguibile utilizzabile solo nella forma e per le operazioni per cui è stato progettato;*
3. *linguaggi semicompilati: in realtà a questa classe appartiene solo Java perché è un linguaggio compilato in un formato intermedio tra il file ASCII e il file binario, tale formato si chiama* bytecode *e va interpretato sul client da una macchina virtuale chiamata* Java Virtual Machine*, in tal modo all'atto della ricezione tale macchina completa la compilazione e rende il file eseguibile;*
4. *linguaggi interpretati: sono quei linguaggi che risultano molto simili all'HTML, ma hanno potenzialità maggiori perché consentono di effettuare controlli e operazioni complesse, vengono inviati in file ASCII, quindi con codice in chiaro che viene* ***interpretato*** *ed eseguito riga per riga dal browser in modalità runtime.*

Il concetto di script è bene espresso con una similitudine di Michael Moncur. Script in inglese significa "copione" o "sceneggiatura", ed, infatti, l'utilizzo è proprio questo: il browser legge una riga, la interpreta e la esegue, poi passa alla successiva e fa la stessa cosa, e così di seguito fino alla chiusura dello script.

Vantaggi e svantaggi

Quali sono i vantaggi e gli svantaggi tra linguaggi di scripting e linguaggi compilati? Cerchiamo di riassumerne qualcuno:

il linguaggio di scripting è più sicuro ed affidabile *perché in chiaro e da interpretare, quindi può essere filtrato; per lo stesso Javascript la sicurezza è quasi totale, perché solo alla sua prima versione erano stati segnalati dal CIAC (Computer Incident Advisory Committee) dei problemi di lieve entità, tra cui la lettura della cache e dei siti visitati, dell'indirizzo e-mail e dei file presenti su disco, tali "falle", però, sono state corrette già con le versioni di Netscape successive alla 2.0;*

gli script hanno limitate capacità, per ragioni di sicurezza, *per cui non è possibile fare tutto con Javascript, ma occorre abbinarlo ad altri linguaggi evoluti, magari più sicuri, come Java, e tale limitazione è ancora più evidente se si desidera operare sull'hardware del computer, come ad esempio il "settaggio" in automatico della risoluzione video o la stampa di un documento;*

un grosso problema è che ***il codice è visibile e può essere letto da chiunque****, anche se tutelato con le leggi del copyright, ma questo è il prezzo da pagare da chi vuole utilizzare il web: la questione dei diritti d'autore è stata rivoluzionata con l'avvento di Internet (si veda soprattutto l'MP3), e la tutela è molto labile e inadeguata alle leggi attuali, per cui occorre prendere la situazione con molta filosofia;*

il ***codice Javascript viene eseguito sul client*** *per cui il server non è sollecitato più del dovuto; uno script eseguito sul server, invece, sottoporrebbe questo ad una dura sollecitazione e i server di capacità più limitate ne potrebbero risentire se interrogati da più utenti;*

il codice dello script deve essere scaricato completamente prima di poter essere eseguito, e questo è il risvolto della medaglia di quanto detto precedentemente, per cui se i dati che uno script utilizza sono tantissimi (ad esempio una raccolta di citazioni da mostrare in maniera casuale), ciò comporterebbe un lungo tempo di scaricamento, mentre l'interrogazione dello stesso database sul server sarebbe più rapida.

Le diverse versioni

E' stato fatto l'accenno all'EMCA, per cui è d'obbligo un approfondimento, anche perché non abbiamo mai trovato riferimento nelle guide Javascript in italiano consultate finora, ma spulciando tra quelle ufficiali, non mancava mai un'indicazione a questa associazione.

Innanzitutto l'ECMA (traducendo fedelmente) è un'associazione internazionale di industrie basate sull'Europa, fondata nel 1961, dedicata alla standardizzazione dei sistemi di comunicazione e di informazione. Per chi volesse saperne di più il sito è: http://www.ecma.ch.

Aderiscono all'associazione in qualità di membri ordinari società come le stesse Netscape e Microsoft, la Sun, l'IBM, la Compaq, la Philips, l'Hewlett-Packard, l'Intel, mentre sono membri associati altre società come la Mitsubishi Electric, la Quantum, la Rockwell.

Netscape Communications presentò Javascript all'ECMA per la standardizzazione nell'autunno 1996. L'Assemblea Generale dell'ECMA ha adottato il linguaggio come standard nel giugno 1997. La versione presentata era la 1.1, per cui è questa ad essere giudicata standard e per tale motivo definita ECMAScript o ECMA-262 (la documentazione completa è reperibile presso il sito della Netscape http://developer.netscape.com/docs/manuals).

Versioni JScript e Javascript

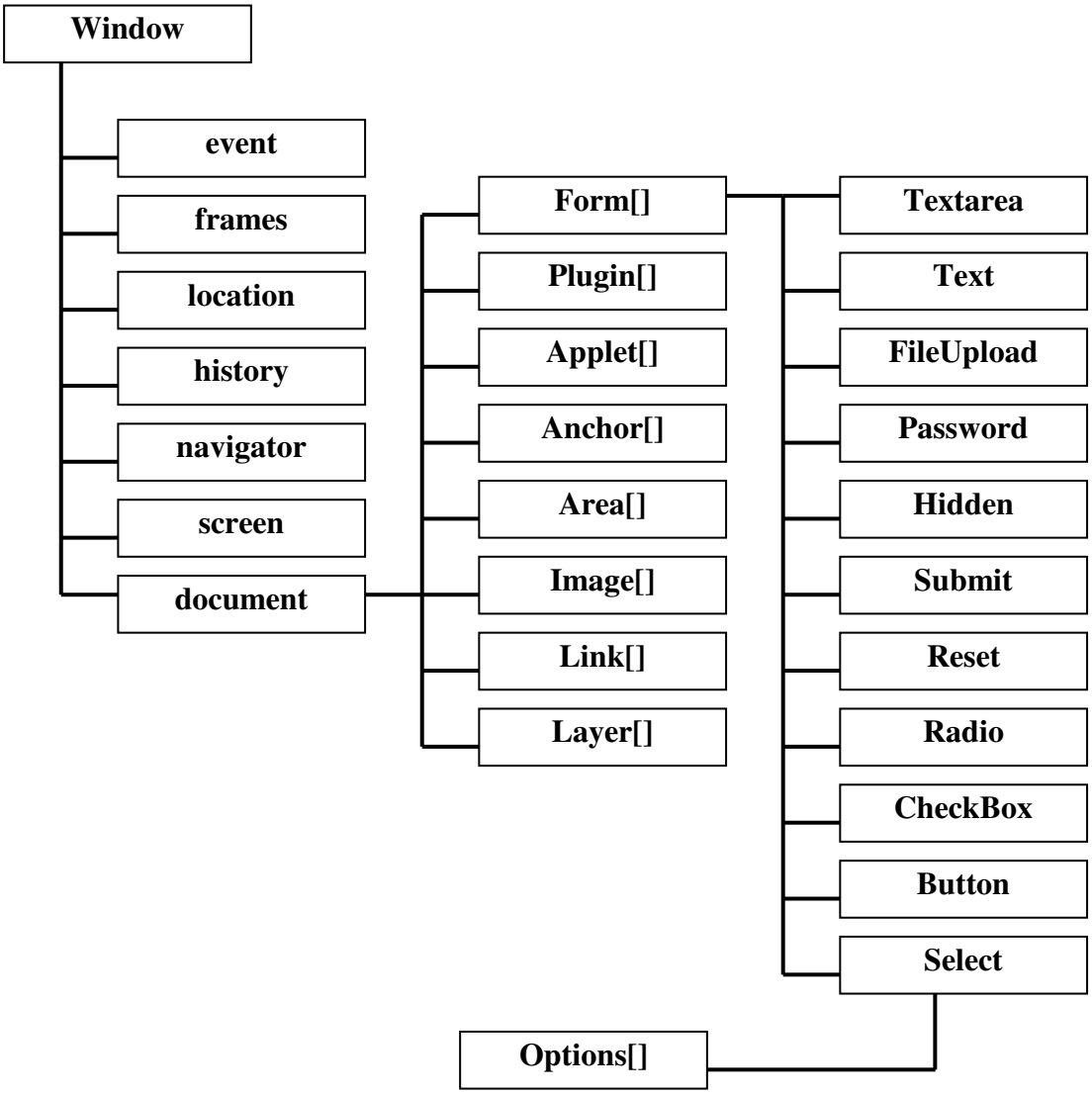
JScript di Microsoft nella sua versione 3.0 (valida per Explorer 4.0), quindi, implementa completamente l'ECMAScript, ma aggiunge anche tutte le caratteristiche di Javascript 1.2. Cerchiamo, comunque, di dare una tabella comparativa tra browser e versioni di Javascript e JScript, la tabella sarà utilissima anche in seguito:

	Javascript				**Jscript**		
	1.0	1.1	1.2	1.3	1.0	3.0	5.0
Netscape 2.0	✓						
Netscape 3.0	✓	✓					
Netscape 4.0	✓	✓	✓				
Netscape 4.06	✓	✓	✓	✓			
Explorer 3.0	✓				✓		
Explorer 4.0	✓	✓	✓		✓	✓	
Explorer 5.0	✓	✓	✓	✓	✓	✓	✓

DHTML

Tuttavia occorre considerare che nelle ultime versioni dei browser, soprattutto da parte di Microsoft, si è introdotto il DHTML: se si dovesse dare una definizione di questo linguaggio, sarebbe un'impresa. Abbiamo spulciato tra la documentazione ufficiale della Microsoft, ma sembra che anche loro abbiano le idee confuse: in una FAQ alla domanda precisa "Che cosa è DHTML", tutto si dice tranne che la risposta. La nostra opinione è questa: JScript era vincolato ad uno standard, VBScript era ormai superato nell'implementazione lato client, la Microsoft ha tirato fuori dal cilindro questo prodotto che si presenta come un HTML avanzato, perché inserisce la struttura DOM (Document Object Model) ovvero, il modello di documento ad oggetti in cui il documento viene diviso in elementi più semplici a cui si applica una programmazione orientata agli oggetti. DHTML ha introdotto diverse novità nella programmazione client di una pagina web, ed è certamente uno strumento potente, anzi, molto potente, tanto da risultare quasi incontrollabile.

Un generico DOM è presentato nella figura sottostante:



3. I primi rudimenti del linguaggio

TAG <SCRIPT>

Dopo aver parlato in generale degli script, occorre passare alla pratica e vedere come inserirli nella pagina HTML. La spiegazione risulta abbastanza complessa soprattutto perché Javascript ormai si integra così bene in HTML da non avere più spazi definiti, ma è possibile trovarlo ovunque.

L'HTML prevede un tag apposito per gli script, e tale tag é:

```
<SCRIPT><!–
   codice
//--></SCRIPT>
```

si noteranno anche dei simboli all'interno dei tag, per il momento accenniamo solo a dire che hanno una loro utilità, si spiegherà poi quale è. Tali tag, possono essere in numero variabile, l'unica attenzione sta nel chiuderli ogni volta che vengono aperti.

I browser ricevono le pagine HTML con tutto il contenuto, quando si incontra il tag <SCRIPT> questo viene eseguito come tutti gli altri tag, dall'alto in basso, ma il suo contenuto è interpretato secondo un codice diverso: in tal modo se il browser comprende il codice, questo viene eseguito, e se si incontra un errore nell'esecuzione dello stesso i casi sono due:

- *la pagina viene visualizzata, ma il codice errato non viene eseguito;*
- *se il codice genera un loop (cioé un ciclo infinito) la pagina resta bianca o visualizzata parzialmente perché l'esecuzione dall'alto in basso del codice HTML è momentaneamente interrotta.*

Così come scritto, però, il tag <SCRIPT> non è completo perché i linguaggi di scripting sono diversi, allora occorre mettere anche la specificazione del linguaggio ed è:

```
<SCRIPT Language="Javascript"><!--
   codice
//--></SCRIPT>
```

Ciò potrebbe bastare, ma negli ultimi riferimenti, soprattutto da parte di Netscape, si consiglia vivamente di indicare anche la versione di Javascript che si adopera, soprattutto perché l'evoluzione del linguaggio è continua e non sempre assicura la compatibilità con i vecchi browser. In tal modo si occulta il codice ai browser che non possono gestire gli aggiornamenti del linguaggio (per le versioni di Javascript si faccia riferimento alla tabella precedente). Alla luce di quanto detto, il precedente script può essere considerato valido per la versione 1.0 di Javascript, e quindi per tutti i browser, mentre uno script del genere:

```
<SCRIPT Language="Javascript1.2"><!--
//--></SCRIPT>
```

diventa leggibile solo da Netscape 4.0 e Explorer 4.0 e dalle loro versioni successive. Vi chiederete anche come fare a conoscere tutte le compatibilità: ebbene non c'è nessun programma che aiuti in ciò, occorre conoscerle a fondo oppure usare un metodo empirico: testare le pagine su diversi browser e segnalare le incompatibilità, se queste dipendono dalla versione di Javascript, mascherarle con l'indicazione della versione.

Richiamo degli script

In linea di principio uno script può essere inserito in due modi all'interno di pagina HTML (un'eccezione è rappresentata dagli script del server creati con LiveWire):

- *inserendo il codice nel documento;*
- *caricandolo da un file esterno.*

l'uso del file esterno, infatti, è dettato dal *limite di dimensione di 32K che deve rispettare una pagina web*.

Script Esterni

In quest'ultimo caso (è anche quello più semplice a spiegarsi) lo script è salvato in un file con estensione .js. Viene richiamato con l'attributo **SRC** del tag SCRIPT:

```
<SCRIPT Language=Javascript SRC="nomefile.js"><!--
//--></SCRIPT>
```

dove la specificazione di **Language** è facoltativa, poiché la stessa estensione del file basta a dimostrare il linguaggio adoperato, ma si consiglia proprio per identificare la versione. Il nome del file può essere indicato con un URL relativo o assoluto.

Tale file esterno viene eseguito all'interno della pagina HTML, per cui lo script viene solo letto come file di testo, trasferito nell'HTML nella posizione di richiamo e qui eseguito. Per tale motivo il file va salvato come testo ASCII, senza caratteri di controllo e senza tag HTML o elementi di altri linguaggi per non generare errori, e si può adoperare un qualsiasi editor molto semplice.

Il vantaggio di usare file esterni è immenso soprattutto perché apporta la caratteristica della modularità per cui uno script che ricorre di frequente può essere scritto una sola volta e richiamato in qualsiasi pagina HTML quando serve, ma tutto ciò ha un prezzo: funziona solo con Netscape 3.0 ed Explorer 4.0 e nelle versioni successive.

Esempio

1.Scrivere con Blocco Note il seguente comando: alert('Sono un file esterno'), e salvarlo con il nome prova.js;
2. Scrivere in un altro file il seguente codice HTML:

```
<HTML><HEAD>
<SCRIPT SRC="prova.js">
</SCRIPT></HEAD>
<BODY></BODY></HEAD></HTML>
```

e salvarlo nella stessa directory del file Javascript con estensione html o html;
3.Caricare la pagina HTML in un browser.

Script Interni

Se lo script è all'interno del documento, può **essere immesso sia nella *sezione di intestazione* (tra i tag <HEAD></HEAD>) sia in quella del *corpo del documento* (tra i tag <BODY></BODY>)**. Occorre tenere presente che la pagina HTML viene eseguita in ordine sequenziale: dall'alto in basso, per cui la differenza tra le due alternative esiste: lo script dell'intestazione viene caricato prima degli altri, quello nella sezione body, invece, viene eseguito secondo l'ordine di caricamento. Che cosa cambia tutto ciò? Bisogna considerare che una variabile o qualsiasi altro elemento di Javascript può essere richiamato solo se caricato in memoria: tutto ciò che si trova nell'intestazione è quindi visibile agli altri script, quello che si trova nella sezione BODY è visibile agli script che lo seguono. La scelta dipende anche da altri fattori (come la creazione della pagina HTML in maniera dinamica), ma sarà poi l'esperienza a suggerirli.

Esempio
1. Scrivere il seguente codice HTML:

```
<HTML><HEAD>
<SCRIPT Language="Javascript">
x=1; alert('TESTA='+x);
</SCRIPT></HEAD>
<BODY>
<SCRIPT Language="Javascript">
x++;alert('CORPO='+x);
</SCRIPT>
<SCRIPT Language="Javascript">
```

```
x++;alert('CORPO='+x);
</SCRIPT>
</BODY></HEAD></HTML>

2.Provare il codice e verificare la sequenza di esecuzione degli script
```

Browser non compatibili

Potrebbero essere utilizzati anche browser non compatibili con Javascript oppure quelli in cui Javascript è disabilitato (è possibile con Netscape). In questo caso ci viene in aiuto il tag **<NOSCRIPT> </NOSCRIPT>** che può contenere testo e grafica alternativi oppure un reindirizzamento in pagine statiche, che non adoperano Javascript, mediante la sequenza:

```
<NOSCRIPT>
<META HTTP-EQUIV REFRESH CONTENT="0; URL=altrapagina.htm">
</NOSCRIPT>
```

Commenti e punteggiatura

Commenti

I commenti sono parti del programma che non sono lette dall'interprete e che, quindi, servono a spiegare e a chiarire. Questi sono **racchiusi tra barre e asterischi** come nell'esempio sotto riportato:

```
/*commento*/
```

Il commento può essere posto su più righe o su una riga singola, mentre *non è accettato dall'interprete il commento annidato*.

