



Informazioni aggiuntive sulle funzioni

http://www.vbsimple.net/info/info_05.htm

In Visual Basic come in molti altri linguaggi esistono due genere di routine: le **Sub** e le **Function**. Le prime sono funzioni che non danno nessun valore di ritorno, mentre le Function permettono di specificare un valore che sarà il risultato della chiamata alla funzione.

La sintassi di base delle funzioni è:

```
Sub NomeDellaSub(Parametro1, Parametro2, ....)
    Istruzioni...
End Sub
```

```
Function NomeDellaFunction(Parametro1, ...) As TipoDati
    Istruzioni...
    NomeDellaFunction = Valore
End Function
```

Ogni funzione può includere uno o più parametri.

Se si dichiara un parametro ogni chiamata alla funzione dovrà includere valori o variabili per tutti i parametri dichiarati nell'intestazione della funzione.

Per esempio se una funzione richiede tre parametri sarà necessario passare per tutti e tre dei valori oppure delle variabili.

La sintassi per ogni parametro è:

```
[Optional] [ByVal / ByRef] NomeParametro [As TipoDati]
```

La parola chiave *Optional* definisce che il parametro è opzionale, non obbligatorio. Se un parametro è opzionale sarà possibile chiamare la funzione senza specificare nessun valore al suo posto.

TipoDati specifica un [tipo di dati](#) per il parametro. In mancata specificazione del tipo dei dati sarà assunto il tipo di dati Variant.

Le parole chiave *ByVal* e *ByRef* definiscono il tipo di chiamata per il parametro. *ByVal* indica una chiamata per valore, mentre *ByRef* indica la chiamata per riferimento. In una chiamata per valore viene creata una copia del valore specificato nella chiamata e verrà utilizzata la copia all'interno della funzione; ragion per cui ogni modifica al parametro non si riflette all'esterno della funzione. Una chiamata per riferimento invece passa un [puntatore](#) all'interno della funzione. Ogni modifica al parametro avrà effetti all'esterno della funzione.

Supponiamo di avere questo segmento di codice:

```
1. Sub CalcolaTriplo(ByVal Numero As Integer)
2.     Numero = Numero * 3
3. End Sub
4.
5. ...
6. X = 5
7. CalcolaTriplo X
8. Print X
```

L'ultima istruzione restituirà il valore 5 e non il valore 15 dato dal prodotto di 5 per 3. Il motivo è che il parametro Numero è stato passato per valore. Alla chiamata della funzione viene effettuata una copia in memoria del valore passato come parametro. Per cui all'interno della funzione il calcolo del Numero * 3 lavora su una copia del valore originale. All'uscita della funzione non rimarrà nulla di tale operazione.

Se invece il parametro Numero fosse stato dichiarato con ByRef il risultato dell'operazione avrebbe dato il numero 15. Poiché quando un parametro viene passato per riferimento ogni modifica fatta sul parametro si riflette sul dato passato nella chiamata.

Alternativamente avremmo potuto definire la stessa operazione con:

```
1. Function CalcolaTriplo(ByVal Numero As Integer) As Integer
2.     CalcolaTriplo = Numero * 3
3. End Function
4.
5. ...
6. X = 5
7. X = CalcolaTriplo(X)
8. Print X
```

Il segmento appena visto stampa il risultato 15, poiché la funzione CalcolaTriplo manda in uscita il prodotto di Numero per 3, pur lavorando su un parametro chiamato per valore.

Se una funzione, invece, riceve come parametro un oggetto caricato in memoria, ad esempio un form, la funzione lavorerà con un puntatore all'oggetto originale. In ogni caso, sia con *ByVal* sia con *ByRef* l'oggetto sarà sempre l'originale.

La spiegazione è molto semplice. Se dichiariamo un parametro con *ByVal* e poi gli passiamo un oggetto, viene trasferita alla funzione, una copia del puntatore - questa è la funzione di *ByVal* - ma essa punterà alla medesima locazione di memoria del puntatore originale.

Questo è il funzionamento di base di tutte le funzioni API, dichiarate tutte con *ByVal*.

Tuttavia, nella creazione di una nuova funzione, è importante non sottovalutare la possibilità di utilizzare *ByRef* invece che *ByVal*. Infatti, se la funzione in questione tratta stringhe, il compilatore in caso di *ByVal* sarà costretto ad [allocare](#) un'area di memoria e copiarvi dentro la stringa con cui lavorare, con un enorme spreco di tempo. Se si è assolutamente sicuri che la funzione non andrà a toccare la stringa passata come parametro, è consigliabile effettuare la chiamata mediante *ByRef*, poiché sarà richiesta soltanto l'allocazione di un [puntatore](#), senza che sia necessario ricopiare la stringa in questione.

Tale soluzione, purtroppo non funziona nel caso che la funzione riceva come stringa una

proprietà di qualche oggetto. Infatti le proprietà, anche se passate con ByRef non riflettono i loro cambiamenti tra le chiamate di funzione ed in ogni caso verrà creata una copia dei dati contenuti in essa.

Un ottimo esercizio su questi concetti è presente nell'[HowTo dedicato all'inversione di una stringa](#).

È estremamente importante definire bene il tipo di chiamata da effettuare: Sub o Function e parametri per valore o per riferimento. Entrambe le scelte hanno vantaggi e svantaggi.

Inoltre sia Sub che Function possono includere dei modificatori di accesso, ovvero una parola chiave che può essere *Private* o *Public*. Se una funzione viene definita con:

```
Private Sub NomeSub()
```

Avremo una Sub di nome NomeSub, senza parametri richiesti. La parola chiave Private indica che tale funzione non sarà accessibile in altre parti del progetto se non quella dove la Sub è dichiarata. Per esempio se avessimo un form di nome Form1 ed un'altro di nome Form2 e la dichiarazione appena vista fosse contenuta all'interno di Form1, il secondo form non potrebbe in nessun modo accedere e sfruttare la Sub dichiarata come Private.

Invece se la Sub fosse stata dichiarata con Public, qualunque segmento di codice all'interno del progetto avrebbe potuto utilizzare la Sub con una chiamata Form1.NomeSub.

I modificatori di accesso trovano nascita nella [programmazione ad oggetti](#) e stanno alla base dell'information hiding, ovvero il non esporre quei dati che dirigono situazioni specifiche interne ad una classe.

[Fibia FBI](#)

14 Novembre 2000
