



DAI VIRUS DELLE MACRO ALLE FUNZIONI INTERNE DI WORD

***Come modificare le funzioni di Word
da Visual Basic***

Visual Basic Italia

Indice

| | |
|--|----|
| Premessa..... | 3 |
| Differenti modi di creare lo stesso oggetto..... | 3 |
| I modelli, i documenti e le funzioni..... | 4 |
| Il contenuto del documento..... | 6 |
| Le funzioni di Word..... | 7 |
| Inserimento delle funzioni nel modulo macro..... | 9 |
| Funzioni create dall'utente..... | 10 |

Premessa

Il primo punto da analizzare è la creazione dell'oggetto 'Applicazione Word', che corrisponde ad una nuova sessione dello stesso. Si desidera creare un programma che apra Word e visualizzi un certo testo all'interno di un nuovo documento.

Risulterebbe immediato a questo punto, includere nel progetto la libreria Word (MSWORDnumeroversione.OLB) corrispondente alla versione posseduta.

Ma cosa accadrebbe se, esportando l'applicazione, ci si trovasse di fronte ad una versione di Word inferiore? E come superare questo primo scoglio?

Il risultato di questa mancanza può essere agevolmente valutato nell'esempio che segue. Studiandone quindi una piccola variazione si giungerà alla conclusione che risolverà la questione..

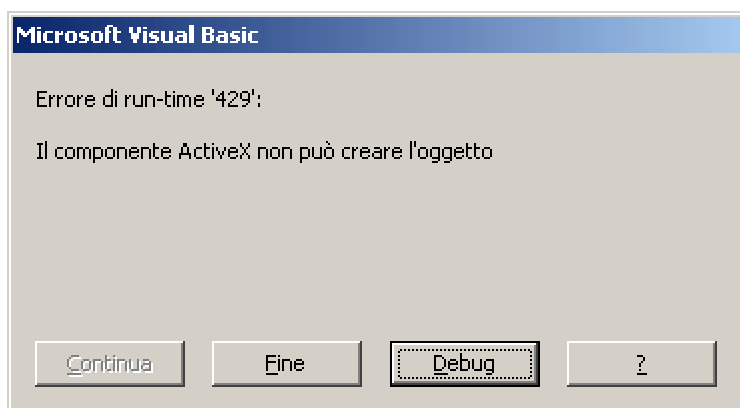
Differenti modi di creare lo stesso oggetto

a creazione di un oggetto può essere operata in molteplici modi. Il primo e più immediato, come già visto, consiste nell'importare la libreria Word nel progetto. Una volta fatta riconoscere a Visual Basic la classe con la quale si desidera operare, sarà un gioco da ragazzi andare a dichiarare e successivamente definire i diversi oggetti di cui si compone la libreria. Se ad esempio si dispone della Microsoft Word 8.0 Object Library, il seguente codice:

```
Dim appWord As Word.Application
Dim docWord As Word.Document
Dim dlgWord As Word.Dialog
```

non fa altro che dichiarare rispettivamente una nuova sessione di lavoro Word 8, un documento ed una finestra di dialogo della medesima classe di oggetti.

Grosso vantaggio di tale soluzione è la disponibilità dell'Intellisense che permette di ridurre sensibilmente il carico di lavoro dal momento che fornisce in tempo reale proprietà e metodi di ogni singolo oggetto. Una volta avviato il progetto su un PC recante una differente versione di Office si otterrà però l'errore "Impossibile creare il componente ActiveX." ossia il famigerato errore 429 che affligge spesso anche gli sviluppatori VBScript.



Si consideri adesso il seguente codice:

```
Dim appWord As Object
appWord = CreateObject("Word.Application.8")
```

Il risultato è esattamente lo stesso a cui portava il blocco di codice precedente. L'unica differenza nonché l'unico passo in avanti verso l'indipendenza da qualsiasi vincolo di versione sta nel fatto che in questa soluzione non rende più necessario il riferimento alla libreria Word.

Si controlli adesso la versione di Word installata sul proprio PC e si sostituisca al numero otto un numero inferiore a tale versione.

Word cercherà di creare un oggetto che in realtà non esiste sulla propria macchina. Si otterrà anche questa volta l'errore "Impossibile creare il componente ActiveX."

Per fortuna però il secondo errore è stato espressamente ricercato e quindi è intenzionale: è bastato indicare una versione di Word differente da quella installata per generarlo.