Un altro tipo di commento è la **doppia barra**, presa a prestito dal linguaggio C++, ma è valida solo per commenti posti su una singola riga, anche se non la occupano per intero:

```
int x: //commento
```

I commenti Javascript non possono essere inseriti al di fuori dei tag che individuano lo script, altrimenti HTML li considererà come parte del testo, e viceversa non si possono utilizzare i tag di commento HTML all'interno dello script. L'unico commento ammissibile è quello che consente **di racchiudere tutti gli script nei tag di commento di HTML**, facendoli aprire dopo il tag di script e chiudere prima della chiusura del tag:

```
<script language="JavaScript">
<!--
alert("Welcome!");
//-->
</script>
```

in tal modo **si maschera il codice javascript ai vecchi browser** che non lo leggono e si evita che l'HTML lo possa considerare come testo e, quindi, visualizzare. Tuttavia occorre tenere presente due accortezze:

- *alcuni browser non riconoscono il commento e visualizzano lo script;*
- *alcuni browser, soprattutto Netscape nelle versioni più vecchie, hanno difficoltà a gestire il segno > di fine commento, per cui conviene posizionare anche un commento Javascript (//) prima della sequenza* **-->**.

Spazi Bianchi

Javascript **non bada agli spazi bianchi**, tranne che per quelli che si trovano nelle stringhe, per cui si possono omettere o anche aumentare. Il loro uso, tuttavia, con l'**identazione** aumenta la leggibilità del programma per cui sono vivamente consigliati.

Apici

Importanti sono gli **apici**, **sia singoli (' ') che doppi (" ")**.

I **doppi apici si adoperano per racchiudere parti di codice Javascript**, e, insieme a quelli singoli, a racchiudere anche le stringhe (sequenze di caratteri), per cui occorre fare attenzione ad annidare due stringhe racchiuse da apici simili, come ad utilizzare i doppi apici per le stringhe se questi già servono a racchiudere codice Javascript.

Se si desidera che in una stringa appaiano apici doppi o singoli come parte integrante della stringa stessa, si fanno precedere da una barra rovesciata (\).

Uno degli errori che si commette di frequente, è proprio quello di non utilizzare correttamente gli apici. Ad esempio il comando:

```
alert('Questo sito e' in costruzione')
```

sembra essere scritto correttamente, ma se eseguito, il browser ne bloccherà l'esecuzione. Netscape mostrerà questo errore:

missing) after argument list.
alert('Questo sito e' in costruzione')

più laconico Explorer, che indica solo: Previsto')'. Si proseguirà oltre, ma l'errore non sarà corretto finché non si scriverà

```
alert('Questo sito e\' in costruzione')
```

Istruzioni

Le *istruzioni hanno la responsabilità di controllare il flusso di elaborazione del codice*. Esse possono:

> ***eseguire iterazioni**, cioè ripetere una parte di codice per un certo numero di volte;*
> ***eseguire decisioni condizionate**;*
> ***richiamare funzioni** (si vedrà in seguito cosa sono);*
> ***consentire** al percorso dell'esecuzione **di essere modificato dinamicamente**.*

In Javascript ogni istruzione inizia ad ogni nuova riga o con il **punto e virgola**, come accade col C e con Java, ma consiglio vivamente di imparare ad utilizzare i punti e virgola, non tanto per compatibilità con gli altri linguaggi, quanto per evidenziare meglio i costrutti.

```
Esempio
1. Individuare le istruzioni nel seguente codice:
<SCRIPT Language="Javascript">
x=1;alert('PROVA');
x++
alert(RIPROVA);y=x
</SCRIPT>
2.Le istruzioni sono cinque.
```

Blocchi di istruzioni (costrutto sequenza)

Un'**istruzione composta**, invece, è formata da un gruppo di due o più istruzioni che formano un gruppo logico, nel senso che l'esecuzione delle stesse è legata, ad esempio, al verificarsi di una condizione: tali istruzioni sono raggruppate in blocchi individuati dalle parentesi graffe.

Modalità di esecuzione

Dopo aver visto le forme tradizionali di interfacciamento del codice Javascript con il codice HTML, effettuiamo un riassunto dei vari concetti rispetto a quest'argomento. Questa parte potrebbe apparire un poco fumosa perché andrà molto per voli pindarici, in quanto la teoria farà la parte da leone, ma a chi interessano, questi concetti possono essere ripresi in seguito e certamente appariranno più chiari.

Le istruzioni in Javascript possono essere eseguite in diverso modo:

> *all'**interno degli script**, individuati dai tag <SCRIPT>, in maniera sequenziale, per cui l'esecuzione è automatica;*
>
> ***caricandoli da file esterni**;*
>
> ***in seguito all'attivazione di un evento** (handler) come un click del mouse o la pressione di un tasto (si vedranno in seguito gli eventi);*
>
> ***in luogo di un link** (a partire da Netscape 3.0) nella forma: <A HREF="Javascript:comando">*
>
> ***valori Javascript** possono essere richiamati dinamicamente dall'HTML includendoli tra i caratteri **&{** e **};%** ad esempio la larghezza di una tabella può essere ricavata in rapporto ad un valore javascript nella forma width="&{barWidth};%"*

logicamente l'utilizzo delle quattro opzioni varia secondo l'obiettivo da raggiungere. Così se il codice Javascript va eseguito in maniera sequenziale basta inserire uno script, mentre se va eseguito in seguito al realizzarsi di uno evento, occorre operare con un handler combinato ad una funzione.

Esempio
1. Scrivere il seguente codice e notare come gli script vengano eseguiti e in seguito a quali eventi:

```
<HTML><HEAD></HEAD>
<BODY>
<SCRIPT Language="Javascript"><!--
alert('Script');
//--></SCRIPT>
<a href="#" onmouseover="alert('Hai passato il mouse')">Passa il
mouse</a>
<a href="javascript:alert('Hai cliccato')">Clicca qui</a>
</BODY></HTML>
```

4. Gli eventi

Innanzitutto occorre fare una premessa: nel decidere l'ordine degli argomenti da trattare eravamo abbastanza indecisi in quale posizione porre quest'argomento, e quelli ad esso collegati, in quanto in alcuni testi gli eventi sono trattati prima delle funzioni, mentre in altri dopo. La scelta non era indifferente perché l'argomento compone una parte abbastanza corposa della teoria Javascript e, se posto in una posizione sbagliata, rischia di rompere la sequenzialità della trattazione. Poi abbiamo optato per la posizione attuale in quanto l'argomento è molto affine alla trattazione degli Script in generale, ma siamo coscienti di rischiare di renderci poco comprensibili ai neofiti, per cui vogliano questi apprendere quanto esposto di seguito dando per scontate alcune affermazioni, soprattutto sulle funzioni e affrontando alcuni esempi focalizzando particolarmente l'attenzione sull'uso degli handler.

Gli *eventi* sono utilizzati per richiamare delle istruzioni. Infatti, lo script va eseguito in maniera sequenziale, ma per fare in modo da inserire la dinamicità e l'interattività occorre che questo resti caricato in memoria e sia attivato o richiamato solo quando si verificano particolari situazioni come il passaggio del mouse, il caricamento di un documento, ecc. Il problema della conciliazione con le funzioni sta nel fatto che ad un evento può essere associata una sola istruzione, ma il più delle volte l'associazione è fatta con un blocco di istruzioni e, quindi, con le funzioni che prendono il nome di **handler** o **gestori di eventi**.

Gli eventi, per poter interfacciare HTML con Javascript, non vengono definiti nel tag <SCRIPT> (tranne che in qualche caso), ma sono inseriti all'interno dei tag HTML: il browser compatibile con Javascript incontrando un evento lo interpreta e lo attiva.

```
Esempio:

<A HREF="jsesterno.html" onclick="alert('ciao')">Link</A>
```

notiamo come l'evento onClick sia inserito nel Tag come se fosse uno specificatore dello stesso.

E' importante capire questo concetto perché Javascript agli inizi aveva pochi eventi ed erano attivabili solo se inseriti in particolari tag e capita spesso di utilizzare gli handler in maniera arbitraria e di considerare come errore la mancata attivazione di un evento quando inserito in tag incompatibili. Internet Explorer nelle ultimissime versioni ha allargato le possibilità di utilizzo degli eventi, per cui possono essere inseriti in tantissimi tag, mentre Netscape è rimasto fedele alle originarie impostazioni. Questo potrebbe essere un bene o un male, ma per esperienza preferisco la strategia di Netscape in quanto gli eventi potrebbero andare in conflitto e, se onnipresenti, potrebbero diventare incontrollabili.

Attivare gli eventi all'interno degli script

Gli eventi, tuttavia, si possono anche attivare direttamente all'interno degli Script, richiamabili come se fossero una proprietà dell'oggetto. La sintassi è:

Oggetto.evento=handler;

Per chiarire questo concetto che è utilissimo, ma raramente si vede applicato, possiamo considerare il caso in cui sia necessario creare uno script che consenta di simulare lo streaming, cioè di caricare un'immagine di una sequenza solo quando siamo sicuri che quella precedente era già stata caricata e quindi visualizzata. Usare un temporizzatore era impossibile perché l'effetto variava secondo la velocità di connessione. Una soluzione potrebbe essere di creare un array di immagini sul quale si opera con il comando:

document.images[num].onload=carica();

dove la funzione carica() serviva a caricare l'immagine successiva. Tutto funzionava alla perfezione, anche se con qualche piccola variazione secondo i browser.

Con Explorer, invece, si può utilizzare uno Script apposito per un oggetto e per un evento tramite la sintassi:

<SCRIPT FOR=Object EVENT=evento>....</SCRIPT>

Raggruppare gli eventi

Negli ultimi tempi gli eventi si sono moltiplicati ed è difficile tenerne traccia, quindi, per facilità cerchiamo di raggrupparli in sezioni omogenee:

> *Eventi attivabili dai tasti del mouse*
> *Eventi attivabili dai movimenti del mouse*
> *Eventi attivabili dal trascinamento del mouse (drag and drop)*
> *Eventi attivabili dall'utente con la tastiera*
> *Eventi attivabili dalle modifiche dell'utente*
> *Eventi legati al "fuoco"*
> *Eventi attivabili dal caricamento degli oggetti*
> *Eventi attivabili dai movimenti delle finestre*
> *Eventi legati a particolari bottoni*
> *Altri e nuovi tipi di eventi*

Nei primi due gruppi sono inseriti quegli eventi tipici del mouse o della tastiera, come il movimento o la pressione, negli altri sono inseriti gli eventi strettamente correlati agli oggetti. Il raggruppamento si richiede perché in tante trattazioni gli eventi sono descritti singolarmente è cio, anche se avvantaggia l'analisi, sgretola la sintesi: che invece è utilissima nelle situazioni più intricate.

Prima della descrizione anticipiamo che tutti gli eventi hanno la propria sintassi composta sintatticamente dal loro nome col prefisso **on**, ad esempio l'evento **click** è richiamato con l'handler **onclick**. Io darò una sintassi fatta di maiuscole e minuscole per evidenziare bene l'evento, ma occorre tenere presente che in Netscape 3.0 l'evento è parte di Javascript per cui deve essere scritto tutto in minuscolo.

Eventi attivabili dai tasti del mouse

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onClick**: attivato quando si clicca su un oggetto;*
> ***onDblClick**: attivato con un doppio click;*
> ***onMouseDown**: attivato quando si schiaccia il tasto sinistro del mouse;*
> ***onMouseUp**: attivato quando si alza il tasto sinistro del mouse precedentemente schiacciato;*
> ***onContextMenu**: attivato quando si clicca il tasto destro del mouse aprendo il ContextMenu.*

Gli eventi onMouseDown e onMouseUp sono attivati dai due movimenti del tasto destro del mouse, il primo quando si preme giù il tasto e il secondo quando si solleva dopo il click.
Il doppio click è un evento che ingloba gli altri e, per la precisione, attiva in successione onmousedown, onmouseup, onclick.

Versioni e compatibilità

L'utilizzo di questi eventi è limitato da diversi fattori. Innanzitutto occorre considerare la versione di Javascript a cui fanno parte e conseguentemente alcuni non sono validi per tutti i browser. Ecco il quadro riepilogativo (NN sta per Netscape e IE per Internet Explorer):

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
onClick	✓	✓	✓	✓	✓	✓	✓	1.0
OnDblClick[1]			✓	✓		✓	✓	1.2
onMouseDown			✓	✓		✓	✓	1.2

OnMouseUp	✓	✓		✓	✓	1.2
OnContextMenu					✓	DHTML

[1]L'evento onDoubleClick non funziona su Mac

Valore true e false

L'evento onClick con JavaScript 1.1 ha aggiunto la possibilità di ricevere un valore true e false tramite il quale può essere attivato o meno. Tale possibilità è valida anche per gli eventi onMouseDown e onMouseUp e per onContextMenu.

```
Esempio:

<A HREF="jsinterno.htm" onclick="return(confirm('Sei sicuro'))">

il link si attiva solo se si risponde OK alla finestra
```

Il vantaggio è che l'evento onClick si attiva prima del tag associato per cui se è un link, questo è caricato dopo il completamento dell'istruzione associata. In tal modo questa caratteristica si può applicare per i radio o i checkbox per non selezionarli, e per i bottoni, compresi quelli Submit e Reset, per considerarli non premuti, tranne che per un piccolo bug che rende non fruibile l'opzione per il Reset su alcune piattaforme.

Interessante l'evento onContextMenu, anche perché spesso si chiede di disabilitare il tasto destro del mouse, ma funziona solo con Internet Explorer 5.0.

Tag Sensibili

Un altro limite per Netscape ed Explorer nelle vecchie versioni è dato dai tag sui quali può essere attivato

Evento	Tag associati in Netscape e Explorer

OnClick	ipt(uestc gestcre è usatc ccr ipulsartidiirvic (subrit), pulsa

n
t
i
d
i
r
e
s
e
t
(
r
e
s
e
t
)
,
c
a
s
e
l
l
e

d
i
c
c
n
t
r
c
l
l
c

(
c
h
e
c
k
b
c
x

e

r
a
d
i
c
)
,
b
c
t
t
t

onDblClick

c
r
i
,
t
a
g

<
I
N
P
U
T
>

d
i
t
i
p
c

(
P
T
I
(
M

e

t
a
g

<
A
>
.
U
s
a
t
c

c
c
r

i
t
a
g

<
B
(
I
Y
>

	e
	<
	A
	>
	U
	s
	a
	t
	c
	c
	c
	r
	i
	b
	c
	t
	t
	c
	r
	i
onMouseDown	e
	i
	t
	a
	g
	<
	P
	C
	I
	Y
	>
	e
	<
	A
	>
	U
	s
	a
	t
	c
	c
	c
	r
onMouseUp	
	i
	b
	c
	t
	t
	c
	r
	i

e

i
t
a
g

<
E
C
I
Y
>

e

<
/
>
M
e
s
s
u
r
c

p
e
r
c
h
é

f
u
r
z
i
c
r
a

s
c
l
c

i
r

E
x
p
l
c
r
e
r

s

onContextMenu

.
(

Molto più numerosi i tag associati in Explorer 4.0 e successivo a tutti i tipi di eventi:
A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP.

Eventi attivabili dai movimenti del mouse

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onMouseOver**: attivato quando il mouse si muove su un oggetto;*
>
> ***onMouseOut**: attivato quando il mouse si sposta da un oggetto;*
>
> ***onMouseMove**: si muove il puntatore del mouse, ma poiché questo evento ricorre spesso (l'utilizzo del mouse è frequente), non è disponibile per default, ma solo abbinato con la cattura degli eventi, come si spiegherà in seguito.*

Gli eventi onMouseOver e onMouseOut sono complementari in quanto il primo è attivato nel momento in cui il puntatore è posto nell'area dell'oggetto il cui tag contiene l'evento e il secondo quando ne esce.
Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
OnMouseOver	✓	✓	✓	✓	✓	✓	✓	1.0
OnMouseOut	✓	✓	✓	✓	✓	✓	✓	1.0
OnMove			✓	✓		✓	✓	1.2

Gli eventi onMouseOver ed onMouseOut assumono dalla versione 1.0 alla 1.1 di Javascript la capacità di essere associati al tag AREA, per cui può operare anche con le mappe cliccabili, ma per Netscape deve essere associato anche al tag HREF, cioè ad un link, anche se fittizio.

