

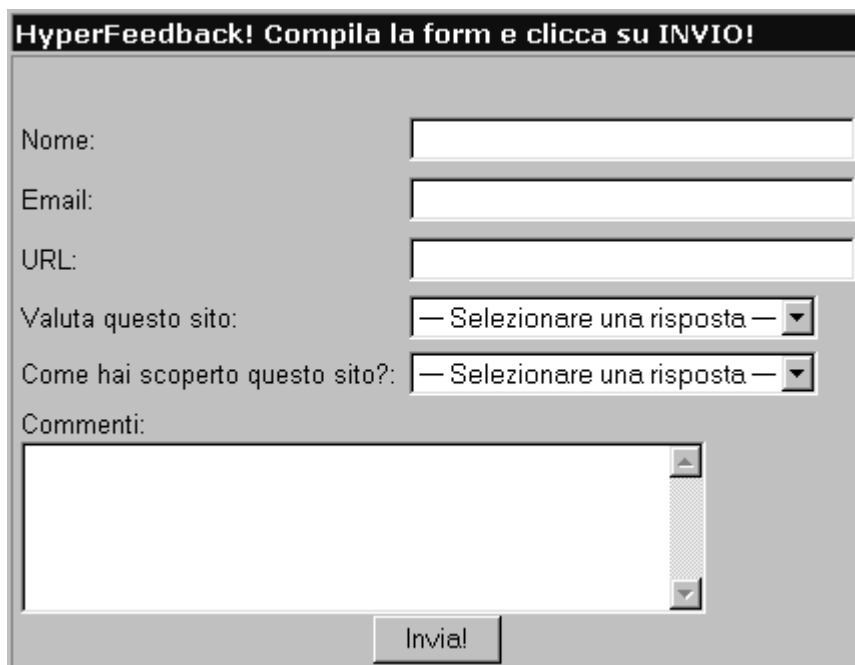
# Appunti di Sistemi Informativi

## Visual Basic Script

<i>Introduzione ai “linguaggio di script”</i> .....	2
<i>Script CGI</i> .....	3
<i>Generalità su VBScript</i> .....	5
<i>Aggiunta di un codice VBScript ad una pagina</i> .....	6
<i>Esecuzione degli script</i> .....	7
Posizionamento dei blocchi script in una pagina HTML .....	9
<i>Variabili in VBScript</i> .....	12
Dichiarazione di variabili.....	12
Area di validità e vita utile delle variabili .....	13
Istruzione DIM .....	14
Istruzione Public .....	14
Istruzione Private .....	15
Istruzione SET.....	15
Assegnazione di valori alle variabili .....	16
Variabili scalari e vettoriali .....	17
<i>Descrizione dei tipi di dati di VBScript</i> .....	19
Sottotipi di variabili Variant .....	19
<i>Controllo dell'esecuzione del programma</i> .....	23
Istruzione If...Then...Else.....	23
Istruzione Select Case .....	24
<i>Utilizzo di cicli per la ripetizione del codice</i> .....	25
Cicli Do .....	26
Uscita da un'istruzione Do...Loop.....	27
<i>Istruzione For...Next</i> .....	28
Istruzione For Each...Next .....	29
<i>Costanti in VBScript</i> .....	20
<i>Operatori in VBScript</i> .....	21
<i>Routine di VBScript</i> .....	31
<i>Convenzioni di scrittura del codice</i> .....	33
Convenzioni di denominazione delle costanti .....	34
Convenzioni di denominazione delle variabili .....	34
Prefissi per l'area di validità delle variabili .....	35
Nomi di routine e variabili descrittivi .....	35
Convenzioni di denominazione degli oggetti.....	36
Convenzioni di scrittura dei commenti .....	36
Formattazione del codice .....	38
<i>VBScript ed i form</i> .....	38
Utilizzo di valori numerici .....	<b>Errore. Il segnalibro non è definito.</b>
Convalida e restituzione di dati al server.....	<b>Errore. Il segnalibro non è definito.</b>
<i>Modalità alternative per l'associazione di codice a eventi</i> .....	40
Messaggi.....	41
Finestre di dialogo .....	45
<i>Utilizzo di VBScript con gli oggetti (cenni)</i> .....	45

## Introduzione ai "linguaggio di script"

Durante la navigazione nel Web, capita spesso di trovare, nelle pagine visitate, *contenuti interattivi* che gli sviluppatori di pagine Web hanno creato utilizzando i cosiddetti script. Per comprendere di cosa si tratta, possiamo partire dalle seguenti considerazioni: immaginiamo di avere davanti, sul nostro monitor, una pagina Web comprendente una scheda (più tecnicamente parliamo di form) da compilare. Ad esempio, una tipica scheda è quella che riempiamo quando visitiamo un sito e il Webmaster ci invita a fornire un giudizio (feedback) sul sito stesso o, più semplicemente, quando indichiamo il nostro indirizzo di posta elettronica per iscriverci ad una mailing list (nel qual caso la scheda è composta da un unico campo).



The image shows a web form with a dark header bar containing the text "HyperFeedback! Compila la form e clicca su INVIO!". Below the header, there are several input fields: "Nome:" with a text box, "Email:" with a text box, "URL:" with a text box, "Valuta questo sito:" with a dropdown menu showing "— Selezionare una risposta —", and "Come hai scoperto questo sito?:" with a dropdown menu showing "— Selezionare una risposta —". Below these is a "Commenti:" label and a large text area with a vertical scrollbar. At the bottom center of the form is a button labeled "Invia!".

*Figura 1 – Esempio di form che l’utente può compilare per fornire il proprio giudizio sul sito appena visitato.*

Quando un utente riempie una scheda e clicca sul tasto per l’inoltro della scheda stessa (ad esempio il tasto *Invia!* della figura 1) , il contenuto della scheda arriva ad un programma che risiede sul server <sup>(1)</sup>; questo programma elabora i dati ricevuti e, in molti casi, crea dinamicamente un documento HTML che viene restituito all’utente: ad esempio, nel caso di un feedback come quello di figura 1, il server potrebbe restituire all’utente un documento HTML con la semplice scritta “*Grazie per aver riempito il*

---

<sup>1</sup> Il server, in questo contesto, è semplicemente il computer che ospita le pagine del sito in questione

*questionario*”, oppure potrebbe fare qualcosa di leggermente più complesso, come ad esempio inviare quella stessa frase all’utente tramite un messaggio di posta elettronica.

Il programma in questione non è altro che una CGI (*Common Gateway Interface*) e può essere scritto con un qualsiasi linguaggio di programmazione come il C , C++ o Perl.

La caratteristica delle CGI è dunque quella di essere programmi eseguiti sul server, per il quale perciò costituiscono un carico elaborativo addizionale. Ciò significa, ad esempio, che la restituzione della pagina HTML all’utente potrebbe non essere immediata, ma richiedere un certo tempo (funzione del carico complessivo del server).

Successivamente alla nascita delle CGI, si è presentata la necessità di ideare un linguaggio che potesse essere eseguito direttamente dal browser dell’utente, al fine di alleggerire il carico dei server e garantire così migliori requisiti di real time nell’interazione dell’utente con il Web: l’idea fu quella di inserire i programmi scritti con questo linguaggio (i cosiddetti script) direttamente all’interno del file HTML, in modo tale che l’esecuzione avvenisse sul client e non sul server.

Tramite gli script possono essere eseguite molteplici operazioni, anche se, in effetti, essi vengono utilizzati soprattutto per definire i comportamenti degli oggetti al verificarsi di un certo evento: ad esempio, si può modificare un paragrafo o un'immagine al passaggio del mouse su una determinata locazione del documento e così via. Il concetto di “evento” è molto importante per questo tipo di linguaggio e quindi ne parleremo diffusamente in seguito.

La prima realizzazione concreta di script interpretabili direttamente sul client fu Javascript, creato dalla Sun Microsystems in collaborazione con Netscape. Successivamente, così come Netscape ha reso disponibile JavaScript attingendo alle funzionalità principali di Java, Microsoft ha fatto lo stesso con il Visual Basic, creando il Microsoft Visual Basic Scripting, più conosciuto come VBScript.

## **Script CGI**

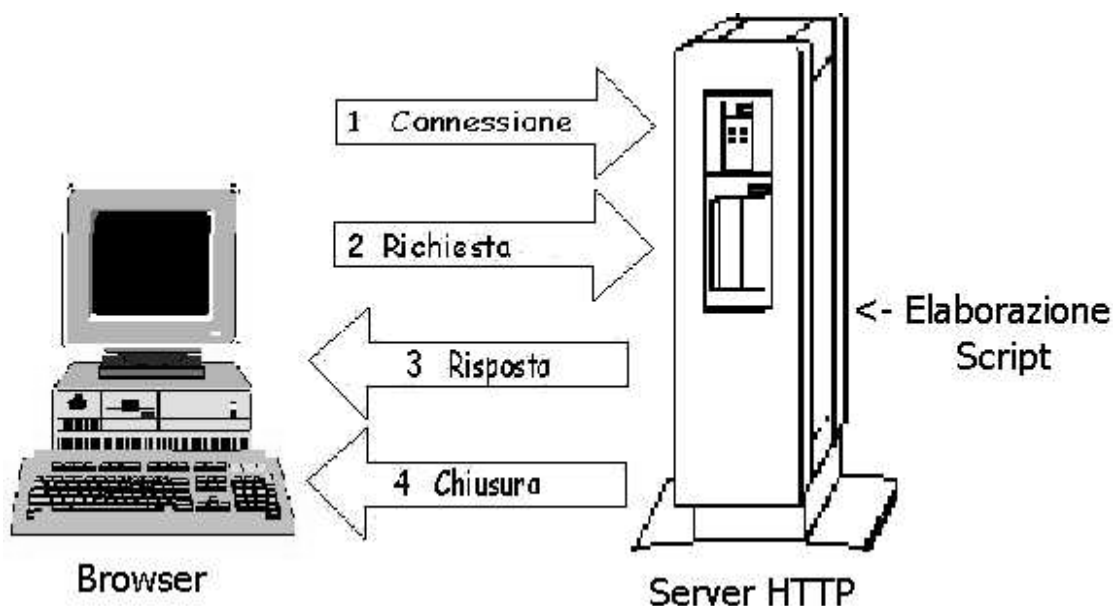
Occupiamoci in primo luogo degli script CGI. Come si è detto, uno script CGI è un programma (scritto in qualsiasi linguaggio di programmazione) che risiede sul server e che elabora l'input dell'utente e opzionalmente crea una risposta tramite un documento HTML che il server invia al browser.