Risulta pertanto semplice comprendere che il metodo migliore per ovviare a questo primo ostacolo è proprio quello di generalizzare il più possibile, lasciando a Visual Basic il compito di riconoscere la versione di Word installata:

```
Dim appWord As Object
```

```
AppWord = CreateObject("Word.Application")
```

I modelli, i documenti e le funzioni

In MS Word le principali attività che possono essere eseguite dall'utente (salvataggio di un file, chiusura di una finestra o dell'applicazione e così via) sono generate da funzioni predefinite.

Tali funzioni sono scritte in VBA e risiedono per impostazione predefinita nel modello principale Normal.dot.

A tale proposito occorre specificare il rapporto tra Normal.dot e qualsiasi altro documento aperto da Word.

Quando infatti si genera un nuovo documento, si crea un foglio di lavoro con determinate caratteristiche: un certo tipo di visualizzazione, una certa lista di macro e così via. Ma come fa Word a capire come impostare un documento alla sua apertura?

Le caratteristiche del nuovo documento vengono importate dal modello Normal.dot, nel caso in cui il documento utilizzi tale modello come predefinito. In Windows 2000 la directory contenente tutti i modelli di Word è C:\Documents and Settings\Administrator\Dati applicazioni\Microsoft\Modelli.

La presenza di un template predefinito (il modello Normal di cui si parlava poco fa) permette di eseguire qualsiasi modifica al documento appena aperto senza che queste siano rese permanenti al salvataggio.

Si prenda il caso di una macro dannosa per l'applicazione, generata tramite l'editor di codice VBA di un documento creato dall'utente e salvato su hard disk con il nome di Documento1.

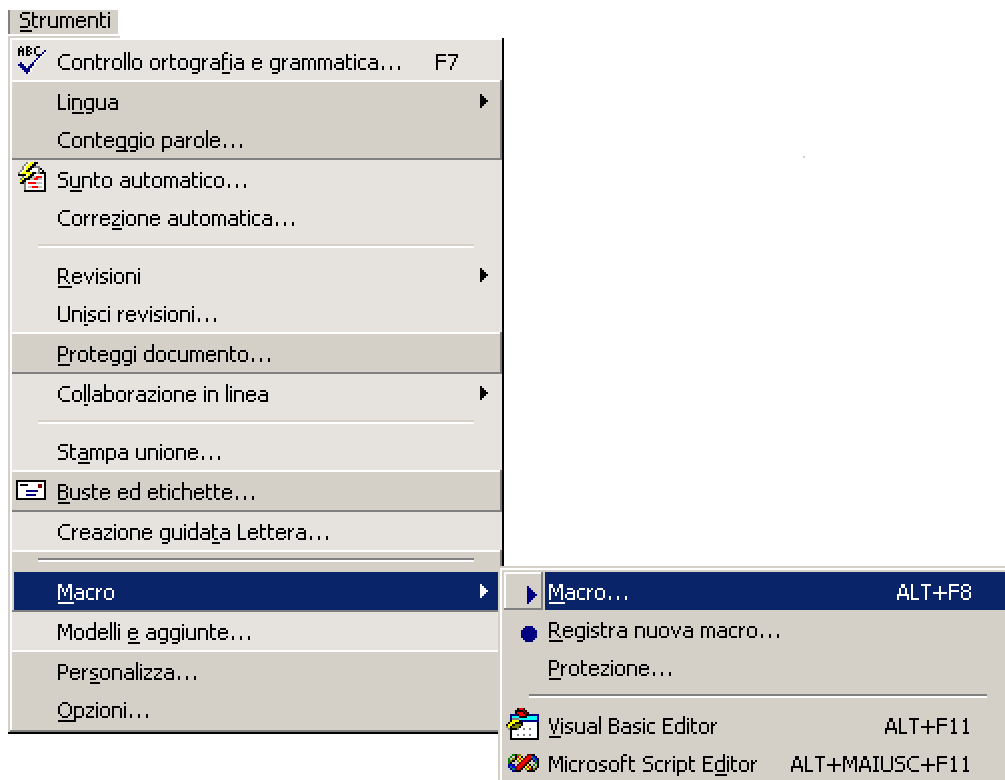
Se tutti i documenti venissero aperti sul modello dell'ultimo file salvato, la macro si propagherebbe su tutte le sessioni di lavoro di Word successive al salvataggio di tale file.