Tag Sensibili

Un altro limite per Netscape ed Explorer nelle vecchie versioni è dato dai tag ai quali può essere associato

Evento	T a g a s s c c i a t i i r N e

	tscape e **J**Script **(**uestc gestcre è usatc ccri i**Pulsartidiirvio** **(**
OnMouseOver	

submit), pulsante di reset (reset), casella di controllo (checkbox

e

r a d i o) , b o t t o n i , t a g

< I N P U T

d i t i p c

(P T I (N

e

t a g

< A > . U s a t c

OnMouseOut

c

OnMouseMove	c r i t a g < E C I Y > e < A > U s a t c c c r i b c t t c r i e i t a g < E C I Y > e < A >

Molto più numerosi i tag associati in Explorer 4.0 e successivo a tutti i tipi di eventi:
A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image,

INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

Rollover

Importantissimo l'evento onMouseOver abbinato ad onMouseOut per creare l'effetto RollOver.

La sintassi è molto semplice:

```
<a href="#" onMouseOver="document.images[0].src='immagine2.gif'"
onMouseOut="document.images[0].src='immagine1.gif'"><img
src="immagine1.gif"></a>
```

dove il cancelletto sostituisce qualsiasi altro link, mentre num è il numero di indice dell'immagine nella pagina HTML.

Qualche anno fa, quando non esistevano programmi come Flash, il rollover era l'effetto grafico più diffuso e certamente quello più adatto a rendere dinamico un sito e a movimentare elementi statici come i menu e le barre di navigazione.

Attenzione! **images[num]** individua il numero di indice della figura nella pagina e si ottiene contando il numero dei tag IMG contando da 0 fino a giungere al tag della figura che ci interessa.

Eventi attivabili dal trascinamento del mouse

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onDragDrop**: evento attivato quando un utente trascina un oggetto sulla finestra del browser o quando rilascia un file sulla stessa;*
>
> ***onMove**: attivato quando un oggetto muove una finestra o un frame;*
>
> ***onDragStart**: evento attivato appena l'utente inizia a trascinare un oggetto;*
>
> ***onDrag**: attivato quando il mouse trascina un oggetto o una selezione di testo nella finestra dello stesso browser o anche di un altro o anche sul Desktop;*
>
> ***onDragEnter**: attivato appena l'utente trascina un oggetto su un obiettivo valido dello stesso o di un altro browser;*
>
> ***onDragOver**: attivato quando l'utente l'utente trascina un oggetto su un obiettivo valido ad ospitarlo, ed è simile all'evento precedente, ma viene attivato dopo quello;*
>
> ***onDragLeave**: attivato quando l'utente trascina un oggetto su un obiettivo adatto per ospitarlo, ma non vi viene rilasciato;*
>
> ***onDragEnd**: attivato quando l'utente rilascia l'oggetto al termine del trascinamento.*
>
> ***onDrop**: attivato quando il mouse si alza il tasto del mouse in seguito ad un'operazione di trascinamento;*

Di questi solo il primo e il secondo sono attivati in Netscape e in Explorer, gli altri funzionano solo in Internet Explorer 5.0 e l'ordine di descrizione è quello di attivazione dell'evento.

Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
OnDragDrop			✓	✓		✓	✓	1.2
OnMove			✓	✓		✓	✓	1.2
OnDrag							✓	DHTML
OnDrop							✓	DHTML

OnDragStart	✓	DHTML
OnDragEnter	✓	DHTML
OnDragLeave	✓	DHTML
OnDragOver	✓	DHTML
OnDragEnd	✓	DHTML

Tag Sensibili

Altro limite è dato, per Netscape e per le vecchie versioni di Explorer, dai tag a cui l'evento può essere associato:

Evento	Tag Associati
OnDragDrop	Questo evento è usato con **window**
OnMove	Questo evento è usato con **window**

Molto più numerosi i tag associati in Explorer 4.0 e successivo a tutti i tipi di eventi:
A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

Eventi legati alla tastiera

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onKeyPress**: evento attivato quando un utente preme e rilascia un tasto o anche quando lo tiene premuto;*
>
> ***onKeyDown**: attivato quando viene premuto il tasto;*
>
> ***onKeyUp**: evento attivato quando un tasto, che era stato premuto, viene rilasciato;*
>
> ***onHelp**: attivato quando il mouse schiacchia il tasto F1;*

L'ultimo evento è stato naturalmente inserito in questo settore, anche se attivabile dall'unico tasto F1.

Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
onKeyPress			✓	✓		✓	✓	1.2
onKeyDown			✓	✓		✓	✓	1.2
onKeyUp			✓	✓		✓	✓	1.2
onHelp							✓	DHTML

Netscape per Unix o Linux non supporta la gestione dei tasti

Tag Sensibili

Altro limite è dato, per Netscape e per le vecchie versioni di Explorer, dai tag a cui l'evento può essere associato:

Evento	Tag Associati
onKeyPress	Questo evento è usato con i tag **<BODY>**, **<IMG>**, **<A>** e **input TEXTAREA**

onKeyDown	Questo evento è usato con i tag **<BODY>**, **<IMG>**, **<A>** e **input TEXTAREA**
onKeyUp	Questo evento è usato con i tag **<BODY>**, **<IMG>**, **<A>** e **input TEXTAREA**

Molto più numerosi i tag associati in Explorer 4.0 e successivo a tutti i tipi di eventi:
A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

Tasti Intercettabili

In Internet Explorer 4.0, l'evento **onkeydown** funziona con i tasti:

> *DELETE, INSERT*
> *Tasti funzione da F1 - F12*
> *Lettere: a - z*
> *Tasti di spostamento: HOME, END, Left Arrow, Right Arrow, Up Arrow, Down Arrow*
> *Numeri: 0 - 9*
> *Simboli: ! @ # $ % ^ & * ( ) _ - + = < > [ ] { } , . / ? \ | ' ` " ~*
> *Tasti di sistema: ESCAPE, SPACE, SHIFT, TAB*

In Internet Explorer 5 funzionano anche i tasti:

> *BACKSPACE*
> *PAGE UP, PAGE DOWN*
> *SHIFT+TAB*

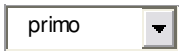
Eventi legati alle modifiche

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onChange**: attivato quando il contenuto di un campo di un form o modulo è modificato o non è più selezionato;*
>
> ***onCellChange**: attivato quando si modifica un elemento in un Database, per questa sua caratteristica ha un uso non propriamente legato a Javascript;*
>
> ***onPropertyChange**: evento attivato quando cambia la proprietà di un elemento;*
>
> ***onReadyStateChange**: evento attivato quando lo stato del caricamento di un elemento cambia, l'evento è utile, ad esempio, per verificare che un elemento sia stato caricato.*

Importante è dire qualcosa in più su onChange: l'evento assomiglia molto ad onBlur, ma verifica anche che l'elemento che lo richiama sia stato modificato. Questo evento, infatti, è attivato quando viene selezionato un altro elemento da una lista o quando si modifica un campo di testo, per cui oltre all'attivazione, occorre anche operare un'azione.

Esempio di onChange



Il codice da mettere nel tag select (fare attenzione all'evento):

```
<select name="select" onChange="if(this.options[1].selected) alert('Hai
selezionato il secondo');
else if(this.options[2].selected) alert('Hai selezionato il terzo') ">
<option value="uno">primo</option>
<option value="due">secondo</option>
<option value="tre">terzo</option>
</select>
```

Gli altri tre eventi sono fortemente legati alle novità apportate da Internet Explorer 5.0 per cui non hanno possibilità di una grande diffusione e vanno utilizzati dopo aver appreso una certa programmazione avanzata.

Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
onChange	✓	✓	✓	✓	✓	✓	✓	1.0
onCellChange							✓	DHTML
onPropertyChange							✓	DHTML
onReadyStateChange							✓	DHTML

Tag Sensibili

Altro limite è dato, per Netscape e per le vecchie versioni di Explorer, dai tag a cui l'evento può essere associato:

Evento	Tag Associati
onChange	Questo evento è usato con i tag **<SELECT>** e **<TEXTAREA>**, e il tag **<INPUT>** di tipo TEXT. Con la versione Javascript 1.1 si aggiunge anche il tag **<FILEUPLOAD>**, ma non funziona con Internet Explorer.
OnCellChange	Questo evento è usato con i tag **<APPLET>**, **<BDO>**, **<OBJECT>**.

Per alcuni eventi di Internet Explorer 5.0 i tag associati sono numerosi, così per **onPropertyChange** e **onReadyStateChange**:

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

Eventi legati al "fuoco"

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onFocus**: Questo handler è l'opposto di* **onBlur** *per cui si attiva quando l'utente entra in un campo;*
>
> ***onBlur**: attivato quando il puntatore del mouse o il cursore esce dalla finestra corrente utilizzando il mouse o il carattere TAB. Applicato ai moduli, invece, tale handler si avvia se si esce dal campo il cui tag contiene il controllo;*
>
> ***onSelect**: attivabile quando si seleziona del testo all'interno di una casella di testo sia col mouse sia tenendo premuto SHIFT e selezionando con i tasti Freccia;*
>
> ***onSelectStart**: si attiva quando si inizia a selezionare un evento;*
>
> ***onbeforeEditFocus**: si attiva con un doppio click o con un clicK su un oggetto che ha già la selezione, quando questo è in DesignMode;*
>
> ***onLoseCapture**: si attiva quando un oggetto perde la cattura del mouse.*

Gli ultimi tre eventi sono particolarità di Internet Explorer 5.0 e richiedono ulteriori conoscenze per poter essere adoperati.

Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
OnFocus	✓	✓	✓	✓	✓	✓	✓	1.0
OnBlur	✓	✓	✓	✓	✓	✓	✓	1.0
OnSelect	✓	✓	✓	✓	✓	✓	✓	1.0
OnSelectStart							✓	DHTML
OnLoseCapture							✓	DHTML
OnbeforeEditFocus							✓	DHTML

Tag sensibili

Altro limite è dato, per Netscape e per le vecchie versioni di Explorer, dai tag a cui l'evento può essere associato:

Evento	Tag Associati
OnFocus	Questo evento è usato con i tag **<SELECT>** e **<TEXTAREA>** e con il tag **<INPUT>** di tipo TEXT. Con Javascript 1.1, cioè da Netscape Navigator 3, questo handler è stato associato anche con i tag **<BODY>** e **<FRAMESET>** e con il resto dei tag di form come **<BUTTON>**, **<CHECKBOX>**, **<FILEUPLOAD>**, **<PASSWORD>**, **<RADIO>**, **<RESET>**, **<SUBMIT>**. In Javascript 1.2 si aggiunge anche il tag **<LAYER>**.
OnBlur	Questo evento è usato con i tag **<SELECT>** e **<TEXTAREA>** e con il tag **<INPUT>** di tipo TEXT. Con Javascript 1.1, cioè da Netscape Navigator 3, questo handler è stato associato anche con i tag **<BODY>** e **<FRAMESET>** e con il resto dei tag di form come **<BUTTON>**, **<CHECKBOX>**, **<FILEUPLOAD>**, **<PASSWORD>**, **<RADIO>**, **<RESET>**, **<SUBMIT>**. In Javascript 1.2 si aggiunge anche il tag **<LAYER>**.
OnSelect	Questo evento è usato con il tag **<TEXTAREA>** e **<INPUT>** di tipo TEXT, anche per Internet Explorer.

In qualche piattaforma gli eventi **onBlur** e **onFocus** non funzionano bene con il tag ***<FRAMESET>***

Molto più numerosi i tag associati in Explorer 4.0 e successivo per gli eventi onBlur, onFocus, SelectStart e onLoseCapture:

A, ADDRESS, APPLET, AREA, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FONT, FORM, HR, I, IMG, INPUT type=button, INPUT type=checkbox, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=radio, INPUT type=reset, INPUT type=submit, INPUT type=text, KBD, LABEL, LEGEND, LI, LISTING, MAP, MARQUEE, MENU, NEXTID, NOBR, OBJECT, OL, P, PLAINTEXT, PRE, RT, RUBY, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, XMP

Eventi da caricamento degli oggetti

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onLoad****: Questo handler funziona nel caricamento di oggetti, per lo più finestre e immagini;*
>
> ***onUnload****: è l'opposto del precedente e funziona quando si lascia una finestra per caricarne un'altra o anche per ricaricare la stessa (col tasto refresh);*
>
> ***onAbort****: funziona quando l'utente ferma il caricamento di un oggetto cliccando su un altro link o premendo il tasto stop del browser;*
>
> ***onError****: si attiva quando il caricamento di un oggetto causa un errore, ma solo se questo è dovuto ad un errore di sintassi del codice e non del browser così funziona su un link errato di un'immagine della pagina, ma non su un link errato di caricamento di una pagina intera;*
>
> ***onBeforeUnload****: questo handler funziona allo stesso modo di onUnload ma si carica in un momento prima;*
>
> ***onStop****: questo handler funziona quando si ferma il caricamento della pagina con il tasto stop del browser e dovrebbe funzionare anche allo stesso modo di onUnload caricandosi prima di questo ma dopo onBeforeUnload.*

Gli ultimi due eventi sono particolarità di Internet Explorer 5.0 e richiedono ulteriori conoscenze per poter essere adoperati.

Per le versioni di Javascript ecco il quadro riepilogativo:

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
OnLoad	✓	✓	✓	✓	✓	✓	✓	1.0
OnUnload	✓	✓	✓	✓	✓	✓	✓	1.0
OnAbort		✓	✓	✓		✓	✓	1.1
OnError		✓	✓	✓		✓	✓	1.1
OnBeforeUnload							✓	DHTML
onStop							✓	DHTML

Tag sensibili

Altro limite è dato dai tag a cui l'evento può essere associato:

	Tag associati in Netscape e Jscript
onLoad	Questo evento è usato con i tag **<BODY>** e **<FRAMESET>** e da Javascript 1.1anche con **<IMG>**

	mentre in Explorer occorre aggiungere anche i tag **<SCRIPT>**, **<LINK>**, **<EMBED>**, **<APPLET>**. In Javascript 1.2 in Netscape si aggiunge anche il tag **<LAYER>**.
onUnload	Questo evento è usato con i tag **<BODY>** e **<FRAMESET>** anche in Internet Explorer.
onAbort	Questo evento è usato solo con il tag**<IMG>** anche in Internet Explorer.
onError	Questo evento è usato solo con il tag**<IMG>** e con Window mentre in Internet Explorer anche con **<OBJECT>** e **<STYLE>**.
onBeforeUnload	Questo evento è usato con i tag **<BODY>** anche in Internet Explorer.
onStop	Questo evento è usato con i tag **<BODY>** anche in Internet Explorer.

onLoad

Questo evento se ben utilizzato potrebbe simulare un effetto streaming mediante caricamento di gif o jpeg. Immaginiamo di avere a disposizione una serie di jpeg da animare. Con questo evento possiamo sincronizzare la visualizzazione delle varie figure alla fine del caricamento della precedente mediante uno script del genere:

```
<script>
var whichImage=0;
var maxImages=5;
function changeAnimation(theImage) {
++whichImage;
if (whichImage <= maxImages) {
var imageName="image" + whichImage + ".gif";
theImage.src=imageName;
}
else {
whichImage=-1;
return;
}
}
</script>
```

applicato all'elemento

```
<img name="changingAnimation" src="image0.gif" border=0 align="top"
onLoad="changeAnimation(this)">
```

onError

Una particolare menzione merita l'evento onError per la sua grande utilità ma per lo scarso utilizzo che se ne fa. Questo evento può:

1. *sopprimere i messaggi di errori che il debugger mostrerebbe (messaggi discreti in Netscape, ma antiestetici in Explorer) il tutto mediante il codice di esempio <img src="corrupt.gif" **onError="null"**>*
2. *se il codice è espresso nella sezione HEAD diventa attivo per tutta la finestra e per tutti gli eventi di errore attivati dagli oggetti ivi contenuti **<script>window.onerror=null</script>***
   tuttavia un evento onError associato alla singola immagine o oggetto cattura l'evento per questa immagine o oggetto a discapito di quello generale

3. *l'evento può fare di più: può indicare il tipo di errore, la riga e l'URL che lo hanno causato e indicarlo in una finestra o memorizzarlo in un array. Di seguito indico l'istruttivo esempio mostrato sul Reference di Netscape che serve a costruire un debugger personalizzato. Nello script ci sono degli errori e cliccando sul primo bottone e poi sul secondo si possono individuare:*

```
<script>
window.onerror = myOnError
msgArray = new Array()
urlArray = new Array()
lnoArray = new Array()
function myOnError(msg, url, lno) {
msgArray[msgArray.length] = msg
urlArray[urlArray.length] = url
lnoArray[lnoArray.length] = lno
return true
}
function displayErrors() {
win2=window.open('','window2','scrollbars=yes')
win2.document.writeln('<B>Error Report</B> <P>')
for (var i=0; i < msgArray.length; i++) {
win2.document.writeln('<B>Error in file:</B> ' + urlArray[i] + '<BR> ')
win2.document.writeln('<B>Line number:</B> ' + lnoArray[i] + '<BR>')
win2.document.writeln('<B>Message:</B> ' + msgArray[i] + '<P>')
}
win2.document.close()
}
</script> </p>
<form>
<input type="button" value="Questo bottone ha un errore di sintassi"
onClick="alert('unterminated string)" name="button">
<p> <input type="button" value="Visualizza l'errore"
onClick="displayErrors()" name="button2">
</form>
```

Eventi dal movimento delle finestre

A questo gruppo si possono ricondurre due soli eventi:

***onResize**: Questo handler si attiva quando l'utente rimpicciolisce o ingrandisce una finestra o un frame o, in caso particolare per Explorer, un oggetto a cui siano stati fissati l'altezza e la larghezza o anche la posizione, come ad esempio un layer;*

***onScroll**: attivato quando si effettua lo scrolling della pagina sia col mouse con i tasti PGUP e PGDOWN o anche con il metodo doScroll.*

Per le versioni di Javascript ecco il quadro riepilogativo in rosso sono indicate le modifiche apportate nelle diverse versioni dei browser):

Evento	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
onResize			✓	✓		✓	✓	1.2
onScroll							✓	DHTML

Tag sensibili

Altro limite è dato dai tag a cui l'evento può essere associato:

Evento	Tag associati in Netscape e JScript
OnResize	Questo gestore è usato con il tag <DOCUMENT>.
OnScroll	Questo gestore è usato solo

	in Explorer e con i tag <APPLET>, <BDO>, <BODY>, <DIV>, <EMBED>, <MAP>, <MARQUEE>, <OBJECT>, <SELECT>, <TABLE>, <TEXTAREA>.