L'utilità delle CGI è generalmente quella di effettuare delle operazioni su Data Base o anche di semplificare le ricerche dell'utente raggruppando dinamicamente le

informazioni. Ad esempio, durante la navigazione sarà capitato a tutti di usare motori di ricerca (come Virgilio, Altavista, Yahoo) o visitare siti Web che richiedono l'inserimento di dati personali per poi inviarli al server. In tutti questi casi, lo script eseguito sul server elabora le informazioni e invia codice formattato HTML da far visualizzare al browser, come ad esempio la pagina contenente l'esito della ricerca, la conferma di successo per il servizio richiesto e altro.

Tramite l'interfaccia CGI, il server può assegnare il compito di creare dinamicamente documenti Web ad un'applicazione creata in modo specifico per rispondere a determinate esigenze. Il programma può essere creato con un linguaggio di programmazione come JavaScript, Perl, C, C++, VBScript e molti altri.

Quando un browser comunica con un server, il programma esegue una connessione HTTP costituita da quattro passi, rappresentati nella figura qui di seguito insieme al posizionamento degli script CGI:



*Figura 2 - Schema di principio del dialogo, secondo il protocollo HTTP, di un client con un server: il client, una volta stabilita la connessione con il server, richiede l'esecuzione di uno script al server stesso, il quale esaudisce la richiesta e, eventualmente, restituisce una risposta, dopodiché chiude la connessione HTTP*

Come si può vedere, gli script risiedono sul server, in cui le applicazioni possono comunicare direttamente fra loro. Questa comunicazione diretta consente ad uno script CGI di ricevere i dati dinamicamente e di passare i dati al server.

Quando un browser vuole prelevare dati prodotti da uno script, si comporta nel modo seguente: innanzitutto, si connette al server e invia la sua richiesta; il server, in base a tale richiesta, richiama lo script coinvolto e gli fornisce tutti i dati provenienti

dal client, nonché i valori delle variabili d'ambiente; lo script, a questo punto, esegue tutta l'elaborazione dei dati e produce l'output richiesto, "consegnandolo" al server; quest'ultimo aggiunge, se necessario, qualche informazione di intestazione e invia tutto al browser e al termine chiude la connessione.

Normalmente, gli script producono un output che poi il server interpreta e invia al browser. In tal modo, il vantaggio è che lo script non è obbligato ad inviare un'intestazione HTTP completa per ogni richiesta. Altri script invece, inviano il loro output direttamente al browser per evitare di sovraccaricare ulteriormente il server. In altre parole, il server non interpreterà l'intestazione dello script, ma sarà compito dello script inviare al browser una risposta HTTP corretta. La distinzione tra i due tipi di Script è fatta dal protocollo CGI, il quale richiede che gli *script con output diretto* inizino con le lettere *nph-* (Not to Parse the Header).

Tutte le comunicazioni che avvengono tra client e server si svolgono notoriamente in accordo al protocollo HTTP.

## Generalità su VBScript

I linguaggi di script sono presenti nel campo informatico già da prima dell'avvento del Web. In ambiente Unix, ad esempio, erano utilizzati per lo svolgimento di compiti ripetitivi connessi all'amministrazione del sistema oppure per l'automatizzazione di alcune procedure di elaborazione.

Oggi i linguaggi di script possono essere utilizzati direttamente all'interno di una pagina HTML. In particolare, noi siamo qui interessati al Visual Basic Script, meglio conosciuto come VBScript.

Cos'è VBScript? Tempo fa, Microsoft creò Visual Basic, un tool di sviluppo scalare per le applicazioni Internet e non solo. Visual Basic Script, meglio conosciuto come VBScript, è tra i più nuovi membri della famiglia del linguaggio di programmazione Visual Basic.

VBScript è sostanzialmente un piccolo sottoinsieme delle caratteristiche e delle librerie di Visual Basic. Esistono due motivi fondamentali per questa riduzione:

- tempi di elaborazione minimi: l'obiettivo della Microsoft nello sviluppo di VBScript era di offrire un tool di sviluppo per la programmazione di pagine Web che fosse compatibile con lo standard Microsoft per lo sviluppo di applicazioni per Windows, ovvero *Visual Basic*; questo nuovo linguaggio, dovendo essere incluso nelle pagine Web, doveva subire una notevole "riduzione" tale da raggiungere la rapidità di interpretazione di JavaScript (che era lo standard allora più diffuso per

la realizzazione di pagine Web interattive). Per ottenere ciò, sono state rimosse tutte le funzioni meno utilizzate di Visual Basic;

- sicurezza: VBScript, a differenza di Visual Basic, non può richiamare le API di Windows o realizzare file di Input / Output; questo per evitare che un hacker possa realizzare pagine in grado di bloccare o danneggiare il sistema che le va a scaricare.

Il codice VBScript, come JavaScript, è usato per aggiungere intelligenza ed interattività ai documenti HTML. Per i milioni di programmatori che già conoscono Visual Basic, VBScript è una valida alternativa a JavaScript nel tentativo di rendere attive le pagine web. Esso è stato progettato in modo da renderne facile ed intuitivo l'apprendimento, specialmente nei confronti dei principianti (la sigla BASIC sta appunto per *Beginners All-purpose Symbolic Instruction Code*, cioè codice di istruzioni simboliche multiuso per principianti). Rispetto a JavaScript, VBScript appare molto più versatile e facile da utilizzare, anche se presenta un grande svantaggio: solo Microsoft Internet Explorer, a partire dalla quarta edizione, lo supporta completamente, mentre altri browser, come Netscape Communicator, riescono a compilare solamente il linguaggio Java. A fronte di questo svantaggio, VBScript è molto più completo (in quanto dispone di oltre 80 funzioni predefinite contro le sole 2 del JavaScript) e molto più versatile (ad esempio non fa distinzione tra lettere maiuscole e minuscole, al contrario dell'altro linguaggio, dove l'intero script non potrebbe funzionare per una semplice svista).

A partire dal prossimo paragrafo ci occuperemo dunque dettagliatamente di VBScript. E' utile allora ricordare che, installando il Microsoft Personal Web Server (accessorio standard di Windows 98) sul proprio computer, viene installata anche un'ampia guida rapida (della quale tra l'altro sono qui ripresi molti concetti) nella directory C:\WINDOWS\HELP\vbscript\.

## **Aggiunta di un codice VBScript ad una pagina**

Per prima cosa, dobbiamo capire come è possibile aggiungere codice VBScript ad una pagina HTML.

Il codice degli script realizzati in VBScript viene inserito all'interno della pagina Web, indifferentemente nell'intestazione o nel corpo del documento <sup>(2)</sup>, racchiuso tra due tag, <script> e </script>:

```
<SCRIPT LANGUAGE="VBScript">
<!--
    codice VBScript
-->
</SCRIPT>
```

Nell'attributo language viene specificato appunto che si utilizza VBScript, esattamente come si fa con JavaScript.

I tag di commento <!-- e --> vengono inseriti in modo da non far eseguire lo script dai browser che non supportano questo linguaggio: in questo modo, l'eventuale browser che non supporta VBScript salta il codice a piè pari e bisogna ovviamente tenerne conto, in quanto tutte le funzionalità svolte dallo script vengono ignorate. Ad esempio, è possibile usare VBScript per dire al browser di caricare una nuova pagina non appena finisce di caricare quella attuale; se però il browser non supporta VBScript, la nuova pagina non sarà mai caricata automaticamente e quindi un programmatore attento dovrà evitare di inserire in quest'ultima informazioni fondamentali, che l'utente potrebbe non vedere mai.

Tra i due tag <SCRIPT> e </SCRIPT> va inserito il codice VBScript desiderato. Non vanno invece inseriti altri tag dell'HTML.

Quando il browser apre la pagina, compila tutti i tag e gli script, traducendoli in un linguaggio comprensibile alla macchina, che quindi li può eseguire.

## Esecuzione degli script

Uno script (a prescindere che sia scritto in VBScript o Javascript o altro) può essere eseguito in occasioni diverse:

- il caso più semplice è quello di eseguirlo direttamente all'apertura del documento;
- un caso più sofisticato (ma altrettanto semplice) è invece quello di eseguirlo solamente al verificarsi di un certo evento;

---

<sup>2</sup> Su questo aspetto torneremo approfonditamente in seguito

- un caso ulteriore è quello di usare una chiamata inserita all'interno di un altro tag.

Gli script più semplici vengono dunque eseguiti automaticamente al caricamento della pagina. Ad esempio, si può realizzare un semplice script che visualizzi l'ora corrente sulla prima riga della pagina Web: in questo caso, basterà inserire il codice dello script nelle prime righe della parte inclusa nel tag <BODY>.

Per eseguire invece uno script al verificarsi di un certo evento, si possono utilizzare vari metodi. Se, ad esempio, si vuole eseguire un determinato script al passaggio del mouse sopra un'immagine, che abbiamo denominato *Img*, basta modificare l'intestazione dello script, inserendo l'oggetto di interesse (l'immagine) e l'evento di interesse (passaggio del mouse), come riportato qui di seguito:

```
<script FOR="Img" EVENT="OnMouseOver" language="VBScript">
<!--
    codice VBScript
-->
</script>
```

Come si vede, l'attributo FOR indica appunto il nome dell'oggetto che gestisce lo script, mentre invece l'evento OnMouseOver, determina la condizione per cui deve essere eseguito lo script. Gli eventi possibili sono diversi e ne parleremo più diffusamente in seguito.

Un altro metodo per richiamare l'esecuzione di script è quello di utilizzare delle routine, cioè dei sottoprogrammi inseriti nella pagina HTML tramite l'istruzione Sub. Questi sottoprogrammi si distinguono in routine Sub e routine di funzione: entrambe queste tipologie possono essere eseguite al verificarsi di un certo evento, ma vengono utilizzate delle procedure diverse per determinare ciò.