La presenza di un modello predefinito permette pertanto di avere uno stampo "pulito" dal quale generare una serie di documenti altrettanto privi di qualsiasi contaminazione.

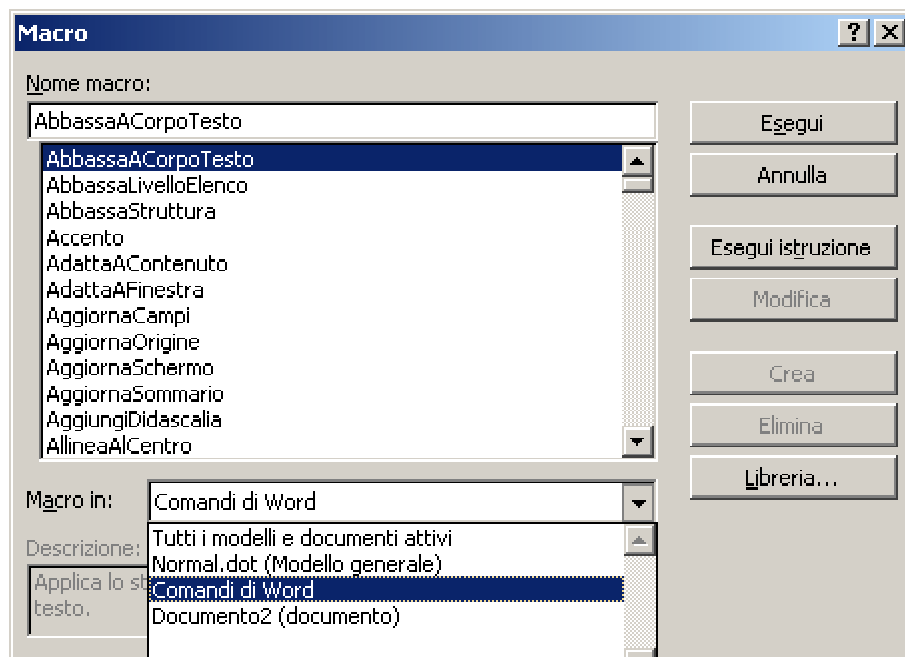
Il problema nasce quando la macro dannosa di cui si parlava poco prima passa da un documento qualsiasi al template Normal.dot, e questo passaggio rappresenta l'innescò della carica distruttiva che si portano dietro i virus delle macro.

Si è parlato all'inizio del paragrafo di funzioni predefinite, ossia una serie di macro richiamate dall'applicazione Word in corrispondenza di eventi generati dall'utente.

La lista è consultabile da un qualsiasi documento Word scegliendo la voce Strumenti -> Macro dal menu principale e cliccando su Macro (o in alternativa premendo i tasti ALT + F8).



Apparirà in tal modo la finestra denominata Macro che non rappresenta altro che una lista delle funzioni generate dall'utente. Per conoscere però le funzioni interne a Word bisogna fare un passo successivo: dalla medesima finestra selezionare la combobox 'Macro in' e scegliere la voce 'Comandi di Word', proprio come mostrato in figura:



La lista della finestra si popolerà con tutte le funzioni interne a Word disponibili. Per conoscere il significato di ciascuna funzione è sufficiente consultare la descrizione posta nella parte bassa della finestra. E' necessario però specificare che tali funzioni sono presenti sia nel modello predefinito che in un qualsiasi

documento generato dall'utente. Ma mentre le funzioni presenti su quest'ultimo sono modificabili senza rischio per l'applicazione, una modifica nel template predefinito si propagherà come visto poc'anzi a tutti i documenti successivamente generati.

Le funzioni elencate rappresentano il punto di partenza di un virus delle macro. Il ragionamento è il seguente: non è possibile modificare direttamente il file Normal.dot aggiungendo macro dannose. E' però possibile riprogrammare le macro predefinite di un qualsiasi documento Word in modo che causino danni all'applicazione e, per renderle presenti ed attive in qualsiasi altro documento, salvarle nel modello Normal.dot.

Naturalmente se lo studio delle funzioni interne di Word fossero utili solo per rendere inutilizzabile l'applicazione non ci sarebbe motivo di analizzarle.

Questo lavoro risulta però molto utile nel caso in cui si voglia comprendere a fondo la struttura dell'interazione utente/applicazione Word per creare soluzioni efficienti. Una delle tante è la generazione in un documento Word di un report utilizzando Visual Basic.