Molto più numerosi i tag associati in Explorer 4.0 e successivo per gli eventi onResize:
A, ADDRESS, APPLET, B, BIG, BLOCKQUOTE, BUTTON, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, EMBED, FIELDSET, FORM, FRAME, Hn, HR, I, IMG, INPUT type=button, INPUT type=file, INPUT type=image, INPUT type=password, INPUT type=reset, INPUT type=submit, INPUT type=text, ISINDEX, KBD, LABEL, LEGEND, LI, LISTING, MARQUEE, MENU, OBJECT, OL, P, PRE, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TEXTAREA, TT, U, UL, VAR, window, XMP

Eventi da tasti particolari

A questo gruppo si possono ricondurre i seguenti eventi:

> ***onSubmit**: Questo handler è attivato dal click su tasto di Invio di un form;*
> ***onReset**: questo handler è attivato dal click su tasto di Annulla di un form.*

sono due eventi particolari e legati a due oggetti specifici che non hanno subito sostanziali modifiche rispetto al passato.
Per le versioni di Javascript ecco il quadro riepilogativo in rosso sono indicate le modifiche apportate nelle diverse versioni dei browser):

	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0	Versione Javascript
onSubmit	✓	✓	✓	✓	✓	✓	✓	1.0
onReset		✓	✓	✓	✓	✓	✓	1.1

Tag sensibili

Altro limite è dato, per Netscape e per le vecchie versioni di Explorer, dai tag a cui l'evento può essere associato:

Evento	Tag associati in Netscape

	e J S c r i p t
onSubmit	Questo gestore è usato col tag <FORM> sia in Netscape che in Explorer.
onReset	Questo gestore è usato col tag <FORM> sia in Netscape che in Explorer.

5. Variabili, espressioni e operatori

Valori letterali

I valori letterali sono quantità **esplicite** per cui non vanno dichiarati e servono a fornire informazioni ad espressioni o a funzioni (es.: il numero 45 è esplicitamente considerato numerico).

In Javascript esistono diversi tipi di valori che sono:

> ***numerici*** *i quali si dividono in:*
>
> ***Interi***
>
> ***decimali*** *o, con linguaggio tecnico, a* ***virgola mobile*** *(in notazione scientifica o in quella standard) (Valori non precisi in Netscape 2.0)*
>
> ***logici*** *o booleani: che possono assumere soltanto due stati (vero o falso);*
>
> ***stringhe*** *(o sequenza di caratteri)*
>
> ***valore nullo***
>
> ***caratteri speciali*** *(es.: \f per l'avanzamento pagina)*
>
> ***oggetti***

Il letterale **intero** viene fornito in tre varianti: *decimale*, *esadecimale* e *ottale* secondo la base con cui viene rappresentato. Per rappresentare un **ottale lo si fa precedere da uno 0**, per rappresentare **un esadecimale lo si fa precedere da 0x**.

Il valore nullo si ha quando le variabili sono indefinite, cioè quando non vi si assegna nessun valore all'atto della dichiarazione.

Importante è sottolineare che ad una variabile può essere assegnato anche un oggetto e si può anche creare un **oggetto generico** tramite l'enunciato:

```
nomeoggetto=new Object();
```

Caratteri speciali

Tra le stringhe occorre indicare i **caratteri speciali** che costituiscono un mezzo per formattare il testo:

Descrizione	Designazione standard	Sequenza
Continuazione	<newline>	\
Nuova riga	NL (LF)	\n
Tab orizzontale	HT	\t
Backspace	BS	\b
Ritorno carrello	CR	\r
Avanzamento pagina	FF	\f
Backslash	\	\\
Virgolette singole	'	\'
Virgolette doppie	"	\"

```
Esempio di codice speciale:
<script>
document.write("questa è una riga \n e questa è un'altra");
alert("questa è una riga \n e questa è un'altra");
</script>

Si noti il diverso effetto sull'alert e sulla finestra del browser.
```

Capita spesso, infatti, di voler indicare all'elaboratore di dover scrivere a video una stringa suddivisa in più righe, in questo caso l'indicazione è fatta con una sequenza di caratteri speciali. Un carattere speciale, ad esempio, è proprio l'apice che spesso sostituisce l'apostrofo perciò, in quest'ultimo utilizzo, va indicato

come carattere speciale facendolo precedere da una barra rovescia. Il computer, in poche parole, quando trova queste particolari sequenze sa di dover interpretare diversamente tali caratteri, quasi come se fossero dei comandi.

Escape ed Unescape

Le stringhe possono essere interpretate sia come sequenze di caratteri o come righe di comando. Un campo di applicazione, ad esempio, è nel trattamento dei dati inviati al server, o anche al client, con il metodo GET. In questo caso lo script riceve i dati come una stringa attaccata (appended) all'URL originale in cui alcune sequenze di caratteri indicano vanno interpretati come caratteri particolari come lo spazio vuoto o il separatore tra variabile e valore. Ad esempio se effettuate una ricerca su Altavista vedrete aggiungersi all'URL una serie di caratteri che segue un punto interrogativo, ebbene, quei caratteri opportunamente trattati, formano una stringa che interroga il motore di ricerca. I caratteri scritti in quella notazione si chiamano **sequenze escape** e utilizzano la codifica URL in cui:

> *le coppie nome=valore sono separate da una &;*
> *gli spazi sono sostituiti da +;*
> *i caratteri alfanumerici sostituiti dall'equivalente esadecimale preceduto da %.*

Parte del lavoro del programma CGI che funziona su server è proprio quello di decifrare la stringa di input in caratteri ISO-Latin-1, e non le sequenze Unicode, ma Javascript può elaborare queste stringhe anche all'interno dei propri script mediante i comandi **escape**:

```
escape("Ecco qui")="Ecco%20qui"
```

o con **unescape** per rendere la situazione contraria:

```
unescape("Ecco%20qui")="Ecco qui"
```

Opportunamente utilizzati, queste funzioni si rivelano utilissime e offrono all'utente un grande grado di interattività.

Dichiarazioni variabili

Le variabili sono dei nomi simbolici che servono ad inviduare delle locazioni di memoria in cui possono essere posti dei valori. Metaforicamente bisogna considerare queste locazioni come delle scatole con un nome in cui si possono inserire questi valori.

In Javascript le variabili sono create con la dichiarazione **var**, attribuendole anche nessun valore, ma anche semplicemente all'*atto di assegnazione di un valore* (ad esempio x=15 crea automaticamente una variabile numerica). La dichiarazione **var** su più variabili va ripetuta per ognuna oppure va fatta con un'interruzione di linea:

```
var miocarattere,
miavariabile;
```

Per le variabili che si dichiarano e si inizializzano senza assegnare un particolare valore si attribuisce il valore **null**. Questo valore può apparire poco importante, ma diventa essenziale se si vuole verificare il caricamento in memoria di una variabile. Ad esempio se si dichiara una variabile

```
var Verifica=null;
```

se si utilizza il comando

```
if(Verifica != null) alert("Non sono stata utilizzata");
```

si può verificare se questa variabile ha avuto un utilizzo o se vuota. Un utilizzo molto utile può essere quello applicato alle finestre popup per verificare se sono rimaste aperte.

Un altro valore che viene assegnato alle variabili quando queste sono state dichiarate ma senza assegnamento è **undefined**. Questo valore è stato introdotto con ***Javascript 1.3*** è accettato nelle norme ECMA.

In tal modo la dichiarazione:

```
var Verifica;
```

assegna alla variabile nessun valore per cui una verifica su di essa restituisce il valore undefined e tale valore si può verificare

```
Esempio:
if(Verifica == undefined))
.....
```

Chi ha già programmato con altri linguaggi noterà con piacere l'estrema flessibilità di Javascript nel trattare le variabili, ma la dichiarazione non è necessaria perché, per la vita breve di queste variabili, e inutile risparmiare memoria. Se si ha familiarità con la programmazione o se si vuole mantenere ordine tra le variabili, allora conviene dichiararle anche in Javascript, possibilmente all'inizio del blocco di codice.

Dove dichiarare le variabili? Dipende dall'utilizzo e dalla distinzione che c'è tra **variabili globali** e **variabili locali**. La distinzione non è poca cosa, anzi è alla base della programmazione orientata agli oggetti:

> *le variabili **globali** hanno valore per tutto il documento HTML e vanno dichiarate all'inizio dello script e fuori da ogni funzione: il posto preferibile è nei tag <SCRIPT> della sezione <HEAD> in modo tale da creare i contenitori dei valori prima di ogni loro utilizzo;*
>
> *le variabili **locali** hanno valore solo all'interno della funzione in cui sono dichiarate, cioè all'interno del blocco di codice compreso tra function(){ e la chiusura della parentesi } e vanno dichiarate entro questi blocchi.*

Da ciò si ricava che le *variabili dichiarate all'interno di parentesi graffe possono essere utilizzate solo in quel blocco* perchè sono **variabili locali**.

I due differenti tipi sono generati da esigenze diverse in quanto le variabili locali hanno una vita brevissima ed esistono finché opera la funzione, alla chiusura della parentesi vengono distrutte liberando memoria, mentre le variabili globali sono dei contenitori che durano per tutta la durata della pagina e servono a veicolare dei valori tra le funzioni e tra gli script, ma anche tra le varie pagine o al server. A questo punto vi chiederete perché utilizzare variabili locali, in quanto quelle globali sono più comode, ma la necessità del loro utilizzo sta nella modularità: le variabili locali rendono uno script riutilizzabile anche in altre pagine HTML, soprattutto se salvato in un file esterno con estensione js.

Identificatori

I nomi dei dati sono chiamati *identificatori* e devono sottostare ad alcune regole:

> - *possono contenere solo lettere, numeri e trattino di sottolineatura, per cui sono esclusi gli spazi bianchi;*
> - *il primo carattere deve essere sempre una lettera. E' utilizzabile come primo carattere anche il trattino di sottolineatura, ma il compilatore tratta quel nome in modo particolare per cui se ne sconsiglia l'uso;*
> - *Javascript è case sensitive per cui tratta diversamente le lettere in maiuscolo e in minuscolo, per tale motivo è convenzione utilizzare l'iniziale maiuscola per i nomi di costanti e quella minuscola per le variabili;*
> - *non si possono utilizzare i nomi che rientrano nelle parole chiave.*
> - *L'uso diffuso è di utilizzare nomi lunghi per identificare meglio il dato, adoperando queste convenzioni:*

- *adoperare il trattino di sottolineatura per definire meglio il dato, così il nome tasso_interesse identifica più di quanto possa fare il semplice nome x;*
- *accanto all'utilizzo del trattino di sottolineatura, si usa anche la notazione a cammello per cui si rende maiuscola una lettera all'interno del nome di una variabile, proprio per identificarla meglio (ad esempio TassoInteresse).*

Parole chiave

In Javascript ci sono delle parole chiave che non si possono utilizzare come identificatori di dati, eccone l'elenco:

Parole Chiave del linguaggio Javascript				
abstract	Do	if	package	Throw
boolean	Double	implements	private	Throws
break	else	import	protected	Transient
byte	extends	in	public	True
case	false	instanceof	return	Try
catch	final	int	short	Var
char	finally	interface	static	Void
class	float	long	super	While
const	for	native	switch	With
continue	function	new	synchronized	
default	goto	null	this	

Tipi di variabili

I linguaggi tradizionali (come il C) utilizzano solo variabili che siano state precedentemente dichiarate per due motivi:

ottimizzare l'uso della memoria;
aumentare l'affidabilità.

Javascript, invece, utilizza un **controllo di tipo lasco** per cui non esiste una sezione di dichiarazione di variabili e non c'è bisogno di farlo, ma **automaticamente viene assegnato il tipo in base alla dichiarazione**. Ad esempio *Prova="testo"* e *Prova=59* sono due dichiarazioni pienamente valide, ma nel primo caso *Prova* è considerata oggetto *string*, nel secondo la stessa è considerata come valore numerico.

Se valori diversi vengono concatenati prevale il valore string, per cui un numero concatenato ad una stringa produce una stringa, tuttavia se la stringa è un numero, il risultato sarà un numero (es.: temval=365+"10" darà 375).

Talora, però, questo controllo non è sempre preciso (quando, ad esempio, si moltiplica una variabile stringa con un numero) e poichè non esiste un compilatore per controllare gli errori, lo script non genera i risultati voluti. In questo caso più che dichiarare basta convertire il risultato con una funzione di conversione.

Per convertire i valori basta utilizzare:

- *gli apici per convertire un valore numerico in stringa o la somma di un numero con uno spazio;*
- *metodo **String()** (da Javascript 1.2) per convertire in stringa o con il costruttore **String()** (es.: miastringa=String(num));*
- *le funzioni eval(), parseInt() e parseFloat() per convertire un valore stringa in numerico;*
- *i valori logici sono ottenuti da valori numerici o stringa ponendo questi uguale a true o false.*

Le funzioni *eval()*, *parseInt()* e *parseFloat()*funzionano in questo modo:

> ***parseInt*** *cerca un intero all'inizio di una stringa, scartando le stringhe, e lo visualizza, ignorando le parti decimali e l'eventuale virgola (es.: parseInt("39 gradi")=39), un secondo parametro, facoltativo, è la base numerica (es.: parseInt(text,16) per cercare numeri esadecimali);*
>
> ***parseFloat*** *opera allo stesso modo, ma conserva l'eventuale virgola presente e il segno.*
>
> ***eval*** *è abbastanza complessa e cerca qualsiasi espressione Javascript valida per trasformarla in numerico.*
>
> *x=10; y=20; z=30; eval("x+y+z+900")=960*

In qualsiasi momento, comunque, si può verificare il tipo della variabile tramite l'operatore ***typeof*** (es.: *typeof 69* restituisce "number") il quale è stato introdotto con Javascript 1.1. I valori restituiti sono: string, boolean, number, function.

```
Ad esempio immaginiamo di avere le seguenti variabili:
var prova=new Function()
var numero=1
var testo="Salve"
con l'operatore typeof si avrà
typeof(prova) restituirà object
typeof(numero) restituirà number
typeof(testo) restituirà string
```

Lifetime dei dati

Nelle prime versioni di Javascript (Netscape 2.0) i dati si conservavano passando da documento a documento o da frame a frame, tuttavia ciò poteva minare la sicurezza della pagina perchè un documento poteva aprire in una cornice, spesso invisibile, l'URL dei file e leggere il contenuto del disco o poteva prendere menzione dei siti visitati.

Per tale motivi si ridusse il **lifetime** delle variabili che durano, perciò, solo finchè la finestra che le ha create rimane aperta.

Già la versione Netscape 2.02 aveva risolto il problema, ma allo stesso modo aveva privato i programmatori di un potente strumento per gestire documenti e frame.