Ad esempio, se avessimo voluto realizzare lo script precedente utilizzando una *routine Sub*, avremmo potuto impostare lo script nel seguente modo:

```
<script language="VBScript">
<!--
    Sub Img_onmouseover()
        codice VBScript
    End Sub
```

```
-->
</script>
```

In pratica, il nome dell'oggetto (*Img*) e l'evento relativo (*OnMouseOver*) vengono uniti tramite il carattere `_` (underscore). Bisogna però aggiungere le doppie parentesi tonde “`()`”, in quanto si tratta pur sempre di una routine e le routine, come si vedrà, hanno la possibilità di ricevere in ingresso dei parametri da usare nell'esecuzione. L'uso delle doppie parentesi tonde `()` senza niente all'interno indica che non ci sono parametri da passare.

In alternativa, sempre usando una routine `Sub`, potremmo procedere nel modo seguente: in primo luogo, si deve attribuire un nome alla routine, nella sua intestazione; in secondo luogo, la si deve richiamare direttamente all'interno del tag dell'oggetto che la gestisce. Ad esempio, volendo eseguire uno script quando si passa col mouse sopra un'immagine, si può utilizzare la seguente sintassi:

```
<script language="VBScript">
<!--
  Sub nomeFunzione()
    codice VBScript
  End Sub
-->
</script>

<BODY>
<imm src="immagine.gif" OnMouseOver="nomeFunzione()">
</BODY>
```

In pratica, la routine viene definita (in termini di nome e codice corrispondente) nella parte `HEAD` del documento (o anche nella parte `BODY`, è indifferente) e poi viene richiamata nel tag che gestisce l'immagine; in particolare, essa viene richiamata in corrispondenza dell'evento `OnMouseOver`.

## **Posizionamento dei blocchi script in una pagina HTML**

Dato che abbiamo parlato delle routine come metodo di esecuzione degli script, possiamo introdurre un altro concetto importante in merito, ossia il criterio con cui conviene posizionare i blocchi `VBScript` all'interno di un documento `HTML`.

In generale, come già detto, è possibile utilizzare i blocchi SCRIPT in qualsiasi punto della pagina HTML, sia nella sezione BODY che nella sezione HEAD. Tuttavia, spesso è opportuno scegliere il punto di inserimento in base alle funzionalità ed agli scopi dello script.

Ad esempio, quando un blocco script corrisponde a una funzione generale, cioè non collegata ad alcun evento (ad esempio *OnMouseOver*) o oggetto (ad esempio una form) esterno, è possibile includerlo, tramite una routine, nella sezione HEAD della pagina:

```
<HTML>
<HEAD>
<TITLE>Immissione ordine</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
    Function Data(Dt)
        varData = (CDate(Dt))
    End Function
-->
</SCRIPT>
</HEAD>
<BODY>
.....
```

In questo modo, inserendo tutto il codice script di utilizzo generale nella sezione HEAD, si rende più leggibile il documento ed inoltre si consente al browser di "decifrare" tutto il codice prima che esso venga utilizzato da una chiamata all'interno della sezione BODY.

Nel caso, invece, uno script sia ad esempio strettamente legato ad una form, è consigliabile sistemarlo direttamente insieme alla form. È, ad esempio, possibile incorporare codice script per rispondere alla pressione di un pulsante in un form:

```
<HTML>
<HEAD>
<TITLE>Verifica eventi dei pulsanti</TITLE>
</HEAD>
<BODY>
<FORM NAME="Form1">
    <INPUT TYPE="Button" NAME="Bottone1" VALUE="Click">
```

```

<SCRIPT FOR="Bottone1" EVENT="onClick" LANGUAGE="VBScript">
    MsgBox "Pulsante premuto"
</SCRIPT>
</FORM>
</BODY>
</HTML>

```

In questo esempio, ogniqualvolta l'utente clicca sul bottone denominato *Bottone1*, il codice VBScript fa comparire una finestra con il messaggio "Pulsante Premuto".

A questo punto, si può intuire la differenza fondamentale tra i vari modi di procedere: quando il codice VBScript è incluso in una routine, esso viene richiamato tutte e sole le volte in cui il codice scritto ne determinerà l'esecuzione; se invece il codice VBScript viene sistemato esternamente alle routine (ma sempre all'interno di un blocco SCRIPT) e senza essere vincolato al verificarsi determinati eventi, esso verrà eseguito una sola volta, quando la pagina HTML viene caricata. In questo secondo caso, è possibile inizializzare i dati o modificare dinamicamente l'aspetto della pagina Web quando viene caricata.

Una situazione in qualche modo atipica è quella in cui si usa l'evento OnLoad inserito nel tag <BODY>:

```

<SCRIPT LANGUAGE="VBScript">
<!--
    Sub Nome_Funzione()
        Codice VBScript
    End Sub
-->
</SCRIPT>
.....
<BODY onLoad="Nome_Funzione()">
.....
</BODY>

```

Con una sintassi di questo tipo, la routine denominata *Nome\_Funzione* viene eseguita non appena il browser ha terminato di caricare la pagina. Si tratta dunque di un caso in cui l'esecuzione dello script avviene automaticamente al caricamento della pagina, con una differenza, però, rispetto a quanto detto in precedenza circa queste

situazioni: infatti, lo stesso script, essendo racchiuso in una routine, può anche essere eseguito altre volte tramite apposite chiamate in altri punti della pagina stessa.

## Variabili in VBScript

Una variabile è un *puntatore* (o *segnaposto* o *identificativo* che dir si voglia) ad una posizione della memoria del computer in cui è possibile memorizzare informazioni di programma soggette a modifiche durante l'esecuzione degli script. La posizione delle variabili in memoria non è importante dal punto di vista dell'utente. Per visualizzarne o modificarne il valore, è sufficiente farvi riferimento tramite il nome.

Come ogni altro linguaggio di programmazione, VBScript è in grado di memorizzare ed elaborare dati (o informazioni) di vari tipi. A differenza degli altri linguaggi, però, in VBScript esiste ufficialmente un solo tipo di variabile, denominato Variant, che può contenere diversi tipi di informazioni a seconda del modo in cui viene utilizzata: questo significa che VBScript riconoscerà automaticamente di che tipo dovrà essere la variabile utilizzata, senza definirla in precedenza.

Come vedremo, i "sottotipi" che può assumere una variabile del tipo Variant sono molti, ma generalmente vengono utilizzate solo variabili numeriche (di tipo intero e reale) e stringhe di caratteri, e più raramente le date.

## Dichiarazione di variabili

Generalmente non è necessario dichiarare le variabili. Potrebbe però rivelarsi necessario quando si utilizzano più *routine* all'interno della stessa pagina: se vogliamo che quelle variabili siano accessibili a tutte le routine, dovremo dichiararle all'inizio dello script utilizzando l'istruzione Public:

```
Public nome_variabile
```

Se, invece, volessimo rendere accessibile la variabile ad una sola routine, bisognerà dichiararla all'interno di essa tramite l'istruzione Private.

Per dichiarare più variabili, è necessario separare i vari nomi con una virgola. Ad esempio:

*Dim MiaVariabile, TuaVariabile, SuaVariabile, NostraVariabile*

È inoltre possibile dichiarare le variabili in modo implicito specificandone semplicemente il nome in un punto qualsiasi dello script. È tuttavia consigliabile non adottare questo metodo, in quanto è possibile inserire errori di ortografia nel nome della variabile in una o più posizioni dello script e ottenere di conseguenza risultati imprevisti.

Esistono precise regole da adottare per la scelta del nome di una variabile:

- deve iniziare con una lettera dell'alfabeto.
- non può includere punti.
- non deve essere composto da più di 255 caratteri.
- deve essere univoco nell'area di validità in cui la variabile è dichiarata.

## **Area di validità e vita utile delle variabili**

L' area di validità di una variabile varia a seconda della posizione in cui la variabile viene dichiarata:

- quando si dichiara una variabile in una routine, è possibile accedere o modificare il corrispondente valore solo tramite il codice di tale routine. Si tratta infatti di una variabile a livello di routine con area di validità locale;
- se invece una variabile viene dichiarata all'esterno di una routine, essa sarà riconosciuta in tutte le altre routine dello script. A questo tipo di variabile, definita a livello di script, è associata un'area di validità a livello di script.

La vita utile di una variabile corrisponde alla sua durata. Per *una variabile a livello di script* la vita utile è compresa tra la dichiarazione e la conclusione dell'esecuzione dello script. A livello di routine, invece, le variabili esistono solo mentre è in esecuzione la routine in cui sono dichiarate. Al termine dell'esecuzione della routine la variabile viene eliminata.

In più routine è possibile includere variabili locali aventi lo stesso nome in quanto ciascuna di esse viene riconosciuta solo dalla routine in cui è dichiarata.

## Istruzione DIM

Questa istruzione consente di dichiarare variabili e assegnare lo spazio di memorizzazione corrispondente. Le variabili dichiarate con l'istruzione Dim a livello di script sono disponibili per tutte le routine incluse nello script. Se dichiarate a livello di routine, le variabili sono disponibili solo nella routine.

La sintassi completa è la seguente:

*Dim nomevar* [ ( [ *indici* ] ) ] [, *nomevar* [ ( [ *indici* ] ) ] ] . . .

Sono qui individuabili le seguenti parti:

Parte	Descrizione
<i>nomevar</i>	Nome della variabile espresso in base alle convenzioni di denominazione standard delle variabili.
<i>indici</i>	<i>Dimensioni</i> di una variabile matrice <sup>(3)</sup> . È possibile dichiarare fino a 60 dimensioni multiple. La sintassi dell'argomento <i>indici</i> è la seguente:  <i>limitesuperiore</i> [, <i>limitesuperiore</i> ] . . .  Il limite inferiore di una matrice è sempre uguale a zero.

Segnaliamo che, qualunque istruzione di dichiarazione venga usata (Dim, Public, Private), le variabili numeriche sono sempre inizializzate su 0, mentre le stringhe sono inizializzate su una stringa di lunghezza zero ("").

## Istruzione Public

E' utilizzata a livello di script per dichiarare variabili pubbliche e assegnare spazio di memorizzazione. La sintassi completa è la seguente:

*Public nomevar*[ ( [ *indici* ] ) ] [, *nomevar* [ ( [ *indici* ] ) ] ] . . .

La sintassi dell'istruzione Public è composta dalle seguenti parti:

Parte	Descrizione
<i>nomevar</i>	Nome della variabile specificato in base alle convenzioni di denominazione standard delle variabili.