Il contenuto del documento

Una volta analizzata la lista delle funzioni dal menu Macro, si può decidere su quale particolare funzione andare ad operare. La creazione di un report in Word rende sicuramente necessario l'inserimento di un testo nel nuovo documento.

Seguendo pertanto le istruzioni dell'articolo precedente si creerà innanzitutto una nuova sessione generica di Word e ad essa si andrà ad aggiungere ed ad attivare un documento:

```
Dim appWord As Object
Dim docWord As Object Private Sub Form_Load()
Dim TestoDaInserire As String
Dim Titolo As String
Set appWord = CreateObject("Word.Application")
appWord.Visible = True
Set docWord = appWord.documents.Add
docWord.Activate
TestoDaInserire = "Questo testo rappresenta il contenuto del Report in Word
richiamato da un'applicazione Visual Basic."
Titolo = "Report"
End Sub
```

Il testo da inserire può essere incluso in una stringa di testo, posizionata subito di seguito all'attivazione del documento, proprio come nel caso del titolo del report.

L'inserimento del titolo unitamente al testo può essere effettuato nel seguente modo:

```
With appWord.Selection
.InsertParagraphAfter
.Font.Bold = True
.Font.Size = 12
.Font.Name = "arial"
.Paragraphs(1).Alignment = wdAlignParagraphCenter
.TypeText Titolo
.TypeParagraph
.TypeParagraph
.InsertParagraphAfter
.Font.Bold = False
.Font.Size = 10
.Font.Name = "arial"
.Paragraphs(1).Alignment = wdAlignParagraphJustify
.TypeText TestoDaInserire
End With
```

Utilizzando il codice sopra si possono visualizzare degli errori dati dalla mancata definizione delle costanti di allineamento dei paragrafi (rispettivamente wdAlignParagraphCenter e wdAlignParagraphJustify). Non avendo importato infatti la libreria Word, anche le costanti ad essa legate non sono definite.

Sarà quindi necessario definirle all'inizio del progetto, nel seguente modo:

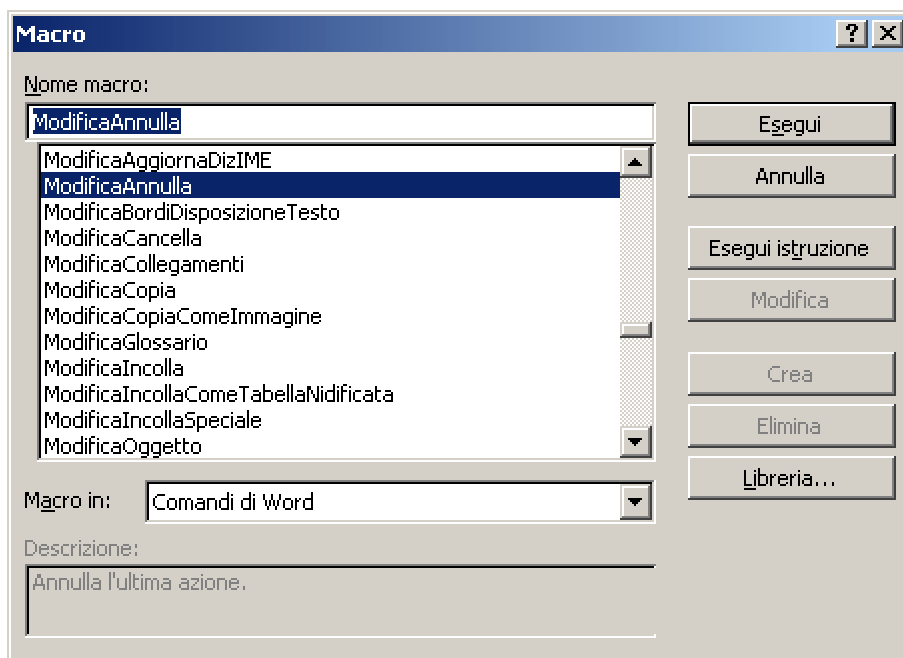
```
Private Const wdAlignParagraphCenter = 1
Private Const wdAlignParagraphJustify = 3
```

Le funzioni di Word

Adesso che il report ha preso una certa forma è necessario ritornare alla trattazione delle funzioni interne di Word. Di certo si vorrà evitare che l'utente rimuova il testo semplicemente premendo il pulsante Annulla (Undo).