Per tenere conto di tali variabili nel passare da una pagina ad un'altra le uniche soluzioni sono:

> - *utilizzo di cookie nel lato client;*
> - *passaggio tramite URL;*
> - *uso di campi nascosti;*
> - *passaggio utilizzando i frame e la forma* ***parent.frameName.variabile*** *o* ***parent.frameName.nomefunzione****;*
> - *passaggio utilizzando le finestre e la forma* ***nomefinestra.variabile*** *o* ***nomefinestra.nomefunzione****.*

L'uso dei cookie lo tralasciamo perché abbastanza complesso e poco indicato nei casi di variabili che non memorizzano dei form. Il passaggio tra frame o tra finestre è molto utilizzato nella pratica, ma è anche molto comodo il passaggio tramite URL, anche se sono state riscontrate alcune difficoltà in Internet Explorer 4.0.

Passaggio dei dati

Invece di spiegare come passare i dati, affrontiamo una situazione concreta.

Immaginiamo di avere un form con campi di tipo button e di voler indicare ad una finestra o alla pagina successiva quale tasto è stato premuto.

```
Inseriamo i tasti e poi dichiariamo nella sezione head una variabile
globale Tasto:
<script language="javascript"><!--
var tasto=null;
//--></script>
```

ad ogni tasto associamo l'evento:

```
onclick="tasto=valore"
```

dove valore sarà uguale a 1 nel primo tasto, uguale a 2 nel secondo e uguale a 3 nel terzo.

Immaginiamo due casi:

1. *di voler trasmettere i dati ad una finestra che viene aperta;*
2. *di trasmettere i dati ad una pagina linkata.*

Nel primo caso abbiamo solo un problema di trasferimento di dati perché la finestra madre, che contiene il valore da passare, conserva sempre questo valore finché non viene chiusa per cui basta inserire una riga di codice del tipo:

```
window.opener.tasto
```

Nel secondo caso, invece, possiamo utilizzare una finestra con dei frame (anche una finestra con un unico frame va bene) e conserviamo la variabile nella finestra top (quella che contiene i frame) e la richiamiamo dalle altre finestre con le proprietà **parent.nomevariabile** oppure **top.nomevariabile**.

I frame, però, sono spesso sconsigliati e restano scarsamente adoperati su siti di un certo livello dove la possibilità di trasferire dati è quella di utilizzare il cookie o l'URL.

In quest'ultimo caso passiamo il valore alla pagina successiva attraverso URL e scriviamo il seguente codice HTML ad un link:

```
<script>
function apri()
{
var link="finestra1.html?"+tasto
window.location=link
}
</script>

<a href="#" onClick="apri()">Link</a>
```

si noti come ci sia il valore indicato dopo il punto interrogativo. Questo tipo di codifica è quella utilizzata dal Perl e da altri linguaggi che operano sul server, ma qui può essere adattata ad HTML (resta solo un'annotazione in quanto questo passaggio non sembra funzionare con IE4).

Il problema ritorna quando bisogna recuperare il valore nella pagina successiva poiché il browser legge l'URL come una sola stringa. In tal caso bisogna utilizzare le proprietà e i metodi dell'oggetto String. Poniamo la locazione in una variabile:

```
var ausilio=String(this.location);
var tasto=ausilio.charAt(ausilio.lastIndexOf("?")+1);
```

nella prima riga viene preso l'URL e trasformato in stringa (ricordate i tipi di variabili?), nella seconda si legge il carattere in posizione successiva a quella dell'ultimo punto interrogativo della stringa.

Array

Gli array sono liste numerate di oggetti, che in Javascript possono anche essere di tipo diverso, che possono essere considerate come un'unità.

Javascript non supporta le matrici come variabili, ma come oggetti per cui si crea una matrice **solo se si dichiara** e tale dichiarazione va fatta come **istanza dell'oggetto** Array:

```
nomematrice=new Array(num)
```

Gli array, in quanto oggetti, verranno trattati in seguito e qui ne diamo solo un accenno anche perché il programmatore tradizionale per abitudine li cercherà tra le variabili.

Per accedere ad un elemento della matrice si utilizza l'indice che indica la posizione dell'elemento all'interno della stessa. Tale posizione si inizia a conteggiare da 0 e non da 1.

```
Ad esempio ecco una matrice
animals=new Object[]
animals[0]="rana";
animals[1]="anatra";
animals[2]="asino";
animals[3]="orso";
animals[4]="gallina";
```

per cui il primo elemento è "rana" ed ha indice 0.

L'uso delle matrici è importantissimo in Javascript perché questo linguaggio indicizza parecchi elementi come le immagini o come il link per cui si possono richiamare con l'oggetto associato e con l'indice.

In Javascript mancano le matrici bidimensionali e tridimensionali, ma si possono facilmente riprodurre come matrici di matrici.

Operatori

Gli operatori si dividono in:

- *operatori di assegnamento;*
- *operatori aritmetici;*
- *operatori logici;*
- *operatori sui bit, adoperati in genere solo per generare colori;*
- *operatori su stringhe.*

possono essere **unari** o **binari**: i primi richiedono un unico operando, i secondi due.

Il primo operatore che occorre conoscere è l'**operatore di assegnamento** *il cui segno è l'uguale* (=), tuttavia il suo uso in Javascript è simile al C ed è diverso da quello della matematica perchè **serve ad**

assegnare un valore e non ad effettuare confronti, per i quali esiste un altro operatore (= =). Adoperando l'operatore di assegnamento si crea come una fotocopia del valore dell'espressione a destra dell'operatore sul dato a sinistra dell'operatore.

Gli **operatori aritmetici** sono binari e unari, gli operatori unari **modificano il valore a cui sono applicati** e sono:

Operatore	Simbolo	Azione
Incremento	++	Incrementa di un'unità
Decremento	--	Decrementa di un'unità
Meno unario	-	Rende negativo un numero

Gli operatori unari matematici *possono essere posti prima* (**prefissi**) o *dopo* (**posfissi**) dell'operando e il loro valore varia secondo questa posizione in quanto l'*operatore prefisso modifica l'operando prima di utilizzarne il valore*, mentre l'*operatore posfisso modifica l'operando dopo averne utilizzato il valore*, un esempio può chiarire:

x=10; y=x++; per cui y=10 e x=11	x=10; y=++x; per cui y=11 e x=11

Gli operatori binari matematici **non cambiano il valore degli operandi, ma memorizzano il risultato in un terzo operando**, comprendono le principali operazioni aritmetiche:

Operatore	Simbolo	Azione
Addizione	+	Somma due operandi
Sottrazione	-	Sottrae il secondo operando dal primo
Moltiplicazione	*	Moltiplica i due operandi
Divisione	/	Divide il primo operando per il secondo
Resto (modulo)	%	Fornisce il resto della divisione tra due operandi

non è possibile, però, utilizzare l'operatore di calcolo del modulo quando gli operandi sono in virgola mobile. L'operatore di divisione, invece, applicato a variabili di tipo intero produce un risultato troncato della parte decimale. Se si divide il modulo per zero, si avrà un'eccezione nell'esecuzione: se l'operazione eccede il limite inferiore (underflow) il risultato sarà zero, se eccede il limite superiore (overflow) si avrà un'approsimazione.

L'operatore di assegnamento può anche essere **compattato**, cioè **abbinato ad un operatore aritmetico**. In genere quando le espressioni sono del tipo:

```
variabile=variabile operatore espressione
```

possono essere cambiati in:

```
variabile operatore = espressione
```

cioè si ha la seguente tabella:

Scrittura compatta	Scrittura equivalente
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Con **operatore relazionale** *si intende la relazione che un valore ha rispetto ad un altro*. Essi si basano sul concetto di verità o di falsità e comunque operano con solo due stati diversi (0/1, acceso/spento, vero/falso).

Gli operatori relazionali sono:

Operatore	Azione
>	Maggiore di
>=	Maggiore o uguale
<	Minore di
<=	Minore o uguale
==	Uguale
!=	Diverso

Gli operatori relazionali, come detto, producono 1 se la relazione è vera, e 0 se la relazione è falsa. E' importante capire che l'output è dato solo da due valori per *evitare confusioni tra l'operatore di assegnamento e quello di uguaglianza*.

Gli operatori logici assomigliano tantissimo a quelli relazionali in quanto danno in output solo due valori, 0 se l'espressione logica è vera, 1 se l'espressione è falsa.

Gli **operatori logici** sono (il NOT è un *operatore unario*):

Operatore	Simbolo	Significato
AND	&	AND logico
OR	|	OR logico
AND	&&	AND valutazione
OR	||	OR valutazione
XOR	^	OR esclusivo
NOT	!	Negazione

La negazione (!) equivale al complemento a uno (~). Gli operatori &, | valutano entrambi i lati dell'argomento decidendo il risultato. Gli operatori && e || possono essere impiegati per evitare la valutazione degli operandi di destra se la valutazione non è necessaria. Le relative tabelle di verità sono:

X	Y	X&&Y	X||Y	X^Y	!X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	1	1	0	0
1	1	0	0	1	0

Questi operatori sono utilissimi per raggruppare più espressioni che altrimenti sarebbero utilizzate con *if* annidati.

Esistono anche operatori di assegnazione combinati con operatori logici (&=, |=, ^=).

Le *operazioni bit a bit si occupano di controllare e impostare i bit*. Occorre ricordare che il risultato di queste operazioni non sempre danno 0 oppure 1, come per gli operatori logici.

Gli operatori sono:

Operatore	Azione
&	AND
|	OR
^	XOR
~	Complemento a uno
>>	Shift a destra
<<	Shift a sinistra
>>>	Shift a destra con riempimento di zero

Le tabelle di verità sono le stesse degli operatori logici.

Si può pensare all'operatore *AND bit a bit come un modo per cancellare un bit portandolo a 0*, l'operatore OR bit a bit è il contrario di AND e *può essere utilizzato per impostare ad 1 un bit*, mentre l'operatore XOR *può essere utilizzato per impostare ad 1 un bit se i bit confrontati sono diversi*.

Gli operatori di shift scorrono tutti i bit di una variabile verso destra o verso sinistra di un numero di posizioni specificate e i nuovi bit che si creano vengono impostati ad 1, mentre i bit che escono sono persi (tranne quello del segno).

Le operazioni di shift sono utili per decodificare l'input di un dispositivo esterno come un convertitore analogico/digitale e consentono anche operazioni velocissime tra interi in quanto *uno scorrimento a destra divide un numero per due e uno scorrimento a sinistra lo moltiplica per due*.

L'operatore di complemento a uno, invece, inverte lo stato dei bit, per cui ogni 1 sarà tramutato in 0 e viceversa. Naturalmente due operazioni di complemento sullo stesso numero producono il numero originale.

Anche le **stringhe** hanno i propri operatori:

Operatore	Nome	Sintassi
+	Addizione	stringa=stringaA+stringaB
+=	Accodamento	stringa=stringaA+="grassa"
==	Uguaglianza	if (stringaA==stringaB)
!=	Disuguaglianza	if (stringaA!=stringaB)

Precedenza degli operatori

Il riepilogo della precedenza degli operatori é:

Priorità	Aritmetici	Relazionali	Logici	Bit a bit	Altri
Alta					**()(funzione) [](vettore) .**
	++ --		**!**	**~**	
	*** / %**				
	+ -				
				<< >>	
		< <= > >=			
		!= ==			
				&	
				^	
				|	
			&&		
			||		
					?:
				<<= >>=	**= += -= *= /=**
				^= |= &=	**%=**
Bassa					**,**

Se gli operatori, però, hanno stessa precedenza, vengono eseguiti in un ordine che va da sinistra a destra, ma è possibile cambiare quest'ordine mediante l'utilizzo di parentesi tonde, per cui saranno calcolate prima le espressioni nelle parentesi più interne. Tuttavia è utile adoperare le parentesi per maggiore chiarezza anche quando la precedenza le rende inutili.

Espressioni al volo

Le espressioni possono essere calcolate anche all'interno di tag HTML permettendo di costruire valori "al volo": la sintassi è:

&{espressione};

un esempio é:

```
<script>
<!-
var tabWidth=50;
//-->
</script>
<table width="&{tabWidth};%">
```

Tali espressioni sono importanti se sono legate ad un valore che varia secondo opportune modalità.

Tuttavia ho potuto notare che Internet Explorer non gestisce questo tipo di espressione che, al contrario, potrebbe rivelarsi utilissimo per i programmatori.

6. Funzioni

Funzioni standard

Nei capitoli precedenti abbiamo utilizzato delle semplici funzioni per la manipolazione di numeri o stringhe ma non abbiamo ancora presentato una loro definizione.

Le funzioni sono istruzioni raggruppate per effettuare un'operazione più complessa che eventualmente è ripetuta nei vari punti del programma. In questo modo l'utente può risparmiarsi di scrivere ripetutamente lo stesso codice in quanto il gruppo di istruzioni sarà interpretato come una singola unità.

L'utilizzo delle funzioni si esegue in due passi: la definizione della funzione e la sua chiamata. La definizione è scritta una sola volta ma il numero di chiamate non è limitato.

La sintassi per la definizione è la seguente:

```
function nome(parametri)  {
   istruzioni;
}
```

Dove *function* è la parola chiave che segnala all'interprete che segue la definizione di una funzione, il nome è scelto dall'utente (ovviamente deve rispettare gli stessi vincoli che hanno i nomi delle variabili) ed i parametri sono i dati in ingresso alla funzione. Le funzioni devono essere definite prima del loro utilizzo. Per essere sicuri di aver definito tutte le funzioni prima dell'esecuzione del codice si potrebbero inserire tra i *tag* <HEAD> </HEAD>. Se la funzione è chiamata prima di essere definita il programma genera un errore.

La definizione della funzione non implica la sua esecuzione ma consiste solo nell'attribuzione di un nome. Quando nel programma si incontra il nome assegnato l'interprete eseguirà le istruzioni tra parentesi graffe.

Per esempio consideriamo una funzione cosi definita:

```
function stampa_nomi(nome1,nome2,nome3)  {
   document.write(nome1);
   document.write(nome2);
   document.write(nome3);
}
```

La funzione scriverà sullo schermo i tre nomi che li sono passati come argomenti. Se, durante la stesura del programma, l'utente osserva che deve inserire ripetutamente le tre istruzioni in sequenza è una buona idea definire una funzione. Per eseguire effettivamente la funzione la si deve chiamare con i parametri oportuni:

```
. . .
var stringa1, stringa2, stringa3;
stringa1="Marco";
stringa2="Mario";
stringa3="Maria";
stampa_nomi(stringa1,stringa2, stringa3);
. . .
```

Quando il programma alla riga che richiama la funzione assegnerà ai parametri della funzione i valori delle variabili stringa1, stringa2 e stringa3. Come risultato si avrà la stampa nel file html dei tre nomi Marco, Mario e Maria. Ovviamente lo stesso risultato si ha anche scrivendo:

```
. . .
stampa_nomi(="Marco","Mario","Maria");
. . .
```

Quando si utilizzano funzioni che effettuano calcoli matematici potrebbe fare comodo avere un valore come risultato della funzione. In questo caso si deve inserire il comando *return* come ultima riga nella

definizione e nella chiamata la funzione può essere considerata come una variabile. Supponiamo di dover utilizzare una funzione che calcoli la somma dei suoi parametri:

```
function somma(n1,n2,n3)  {
   var result = n1+n2+n3;
   return result;
}
```

La chiamata alla funzione

```
risultato=somma(2,5,7);
document.writeln(risultato);
```

In questo modo il risultato dei calcoli che la funzione esegue è assegnato alla variabile *risultato* che può essere successivamente utilizzata nel programma. La funzione potrebbe essere chiamata anche nel modo tradizionale ma in questo caso risulterebbe alquanto inutile.

Funzioni costruttrici

Abbiamo detto in precedenza che Javascript è un linguaggio di scripting orientato agli oggetti. I linguaggi di programmazione orientati agli oggetti (C++ e Java) si basano su strutture chiamate ***classi***. Le classi possiedono ***attributi*** e ***metodi***, ma per utilizzarli sono necessari gli oggetti. Gli ***oggetti*** sono ***istanze*** delle classi ed ereditano tutte le loro caratteristiche.

Nel caso di Javascript si ha un elemento analogo alle classi che prende il nome di ***funzione costruttrice***. Tutti gli oggetti creati tramite una funzione costruttrice ne ereditano le caratteristiche. Tutte le funzioni definite in Javascript possono essere utilizzate come costruttrici.