<sup>3</sup> Ne parliamo dettagliatamente più avanti

<i>indici</i>	<p>Dimensioni di una variabile matrice; è possibile dichiarare fino a 60 dimensioni multiple. La sintassi dell'argomento <i>indici</i> è la seguente:</p> <p><i>limitesuperiore</i>[,<i>limitesuperiore</i>] . . .</p> <p>Il limite inferiore di una matrice è sempre zero.</p>
---------------	---

Le variabili dichiarate utilizzando l'istruzione `Public` sono disponibili nelle routine di tutti gli script in tutti i progetti.

Per poter utilizzare le variabili che fanno riferimento a un oggetto, è prima necessario assegnare a tali variabili un oggetto esistente, utilizzando l'istruzione `Set` di cui parleremo tra poco. Fino al momento dell'assegnazione di un oggetto, il valore della variabile oggetto dichiarata è `Nothing`.

### ***Istruzione Private***

Utilizzata a livello di script per dichiarare variabili private e assegnare spazio di memorizzazione. La sintassi completa è la seguente:

*Private nomevar* [ ( [ *indici* ] ) ] [, *nomevar* [ ( [ *indici* ] ) ] ] . . .

La sintassi dell'istruzione `Private` è composta dalle seguenti parti:

Parte	Descrizione
<i>nomevar</i>	Nome della variabile specificato in base alle convenzioni di denominazione standard delle variabili.
<i>indici</i>	<p>Dimensioni di una variabile matrice; è possibile dichiarare fino a 60 dimensioni multiple. La sintassi dell'argomento <i>indici</i> è la seguente:</p> <p><i>limitesuperiore</i>[, <i>limitesuperiore</i>] . . .</p> <p>Il limite inferiore di una matrice è sempre zero.</p>

Le variabili `Private` sono disponibili soltanto nello script in cui vengono dichiarate.

Anche in questo caso, per poter utilizzare le variabili che fanno riferimento a un oggetto, è prima necessario assegnare a tali variabili un oggetto esistente utilizzando l'istruzione `Set`.

### ***Istruzione SET***

Questa istruzione assegna un riferimento di oggetto a una variabile o *proprietà* (4). La sintassi completa è la seguente:

*Set varoggetto = { espressioneoggetto | Nothing}*

La sintassi dell'istruzione Set è composta dalle seguenti parti:

Parte	Descrizione
<i>varoggetto</i>	Nome della variabile o proprietà espresso in base alle convenzioni di denominazione standard delle variabili.
<i>espressioneoggetto</i>	Espressione costituita dal nome di un oggetto, da un'altra variabile dichiarata dello stesso <i>tipo di oggetto</i> oppure da una funzione o da un metodo che restituisce un oggetto dello stesso tipo.
Nothing	Annulla l'associazione dell'argomento <i>varoggetto</i> a un oggetto specifico. Assegnando <i>varoggetto</i> al valore Nothing, vengono liberate la memoria e le risorse di sistema associate all'oggetto a cui è stato fatto riferimento precedentemente, a condizione che nessun'altra variabile faccia riferimento a tale oggetto.

Per essere valido, l'argomento *varoggetto* deve essere un oggetto dello stesso tipo dell'oggetto che vi viene assegnato.

Le istruzioni Dim, Private, Public e ReDim consentono semplicemente di dichiarare variabili che fanno riferimento a un oggetto. L'assegnazione di un oggetto viene eseguita solo quando si utilizza l'istruzione Set indicando un oggetto specifico.

Quando si utilizza l'istruzione Set per assegnare un riferimento di oggetto a una variabile, in genere non viene creata alcuna copia dell'oggetto per la variabile, ma un riferimento a tale oggetto. Più variabili oggetto possono fare riferimento allo stesso oggetto. Dato che queste variabili sono riferimenti e non copie dell'oggetto, le modifiche apportate all'oggetto si riflettono automaticamente in tutte le variabili che fanno riferimento a tale oggetto.

## Assegnazione di valori alle variabili

Per assegnare valori alle variabili, è necessario creare un'espressione indicando a sinistra la variabile e a destra il valore che si desidera assegnare. Ad esempio:

---

<sup>4</sup> Una **proprietà** è un attributo predefinito di un oggetto. Le proprietà definiscono le caratteristiche degli oggetti, quali le dimensioni, il colore, la posizione sullo schermo, oppure lo stato degli oggetti, ad esempio se sono attivati o meno.

*MiaVariabile = 10*

*MiaVariabile = "Ciao"*

## Variabili scalari e vettoriali

Nella maggior parte dei casi alle variabili dichiarate viene assegnato un solo valore. Una variabile contenente un solo valore è definita variabile scalare.

A volte, tuttavia, risulta utile assegnare a una singola variabile più valori. Una variabile contenente una serie di valori è definita variabile matrice.

La dichiarazione di queste variabili è simile a quella delle variabili scalari, con la sola differenza che, nelle dichiarazioni di *variabili matrici*, il nome della variabile è seguito da parentesi. Nell'esempio seguente, viene dichiarata una matrice a una sola dimensione contenente 11 elementi:

*Dim MiaVariabile(10)*

Anche se tra parentesi è indicato il numero 10, la matrice include in effetti 11 elementi (numerati da 0 a 10). In VBScript infatti le matrici sono sempre in base zero e di conseguenza il numero degli elementi in esse contenuti corrisponde sempre al numero indicato tra parentesi più uno. Queste matrici sono definite a *dimensione fissa*.

Per assegnare dati a ciascun elemento di una matrice, è necessario utilizzare un indice. Nell'esempio seguente vengono assegnati dati numerici ai vari elementi della matrice, usando appunto gli indici:

*MiaVariabile(0) = 256*

*MiaVariabile(1) = 324*

*MiaVariabile(2) = 100*

.....

*MiaVariabile(10) = 55*

Per recuperare dati da un elemento della matrice, è possibile procedere nello stesso modo, ovvero inserire un indice nell'elemento desiderato. Ad esempio:

*AltraVariabile = MiaVariabile(8)*

Le matrici non sono limitate a una sola dimensione. È infatti possibile specificare fino a 60 dimensioni, anche se nella maggior parte dei casi ne vengono specificate solo tre o quattro. Per dichiarare dimensioni multiple, è necessario separare con una virgola i valori delle dimensioni indicati tra parentesi. Nell'esempio seguente la variabile *MiaTabella* è una matrice a due dimensioni composta da 6 righe e 11 colonne:

```
Dim MiaTabella(5, 10)
```

In una matrice a due dimensioni il primo numero corrisponde sempre al numero di righe, il secondo al numero di colonne.

È inoltre possibile dichiarare matrici le cui dimensioni vengono modificate durante l'esecuzione dello script. Questo tipo di matrici sono definite matrici dinamiche e vengono inizialmente dichiarate in una routine utilizzando l'istruzione *Dim* come qualsiasi altra matrice, oppure l'istruzione *ReDim*. La differenza è che tra parentesi non viene indicata alcuna grandezza o numero di dimensioni. Ad esempio:

```
Dim MioArray() ReDim AltroArray()
```

Per utilizzare una matrice dinamica, è necessario specificare più istruzioni *ReDim* in modo da determinare il numero di dimensioni e la grandezza di ciascuna dimensione. Nell'esempio seguente l'istruzione *ReDim* imposta la grandezza iniziale della matrice dinamica su 25. Una successiva istruzione *ReDim* ridimensiona quindi la matrice impostandola su 30 e viene utilizzata la parola chiave *Preserve* per mantenere il contenuto della matrice durante l'operazione di ridimensionamento.

```
ReDim MioArray(25)
```

```
ReDim Preserve MioArray(30)
```

Non esiste alcun limite alla frequenza delle operazioni di ridimensionamento delle matrici dinamiche. È tuttavia importante ricordare che, quando si riducono le dimensioni di una matrice, i dati contenuti negli elementi eliminati vanno perduti.

## Descrizione dei tipi di dati di VBScript

Come si è già detto, in VBScript è disponibile solo il tipo di dati Variant, un tipo di dati speciale che, a seconda della modalità in cui viene utilizzato, può includere vari tipi di informazioni. Il tipo di dati Variant, essendo l'unico disponibile, è il tipo di dati restituito da tutte le funzioni di VBScript.

Nella forma più semplice, una variabile Variant può includere informazioni numeriche o stringhe. Essa è di tipo numerico se viene utilizzata in un contesto numerico, è di tipo stringa se utilizzata in un contesto stringa. Ciò significa che, se si lavora con dati simili a valori numerici, VBScript interpreterà tali dati come numeri e verranno eseguite le operazioni più appropriate. In modo analogo, se si lavora con dati che possono essere solo stringhe, essi verranno gestiti come dati stringa. È tuttavia possibile fare in modo che i numeri vengano gestiti come stringhe racchiudendoli tra apici doppi ("").

### Sottotipi di variabili Variant

Oltre alla semplice classificazione di valore numerico o stringa, con le variabili Variant è possibile definire ulteriori distinzioni in base alla natura specifica delle informazioni numeriche. Se, ad esempio, si utilizzano informazioni numeriche che rappresentano una data o un orario insieme ad altri dati relativi alla data o all'orario, il risultato verrà sempre espresso come *data* o come *orario*.

È tuttavia possibile utilizzare informazioni numeriche di vario tipo, dai valori booleani ai numeri in virgola mobile di grandi dimensioni. Le varie categorie di informazioni che possono essere incluse in variabili Variant sono definite sottotipi.

Nella tabella seguente sono elencati i possibili sottotipi di variabili Variant.

Sottotipo	Descrizione
Empty	Variabile Variant non inizializzata. Il valore è 0 nel caso di variabili numeriche e una stringa di lunghezza zero ("" ) nel caso di variabili stringa.
Null	Variabile Variant che include dati non validi inseriti volutamente.
Boolean	Può assumere valore Vero(True) o Falso(False).

Byte	Contiene un intero compreso tra 0 e 255.
Integer	Contiene un intero compreso tra -32.768 e 32.767.
Currency	Contiene un valore compreso tra -922.337.203.685.477,5808 e 922.337.203.685.477,5807.
Long	Contiene un intero compreso tra -2.147.483.648 e 2.147.483.647.
Single	Contiene un numero in virgola mobile e precisione singola compreso tra -3,402823E38 e -1,401298E-45 per valori negativi e tra 1,401298E-45 e 3,402823E38 per valori positivi.
Double	Contiene un numero in virgola mobile e precisione doppia compreso tra -1,79769313486232E308 e -4,94065645841247E-324 per valori negativi e tra 4,94065645841247E-324 e 1,79769313486232E308 per valori positivi.
Date (Time)	Contiene un numero che rappresenta una data compresa tra l'1 gennaio dell'anno 100 e il 31 dicembre del 9999.
String	Contiene una stringa di lunghezza variabile composta da un massimo di circa 2 miliardi di caratteri.
Object	Contiene un oggetto.
Error	Contiene un numero di errore.