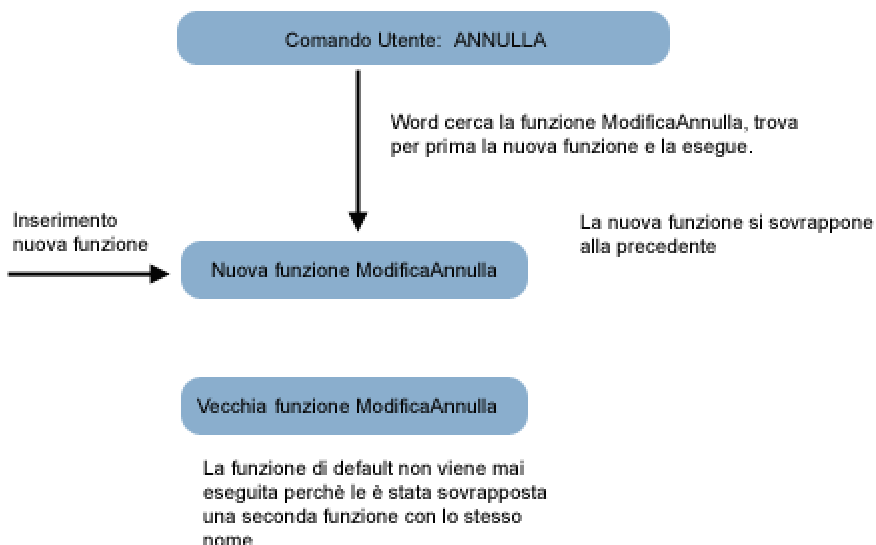
E' pertanto necessario disabilitare questa caratteristica.

La funzione di Word che si occupa di compiere tale operazione è ModificaAnnulla come mostrato nella finestra Macro sottostante:



Disabilitare l'annullamento di un comando o della digitazione di un testo significa andare a scrivere una funzione ModificaAnnulla (quindi col medesimo nome della funzione interna di Word), con un codice differente, in modo che questa si sovrapponga ed annulli la funzione di default.

Il grafico sottostante schematizza questo processo:



In realtà esistono varie relazioni tra una funzione interna a Word ed una con lo stesso nome generata tramite una macro.

Esiste la sovrapposizione come nel caso del grafico: mentre la funzione di default si trova nel modello Normal, la nuova funzione si trova esclusivamente in un modulo di codice dell'editor VBA all'interno del documento aperto.

In questo modo la nuova funzione sostituirà quella di default finché il documento non verrà chiuso.

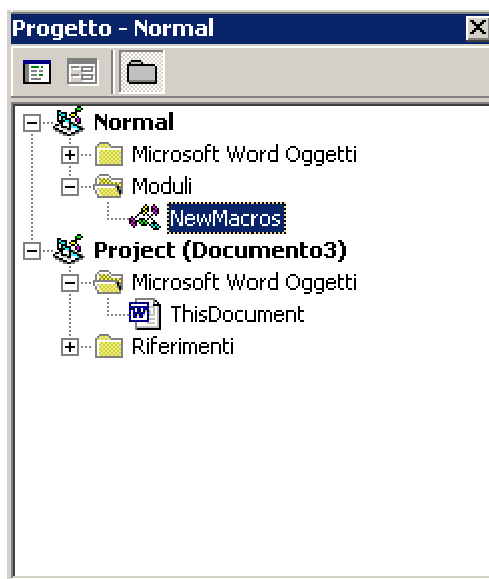
Esiste poi la sostituzione, modalità di cui fanno uso i virus delle macro: la funzione creata viene salvata proprio nel modello Normal, andando così a sovrascrivere la funzione di Word.

All'apertura di qualsiasi documento generato da quel modello Normal, la nuova funzione verrà così attivata.

Per comprendere come e dove scrivere una nuova funzione, è necessario analizzare la struttura dei moduli dell'editor di codice VBE.

A tale scopo aprire l'editor tramite Strumenti->Macro->Visual Basic Editor.