Consideriamo il seguente esempio:

```
function veicolo(tipo, colore, consumo) {
   this.tipo_veicolo = tipo;  // auto, moto, bici
   this.colore_veicolo = colore;
   this.consumo_veicolo = consumo;
}
```

La parola chiave *this* fa riferimento all'oggetto chiamato nella funzione costruttrice ed il suo uso è una delle differenze sostanziali tra funzioni standard e costruttrici. Per istanziare oggetti di tipo veicolo si deve utilizzare la seguente sintassi:

```
ferrari = new veicolo("auto","rosso","grande");
williams = new veicolo("auto","bianco","medio");
williams.motore = "benzina";
```

In questo modo si sono definiti due oggetti che hanno lo stesso tipo di caratteristiche anche se di valore differente. Si osservi che all'oggetto si possono aggiungere delle nuove proprietà che la funzione costruttrice non definisce. E' possibile anche istanziare un nuovo oggetto senza passare nessun parametro alla funzione costruttrice e assegnarle in un secondo tempo.

Il Javascript permette anche di creare degli oggetti più specifici, come per esempio definire degli oggetti che sono particolari tipi di veicoli con delle proprietà aggiuntive.

```
function veicoloStradale(targa_veicolo, scadenza_assicurazione) {
   this.targa = targa_veicolo;
   this.scad_ass = scadenza_assicurazione;
}
veicoloStradale.prototype = new veicolo;
```

Gli oggetti cosi definiti sono sicuramente veicoli (tramite l'uso della proprietà prototype) ma in più hanno una targa ed una scadenza assicurativa. Quando si vuole istanziare un oggetto di tipo veicoloStradale si deve utilizzare la seguente sintassi:

```
golf = new veicoloStradale(“aa999ff”,”gennaio2002” )
golf.tipo_veicolo = “auto”;
golf.colore_veicolo = “argento”;
golf.consumo_veicolo = “medio”;
```

In questo modo il veicolo golf può essere utilizzato nel codice sfruttando le caratteristiche di entrambe le funzioni costruttrici.

7. Istruzioni condizionali

Javascript, per capacità volutamente limitata, dispone di tutte le istruzioni condizionali di base, presenti già da Javascript 1.0 e quindi riconosciute anche dall'ECMA, mentre solo due sono più recenti: la **switch** e il **do... while** introdotte con Javascript 1.2.

Il cuore di queste istruzioni è nella condizione che ne implicano l'esecuzione. Per tale motivo l'istruzione va scritta con dovizia e con accuratezza.

Innanzitutto occorre tenere presente che l'operatore di uguaglianza da usarsi nelle espressioni è **==** e non **=** che, invece, è l'operatore di assegnamento: è questo un errore frequentissimo che pregiudica spesso l'esecuzione dello script in quanto Javascript semplicemente ignora l'espressione.

Inoltre le condizioni possono essere tra loro abbinate prestando attenzione alla precedenza degli operatori (per tale motivo guardare la lezione relativa a quest'argomento).

Ecco un esempio di espressione condizionale:

```
if (valore<10 && valore>0)
```

per cui l'espressione restituisce il valore true se il numero è compreso tra 0 e 10 (ma non uguale agli estremi).

If, If...else e switch

L'espressione base del controllo di flusso è:

```
if(espressione) {istruzione}
[else istruzione]...
```

dove l'espressione è booleana, cioè può assumere i valori true o false. L'istruzione principale può essere seguita dall'else che indica un'istruzione alternativa da eseguire quando la prima non è verificata.

Se occorre effettuare una serie di test si può iterare l'*else* in questo modo:

```
if (espressione) {istruzione;}
else (espressione) {istruzione;}
else (espressione) {istruzione;}
else (espressione) {istruzione;}
else istruzione;
```

per cui si valuta l'espressione accanto all'*else* e viene eseguita se l'espressione è vera, altrimenti si esegue l'*else* finale.

L'alternativa a questa disposizione è l'istruzione ***switch***. La forma sintattica è:

```
switch (espressione) {
   case costante1:
      istruzioni;
      break;
   case costante 2:
      istruzioni;
      break;
....
   default:
      istruzioni;
}
```

Il valore dell'espressione viene confrontato con i diversi valori dei *case* e quando viene trovata corrispondenza, si esegue l'istruzione o le sequenze di istruzioni associati, anche se al *case* è associato uno

statement vuoto oppure un'ulteriore *switch*. L'istruzione di *default* è opzionale e viene eseguita solo se non è trovata corrispondenza. L'istruzione *break* è opzionale in quanto consente solo al programma di uscire dal ciclo di *switch*, se fosse mancante il programma continuerebbe a confrontare il valore.

Molto valida è anche l'espressione condizionale ternaria che fonde l'if...else in un unico comando che è il **?** conosciuto anche come **operatore ternario**. La sua forma è (fare attenzione ai due punti):

```
Espressione1 ? Espressione2 :Espressione3
```

per cui se è vera la prima espressione viene eseguita la seconda, se falsa viene eseguita la terza. Si noti l'esempio:

```
"Ho trovato", counter, (counter==1)?"parola.":"parole."
```

che nel caso in cui la variabile counter sia diversa da uno scriverà parole al posto di parola.

Un esempio molto elegante di operatore ternario è quello che serve a determinare il browser utilizzato:

```
ie=document.all?1:0
nn=document.layers?1:0
```

le espressioni vanno lette così: Explorer sa interpretare il comando **document.all**, Netscape sa interpretare **document.layers** e non viceversa per cui se il browser interpreta il comando allora sarà la variabile **ie** oppure **nn** uguale ad 1.

For e for … in

Un altro potente strumento di iterazione di istruzioni è il ciclo **for** che esegue una serie di istruzioni fino a che non è stato raggiunto il limite indicato da una condizione. La variabile assume spesso il nome di contatore e va inizializzata, poi ad ogni passaggio questa viene incrementata e poi si confronta con la condizione, se falsa il ciclo continua, se vera il ciclo esce. L'istruzione è:

```
for (inizializzazione; condizione; incremento) {istruzioni};
```

dove la prima espressione dice da dove inizializzare la variabile contatore, la seconda espressione pone il limite condizionale entro il quale l'istruzione deve essere reiterata, mentre l'ultima espressione dice al loop di quanto deve incrementare o decrementare la variabile.

Ad esempio molto utile è il ciclo che serve ad inizializzare un array (nel nostro caso prendiamo un array di 10 elementi:

```
for (i=0; i<10; i++){
matrice[i]=0;
}
```

osserviamo che le matrici si iniziano a contare da 0 per cui quando il contatore assume valore 10 il ciclo non è ripetuto.

E' possibile anche omettere qualsiasi parte del costruttore della funzione, ma si deve essere sicuri che il ciclo s'interromperà controllando la variabile tramite istruzioni esterne. Osservate che nel costruttore della funzione for che segue manca la prima parte (d'inizializzazione) ma è presente il punto e virgola.

```
var i=5;
for (; i<10; i++){
document.write(i);
}
```

Un altro tipo di ciclo è ***for...in*** che lavora con le proprietà di un oggetto.

La sintassi è:

```
for (indice in oggetto) {istruzioni}
```

Tale funzione è utilizzata per analizzare le proprietà di un oggetto indicato. E' un'istruzione un po' complessa ma utilissima per conoscere il valore che in un certo momento possiedono le proprietà di un oggetto. Qui se ne tralascia ogni approfondimento, ma se ne dà solo un esempio:

```
for (i in navigator){
document.write("Proprieta\' :"+i);
document.writeln (" valore: " +navigator[i]);
}
```

While

Altro controllo di flusso si può avere con:

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

che esegue l'espressione finché la condizione è vera. Come nel caso delle istruzioni if o switch, la funzione while verifica la condizione racchiusa tra parentesi tonde e per tutto il tempo che essa è valutata true sono eseguite le espressioni che seguono. Nel momento in qui la condizione è valutata come falsa il ciclo è interrotto e si esegue l'istruzione che lo segue.

Per creare un ciclo infinito può andare molto bene l'istruzione:

```
while (true) {...}
```

Nell'esempio che segue si mostra un semplice programma che contiene un ciclo while:

```
var count=1;
while (count<5) {
   document.writeln(count);
   ++count;
}
```

è molto importante inserire nel ciclo while un'istruzione che prima o poi porti la condizione a false. Nell'esempio la variabile count è inizializzata a 1 ed è incrementata dentro al ciclo. In questo modo il ciclo stesso sarà eseguito per quattro volte.

do ... While

Se occorre eseguire il blocco di istruzioni almeno una volta ci viene in aiuto un altro controllo di flusso:

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

che esegue l'espressione finchè la condizione è vera. Considerando l'esempio che segue il ciclo sarà eseguito una volta per poi interrompersi poiché la condizione non è vera.

```
var count=5;
do {
   document.write("siamo arrivati al numero "+count);
   count++;
} while (count<2)
```

Break e continue

I comandi ***break*** e ***continue*** servono ad ottimizzare i cicli *for* e *while* oltre che all'operatore condizionale *if*.

Il comando ***break***, infatti, interrompe un blocco di istruzioni saltando alla prima istruzione seguente il blocco contenente il ***break***. L'utilizzo appropriato è quello di evitare la formazione di loop senza uscita:

```
function interrompi() {
while (x>0) {
```

```
if (x>3)
break; //qui l'istruzione si interrompe ed esce dall'while
x++;
}
return x;
}
```

L'esempio mostra come il ciclo continua ad incrementare la variabile x finchè questa è maggiore di 3, nel qual caso incontra l'istruzione ***break*** che interrompe il ciclo e continua con l'istruzione successiva al blocco (*return x;*).

Il comando ***continue***, invece, indica di continuare il blocco ma interrompendo l'iterazione in quel punto e riprendendo dall'inizio del blocco.

```
var x=0;
while (x<10) {
x++;
if (x==3) continue;
document.write("<br>... "+x+" ")
}

var x=0;
while (x<10) {
x++;
if (x==3) break;
document.write("<br>... "+x+" ")
}
```

L'esempio mostra due cicli while che si ripetono finché x rimane minore di 10. Nel primo ciclo il programma scriverà sullo schermo tutti i numeri da 1 a 10 saltando il 3. Nel secondo ciclo il programma scriverà solo 1 e 2 in quanto trovando il numero 3 il ciclo si interrompe.

8. Imparare tramite gli esempi

Aprire una nuova finestra

Nella pratica JavaScript consente di arricchire documenti HTML con script più o meno complessi e più o meno utili. L'apertura di finestre indipendenti da quella principale del browser è una delle peculiarità di JavaScript, che oltre a dare esempi pratici consente di definire approfondire elementi concettuali.

Questo è il codice HTML necessario all'apertura di una finestra indipendente da quella principale del browser:

```
<HTML>
<HEAD>
<title>Nuova finestra</title>
</HEAD>
<body>
<FORM>
<INPUT TYPE="button" VALUE="Nuova finestra"
onClick='window.open("finestra.htm");'>
</FORM>
</BODY>
</HTML>
```

Per comprendere questo script è necessario introdurre il metodo open(). I metodi sono delle particolari funzioni associate agli oggetti, dei quali richiedono l'elaborazione senza definirne le caratteristiche. I metodi sono facilmente riconoscibili dalle due parentesi che seguono la parola.

OnClick è, invece, uno dei principali gestori di eventi, e come tale è utilizzato per iniziare o richiamare uno script. Viene utilizzato con il tag <INPUT> di tipo pulsante, col tag <A>, con i pulsanti di reset e di invio, e finalmente con le caselle di controllo. In risposta ad un click del mouse viene eseguito lo script specificato dal gestore di eventi, in questo caso window.open (si può anche omettere il riferimento a window, visto che il metodo open gli appartiene di default).

JavaScript permette di definire non solo l'apertura di una finestra indipendente da quella principale, ma anche di regolarne alcune caratteristiche. Per fare questo è necessario impostare alcuni paramentri che, è bene precisare, generano i propri effetti esclusivamente sulla nuova finestra, e non sulla finestra principale del browser.

JavaScript supporta i segueni argomenti, in base ai quali è possibile definre le caratteristiche della nuova finestra:

Menubar=yes/no	Attiva o disattiva la barra dei menu del browser
Location=yes/no	Attiva o disattiva la barra dell'URL del browser
Status=yes/no	Attiva o disattiva la barra di stato del browser
Scrollbars=yes/no	Attiva o disattiva le barre di scorrimento del browser
Directories=yes/no	Attiva o disattiva i bottoni del browser
Toolbar=yes/no	Attiva o disattiva i pulsanti della barra degli strumenti
Resizeable=yes/no	Attiva o disattiva la possibilità di ridimensionare la finestra
Width=X	Larghezza della finestra in pixel
Height=X	Altezza della finestra in pixel

Ecco il codice completo di una finestra in JavaScript di cui si stabiliscono le caratteristiche

```
<HTML>
<HEAD>
<TITLE>Finestra indipendente</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function NewWindow() {
win2=window.open("http://www.clarence.com/home/htmlpoint","NewWindow",
"toolbar=no,directories=no,menubar=no,scrollbars=no,width=400,height=30
0");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<input type="button" value="Nuova finestra" onclick='NewWindow();'>
</FORM>
</BODY>
</HTML>
```

NewWindow è una funzione, ovvero una istruzione speciale contenente, a sua volta, altre istruzioni. Tali istruzioni devono essere obbligatoriamente racchiuse tra parentesi graffe: {}.

Se le finestre da aprire sono più di una, si definiscono più funzioni NewWindow, assegnando ad ognuna di esse un numero diverso:

NewWindow()
NewWindow2()

Aperta una finestra indipendente dal browser può essere utile, ai fini di una maggiore funzionalità delle pagine, creare un bottone "close". In questo caso si fa uso di un altro metodo, close(), in tutto simile al già esaminato open(), con l'unica non marginale eccezione di procurare l'effetto inverso, ovvero chiudere la finestra:

```
<form>
<input type="button" value="Chiudi questa finestra" name="B1"
onclick="window.close()">
</form>
```

Il contenuto di una finestra indipendente del browser richiamata con JavaScript, può essere definito in due modi:

> *direttamente all'interno della pagina principale del browser;*
> *da un file HTML esterno.*

Richiamare il contenuto direttamente dalla pagina principale del browser ha il vantaggio di allegerire il traffico sul server, visto che comunque il browser non deve richiamare un nuovo documento HTML, ma interpretare quello posto all'interno dello script.

```
<HTML>
<HEAD>
<script language="JavaScript">
function NewWindow()
{
msg=open("","schermo","toolbar=no,directories=no,menubar=no,width=170,h
eight=250,resizable=yes");
msg.document.write("<HEAD><TITLE>HTML.it</TITLE></HEAD><body>");
```

```
msg.document.write("<B><CENTER><font
size=6>HTML.it</font></CENTER></B><BR>");
msg.document.write("<A
HREF=http:www.clarence.com/home/htmlpoint/java.htm
target=home>Javascript</A><BR>");
msg.document.write("<A
HREF=http:www.clarence.com/home/htmlpoint/sfondi.htm
target=home>Sfondi</A><BR>");
msg.document.write("<A
HREF=http:www.clarence.com/home/htmlpoint/gif.htm target=home>Gif
animate</A><BR>");
msg.document.write("<A
HREF=http:www.clarence.com/home/htmlpoint/guida_go.htm
target=home>Guida HTML</A><BR>");
msg.document.write("<A
HREF=http:www.clarence.com/home/htmlpoint/controllo.htm
target=home>Controllo di qualita'</A><BR>");
}
</script>
</HEAD>
<body>
<form>
<input type="button" value="Apri la finestra" onclick="NewWindow()">
</form>
</body>
</HTML>
```

La seconda ipotesi, ovvero richiamare il contenuto di una finestra da un file HTML esterno, è consigliabile quando tale contenuto non si limiti a semplice testo, ma contenga immagini, suoni e una struttura complessa.

Messaggi sulla status bar

La barra di stato del browser è il luogo solitamente deputato alla visualizzazione di informazioni sui link presenti nella pagina.

JavaScript permette attraverso semplici operazioni, di modificarne il contenuto testuale in relazione ad alcuni eventi.

Si parta, per un successivo approfondimento, dal seguente codice:

```
<A HREF="link.htm" OnMouseOver="window.status='questo testo viene
visualizzato sulla barra di stato del browser' ; return true">Clicca
qui</A>
```

Il gestore di eventi "OnMouseOver" viene attivato quando il puntore del mouse passa sopra il collegamento ipertestuale. La proprietà "window.status" visualizza il testo inserito tra gli apici doppi, nella barra di stato del browser. L'istruzione "return" specifica il valore restituito da una funzione, che può essere "true" o "false".