Per le conversioni tra sottotipi è disponibile un'ampia gamma di funzioni di conversione.

## Costanti in VBScript

Una costante è un nome significativo non soggetto a modifiche utilizzato in sostituzione di un numero o di una stringa.

Per creare costanti in VBScript, è necessario utilizzare l'istruzione Const con cui vengono create costanti stringa o numeriche con nomi significativi, che è quindi possibile associare a valori letterali e utilizzare negli script. Ad esempio:

```
Const MiaStringa = "Questa è la mia stringa."
```

```
Const MioNumero = 49
```

Nell'esempio, il valore letterale stringa è racchiuso tra virgolette doppie ("), il modo più chiaro per contraddistinguere i valori stringa dai valori numerici. È possibile rappresentare i valori letterali di data e ora racchiudendoli tra simboli di cancelletto (#); ad esempio:

```
Const LaData = #6-1-97#
```

È consigliabile adottare una convenzione di denominazione in modo da contraddistinguere le costanti dalle variabili ed evitare quindi di riassegnare valori alle costanti durante l'esecuzione dello script.

## Operatori in VBScript

Come per i programmi scritti in Visual Basic, anche per gli script l'elaborazione dei dati è una delle attività principali. Le operazioni matematiche, per esempio, vengono usate per calcolare importi, sconti ecc. Un insieme di operazioni da svolgere prende il nome di espressione.

Quando in un'espressione si eseguono varie operazioni, ciascuna parte viene valutata e risolta secondo un ordine specifico definito precedenza tra operatori. Per ignorare l'ordine di precedenza in modo che determinate parti di un'espressione vengano valutate prima di altre, è necessario utilizzare le parentesi. Le operazioni racchiuse tra parentesi vengono sempre valutate per prime. All'interno delle parentesi tuttavia viene mantenuta la normale precedenza tra operatori.

Nelle espressioni che includono operatori di categorie diverse, vengono innanzitutto valutati gli operatori aritmetici, quindi gli operatori di confronto e infine gli operatori logici.

Gli operatori di confronto hanno la stessa precedenza, ovvero vengono valutati da sinistra a destra nello stesso ordine in cui sono stati specificati. Gli operatori aritmetici e logici vengono valutati in base all'ordine di precedenza indicato nella tabella seguente:

Aritmetici		Di confronto		Logici	
Descrizione	Simb	Descrizione	Simb	Descrizione	Simb
Elevamento a potenza	^	Uguaglianza	=	Negazione logica	Not
Negazione unaria	-	Disuguaglianza	<>	Congiunzione logica	And
Moltiplicazione	*	Minore di	<	Disgiunzione logica	Or
Divisione	/	Maggiore di	>	Esclusione logica	Xor
Divisione intera	\	Minore o uguale a	<=	Equivalenza logica	Eqv
Modulo aritmetico	Mod	Maggiore o uguale a	>=	Implicazione logica	Imp
Addizione	+	Equivalenza tra oggetti	Is		
Sottrazione	-				
Concatenamento di stringhe	&				

Se in un'espressione vengono eseguite la moltiplicazione e la divisione oppure l'addizione e la sottrazione, ciascuna operazione verrà valutata nello stesso ordine in cui è indicata da sinistra verso destra.

L'operatore di concatenamento di stringhe (&) non è un operatore aritmetico, ma per quanto riguarda la precedenza segue tutti gli operatori aritmetici e precede tutti gli operatori di confronto.

L'operatore Is è un operatore di confronto per riferimenti a oggetti, ovvero non esegue il confronto degli oggetti o dei corrispondenti valori, ma verifica semplicemente se due riferimenti sono relativi allo stesso oggetto.

## Controllo dell'esecuzione del programma

È possibile controllare il flusso di script tramite istruzioni condizionali e istruzioni di ciclo.

Con le istruzioni condizionali è possibile scrivere codice VBScript che consente di prendere decisioni e ripetere azioni. In VBScript sono disponibili le seguenti istruzioni condizionali:

*If...Then...Else*

*Select Case*

### Istruzione If...Then...Else

L'istruzione If...Then...Else consente di valutare se una condizione è True o False e, a seconda del risultato, di specificare una o più istruzioni da eseguire. In genere, la condizione corrisponde a un'espressione in cui due valori o due variabili vengono confrontati tramite un *operatore di confronto*.

Le istruzioni *If...Then...Else* possono essere nidificate in un numero qualsiasi di livelli di nidificazione.

Per eseguire una sola istruzione quando una condizione è True, è necessario utilizzare la cosiddetta sintassi a riga singola dell'istruzione If...Then...Else, come illustrato nell'esempio riportato di seguito:

```
Sub FissaLaData()
    Dim miaData
    miaData = #2/13/95#
    If miaData < Now Then miaData = Now
End Sub
```

Si noti che, in questo caso, la parola chiave *Else* viene omessa.

Per eseguire più righe di codice, è necessario invece utilizzare la sintassi a righe multiple o a blocco, in cui è inclusa l'istruzione End If, come illustrato nell'esempio riportato di seguito:

```
Sub AvvertiUtente()
    Dim MioForm
```

```
Set MioForm = Document.Form1
valore = MioForm.TxtTest.Value
If valore = 0 Then
    MsgBox "ciao"
    .....
End If
End Sub
```

È possibile utilizzare un'istruzione If...Then...Else per definire due blocchi di istruzioni da eseguire l'uno quando la condizione è True e l'altro quando la condizione è False:

```
Sub AvvertiUtente2()
    Dim MioForm2
    Set MioForm2 = Document.Form2
    valore = MioForm2.TxtTest2.Value
    If valore = 0 Then
        MsgBox "ciao"
    Else
        MsgBox "Buonasera"
    End If
End Sub
```

Per fornire più situazioni diverse, è possibile aggiungere il numero necessario di proposizioni ElseIf. Un utilizzo eccessivo di queste proposizioni può tuttavia creare un codice poco chiaro e contorto. Un metodo alternativo migliore consiste nell'utilizzare l'istruzione Select Case.

## **Istruzione Select Case**

La struttura Select Case rappresenta un'alternativa all'istruzione If...Then...ElseIf per la selezione di un blocco di istruzioni specifico tra più blocchi diversi. La funzione dell'istruzione Select Case è simile a quella dell'istruzione If...Then...Else, con la differenza tuttavia che il codice risulta più efficiente e spesso di più facile lettura.

Una struttura Select Case include una singola espressione di testo valutata una sola volta all'inizio della struttura stessa. Il risultato dell'espressione viene quindi confrontato con i valori di ciascun blocco Case della struttura. Se viene individuata

una corrispondenza, il blocco di istruzioni associato al blocco Case specifico verrà eseguito:

```
Select Case valore
  Case "0"
    MsgBox "ciao"
  Case "2"
    MsgBox "Buonsera"
  Case Else
    MsgBox "Buon Anno"
End Select
```

A differenza della struttura Select Case, con cui un'espressione viene valutata una sola volta all'inizio della struttura stessa, la struttura If...Then...Elseif consente di valutare un'espressione diversa per ciascuna istruzione Elseif. È possibile sostituire una struttura If...Then...Elseif con una struttura Select Case solo se con ciascuna istruzione Elseif viene valutata la stessa espressione.

## Utilizzo di cicli per la ripetizione del codice

L'esecuzione di cicli consente di eseguire ripetutamente un gruppo di istruzioni. In alcuni cicli le istruzioni vengono ripetute fino a quando una condizione risulta False, mentre in altri vengono ripetute fino a quando una condizione risulta True. Esistono inoltre cicli in cui le istruzioni vengono ripetute un numero specifico di volte (*cicli a contatore*).

In VBScript sono disponibili le seguenti istruzioni per l'esecuzione di cicli:

- Do...Loop: le istruzioni vengono ripetute fino a quando una condizione risulta True.
- While...Wend: le istruzioni vengono ripetute fino a quando una condizione risulta True.
- For...Next: le istruzioni vengono ripetute il numero di volte specificato da un contatore.
- For Each...Next: un gruppo di istruzioni viene ripetuto per ciascun elemento incluso nell'insieme di una matrice.

## Cicli Do

E' possibile utilizzare le istruzioni Do...Loop per eseguire un blocco di istruzioni un numero indefinito di volte. Le istruzioni vengono ripetute mentre una condizione è True oppure fino a quando risulta True.

Per verificare una condizione in un'istruzione Do...Loop, è necessario utilizzare la parola chiave While. La verifica può essere eseguita prima dell'inizio del ciclo, come illustrato nell'esempio *ChkPrimaWhile* riportato di seguito, oppure dopo almeno una esecuzione del ciclo, come illustrato nell'esempio *ChkDopoWhile*:

```
Sub ChkPrimaWhile()  
    Dim counter, mioNum  
    counter = 0  
    mioNum = 20  
    Do While mioNum > 10  
        mioNum = mioNum - 1  
        counter = counter + 1  
    Loop  
    MsgBox "Il ciclo ha eseguito " & counter & " ripetizioni."  
End Sub
```

```
Sub ChkDopoWhile()  
    Dim counter, mioNum  
    counter = 0  
    mioNum = 9  
    Do  
        mioNum = mioNum - 1  
        counter = counter + 1  
    Loop While mioNum > 10  
    MsgBox "Il ciclo ha eseguito " & counter & " ripetizioni."  
End Sub
```

Nella routine *ChkPrimaWhile*, se *myNum* è impostato su 9 anziché su 20, le istruzioni incluse nel ciclo non verranno mai eseguite. Nella routine *ChkDopoWhile*, le istruzioni incluse nel ciclo vengono eseguite una sola volta, in quanto la condizione è già False.