Apparirà un ambiente di sviluppo simile al classico di Visual Basic. E sulla colonna di sinistra sarà riportata una struttura simile a quella mostrata in figura:



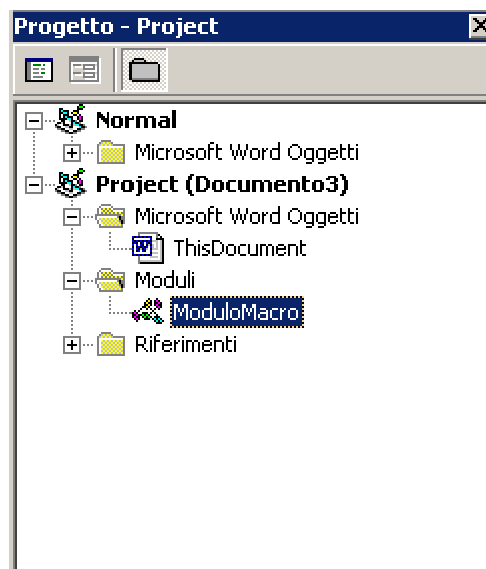
dove Normal rappresenta il modello col quale è stato generato il documento e Project (NomeDocumento) rappresenta il documento corrente.

Il modulo di codice evidenziato e denominato NewMacros rappresenta il contenitore delle macro generate dallo sviluppatore ed applicabili a tutti i documenti Word. Questo significa che è stato deciso di scrivere delle macro all'interno del modello Normal.

Per generare un nuovo modulo di macro applicabile unicamente al documento attivo (cioè al report generato dall'applicazione Visual Basic) si dovrà inserire il seguente blocco di codice:

```
Dim NewModule As Object
Set NewModule = appWord.activatedocument.VBProject.VBComponents.Add(1)
NewModule.Name = "ModuloMacro"
```

Il risultato dovrebbe rispecchiare il contenuto della figura sottostante:



Questo nuovo modulo di codice sarà il punto di partenza per andare a scrivere la nuove funzioni.

Inserimento delle funzioni nel modulo macro

Ciò che adesso si andrà a fare sarà il riempimento del modulo di codice denominato ModuloMacro. Com'è già stato anticipato si andrà a definire alcune funzioni che avranno lo stesso nome di alcune delle funzioni standard di Word.

In tal modo l'applicazione sostituirà le funzioni predefinite con le nuove, senza però eliminare permanentemente le prime.

Come esempio si consideri lo scenario prospettato nel corso dell'articolo precedente e non ancora completamente sviluppato: si desidera creare un report che inserisca in un nuovo documento un determinato titolo ed un testo, senza che l'utente finale abbia la possibilità ad esempio di salvare il documento come pagina Web. Questo caso è puramente casuale e si può facilmente estendere a qualsiasi altro evento.

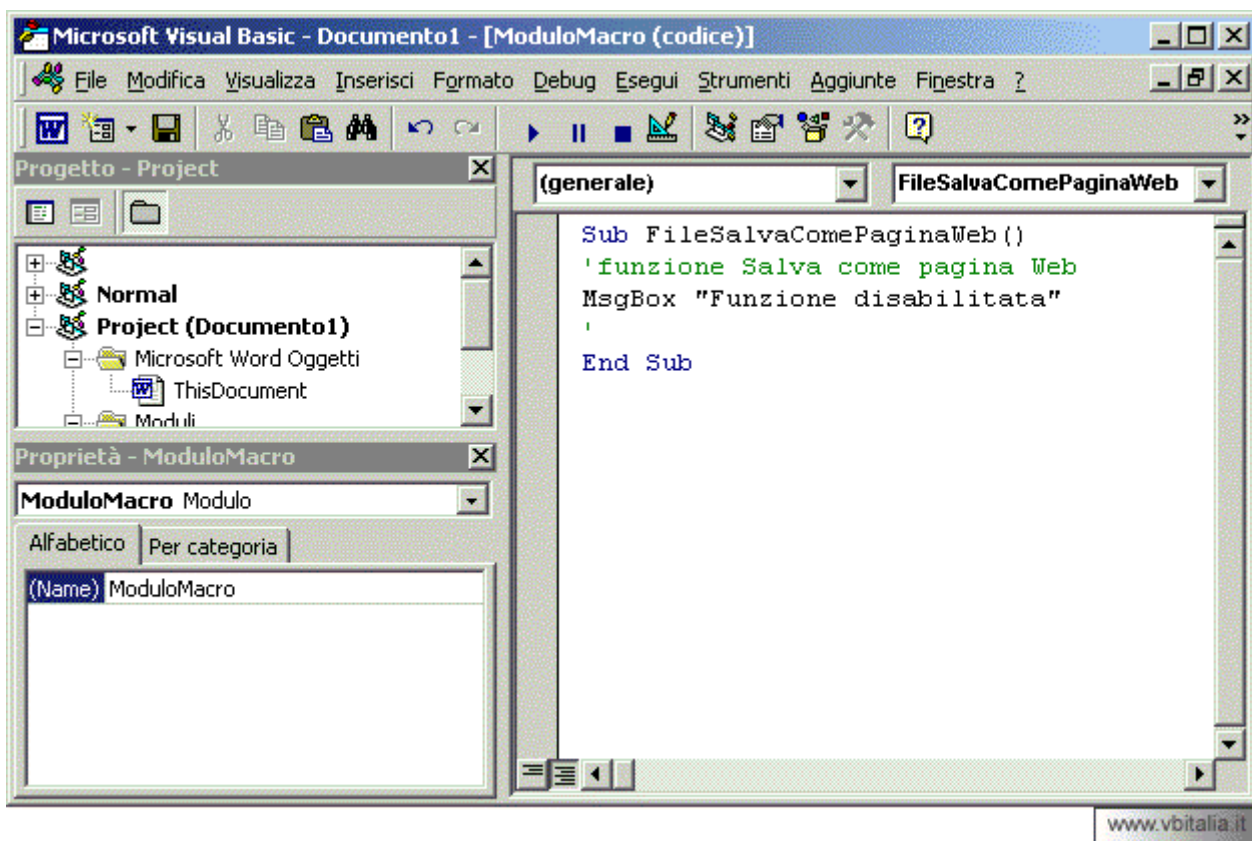
Per ottenere questo risultato sarà sufficiente utilizzare il metodo **InsertLines** come mostrato sotto:

```
NewModule.CodeModule.InsertLines 1, "Sub FileSalvaComePaginaWeb()" & _
vbCrLf & "'funzione Salva come pagina Web" & vbCrLf & _
"Msgbox ""Funzione disabilitata""" & vbCrLf & _
"" & vbCrLf & "End Sub"
```

