Manuale Javascript

•La nascita di Javascript

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, per cui 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. L'osservazione può sembrare banale, ma io, personalmente, quando mi sono avvicinato ad Internet, nel 1997, difficilmente trovavo siti dinamici e quei pochi che riuscivo a scovare risultavano per me di notevole fascino, tanto da avvicinarmi a Javascript con il massimo entusiasmo.
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. 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 per cui 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.
In queste brevi lezioni 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 capisco 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).

•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 approccia per la prima volta a questo linguaggio data la sua semplicità sintattica e la sua maneggevolezza. Tuttavia ciò può rappresentare un'arma a doppio taglio perché la semplicità si gioca anche su una disponibilità limitata di oggetti per cui alcuni procedimenti, all'apparenza molto semplici, richiedono *script* abbastanza complessi.
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 comprensione da parte dell'utente, ma immediatamente eseguibili dal client, senza richiedere passaggi intermedi.

Effettuata questa premessa si possono suddividere i linguaggi di solito utilizzati per il Web 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 nel testo di Michael Moncur su Javascript, apparso di recente e pubblicato in Italia da Tecniche Nuove, dove la spiegazione è didattica ma molto efficace e merita la citazione: 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:

1. **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;
2. **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;
3. 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, che secondo me è un vantaggio, 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;
4. 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;
5. 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 io non ne ho mai trovato riferimento nelle guide Javascript in italiano consultate finora, ma spulciando tra quelle ufficiali, non mancava ma un'indicazione a questa associazione. Innanzitutto l'ECMA (e traduco 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, e spero riesca a cogliere più informazioni di quanto sia riuscito a fare io, poiché il sito non appare certo all'altezza della fama dell'associazione, non tanto per la semplicità spartana, ma per la penuria di informazioni, comunque 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 alla 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. Ho spulciato tra la documentazione ufficiale della Microsoft, ma penso 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 mia 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 oggetto in cui il documento viene diviso in elementi più semplici a cui si applica una programmazione orientata agli oggetti, ma oltre i Data-Binding, che sono interfacce client per la visualizzazione dei dati, e gli scriplets, questa mi sembra la "scoperta dell'acqua calda", perché fino ad ora, mi chiedo, cosa ha fatto Javascript? Comunque DHTML ha introdotto diverse novità nella programmazione client di una pagina web, ed è certamente uno strumento potente, anzi, molto potente, tanto da risultare incontrollabile; se ci fosse un po' più di ordine e meno entropia potrebbe essere una svolta decisiva per Internet.
Tuttavia voglio anticipare che i miei richiami a DHTML saranno ridotti al minimo per tanti motivi: innanzitutto esiste già un corso di DHTML (ed altri ad esso correlati come CSS e XML) su questo sito, in più si aggiunge che la spiegazione approfondita di questo nuovo linguaggio potrebbe certamente minare la semplicità del corso, perdendoci in meandri e in "oggetti" che sarebbero attualmente poco utilizzabili perché visibili da poco più della metà dei navigatori. Occorre attendere qualche anno prima che DHTML prenda piede, perciò nel frattempo impariamo almeno ad utilizzare alla meglio Javascript.

•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><!--
//--></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:

1. la pagina viene visualizzata, ma il codice errato non viene eseguito;
2. se il codice genera un loop (cioé un ciclo infinito) la pagina resta bianca o è 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"><!--
//--></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 nella lezione sulle versioni). 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):

1. inserendo il codice nel documento;
2. 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 (in Windows è consigliato NotePad o Blocco Note).
Il vantaggio di usare file esterni è immenso soprattutto perché apporta la caratteristica della modularità per cui uno script che ricorre di frequente (ad esempio il rollover) 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>
```

e salvarlo nella stessa directory del file Javascript;
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 tener 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. 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>
```

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 vengono 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 commenti 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 (//) alla 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:

1. **eseguire iterazioni**, cioè ripetere una parte di codice per un certo numero di volte;
2. **eseguire decisioni condizionate**;
3. **richiamare funzioni** (si vedrà in seguito cosa sono);
4. **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 :

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

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 concetti sparsi qua e là nelle lezioni precedenti rispetto a questo argomento. La lezione 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:

1. all'**interno degli script**, individuati dai tag <SCRIPT>, in maniera sequenziale, per cui l'esecuzione è automatica;
2. **caricandoli da file esterni**;
3. **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);
4. **in luogo di un link** (a partire da Netscape 3.0) nella forma: <A HREF="Javascript:comando">
5. **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>
```

•Gli eventi

Innanzitutto occorre fare una premessa: nel decidere l'ordine degli argomenti da trattare ero abbastanza indeciso in quale posizione porre questa lezione, e quelle ad essa collegate, 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, ma poi ho optato per la posizione attuale in quanto l'argomento è molto affine alla trattazione degli Script in generale, ma sono cosciente di rischiare di rendermi poco comprensibile ai neofiti, per cui vogliano questi apprendere le successive lezioni dando per scontato 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 venga 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.

Ecco un esempio:

```
<A HREF="pagina.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 vedo applicato, posso raccontare un aneddoto: agli inizi del mio uso con Javascript cercavo uno script che mi consentisse di simulare lo streaming, cioè di caricare dalla rete un'immagine di una sequenza solo quando ero sicuro che quella precedente era già stata caricata e quindi visualizzata. Usare un temporizzatore era impossibile perché l'effetto variava secondo la velocità di connessione. Da un ottimo testo di Javascript, trovai un trafiletto che mi offrì lo spunto. In tal modo creai un array di immagini e operavo 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 e difficile è tenerne traccia, quindi, per facilità cerchiamo di raggrupparli in sezioni omogenee:

1. **Eventi attivabili dai tasti del mouse**
2. **Eventi attivabili dai movimenti del mouse**
3. **Eventi attivabili dal trascinamento del mouse (drag and drop)**
4. **Eventi attivabili dall'utente con la tastiera**
5. **Eventi attivabili dalle modifiche dell'utente**
6. **Eventi legati al "fuoco"**
7. **Eventi attivabili dal caricamento degli oggetti**
8. **Eventi attivabili dai movimenti delle finestre**
9. **Eventi legati a particolari bottoni**
10. **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 tener 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

Evento	Versione Javascript	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0
onClick	1.0	*	*	*	*	*	*	*
onDblClick	1.2			*	*		*	*
onMouseDown	1.2			*	*		*	*
onMouseUp	1.2			*	*		*	*
onContextMenu	DHTML							*

Evento	Tag
onClick	**it)**,
onDblClick	**<BOD**
onMouseDown	**<BOD**
onMouseUp	**<BOD**
onContextMenu	Explore

•Eventi 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, che si spiegherà in seguito.
Gli eventi onMouseOver e onMouseOut sono complenmentari 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 (il colore diverso dell'asterisco indica le modifiche rispetto alla versione precedente, NN sta per Netscape e IE per Internet Explorer):

Evento Versione Javascript
NN2.0 NN3.0 NN4.0 NN4.06 IE3.0 IE4.0 IE5.0
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

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 JScript

onMouseOver Questo gestore è usato con i pulsanti di invio (submit), pulsanti di reset (reset), caselle di controllo (checkbox e radio), bottoni, tag <INPUT> di tipo OPTION e tag <A>.
onMouseOut Usato con i tag <BODY> e <A>
onMouseMove Usato con i bottoni e i tag <BODY> 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[num].src='immagine.gif'" onmouseout=document.images[num].src='immagine1.gif'">

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 amovimentare elementi statici come i menu e le barre di navigazione.

Per esempio inserisco un rollover un po' complesso, quello attivabile da una mappa cliccabile. Da notare che per funzionare in Netscape c'è bisogno di aggiungere comunque un link all'area sensibile, evidenziata in rosso (anche se sostituito dal segno del cancelletto), mentre in Explorer si può omettere.