Per verificare una condizione in un'istruzione Do...Loop, è possibile utilizzare la parola chiave Until in due modi diversi: la verifica può essere eseguita prima dell'inizio del ciclo, come illustrato nell'esempio *ChkPrimaUntil* riportato di seguito, oppure dopo almeno una esecuzione del ciclo, come illustrato nell'esempio *ChkDopoUntil*. Il ciclo viene ripetuto finché la condizione risulta False:

```
Sub ChkPrimaUntil()  
    Dim counter, mioNum  
    counter = 0  
    mioNum = 20  
    Do Until mioNum = 10  
        mioNum = mioNum - 1  
        counter = counter + 1  
    Loop  
    MsgBox "Il ciclo ha eseguito " & counter & " ripetizioni."  
End Sub
```

```
Sub ChkDopoUntil()  
    Dim counter, mioNum  
    counter = 0  
    mioNum = 1  
    Do  
        mioNum = mioNum + 1  
        counter = counter + 1  
    Loop Until mioNum = 10  
    MsgBox "Il ciclo ha eseguito " & counter & " ripetizioni."  
End Sub
```

### ***Uscita da un'istruzione Do...Loop***

Per uscire da una struttura Do...Loop, è possibile utilizzare l'istruzione Exit Do. Dato che l'uscita da tale struttura viene in genere eseguita solo in determinate situazioni, ad esempio per evitare la ripetizione infinita di un ciclo, l'istruzione Exit Do deve essere inclusa nel blocco di istruzioni True di un'istruzione If...Then...Else. Se la condizione è False, il ciclo viene eseguito normalmente.

Nell'esempio riportato di seguito, a *myNum* viene assegnato un valore in base al quale viene creato un ciclo infinito. L'istruzione *If...Then...Else* consente di verificare questa condizione, impedendo pertanto un'esecuzione all'infinito:

```
SubEsempioExit()  
    Dim counter, mioNum  
    counter = 0  
    myNum = 9  
    Do Until mioNum = 10  
        mioNum = myNum - 1  
        counter = counter + 1  
        If mioNum < 10 Then Exit Do  
    Loop  
    MsgBox "Il ciclo ha eseguito " & counter & " ripetizioni."  
End Sub
```

## Istruzione For...Next

L'istruzione *For...Next* consente di eseguire un blocco di istruzioni un numero specifico di volte. Nel caso di cicli, è necessario utilizzare una variabile contatore il cui valore viene aumentato o diminuito a ogni ripetizione del ciclo.

Nella routine riportata di seguito, ad esempio, la routine *MioCiclo* viene eseguita 50 volte. L'istruzione *For* specifica la variabile contatore *x* e il corrispondente valore iniziale e finale. L'istruzione *Next* consente di aumentare la variabile contatore con incrementi di una unità:

```
Sub DoMioCiclo50Volte()  
    Dim x  
    For x = 1 To 50  
        MioCiclo  
    Next  
End Sub
```

La parola chiave *Step* consente di aumentare o diminuire la variabile contatore del valore specificato. Nell'esempio riportato di seguito la variabile contatore viene

incrementata di due unità a ogni ripetizione del ciclo. Al completamento del ciclo, il totale corrisponde alla somma di 2, 4, 6, 8 e 10.

```
Sub DoppioTotale()  
    Dim j, totale  
    For j = 2 To 10 Step 2  
        totale = totale + j  
    Next  
    MsgBox "Il totale è " & total  
End Sub
```

Per diminuire la variabile contatore, è necessario utilizzare un valore Step negativo specificando un valore finale minore del valore iniziale. Nell'esempio riportato di seguito, la variabile contatore *myNum* viene diminuita di due unità a ogni ripetizione del ciclo. Al completamento del ciclo, il totale corrisponde alla somma di 16, 14, 12, 10, 8, 6, 4 e 2:

```
Sub NuovoTotal()  
    Dim mioNum, totale  
    For mioNum = 16 To 2 Step -2  
        totale = totale + mioNum  
    Next  
    MsgBox "Il totale è " & total  
End Sub
```

Per uscire da un'istruzione For...Next prima che il valore del contatore abbia raggiunto il valore finale, è possibile utilizzare l'istruzione Exit For. Dato che l'uscita da un'istruzione For...Next viene in genere eseguita solo in determinate situazioni, ad esempio quando si verifica un errore, l'istruzione Exit For deve essere inclusa nel blocco di istruzioni True di un'istruzione If...Then...Else. Se la condizione è False, il ciclo viene eseguito normalmente.

## Istruzione For Each...Next

L'istruzione For Each...Next è simile a For...Next, con la sola differenza che le istruzioni non vengono ripetute il numero di volte specificato, ma per ciascun elemento

di un insieme di oggetti o per ciascun elemento di una matrice. Ciò risulta particolarmente utile quando non si conosce a priori il numero di elementi di un insieme.

Nell'esempio di codice HTML riportato di seguito, il contenuto di un oggetto *Dictionary* viene utilizzato per inserire testo in varie caselle di testo:

```
<HTML>
<HEAD><TITLE>Forms and Elements</TITLE></HEAD>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub cmdChange_OnClick
    Dim d                                'Crea una variabile
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "0", "Atene"                    'Aggiunge alcune chiavi ed elementi
    d.Add "1", "Belgrado"
    d.Add "2", "Cairo"

    For Each I in d
        Document.frmForm.Elements(I).Value = D.Item(I)
    Next
End Sub
-->
</SCRIPT>
<BODY>
<CENTER>
<FORM NAME="frmForm"

<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Text"><p>
<Input Type = "Button" NAME="cmdChange" VALUE="Fare clic qui"><p>
</FORM>
</CENTER>
</BODY>
</HTML>
```

## Routine di VBScript

Nella stesura di un programma, può capitare che alcune istruzioni vengano ripetute più volte nel codice del programma stesso. La soluzione migliore per evitare di riscrivere inutilmente lunghe porzioni di codice, e rendere allo stesso tempo il sorgente più leggibile, è quella di adoperare le cosiddette routine. Un altro notevole vantaggio che si viene a presentare, utilizzando le routine, è evidentemente quello della riutilizzabilità di quest'ultime anche in altri programmi, senza doverle riscrivere ogni volta.

In VBScript sono disponibili due tipi di routine, ovvero le routine Sub e le routine Function.

Una routine Sub è una serie di istruzioni VBScript, incluse tra le istruzioni Sub ed End Sub, che eseguono azioni senza restituire alcun valore. In una routine Sub è possibile specificare argomenti, ovvero costanti, variabili o espressioni passate dalla routine che sta eseguendo una chiamata. Se non si specifica alcun argomento, nella corrispondente istruzione Sub è necessario includere parentesi vuote ().

Nella routine Sub riportata di seguito vengono utilizzate le due funzioni VBScript intrinseche, ovvero incorporate, MsgBox e InputBox, che consentono di richiedere informazioni all'utente. Viene quindi visualizzato il risultato di un calcolo basato sulle informazioni fornite ed eseguito in una routine Function creata in VBScript, come descritto nella sezione successiva:

```
Sub ConvertTemp()  
    temp = InputBox("Immettere la temperatura in gradi Fahrenheit.", 1)  
    MsgBox "La temperatura è pari a " & Celsius(temp) & " gradi C."  
End Sub
```

Come si può notare, il nome della routine è *ConvertTemp* e non sono previsti parametri in ingresso. La prima istruzione è la chiamata di un'altra routine, di nome *InputBox*, che semplicemente richiede all'utente l'inserimento di un valore <sup>(5)</sup>. La seconda istruzione si compone di varie parti: è sicuramente una chiamata della routine *MsgBox*, anch'essa predefinita, che consente di visualizzare un messaggio in una finestra sul video; in particolare, il contenuto del messaggio viene costruito concatenando tre stringhe: la stringa "La temperatura è pari a" viene concatenata al valore fornito da una routine Function di nome *Celsius*, alla quale viene passato il

---

<sup>5</sup> Di queste routine parleremo in seguito

valore *temp* digitato dall'utente e che, sulla base di questo, restituisce un determinato valore (in particolare, il valore di temperatura in gradi Celsius). Il tutto viene infine concatenato all'altra stringa "gradi C."

Una routine Function, come quella appena citata di nome *Celsius*, è una serie di istruzioni VBScript incluse tra le istruzioni Function ed End Function. È simile a una routine Sub, ma a differenza di questa restituisce un valore. In una routine Function è possibile specificare argomenti, ovvero costanti, variabili o espressioni passate alla routine dalla routine chiamante. Se non si specifica alcun argomento, nella corrispondente istruzione Function è necessario includere parentesi vuote.

Una routine Function restituisce un valore quando si assegna un valore al nome della routine in una o più istruzioni. Il tipo restituito di una routine Function è sempre Variant.

Nell'esempio riportato di seguito, la funzione *Celsius* consente di eseguire la conversione da Fahrenheit a Celsius. Quando la funzione viene richiamata nella routine *Sub ConvertTemp*, viene passata alla funzione una variabile contenente il valore dell'argomento. Il risultato del calcolo viene quindi restituito alla routine che ha eseguito la chiamata e visualizzato in una finestra di messaggio:

```
Sub ConvertTemp()  
    temp = InputBox("Immettere la temperatura in gradi Fahrenheit.", 1)  
    MsgBox "La temperatura è pari a " & Celsius(temp) & " gradi C."  
End Sub  
  
Function Celsius(fDegrees)  
    Celsius = (fDegrees - 32) * 5 / 9  
End Function
```

Se il nome della routine non venisse utilizzato all'interno delle istruzioni della routine stessa, non sarebbe possibile ottenere il risultato voluto.

I dati vengono passati alla routine tramite argomenti che fungono da segnaposto per i dati stessi. Agli argomenti è possibile assegnare un nome valido per le variabili.

Quando si crea una routine utilizzando l'istruzione Sub o Function, è necessario far seguire il nome della routine da parentesi, all'interno delle quali vengono inseriti gli argomenti separati da una virgola. Nell'esempio seguente, *fDegrees* è il segnaposto del valore da convertire passato alla funzione *Celsius*:

```
Function Celsius(fDegrees)
```

```
Celsius = (fDegrees - 32) * 5 / 9
```

```
End Function
```

## Istruzione Call

Per richiamare una routine Sub da un'altra routine, è sufficiente specificarne il nome insieme ai valori degli argomenti obbligatori, separati da una virgola. L'istruzione Call non è obbligatoria. Se viene utilizzata, è necessario racchiudere gli eventuali argomenti tra parentesi.

Nell'esempio seguente vengono eseguite due chiamate alla routine *MiaRoutine*. Sebbene l'istruzione Call sia utilizzata nel codice di una sola chiamata, entrambe le chiamate eseguono la stessa operazione.

```
Call MiaRoutine(primoarg, secondoarg)
```

oppure

```
MiaRoutine primoarg, secondoarg
```

Nella chiamata in cui l'istruzione Call non è utilizzata, le parentesi sono state omesse.