Impostato in questo modo, lo script visualizza una stringa di testo fissa, che viene sostiuita soltanto quando il mouse passa su un altro link. Per evitare questo piccolo inconveniente, facendo sparire il testo quando il mouse esce dall'area di link, è sufficiente scrivere il seguente codice:

```
<HTML>
<HEAD>
<script language="JavaScript">
<!-- Hide
function link(txt) {
window.status = txt;
setTimeout("cancella()",500);
```

```
}

function cancella() {
window.status="";
}
// -->
</script>
</HEAD>
<BODY>
<a href="link.htm" onMouseOver="link('Messaggio sulla status
bar');return true;">altro link</a>
</BODY>
</HTML>
```

La funzione "link" viene invocata quando il puntore del mouse passa sul link testuale, mentre "setTimeout" stabilisce che entro 500 millesecondi (mezzo secondo) debba verificarsi la funzione "cancella".
Negli esempi finora considerati il testo è visualizzato sulla barra di stato in modo statico. Complicando un po' il codice è possibile generare testo in movimento.

Ecco un esempio di testo in movimento sulla status bar da destra a sinistra:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var timed = 0;
var scrollGo = false;
var delay=100;
var space=100;
function scroll_start() {
var i=0;
msg="Questo è il messaggio visualizzato sulla status bar";
for (i=0;i<space;i++)
msg=" "+msg;
scrollGo=true;
timerid=window.setTimeout("scrollmsg(0)",delay);
}

function scrollmsg(pos) {
var out = "";
scrollGo=false;
if (pos < msg.length)
self.status = msg.substring(pos, msg.length);
else
pos=-1;
++pos;
scrollGo=true;
timerid=window.setTimeout("scrollmsg("+pos+")",delay);
}
</SCRIPT>
</HEAD>
<BODY onLoad="scroll_start();">
</BODY>
</HTML>
```

Un valore inferiore della variabile "delay" aumenta la velocità di scorrimento mentre, al contrario, un valore più alto la diminuisce. Il tempo è espresso in decimi di secondo (millisecondi) e viene manipolato attraverso il metodo "setTimeout()".

Questo script usa la proprietà "status" dell'oggetto "window" e grazie al gestore di eventi "onLoad" viene eseguito automaticamente all'apertura della pagina. Il gestore "onLoad" viene solitamente usato con i tag <BODY>, <FRAMESET> ed <IMG>. Nel caso specifico il gestore di eventi richiama la funzione "scroll_start()", la quale a sua volta avvia il messaggio sulla status bar del browser.

Un abuso di questo script rischia di generare confusione nel visitatore ma, se usato con parsimonia, da' buoni risultati. La presenza di testo nella status bar preclude, nella maggior parte di questi script, la conoscenza, per esempio, degli URL collegati ad un certo link testuale o d'immagine. La buona versatilità di JavaScript permette di personalizzare a piacimento il modo in cui i messaggi appaiono nella status bar. Grazie ad un semplicissimo script, per esempio, è possibile visualizzare un messaggio fisso, che non preclude l'indicazione degli URL di riferimento ai link presenti nella pagina:

```
<script language="JavaScript"><!--defaultStatus = "Questo messaggio
appare fisso sulla barra di stato"//--></script>
```

La proprietà "defaultStatus" indica il messaggio che deve apparire nella barra di stato quando il documento è aperto per la prima volta.

La questione relativa alla visualizzazione di testo nella barra di stato permette di introdurre il concetto di tempo in JavaScript.

JavaScript misura il tempo come numero di millesimi di secondo trascorsi dal 1 gennaio 1970, e viene manipolato grazie all'oggetto "date", il quale cambia con l'ora. Le istanze dell'oggetto "date" restituiscono un valore statico in base ad un valore corrente. L'ora, in JavaScript, si basa sulla macchina client, non sul server. Ciò significa che chiunque incontri un orologio scritto in JavaSctipt vedrà visualizzata l'ora secondo il proprio pc, e non secondo il computer remoto.

Ecco un primo, semplice, esempio di orario generato con JavaScript:

```
<script language="JavaScript">
<!--
oggi = new Date()
document.write("Sono le ",oggi.getHours(),":",oggi.getMinutes())
document.write(" del ", oggi.getDate(),"/",oggi.getMonth() +
1,"/",oggi.getYear());
// -->
</script>
```

L'oggetto Date utilizza il modello "new Date()", ed il suo valore è memorizzato nella variabile "oggi". I metodi "getHours()" e "getMinutes()" restituiscono, l'uno per le ore e l'altro per i minuti, valori compresi tra 0 e 23, e tra 0 e 59.

Il metodo "getMonth" restituisce un numero compreso tra 0 e 11, per cui è necessario aggiungere 1 al numero restituito per ottenere il mese corretto.

Se non si specifica una certa ora e una data, il browser utilizza l'ora locale. Per fare questo è necessario scrivere "oggi" prima di ciascun metodo "get", altrimenti il browser non sa a quale oggetto fare riferimento.
E' possibile, grazie all'uso congiunto di script e moduli, creare orologi dinamici che visualizzino l'ora secondo dopo secondo. Ecco un esempio:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function TimeOutfunc() {
timeout = window.setTimeout("TimeOutfunc()", 1000);
var today = new Date();
document.forms[0].elements[0].value = today.toString();
```

```
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="timeout = setTimeout('TimeOutfunc();',1000);">
<FORM>
<INPUT TYPE="text" NAME="ora" VALUE="" SIZE="25">
</FORM>
</BODY>
</HTML>
```

Il valore 1000, rappresenta il numero di millesecondi entro i quali si aggiorna l'ora nella casella tramite il metodo "timeout()".

L'associazione degli eventi JavaScript relativi alla status bar e all'orario trattati in questo esempio, permettono di visualizzare la data e l'ora all'interno della barra di stato del browser, attraverso un semplice script:

```
<HEAD>
<SCRIPT language="JavaScript">
function doClock() {
window.setTimeout( "doClock()", 1000 );
today = new Date();
self.status = today.toString();
}
</SCRIPT></HEAD>
<BODY onLoad="doClock()">
</BODY>
```

Utilità interessante per le coloro che aggiornano periodicamente le proprie pagine, è quella che consente l'inserimento automatico della data riferita all'ultima modifica apportata alla pagina. In questo modo è possibile aggiornare periodicamente i visitatori sulle modifiche intervenute nei documenti HTML.

Il codice HTML per ottenere un tale risultato è molto semplice:

```
<html>
<body>
Ultima modifica:
<script language="JavaScript">
<!--
document.write(document.lastModified)
// -->
</script>
</body>
</html>
```

La proprietà "lastModified" è di sola lettura e restituisce la data in cui il documento è stato modificato per l'ultima volta. In questo modo, e senza alcun ulteriore intervento, la data della modifica cambierà automaticamente.

Bottoni attivi in JavaScript

E' sempre più frequente imbattersi in siti Web che fanno largo uso di bottoni cliccabili. Con questo termine si intendono quelle immagini che, sfiorate dal puntatore del mouse, cambiano aspetto generando effetti grafici graficamente accattivanti. Questi effetti, da un punto di vista meramente tecnico, sono generati da applet Java o da codice JavaScript. Sulla disamina tecnica di questi ultimi si incentra il presente esempio, attraverso la presentazione del codice fonte e il relativo esempio pratico.

Il funzionamento di tali script è concettualmente molto semplice: nel momento in cui il mouse passa sopra l'area d'immagine, questa viene sostituita da un'altra precedentemente impostata, con la conseguenza di generare un effetto simile alla pressione di un bottone.

Prima di cominciare, quindi, è necessario creare due immagini:

immagine01.gif: che viene caricata insieme alla pagina e finchè il puntatore del mouse non vi passa sopra rimane visibile;

immagine02.gif: che è, invece, visualizzata quando il puntatore del mouse passa nell'area di link e scompare quando lo stesso ne esce.

Le due immagini possono anche avere estensioni diverse (una JPG e l'altra GIF, per esempio), ma devono obbligatoriamente avere la stessa altezza e larghezza. Il mancato rispetto di tale condizione genera pessimi effetti grafici.

Il primo esempio mostrato è volto alla realizzazione di un'immagine cliccabile del logo di HTML.it che, dall'iniziale bianco e nero, assume i colori originali nel momento in cui viene toccato dal puntatore del mouse. Per far questo è sufficiente creare due immagini identiche del logo e, solo per una di esse, assegnare i colori in scala di grigio. Un qualsiasi programma di fotoritocco compie agevolmente quest'operazione.

Create le immagini è necessario impostare correttamente lo script, che va inserito in due parti distinte del codice fonte del documento.

La prima parte va inserita tra i Tag <HEAD></HEAD>, ed è quella che riguarda principalmente la sintassi JavaScript:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
browserName = navigator.appName;
browserVer = parseInt(navigator.appVersion);
if(browserName=="Netscape" && browserVer >= 3) ver = "yes";
else ver = "yes";
if(ver == "yes") {
off = new Image(); off.src ="off.gif";
on= new Image(); on.src ="on.gif";
}

function active(imgName)
{
if (ver == "yes") {
document [imgName].src = "off.gif";
}}

function inactive(imgName)
{
if (ver == "yes") {
document [imgName].src = "on.gif";
}}
// -->
</SCRIPT>
</HEAD>
```

La seconda parte del codice va inserita all'interno dei Tag, e richiama l'immagine associandola agli eventi dello script:

```
<BODY>
<A HREF="jsinterno.html"
onMouseover="inactive('Premi');window.status='Pulsante animato'"
onMouseout= "active('Premi')">
<IMG NAME="Premi" SRC="off.gif" ALT="Premi qui" width=149 height=55
border=0>
</A>
</BODY>
```

Impostato il codice è opportuno esaminarne i passaggi più importanti.

La prima parte dello script compreso tra i Tag <HEAD> stabilisce quali browser possono accedere al documento ed interpretare correttamente il codice JavaScript.

Il codice inserito tra i tag <SCRIPT> viene elaborato dopo il caricamento della pagina. In questo modo le funzioni sono memorizzate ma vengono eseguite soltanto dagli eventi presenti nella pagina. Il Tag <SCRIPT> è contenuto all'interno di <HEAD>, che è il luogo deputato a contenere i dati relativi all'intestazione del documento, perchè questo venga caricato per primo e garantisca che le funzioni siano interpretate prima che l'utente effettui altre operazioni.

"navigator.appName" e "navigator.appVersion" sono proprietà di sola lettura che forniscono rispettivamente: il nome del browser utilizzato nel primo caso e il numero di versione nel secondo. Dal codice si evince come questo script sia supportato da Netscape 3 o successivo (>= 3) e da MS Internet Explorer 4, nonchè dall'ultima release di Opera.

La stringa successiva di codice è particolarmente importante, e richiama le due immagini da visualizzare:

```
off = new Image(); off.src ="off.gif";
on= new Image(); on.src ="on.gif";
```

La funzione "active" utilizza una semplice istruzione "if" per controllare la condizione dello script. L'operatore di confronto "==", restituisce un valore booleano vero o falso come richiesto dall'istruzione condizionale. Quando la funzione "active" viene richiamata, il browser visualizza "off.gif", mentre la funzione "inactive" richiama "on.gif".

All'interno del link vanno inseriti i due gestori di eventi: "onMouseOver" ed "onMouseOut". Il primo si attiva quando l'utente posiziona il puntatore del mouse sull'area di link; mentre il secondo quando lo stesso ne esce.

L'immagine da inserire all'interno del link è "off.gif". Anche quest'immagine deve contenere un riferimento allo script, attraverso il codice NAME="Premi". Senza di esso lo script non funziona.

All'immagine iniziale è possibile affiancarne altre, senza alcuna restrizione, che non sia il rispetto del codice JavaScript.

Si consideri, per un esempio chiarificatore, di voler inserire due nuovi link (uno a JavaScript e l'altro a Html) associati a bottoni cliccabili. In questo caso dovranno crearsi quattro immagini, due per ogni link:

javascript01.gif
javascript02.gif
html01.gif
html02.gif

Partendo dal codice precedentemente esaminato sarà sufficiente aggiungere solo piccoli pezzi di codice JavaScript. All'interno dei Tag <SCRIPT></SCRIPT> va aggiunto:

```
<A HREF="link.htm" onMouseover="inactive('javascript')" onMouseout=
"active('javascript')">
<IMG NAME="javascript" SRC="javascript01.gif" ALT="Javascript"
width=130 height=36 border=0></A>
<BR>
```

```
<A HREF="link.htm" onMouseover="inactive('html')" onMouseout=
"active('html')">
<IMG NAME="html" SRC="html01.gif" ALT="HTML" width=130 height=36
border=0></A>
```

Il codice presentato in questo esempio non è il solo adottabile per ottenere l'effetto tipico dei bottoni cliccabili, ma è certamente tra i più diffusi e semplici da personalizzare.

Un abuso di questo script rischia di rallentare pesantemente la velocità di caricamento delle pagine. A tal proposito si consiglia di inserire piccole immagini in formato GIF con il più basso numero possibile di colori.

Password in Javascript

La protezione di una pagina Web con password avviene solitamente attraverso un CGI (Common Gateway Interface), che richiede al visitatore la parola d'ordine e l'eventuale login di accesso. L'impostazione e la configurazione di un CGI richiedono conoscenze tecniche approfondite, senza contare che molti server gratuiti (Geocities, Aspide ecc.) non permettono, per ragioni di sicurezza, di depositare propri CGI o altri programmi residenti sui propri hard disk.

Javascript semplifica questa procedura permettendo di impostare password di accesso selezionato a un sito Web o soltanto ad alcune pagine di esso. E' bene premettere fin d'ora che questa soluzione non dà le stesse garanzie di sicurezza di una password generata da un programma CGI, che rimane indubbiamente la soluzione migliore per chi ha necessità professionali.

La duttilità del linguaggio di scripting consente varie soluzioni, più o meno efficaci, per la generazione di password su pagine Web. In questo esempio si analizzeranno due esempi indicativi.

Il primo esempio è di semplice realizzazione oltre che di facile comprensione. Questo il codice HTML da inserire tra i tag <HEAD></HEAD> di un documento Web:

```
<script language="JavaScript">
function passWord() {
var testV = 1;
var pass1 = prompt('Inserisci la tua Password');
while (testV < 3) {
if (!pass1) history.go(-1)
if (pass1.toLowerCase() == "HTML.it") {
alert('La tua password è esatta');
break;
}
testV+=1;
var pass1 = prompt('Sbagliato! Puoi riprovare.','Password');
}
if (pass1.toLowerCase()!="altro" & testV ==3) history.go(-1);
return " ";
}
document.write(passWord());
</script>
```

Il risultato generato da questo codice è, per tutti i browser che supportano javascript (Netscape, Internet Explorer, Opera e altri), la visualizzazione di una finestra di avvertimento che richiede quanto scritto tra apici singoli nella riga di codice:

```
var pass1 = prompt('Inserisci la tua Password');
```

La password che si intende richiedere al visitarore (nell'esempio "HTML.it") va inserita tra apici doppi nella riga di codice:

if (pass1.toLowerCase() == "ciao")

La finestra di avvertimento prevede una stringa bianca da riempire con la password di accesso.

"toLowerCase" è un metodo che converte la stringa in caratteri minuscoli. Non esiste un equivalente in HTML. I metodi sono tipi particolari di funzioni che richiedono degli argomenti e vengono associati a oggetti ed elaborano gli oggetti senza definirne le caratteristiche. Come le funzioni, anche i metodi si riconoscono dalla presenza di parantesi tonde che seguono la parola, ma a differenza delle prime non sono precedute dal termine "function".

Modificando il codice script è possibile impostare il numero massimo di tentativi concessi in caso di inserimento di parole d'ordine errate. Nell'esempio considerato il numero massimo di tentativi concessi è 3:

```
if (pass1.toLowerCase()!="altro" & testV ==3) history.go(-1);
```

Il metodo history.go(-1) indica al browser la pagina da visualizzare successivamente al fallimento del terzo e ultimo tentativo. In questo caso (-1) singnifica che la pagina visualizzata sarà la stessa dalla quale si è tentato di accedere alla pagina con password. In altre parole tale codice simula la pressione del tasto "back" o "indietro" del browser. Per rendere questa funzione maggiormente "user friendly" è possibile sostituire a (-1) il link a un'altra pagina che avverta gentilmente dell'errore intervenuto nella digitazione della password. Considerando di chimare la nuova pagina "risposta.htm", il codice è:

```
if (pass1.toLowerCase()!="altro" & testV ==3) location.href =
"risposta.htm";
```

Nel caso in cui la password sia digitata correttamente, il messaggio di conferma che appare in una nuova finestra prima del caricamento della pagina protetta viene generato dal codice:

```
alert('La tua password è esatta');
```

Il secondo esempio riguarda una password numerica, nella quale è necessario digitare da un'apposita pulsantiera un codice di quattro numeri. Ecco il codice completo:

```
<HEAD>
<SCRIPT language="JavaScript">
<!--
var usermulcode=40
var code=0
var mul=1
var digit=0
var fails=0
function Enter_code(number)
{
code=code*10+number
mul=mul*number
document.codepad.thecode.value=code
digit++
if (digit==4)
{
if (mul==40)
{
location=code+".htm"
}
else
{
fails++
code=0
mul=1
digit=0
if (fails<3)
```

```
{
if (fails==1)
{document.codepad.thecode.value="riprova"}
if (fails==2)
{document.codepad.thecode.value="ultima volta"}
}
else
{
location="risposta.htm"
document.codepad.thecode.value="A presto!"
}
}
}
}

function keycodepad(mulcode)
{
usermulcode=mulcode
document.write("<form name=\"codepad\">");
document.write("<input type=\"button\" value=\" 1 \"
onClick=\"Enter_code(1)\">");
document.write("<input type=\"button\" value=\" 2 \"
onClick=\"Enter_code(2)\">");
document.write("<input type=\"button\" value=\" 3 \"
onClick=\"Enter_code(3)\"><br>");
document.write("<input type=\"button\" value=\" 4 \"
onClick=\"Enter_code(4)\">");
document.write("<input type=\"button\" value=\" 5 \"
onClick=\"Enter_code(5)\">");
document.write("<input type=\"button\" value=\" 6 \"
onClick=\"Enter_code(6)\"><br>");
document.write("<input type=\"button\" value=\" 7 \"
onClick=\"Enter_code(7)\">");
document.write("<input type=\"button\" value=\" 8 \"
onClick=\"Enter_code(8)\">");
document.write("<input type=\"button\" value=\" 9 \"
onClick=\"Enter_code(9)\"><br>");
document.write("<input type=\"text\" name=\"thecode\" size=10
value=\"\"><br>");
document.write("</form>");
}

// -->

</SCRIPT>
</HEAD><BODY>
<FORM name="codepad"><INPUT type="button" value=" 1 "
onClick="Enter_code(1)"><INPUT type="button" value=" 2 "
onClick="Enter_code(2)"><INPUT type="button" value=" 3 "
onClick="Enter_code(3)"><BR><INPUT type="button" value=" 4 "
onClick="Enter_code(4)"><INPUT type="button" value=" 5 "
onClick="Enter_code(5)"><INPUT type="button" value=" 6 "
onClick="Enter_code(6)"><BR><INPUT type="button" value=" 7 "
onClick="Enter_code(7)"><INPUT type="button" value=" 8 "
onClick="Enter_code(8)"><INPUT type="button" value=" 9 "
onClick="Enter_code(9)"><BR><INPUT type="text" name="thecode" size=10
value=""><BR></FORM>
</BODY>
```

La password di questo esempio viene generata dalla riga di codice

```
var usermulcode=40
```

Che non è altro che il risultato della moltiplicazione di: 2*5*1*4, che fa appunto 40. Alla digitazione di questi numeri lo script rimanderà alla pagina "2514.htm", cioè la pagina protetta e creata allo scopo.

La prima considerazione che viene in mente è che il risutalto di 40 può raggiungersi anche in altri modi: 2*5*2*2, 1*2*4*5, 5*2*1*4 ecc. Ma in questo caso l'unica pagina che lo script raggiungererà è quella con i numeri della password reale, mentre negli altri casi molto semplicemente non troverà il link in quanto inesistente.

Se per esempio inseriamo un valore di 128, la password potrà essere: 8*8*1*2*, ovvero 2*4*8*2, ovvero 2*1*8*8*, ovvero 8*1*2*8, ovvero 4*4*4*2 ecc. Se l'intenzione è di sfruttare la prima di queste password (8812), la pagina protetta dovrà chiamarsi "8812.htm".

La parte successiva di codice JavaScript:

```
if (fails<3)
{
if (fails==1)
{document.codepad.thecode.value="riprova"}
if (fails==2)
{document.codepad.thecode.value="ultima volta"}
}
else
{
location="risposta.htm"
document.codepad.thecode.value="A presto!"
```

determina nell'ordine: il numero di tentativi permessi (in questo caso 3); il messaggio visualizzato in caso di errata digitazione del primo tentativo (in questo caso "riprova"); il messaggio visualizzato dallo script dopo il secondo errore (in questo caso "ultima volta); la pagina visualizzata in seguito al terzo errore e che dovrà essere creata e messa sul server (in questo caso "risposta.htm") e finalmente il messaggio che appare dopo l'ultimo tentativo fallito.

La seconda delle due password è più sicura della prima, anche se come accennato sopra, la maggior sicurezza si ottiene soltanto con programmi creati allo scopo.

Controllo dati inseriti in un form

Il successo di un sito Web dipende da fattori eterogenei che ben integrati avviano un circolo virtuoso fatto di accessi e seguito tra i navigatori. Non esiste una formula empirica che ne spieghi le ragioni, ne' è questa la sede per cercarne. Ad un'analisi superficiale, però, sembra evidente che il successo di un sito dipende in buona misura dalla costanza delle visite da parte di uno "zoccolo duro" di visitatori. Perché un sito crei un'utenza "affezionata" è necessario che metta in atto iniziative volte a creare tale affinità. Un rapporto personalizzato e personale "one to one" è in questo senso la soluzione ideale. Ma se questa soluzione è percorribile per siti di piccole dimensioni, risulta impraticabile per grosse realtà con centinaia di migliaia di contatti al giorno. Si pensi soltanto che rispondere a 100 mail giornaliere potrebbero tenere occupate 5 persone a tempo pieno. Tralasciando gli aspetti più intimamente psicologici, la soluzione tecnica a questi problemi è costituita dai moduli HTML. Introdotti dalle prime versioni di HTML, i moduli sono dei questionari precompilati con campi di diverso tipo. Spesso sono collegati a CGI (Common Gateway Interface), programmi solitamente scritti in Perl e residenti su server. Questi programmi ricevono i moduli compilati, li elaborano secondo schemi predefiniti e li spediscono agli indirizzi e-mail indicati.

L'automazione consentita da un CGI è in massima parte determinata dalla sua complessità.

La diffusione dei moduli HTML consente di creare guestbook, questionari, quiz interattivi, curriculum vitae sia ai grandi siti che a quelli più piccoli, fino alle home page personali. Esistono decine, se non centinaia, di programmi gratuiti ed altrettanti servizi on line di supporto.

Tralasciando le basi tecniche sulle quali si reggono i form HTML, quello che si intende sviluppare in questa sede è un controllo che Javascript consente di effettuare sulla compilazione dei moduli. E' frequente che alcuni moduli vengano compilati privi di parti fondamentali, senza le quali gli stessi non hanno ragion d'essere. In questo modo si ha il duplice svantaggio di ricevere informazioni parziali e inservibili, e

apparire villani nei confronti dei visitatori che immaginano di ricevere una risposta che non verrà mai, per esempio, perchè il campo "e-mail" non è stato compilato per dimenticanza.

Per evitare certi inconvenienti è possibile un controllo in tempo reale sulla completa compilazione del form da parte del visitatore. In altre parole, se l'utente non compila tutti i campi, o solo alcuni, otterrà un messaggio di avviso che indica l'errore o la dimenticanza.

Per semplificare la comprensione di quanto esposto in questo esempio, si consideri il caso pratico di un modulo informativo composto da tre campi: nome e cognome, e-mail e telefono. Di questi dati soltanto il numero di telefono è facoltativo, mentre gli altri sono considerati obbligatori.

Il codice HTML che crea il form è il seguente (l'aterisco indica i campi obbligatori):

```
<FORM>
Nome e cognome *<BR>
<input type=text name="nome" size=40><BR><BR>
E-mail *<BR>
<input type=text name="mail" size=30><BR><BR>
Telefono<BR>
<input type=text name="telefono" size=20><BR><BR>
<INPUT TYPE="SUBMIT" VALUE="Spedisci">
</FORM>
```

Impostato in questo modo il form è certamente operativo ma se spedito senza un campo obbligatorio compilato non dà alcun messaggio di avviso col rischio, già indicato in precedenza, di ricevere informazioni parziali o peggio inservibili.

Javascript interviene in aiuto con una sintassi semplice e facilmente configurabile. Il codice va inserito in due fasi successive.

Innanzitutto va impostato il codice all'interno dei tag <HEAD> del documento:

```
<script>
function checkrequired(which){
var pass=true
if (document.images){
for (i=0;i<which.length;i++){
var tempobj=which.elements[i]
if (tempobj.name.substring(0,8)=="required"){
if
(((tempobj.type=="text"||tempobj.type=="textarea")&&tempobj.value=='')|
|(tempobj.type.toString().charAt(0)=="s"&&tempobj.selectedIndex==-1)){
pass=false
break
}
}
}
}
if (!pass){
alert("Non hai compilato correttamente il form. Alcune informazioni non
sono state inserite. Clicca su OK e verifica l'inserimento")
return false
}
else
return true
}
</script>
```

La riga di codice:

```
alert("Non hai compilato correttamente il form. Alcune informazioni non
sono state inserite. Clicca su OK e verifica l'inserimento ")
```

imposta il messaggio che il browser dà in risposta quando verifica l'assenza di alcuni campi obbligatori. Ovviamente il messaggio può essere modificato secondo le proprie esigenze.

Il resto del codice esaminato non va oltremodo modificato.

La funzione checkrequired è quella che regge l'interno struttura dello script e stabilisce che i campi controllabili siano text, textarea e select.

Conclusa l'impostazione del codice Javascript è necessario apportare alcune modifiche ai singoli campi del form visto in precedenza.

Per prima cosa va inserito il riferimento alla funzione checkrequired nell'apertura del form:

```
<FORM onSubmit="return checkrequired(this)">
```

Successivamente è necessario anteporre ai nomi dei diversi campi che si vogliono obbligatori il termine "required". Per esempio il nome del primo campo è "nome", e per essere impostato come obbligatorio deve trasformarsi in "requirednome". Così per il campo "mail" che diventa "requiredemail".

L'unico campo non modifato è "telefono", che come scritto in precedenza è facoltativo.

Apportate le dovute modifiche, il form visto in precedenza assume la seguente struttura (fermo restando il codice inserito tra i tag <HEAD>):

```
<FORM onSubmit="return checkrequired(this)">
Nome e cognome *<BR>
<input type=text name="requirednome" size=40><BR><BR>
E-mail *<BR>
<input type=text name="requiredmail" size=30><BR><BR>
Telefono<BR>
<input type=text name="telefono" size=20><BR><BR>
<INPUT TYPE="SUBMIT" VALUE="Spedisci">
</FORM>
```

Compilato il form e premuto il tasto li tasto "spedisci" non rimane altro che verificare gli effetti dello script.

Proteggere il codice sorgente

Tra le esigenze più sentite da chi realizza siti Web o progetti editoriali in Internet, la tutela del proprio diritto d'autore occupa una posizione preminente. Se l'editoria tradizionale è garantita da un'ampia giurisprudenza, molti settori del Web publishing restano vulnerabili a violazioni palesi di diritti d'autore altrui. Questo stato di insicurezza è causato da fattori strettamente legati alla novità del media Internet, alle sue dimensioni ed alla possibilità di visualizzare il codice HTML di un documento e modificarlo a proprio piacimento. In altre parole, l'assenza di un codice compilato in HTML e la possibilità che chiunque possa appropriarsi di immagini e testi, spesso causa vere e proprie clonazioni di Website e relativi danni economici e d'immagine. Senza addentrarsi oltre in dissertazioni giuridiche, ciò che preme mostrare in questo esempio è come Javascript possa, con tutti i limiti di questo linguaggio di scripting, risolvere parzialmente il problema succitato.

Gli esempi Javascript che seguono consentono di disabilitare, in tutto o in parte, l'uso del tasto destro del mouse. L'importanza di questo tasto nell'ambito di navigazioni su WWW è dovuta a due peculiarità:

1. *attivando il tasto destro su un punto qualsiasi di un documento Web e selezionando la voce "HTML" in Internet Explorer e "visualizza sorgente" in Netscape, viene mostrato il codice fonte completo della pagina;*
2. *attivando il tasto destro del mouse su un'immagine di una pagina Web e selezionando la voce "Salva immagine con nome" in Ms Internet Explorer e Netscape, si possono salvare su Hard Disk tutti gli elementi grafici del documento.*

Javascript riesce parzialmente ad annullare l'uso del tasto destro del mouse con codice relativamente semplice. Si sottolinea la natura parziale e non definitiva della soluzione perché, come i più perspicaci avranno notato, l'espediente è aggirabile attivando i comandi della toolbar del browser:

VISUALIZZA / SORGENTE PAGINA per Netscape

e

VISUALIZZA / HTML per MsIe

Esiste, quindi, un altro modo per visualizzare il codice fonte di un documento, ma se la pagina viene priva della toolbar (semplice da nascondere con comandi Javascript) il problema è in parte risolto.

In parte perché disabilitando l'uso di Javascript dal browser il tutto verrebbe vanificato, ed anche nel caso in cui ciò non avvenisse sarebbe sufficiente ripescare i file dalla cache del browser.

Questi esempi Javascript non risolvono alla radice il problema ma sono comunque un buon deterrente.

Il primo esempio si riferisce alla possibilità di eliminare, tout court, l'uso del tasto destro del mouse all'interno di una pagina Web. Quindi, non si limita a disabilitare il comando "HTML" o " visualizza sorgente", ma tutte le opzioni proprie del tasto destro.

Il codice javascript necessario al corretto funzionamento dell'esempio va inserito in due punti distinti del documento.

La prima parte va inserita all'interno degli elementi <HEAD></HEAD>:

```
<SCRIPT LANGUAGE="JavaScript1.1">
<!-- Begin
function right(e) {
if (navigator.appName == 'Netscape' &&
(e.which == 3 || e.which == 2))
return false;
else if (navigator.appName == 'Microsoft Internet Explorer' &&
(event.button == 2 || event.button == 3)) {
alert("Spiacenti, il tasto destro del mouse e' disabilitato");
return false;
}
return true;
}
document.onmousedown=right;
if (document.layers) window.captureEvents(Event.MOUSEDOWN);
window.onmousedown=right;
// End -->
</script>
```

Il messaggio di Alert che in Internet Explorer avverte che il tasto destro è disabilitato va riportato nella riga di codice:

```
alert("Spiacenti, il tasto destro del mouse e' disabilitato");
```

In Nescape, al contrario, cliccando sul tasto destro del mouse non appare alcun messaggio di Alert; questa è l'unica differenza di approccio allo script tra di due browser.

Impostato il codice tra i campi <HEAD> è necessario inserire altre righe di codice alla fine del documento, dopo la chiusura del tag </HTML>:

```
<SCRIPT LANGUAGE"JavaScript1.1">
<!-- Begin
for (var i=0; i<document.images.length; i++)
document.images[i].onmousedown=right;
```

```
for (var i=0; i<document.links.length; i++)
document.links[i].onmousedown=right;
// End -->
</script>
```

Tale codice è necessario percè venga escluso il menu del tasto destro sulle immagini (document.images.length) e sui link (document.links.length).

La disabilitazione totale del tasto destro del mouse è una soluzione radicale non consigliabile, quando l'unica esigenza è di evitare il salvataggio degli elementi grafici. In questi casi è preferibile servirsi di un Javascript che si limita a disabilitare l'uso del tasto destro soltanto quando si cerca di salvare un'immagine. In altre parole se il tasto viene premuto in un qualsiasi altro punto della pagina produce i suoi effetti, mentre sopra un'immagine gif o jpg appare una finestra di Alert che avvisa gentile ma ferma "Spiacenti, non puoi salvare l'immagine". In realtà non viene disabilitata la sola opzione di salvataggio dell'immagine, ma anche le altre "imposta come sfondo", "copia" e "proprietà", ma questo è uno scotto che lo script non risparmia.

Considerando come esempio una pagina HTML sulla quale si vuole evitare il salvataggio dell'immagine "HTML.it.gif", si deve, innanzitutto, scrivere il seguente codice tra gli elementi <HEAD>:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Begin
function protect(e) {
alert("Spiacenti, non puoi salvare l'immagine");
return false;
}

function trap() {
if(document.images)
for(i=0;i<document.images.length;i++)
document.images[i].onmousedown = protect;
}

// End -->
</SCRIPT>
```

La riga di codice:

```
alert("Spiacenti, non puoi salvare l'immagine");
```

determina il testo visualizzato dalla finestra di alert del browser.

Fondamentale è poi inserire il gestore onload all'interno del tag <BODY>:

```
<BODY OnLoad="trap()">
```

Senza OnLoad="trap()" lo script non produce i suoi effetti.

Anche in questo caso esistono due modi per aggirare gli effetti dello script: disabilitare l'uso di Javascript dal browser e salvare l'intero documento con l'opzione "salva con nome" di Ms Internet Explorer 5, che salva sia il testo che le immagini. Richiamare la pagina HTML in una finestra del browser priva di toolbar risolve in parte questo problema.