<area shape="rect" coords="2,2,59,26" onmouseover="document.images[num].src='images/tre.gif'" href="#">

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 dal trascinamento del mouse

Evento	**Versione Javascript**	**NN2.0**	**NN3.0**	**NN4.0**	**NN4.06**	**IE3.0**	**IE4.0**	**IE5.0**
onDragDrop	1.2			*	*		*	*
onMove	1.2			*	*		*	*

onDrag	DHTML	*
onDrop	DHTML	*
onDragStart	DHTML	*
onDragEnter	DHTML	*
onDragLeave	DHTML	*
onDragOver	DHTML	*
onDragEnd	DHTML	*

Tag sensibili

Evento	Tag
onDragDrop	**Windo**
onMove	**Windo**
Molto più	**w**.

- Eventi legati alla tastiera

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

Evento	Tag
onKeyPress	**<BOD**
onKeyDown	**<BOD**
onKeyUp	**<BOD**

- Eventi legati alle modifiche

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

Evento	Tag
onChange	**<SELE**
onCellChange	**<APPL**
onKeyUp	**<BOD**

- Eventi legati al "fuoco"

Evento	Versio	NN2.0	NN3.0	NN4.0	NN4.0	IE3.0	IE4.0	IE5.0

	Versione Javascript				6			
onFocus	1.0	*	*	*	*	*	*	*
onBlur	1.0	*	*	*	*	*	*	*
onSelect	1.0	*	*	*	*	*	*	*
onSelectStart	DHTML							*
onLoseCapture	DHTML							*
onbeforeEditFocus	DHTML							*

Evento	Tag
onFocus	**<SELE**
onBlur	**<SELE**
onSelect	***<TEXT***

- Eventi da caricamento degli oggetti

Evento	Versione Javascript	NN2.0	NN3.0	NN4.0	NN4.06	IE3.0	IE4.0	IE5.0
onLoad	1.0	*	*	*	*	*	*	*
onUnload	1.0	*	*	*	*	*	*	*
onAbort	1.1		*	*	*		*	*
onError	1.1		*	*	*		*	*
onBeforeUnload	DHTML							*
onStop	DHTML							*

Evento	Tag associ
onLoad	i tag **<BOD**
onUnload	i tag **<BOD**
onAbort	tag**<I MG>**
onError	tag**<I MG>** e
onBeforeUnload	Questo gestore è usato

onStop	Questo gestore

- Eventi dal movimento delle finestre

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

Evento	Tag
onResize	**<DOC**
onScroll	**Explor**

- Eventi da tasti particolari

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

Evento	Tag
onSubmit	**<FOR**
onReset	**<FOR**

- 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:

1. **numerici** i quali si dividono in:
   a. Interi
   b. decimali o, con linguaggio tecnico, a virgola mobile (in notazione scientifica o in quella standard) (Valori non precisi in Netscape 2.0)
2. **logici** o booleani: che possono assumere soltanto due stati (vero o falso);
3. **stringhe** (o sequenza di caratteri)
4. **valore nullo**
5. **caratteri speciali** (es.: *f* per l'avanzamento pagina)
6. **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();

Esempio di variabile con oggetto

•Caratteri speciali

Tra le stringhe occorre indicare

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	"	\"

un esempio di codice speciale

•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:

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

Parte del lavoro del programma CGI che funzionano 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.

•Dichiarazione 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 vengono 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 darle 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 (es.: 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 perchè, 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:

1. 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;
2. 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:

1. possono contenere solo lettere, numeri e trattino di sottolineatura, per cui sono esclusi gli spazi bianchi;
2. 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;
3. 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;
4. 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:

1. 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*;
2. 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*).