## Convenzioni di scrittura del codice

Le convenzioni di scrittura del codice consentono di facilitare la scrittura del codice. Tali convenzioni sono le seguenti:

1. Convenzioni di denominazione per oggetti, variabili e routine
2. Convenzioni per l'aggiunta di commenti
3. Indicazioni generali per la formattazione e il rientro del testo

Lo scopo principale delle convenzioni di scrittura del codice è uniformare la struttura e lo stile di uno script o gruppo di script in modo che il codice risulti semplice da leggere e comprensibile a tutti. Grazie alla definizione di buone

convenzioni è possibile ottenere codice sorgente preciso, leggibile e chiaro, in conformità con il codice di altri linguaggi e quanto più possibile intuitivo.

## Convenzioni di denominazione delle costanti

Nelle versioni precedenti di VBScript non era disponibile alcun metodo di creazione di costanti definite dall'utente. Se utilizzate, le costanti venivano implementate come variabili e distinte dalle altre variabili con l'utilizzo dei caratteri maiuscoli. Le parole che componevano il nome venivano separate con un carattere di sottolineatura (\_). Ad esempio:

*USER\_LIST\_MAX*

*NEW\_LINE*

È ancora possibile utilizzare questo metodo per identificare le costanti, ma si consiglia di utilizzare uno schema di denominazione alternativo e l'istruzione Const per creare vere costanti. Questa convenzione consente di utilizzare un formato con caratteri maiuscoli e minuscoli in cui i nomi delle costanti sono preceduti dal prefisso "con", ad esempio:

*conCostantePersonale*

## Convenzioni di denominazione delle variabili

Per motivi di leggibilità e uniformità, in VBScript i nomi delle variabili devono essere descrittivi e preceduti dai prefissi elencati nella tabella seguente.

Sottotipo	Prefisso	Esempio
Boolean	bln	blnFound
Byte	byt	bytRasterData
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance

Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

## Prefissi per l'area di validità delle variabili

Con script di grandi dimensioni diventa particolarmente importante poter contraddistinguere l'area di validità delle variabili. A tale scopo è sufficiente aggiungere un prefisso di una lettera al prefisso del tipo di variabile in modo da evitare nomi eccessivamente lunghi.

Area di validità	Prefisso	Esempio
A livello di routine	Nessuno	dblVelocity
A livello di script	s	sblnCalcInProgress

## Nomi di routine e variabili descrittivi

Il corpo del nome di variabili e routine deve descrivere nel modo più completo possibile la funzione della variabile o routine, nonché includere una combinazione di lettere maiuscole e minuscole. I nomi di routine devono inoltre iniziare con un verbo, ad esempio *InitNameArray* o *CloseDialog*. Per termini utilizzati di frequente o particolarmente lunghi, è consigliabile utilizzare abbreviazioni standard, in modo da mantenere ridotta la lunghezza del nome. In generale, i nomi di variabili composti da più di 32 caratteri risultano di difficile lettura. Le abbreviazioni devono essere utilizzate in modo uniforme. Se in uno script o gruppo di script vengono, ad esempio, alternate le abbreviazioni *Cnt* e *Count* senza alcun criterio, è possibile creare confusione.

## Convenzioni di denominazione degli oggetti

Nella tabella seguente sono elencate le convenzioni di denominazione di vari oggetti utilizzati nella programmazione in VBScript:

Tipo di oggetto	Prefisso	Esempio
Pannello 3D	pnl	pnlGroup
Pulsante animato	ani	aniMailBox
Casella di controllo	chk	chkReadOnly
Casella combinata o di riepilogo a discesa	cbo	cboEnglish
Pulsante di comando	cmd	cmdExit
Finestra di dialogo comune	dlg	dlgFileOpen
Cornice	fra	fraLanguage
Barra di scorrimento orizzontale	hsb	hsbVolume
Immagine	img	imgIcon
Etichetta	lbl	lblHelpMessage
Linea	lin	linVertical
Casella di riepilogo	lst	lstPolicyCodes
Pulsante di selezione	spn	spnPages
Casella di testo	txt	txtLastName
Barra di scorrimento verticale	vsb	vsbRate
Dispositivo di scorrimento	sld	sldScale

## Convenzioni di scrittura dei commenti

All'inizio delle routine è opportuno includere un breve commento per descrivere il funzionamento della routine stessa. Tale commento non deve tuttavia includere informazioni dettagliate sulla modalità di implementazione. Tali informazioni, essendo soggette a modifiche, comporterebbero infatti un inutile lavoro di manutenzione o addirittura la presenza di commenti non corretti. La modalità di implementazione viene

comunque descritta nel codice stesso e nei commenti inseriti sulla stessa riga delle istruzioni.

È necessario descrivere gli argomenti passati a routine la cui funzione non risulta evidente o che sono inclusi in un determinato intervallo. All'inizio delle routine è inoltre necessario includere una descrizione dei valori restituiti da funzioni e altre variabili che vengono modificate dalla routine, soprattutto se le modifiche vengono apportate tramite argomenti di riferimento.

Nei commenti di intestazione delle routine è necessario includere le intestazioni di sezione indicate di seguito. Per alcuni esempi, vedere la sezione successiva "Formattazione del codice".

Intestazione di sezione	Contenuto del commento
Scopo	Operazioni eseguite dalla routine, non la modalità di esecuzione.
Presupposti	Elenco di variabili esterne, controlli e altri elementi il cui stato ha effetto sulla routine.
Effetti	Elenco dell'effetto della routine su variabili esterne, controlli e altri elementi.
Input	Descrizione degli argomenti con significato non evidente. Ciascun argomento deve essere incluso in una riga distinta insieme a un commento specifico sulla stessa riga.
Valori restituiti	Descrizione del valore restituito.

È importante tenere presente i seguenti punti:

- le dichiarazioni di variabili importanti devono includere un commento sulla stessa riga in cui viene descritto l'utilizzo della variabile dichiarata;
- i nomi di variabili, controlli e routine devono essere sufficientemente descrittivi in modo che sia necessario aggiungere commenti sulla stessa riga solo per la descrizione di implementazioni complesse;

- all'inizio degli script è consigliabile includere cenni preliminari che descrivono brevemente lo script e in cui siano elencati gli oggetti, le routine, gli algoritmi, le finestre di dialogo e altre dipendenze di sistema. A volte può risultare utile aggiungere pseudocodice che descrivi l'algoritmo.

## Formattazione del codice

Lo scopo è quello di risparmiare spazio sullo schermo, ma allo stesso tempo applicare al codice una formattazione che ne rifletta in modo chiaro la struttura logica e i livelli di nidificazione. Di seguito sono indicati alcuni suggerimenti:

- i blocchi nidificati standard devono rientrare di quattro spazi.
- i commenti introduttivi di una procedura devono essere rientrati di uno spazio.
- le istruzioni di livello principale che seguono i commenti iniziali devono rientrare di quattro spazi, con ciascun blocco nidificato rientrato di altri quattro spazi. Ad esempio:

## VBScript ed i form

È possibile utilizzare Visual Basic Scripting per le operazioni di elaborazione di form che in genere è necessario eseguire in un server (tramite le CGI), nonché per le operazioni che non possono essere svolte nel server.

Ad esempio, è possibile usare VBScript per controllare che un utente, nel riempire una form, abbia riempito correttamente tutti i campi della stessa. Tipico caso è quello in cui l'utente deve inserire il proprio indirizzo di posta elettronica, ma, per un motivo qualsiasi, dimentica di inserire la @, il che rende l'indirizzo inutilizzabile. In una situazione del genere, le strategie perseguibili sono almeno due: la prima è quella di lasciare che sia il server a segnalare l'errore, nel qual caso quindi la form errata viene comunque inviata al server, il quale risponde dicendo che c'è stato un errore nel riempimento dei campi ed è quindi necessario ripetere la procedura; è ovvio che, con questa soluzione, non solo si perde più tempo perchè il server potrebbe essere già molto impegnato e quindi potrebbe rispondere con un certo ritardo, ma anche perchè si va ad incrementare il lavoro dello stesso server. L'alternativa è perciò quella di demandare direttamente al client il compito di controllare che la form sia stata

riempita in modo corretto, il che equivale a dire che la form giungerà al server solo quando tutti i campi saranno stati riempiti in modo giusto.

Per ottenere questo, si può procedere nel modo seguente: bisogna fare in modo che, alla pressione di uno specifico tasto della form (che non deve essere quello di invio), parta una routine VBScript che si occupi di controllare i dati contenuti nella form; nel caso riscontrasse errori, la routine può segnalarli ed eventualmente permettere di correggerli; in assenza di errori, la routine esegue l'invio della form al server.

Un esempio di questo procedimento è riportato qui di seguito:

```
<HTML>
<HEAD><TITLE>Convalida semplice</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Submit_OnClick
  Dim MioForm
  Set MioForm = Document.ValidForm
  If IsNumeric(MioForm.Text1.Value) Then
    If MioForm.Text1.Value < 1 Or MioForm.Text1.Value > 10 Then
      MsgBox "Immettere un numero compreso tra 1 e 10."
    Else
      MsgBox "Valore corretto."
      MioForm.Submit      ' Dati inoltrati al server.
    End If
  Else
    MsgBox "Immettere un valore numerico."
  End If
End Sub
-->
</SCRIPT>
</HEAD>
<BODY>
<H3>Convalida semplice</H3><HR>
<FORM NAME="ValidForm">
Immettere un valore compreso tra 1 e 10:
<INPUT NAME="Text1" TYPE="TEXT" SIZE="2">
<INPUT NAME="Submit" TYPE="BUTTON" VALUE="Invia">
</FORM>
</BODY>
</HTML>
```

### Osservazioni sul codice utilizzato

Viene utilizzata la proprietà Value della casella di testo, al fine di verificare il valore immesso. Per leggere la proprietà Value, è necessario che nel codice venga specificato il riferimento al nome della casella di testo. È comunque possibile scrivere il riferimento completo *Document.ValidForm.Text1*.

Nel caso di più riferimenti a controlli di form, è tuttavia consigliabile procedere come indicato nell'esempio, ovvero dichiarare innanzitutto una variabile e assegnare quindi il form alla variabile *MioForm* tramite l'istruzione Set. In questo caso non è possibile utilizzare una normale istruzione di assegnazione quale Dim. L'istruzione Set consente infatti di mantenere il riferimento a un oggetto.