L'intero che segue il metodo InsertLines indica il numero di linea sulla quale andare a scrivere la funzione o qualsiasi altro testo, mentre il testo che segue l'intero è ciò che si desidera inserire nel modulo di codice.

Come si può notare facilmente è stata riprodotta in formato testuale la struttura delle funzioni tipiche di Visual Basic.

Per averne una conferma sarà sufficiente aprire l'editor di codice VBE. Si otterrà un risultato simile a quello mostrato in figura:



Quando l'utente tenterà di salvare il documento come pagina Web otterrà come unico risultato una finestra di messaggio che lo avverte dell'impossibilità di utilizzare la funzione scelta.

NOTA: una volta inserite le funzioni nel modulo di codice queste rimangono di proprietà del documento. La chiusura dell'applicazione Visual Basic dalle quali sono state generate non significa pertanto l'eliminazione di tali funzioni dal documento.

Queste funzioni create appositamente possono poi risultare utili per evitare il prompt che richiede il salvataggio del lavoro alla chiusura della sessione o del documento: basterà sostituire alla funzione FileSalvaComePaginaWeb una qualsiasi funzione di Word elencata nell'editor di Macro come già visto in precedenza.

Funzioni create dall'utente

E' poi possibile creare una macro da eseguire in determinate occasioni. Ad esempio si potrebbe pensare ad una funzione che, all'apertura del documento Word visualizzi una finestra di benvenuto. Sarà necessario innanzitutto creare la macro. A tale scopo si può pensare a qualcosa di questo genere:

```
NewModule.Codemodule.InsertLines 1, "Sub Benvenuto()" & _
vbCrLf & "'funzione di benvenuto" & vbCrLf & _
"Msgbox ""Benvenuto "" & application.UserName & ""."" & vbCrLf & _
""" & vbCrLf & "End Sub"
```

Il messaggio visualizzerà pertanto il testo "Benvenuto" seguito dall'utente nel nome del quale è stata registrata l'applicazione Word.

E' poi sufficiente richiamare in qualsiasi punto si desideri dell'applicazione Visual Basic il metodo Run come mostrato di seguito:

```
appWord.Run "Benvenuto"
```

per visualizzare il messaggio di benvenuto.

Il metodo Run ha dalla sua parte una lunga lista di argomenti (per la precisione ben 31). Il primo è predefinito e corrisponde al nome della macro che si desidera avviare.

Dal parametro 2 al parametro 31 si possono definire a scelta i valori parametro di tipo Variant da passare alla funzione.