•Tipi di variabili

chiave che non si possono

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	

Le parole chieve riservate, ma

•Tipi di variabili

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

1. ottimizzare l'uso della memoria;
2. 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:

1. gli apici per convertire un valore numerico in stringa o la somma di un numero con uno spazio;
2. metodo **String()** (da Javascript 1.2) per convertire in stringa o con il costruttore **String()** (es.: miastringa=String(num));
3. le funzioni *eval()*, *parseInt()* e *parseFloat()* per convertire un valore stringa in numerico;
4. 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.

La funzione **eval** è abbastanza complessa e cerca qualsiasi espressione Javascript valida per trasformarla in numerico. In Netscape 2.0, però, provoca cadute improvvise:
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 carattere="Salve"
cliccare sui bottoni per verificare il tipo

Variabile pro	Variabile nu	Variabile car

con l'operatore typeof si avrà
typeof prova restituirà object
typeof numero restituirà number
typeof carattere 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. Nella mia esperienza personale ho utilizzato sempre il passaggio tra frame o tra finestre, ma ho trovato ultimamente molto comodo il passaggio tramite URL, anche se ho trovato delle 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
per indicare: "prendimi il valore della variabile tasto contenuto nella finestra che ha aperto la finestra secondaria".

Clicca su uno dei tre tasti e poi clicca qui per verificare

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:

```
<A HREF="prova25b.htm?tasto">
```

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.

[Premere uno qualsiasi dei tre tasti e cliccare qui (fare attenzione all'URL della pagina successiva](#)

•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

Operatore	**Si**		**A**
Incremento	++		
Decremento	--		
Meno unario	-		

Gli operatori binari matematici **non cambiano il valore degli operandi, ma memorizzano il**

Operatore	**Si**	**Azio**	
Addizione	+		x=10;y
Sottrazione	-		
Moltiplicazione	*		

x=10;
Divisione / Di
Resto (modulo) F
non è possibile, però, % or
Scrittura compatta ni **Sc**
x += y sc x
x -= y e x
x *= y il x
x /= y re x
x %= y x
Con **operatore** =
Operatore **A**
> M
>= M
< Mi
<= Mi
== U
!= Di
Gli operatori ve

Operatore	**Si**	**Si**
AND	&	A
OR	|	O
AND	&	A
OR	| |	O
XOR	^	O
NOT	!	N

La negazione (!) eg

X	Y	X	X| |		X^Y	!X
0	0	0	0	az	0	1
0	1	0	1	io	1	1
1	0	1	1	ne	0	0
1	1		0		1	0

Questi operatori sono

Operatore	**Azi**
&	AND
|	OR
^	XOR
~	Co
>>	Shif
<<	Shif
>>>	Shif

0
mpl
eme
nto
a
uno

Le tabelle di verità sono le stesse degli operatori logici.
Si può pensare

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

•Precedenza degli operatori

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

•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 vengono 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.

•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 facendo attenzione alla precedenza degli operatori (per tale motivo guardare la lezione relativa a questo 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...else

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.
Alternativa a questa disposizione e 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."

per cui se x è maggiore di 9 ad y sarà assegnato il valore 100, altrimenti gli sarà assegnato il valore 200.
Un esempio molto elegante di operatore ternario è quello che serve a determinare il browser utilizzato:
ie=document.all?1:0

n=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 **n** uguale ad 1.

- For

Un altro potente strumento di iterazione di istruzioni è il loop **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) istruzione;*
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 e' 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;
}
```

ricordiamo che le matrici si iniziano a contare da 0 per cui quando il contatore assume valore 10 il ciclo non è ripetuto.
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 posseggono 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 e do ... While

Altro controllo di flusso si può avere con:

while (*condizione*) *espressione*

che esegue l'espressione finchè la condizione è vera.
Per creare un ciclo infinito può andare molto bene l'istruzione:
while (true) {...}
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.

- 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'if
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.

```
while (x<10) {
x++;
if (x>3)
continue;
a+=x;
}
```

L'esempio mostra come il ciclo si ripete finchè x è minore di 10, mentre se è uguale a 8 l'istruzione *continue* interrompe il ciclo e riprende dall'inizio.