Il valore della variabile *Text1* viene verificato direttamente in base a un numero, ovvero viene verificato che la stringa inclusa nella casella di testo sia un numero tramite la funzione IsNumeric. Anche se in VBScript stringhe e numeri vengono convertiti automaticamente in modo adeguato, è consigliabile controllare sempre il sottotipo dei valori immessi dall'utente e utilizzare, se necessario, le funzioni di conversione.

Si nota subito, nell'esempio appena riportato, che manca il classico bottone di tipo *Submit* per l'invio della form: questo perché, se venisse usato questo bottone, i dati da controllare non sarebbero mai visualizzati, ma verrebbero inoltrati immediatamente al server. Non utilizzando il controllo Submit, è possibile verificare i dati, ma poi è necessario aggiungere una apposita istruzione VBScript per inoltrare il tutto al server. In particolare, viene richiamato il metodo Submit dell'oggetto form quando i dati sono corretti. Il client gestisce quindi i dati secondo la normale modalità, ma i dati vengono corretti prima di essere inoltrati al server.

## **Modalità alternative per l'associazione di codice a eventi**

Uno dei grandi vantaggi offerti dall' HTML dinamico è che tutti i componenti di una pagina WEB vengono considerati degli oggetti: le immagini, i paragrafi, i titoli e così via. Grazie a questa possibilità, si possono scrivere degli script che vengono attivati al verificarsi di un determinato evento su di un oggetto. Ad esempio, si può modificare l'aspetto del testo al passaggio del mouse, o, allo stesso modo, far apparire un'immagine al posto di un'altra.

Un esempio semplice è il seguente: supponiamo di voler visualizzare sul monitor la scritta "PROVA MOUSE" e di voler fare in modo che, al passaggio del mouse sopra la scritta, quest'ultima cambi colore e si ingrandisca, mentre invece torni uguale a come

era prima spostando il mouse. Per ottenere quest'effetto, si può inserire il codice necessario direttamente all'interno del tag relativo alla caratteristica dell'oggetto da cambiare. In questo esempio, il testo cambia colore e dimensione al passaggio del mouse (evento `OnMouseOver`), e ritorna poi alla condizione iniziale quando il puntatore si allontana (`OnMouseOut`): entrambi gli eventi dovranno essere definiti all'interno del tag `<FONT>`:

```
<p align="justify"><font color="#00FFFF" size="4"
onmouseover="color='#FF0000'; size='6'"
onmouseout="color='#00FFFF'; size='4'">PROVA MOUSE</p></font>
```

Se, invece, avessimo voluto cambiare allineamento al passaggio del mouse, l'evento doveva essere inserito nel tag `<p>`:

```
<p align="justify" onmouseover="align='center'"
onmouseout="align='justify'">TESTO</p>
```

Allo stesso modo, si può far verificare l'evento del cambiamento dell'aspetto del testo invece che al passaggio del mouse, quando l'oggetto viene cliccato una o due volte: invece di usare l'evento `OnMouseOver`, si possono utilizzare `OnClick` e `OnDbClick`:

```
<P><FONT color=#0000ff onclick="color='#FF0000'; size='6'"
ondblclick="size='4'; color='#0000FF'" size=4>PROVA MOUSE</FONT></P>
```

Analogamente, si possono applicare gli stessi eventi alle immagini, sostituendo l'attributo `color` o `size` con `src` (`onmouseover="src='...'"`), in modo da cambiare immagine ad esempio al passaggio del mouse:

```
<P align=center>
<IMG border=0 height=150 onmouseout="src='image1.jpg'"
onmouseover="src='image2.jpg'" src="image1.jpg" width=180></P>
```

## Messaggi

Tramite la funzione `MsgBox()` è possibile visualizzare, all'apertura della pagina o al verificarsi di un determinato evento, un messaggio o una domanda personalizzati. Per

ottenere ciò, basta inserire, all'interno di uno script realizzato in VBScript, questa funzione, che presenta la seguente sintassi:

NomeVariabile = MsgBox (testo [, pulsanti] [,titolo])

*NomeVariabile* indica il nome di una variabile che inseriamo nello script per contenere la risposta al messaggio. *Testo* indica il messaggio che dovrà essere visualizzato: se si tratta di una stringa, dovrà essere contenuta all'interno delle virgolette (ad esempio "Questo è il testo visualizzato"). Se, invece, si tratta del nome di una variabile di tipo stringa, che contiene il testo, non deve essere separata dalle virgolette.

All'interno delle parentesi quadre [ ] sono stati inseriti due parametri non indispensabili: con *pulsanti* si indica un numero intero (stabilito secondo una tabella che riporteremo tra poco) tramite il quale si determinano sia i pulsanti che si vogliono visualizzare nel messaggio sia le eventuali icone; con *titolo* s'indica appunto la frase da inserire nella barra del titolo della finestra di messaggio.

Il seguente script viene eseguito quando si verifica l'evento *onclick* sul testo che abbiamo denominato *esempio1*:

```
<script for="esempio1" event="onclick" language="VBScript">
<!--
  nome= ucase(Nome)
  risposta = msgbox ("CIAO " & nome,4144,"Ehy!")
-->
</script>
```

Questo esempio presuppone che la variabile *nome* sia stata precedentemente inizializzata, ad esempio tramite un ulteriore script che chieda all'utente di digitare il proprio nome e inserisca tale nome nella suddetta variabile.

La funzione `Ucase()` è semplicemente un sottoprogramma predefinito di VBScript, che trasforma i caratteri minuscoli in maiuscoli.

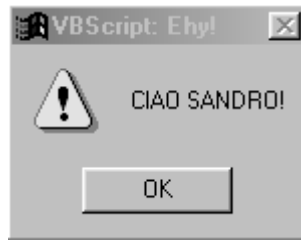
Provando concretamente questo script tramite un browser che supporti VBScript, si vede che nel messaggio, oltre al testo ("CIAO " & Nome) e al titolo (Ehy!), viene mostrato un'icona contenente un punto esclamativo; non solo, ma il sistema impedisce di eseguire nessun'altra applicazione prima di aver dato una risposta al messaggio: ciò si ottiene combinando insieme dei numeri interi, che in questo caso danno come

risultato 4144, i quali, seguendo la seguente tabella, definiscono le caratteristiche del messaggio visualizzato:

Numero	Caratteristiche
<b>Pulsanti</b>	
0	Visualizza solo il pulsante Ok (predefinito se viene omissso)
1	Ok - Annulla
2	Termina - Riprova - Ignora
3	Sì - No - Annulla
4	Sì - No
5	Riprova - Annulla
<b>Icone</b>	
16	Icona Messaggio Critico
32	Punto Interrogativo
48	Punto Esclamativo
64	Messaggio Informativo
<b>Pulsante Predefinito</b>	
0	Il primo pulsante è predefinito
256	Secondo pulsante predefinito
512	Terzo pulsante predefinito
768	Quarto pulsante predefinito
<b>Modalità</b>	
0	L'utente deve rispondere prima di poter riutilizzare l'applicazione corrente
4096	Tutte le applicazioni vengono sospese finché non si risponde

Questi numeri possono essere combinati insieme, sommandoli: ad esempio, se si vogliono visualizzare su un messaggio i pulsanti Sì - No e l'icona del punto interrogativo, si dovrà inserire nello script il numero 36 (32+4). Nell'esempio di prima, il numero 4144 deriva da 0+48+4096, il che significa che vengono visualizzati solo il tasto ok (0) e l'icona del punto esclamativo (48) e che tutte le applicazioni vengono

sospese fin quando non si risponde (4096), che in questo caso equivale semplicemente a cliccare su ok:



Una volta che si conosce il valore della risposta, cioè quale pulsante è stato premuto dal visitatore, si possono usare le istruzioni If..Then o Select Case per agire di conseguenza. Questa è la tabella dei risultati relativi ad ogni pulsante:

Pulsante	Valore assegnato alla Variabile
Ok	1
Annulla	2
Termina (Interrompi)	3
Riprova	4
Ignora	5
Sì	6
No	7

Ad esempio, se si vuole porre una domanda e s'inseriscono nel messaggio i pulsanti Sì e No insieme all'icona del punto interrogativo (numero corrispondente: 32+4=36) , se il valore della variabile dove viene riposto il risultato della funzione MsgBox() sarà 6, la risposta sarà affermativa, altrimenti sarà negativa.

L'esempio seguente aiuta a chiarire questi concetti:

```
<script language="VBScript">
<!--
risposta = msgbox("Vuoi continuare?",36)
Select Case risposta
  Case 6
    msg = "Hai scelto SI!!"
  Case else
    msg = "Hai paura?"
End Select
alert (msg)
```

```
-->  
</script>
```

## Finestre di dialogo

All'apertura della pagina o al verificarsi di un certo evento, si può visualizzare una finestra di dialogo che accetti l'inserimento di una stringa o di un numero. Ad esempio, per visualizzare la finestra di dialogo all'apertura della pagina si può utilizzare la seguente sintassi:

```
<script language="VBScript">  
<!--  
    testo = "Per favore, inserisci il tuo nome?"  
    titolo= "Qual è il tuo nome?"  
    messaggio = "Nome"  
    leggiRisposta = InputBox(testo, titolo, messaggio)  
    leggiRisposta = Ucase(leggiRisposta)  
    alert ("CIAO " & leggiRisposta & "!")  
-->  
</script>
```

In questo caso, abbiamo inserito anche la funzione `alert()`, che visualizza un messaggio critico. *leggiRisposta* è la variabile alla quale viene attribuito il valore della risposta, cioè quello che viene scritto nella casella di testo. Si tratta in questo caso del nome inserito dall'utente.

## Utilizzo di VBScript con gli oggetti (cenni)

Microsoft Visual Basic Scripting Edition e Microsoft® Internet Explorer gestiscono sia i controlli ActiveX, precedentemente definiti controlli OLE, che gli oggetti Java.

Per includere un oggetto, è necessario utilizzare il tag `<OBJECT>`, mentre per impostare i valori iniziali delle proprietà dell'oggetto, è necessario utilizzare il tag `<PARAM>`. L'utilizzo del tag `<PARAM>` è equivalente all'impostazione dei valori iniziali delle proprietà per un controllo di un form in Visual Basic.