Introduzione alle collezioni in VBA

Le **Collection**, (o Collezioni) sono una serie di elementi in cui ogni elemento ha le stesse caratteristiche, in altre parole, tutti gli elementi possono essere descritti nello stesso modo. In programmazione, una collezione è una serie di elementi in cui tutti gli elementi condividono le stesse proprietà e metodi, se presenti. VBA fornisce un oggetto Collection che è possibile utilizzare per memorizzare oggetti e dati e dispone di quattro proprietà:

- Add
- Count
- Item
- Remove

Non ci sono restrizioni sul tipo di dati che possono essere memorizzati in un oggetto Collection, e oggetti con tipi di dati diversi possono essere memorizzati nello stesso oggetto Collection. Excel dispone di molte collezioni incorporate, per esempio una cartella di lavoro ha una collezione di fogli, un foglio di lavoro ha un insieme di celle e così via. Possiamo fare riferimento a un foglio di lavoro con il suo indice nella collezione. Per esempio:

```
ActiveWorkbook.Worksheets (1).Visible
```

Oppure se il foglio è stato nominato, è possibile anche fare riferimento col nome.

```
ActiveWorkbook.Worksheets ("Dati"). Visible
```

Un altro aspetto delle Collection è che sono simili agli array, ma molto spesso viene preferita la collezione rispetto alla matrice. La ragione per cui si desidera lavorare con le collezioni piuttosto che con gli array è che si vuole evitare di dover ridimensionare le matrici ogni volta. Gli array sono fondamentalmente utile solo a patto che si sa, in anticipo, le dimensioni della nostra raccolta dei dati.

Le collezioni e gli array sono entrambi utilizzati con delle variabili di gruppo ed entrambi memorizzano una serie di oggetti simili, ad esempio un elenco di nomi o di paesi, ed è possibile manipolare facilmente e rapidamente un gran numero di elementi. Vediamo ora molto brevemente la differenza tra una variabile normale e un array. Se dobbiamo memorizzare i dati di uno studente si può facilmente farlo con una singola variabile in questo modo

```
Dim stud As Long
Stud = sheets(“Foglio1”).Range(“A1”)
```

Tuttavia si avrà da affrontare la manipolazione di dati per più studenti, pertanto immaginate di voler archiviare i dati di 100 studenti, se non è stata utilizzata una collezione o un array si avrebbe bisogno di creare un centinaio di variabili, una variabile per ogni studente. Inoltre un altro problema è che si devono utilizzare queste variabili singolarmente, cioè se si desidera memorizzare 100 valori allora abbiamo bisogno di una riga di codice ogni volta che si desidera memorizzare un valore in una variabile.

```
Dim stud1 As Long
Dim stud2 As Long
Dim stud3 As Long
stud1 = sheets("Foglio1").Range ("A1")
stud2 = sheets("Foglio1").Range ("A2")
stud3 = sheets("Foglio1").Range ("A3")
```

Come si può vedere nell'esempio sopra, la scrittura di codice come questo significherebbe dover scrivere centinaia di righe di codice ripetitivo, mentre se si utilizza una raccolta o un array è solo necessario dichiarare una variabile e utilizzare un ciclo per scorrerne tutti gli elementi, per cui con una raccolta è sufficiente una riga per poter leggere ulteriori valori. Se riscriviamo l'esempio precedente utilizzando una collezione, abbiamo solo bisogno di un paio di righe di codice

```
'crea la collezione
Dim Cstud As New Collection
'Leggi 100 valori della raccolta
Dim Col1 As Range
For Each Col1 In Foglio1.Range("A1:A100")
Cstud.Add Col1.Value
Next Col1
```

Con questo esempio abbiamo visto quello che le collezioni e gli array hanno in comune, allora, qual è la differenza e perché usare uno al posto dell'altro? La differenza principale è che con una serie normalmente si impostano le dimensioni una sola volta, questo significa che si conosce la dimensione prima di iniziare ad aggiungere elementi. Mi spiego con un esempio. Immaginate di avere un foglio di lavoro con un elenco di studenti e uno studente per riga

Cognome	*Nome*	*Materia*	*Nazione*
Abate	Carmine	Storia	Italia
Aiello	Michele	Lettere	Spagna
Aliberti	Filly	Lingue	Francia
Apostolico	Maria	Storia	Italia
Apreda	Daniele	Lettere	Germania
Boccardo	Tina	Storia	Grecia
Caldieri	Giuseppina	Matematica	Portogallo
Cantiello	Lara	Storia	Spagna
Carrella	Biagio	Lingue	Spagna
Cerrato	Maurizio	Scienze	Italia
Colombo	Verusca	Matematica	Germania
Cosenza	Biagio	Informatica	Inghilterra
Cretella	Giuseppe	Fisica	Grecia
D'amaro	Michele	Filosofia	Russia
De Crescenzo	Ilaria	Informatica	Polonia

Fig. 1

Ora se si desidera memorizzare le informazioni di ogni studente, in questo esempio si può facilmente

contare il numero di righe per ottenere il numero di studenti, in altre parole si conosce il numero di elementi in anticipo.

```
'trova l'ultima riga
Dim stud1 As Long
stud1 = Sheets("Foglio1").Range("A" & Rows.Count).End(xlUp).Row
'Crea un array di dimensione corretta
Dim arr() As Long
ReDim arr(1 To stud1)
```

Nel codice di esempio, si può vedere che si ottiene il numero di studenti contando le righe, per cui possiamo quindi utilizzare questo sistema per creare un array di dimensioni corrette. Vediamo ora in un secondo esempio in cui non conosciamo il numero di elementi in anticipo e vogliamo estrarre solo gli studenti con un dato criterio. Ad esempio solo gli studenti provenienti dalla Spagna o dall'Italia che studiano la matematica o la storia, in altre parole non sarà come selezionare uno studente e leggere i loro dati dal foglio di lavoro.

Immaginate anche che gli studenti possono essere aggiunti o rimossi dalla lista, quindi il numero di studenti non è fisso e varia, e non si conosce il numero di studenti in anticipo, quindi non sappiamo che dimensione assegnare alla matrice per crearla. Si potrebbe creare un array di grandi dimensioni, ma si avrebbero un sacco di slot vuoti, oppure leggere 50 studenti da un massimo di 1.000, ma allora si avrebbe 950 slot di matrice non utilizzati. Si potrebbe anche ridimensionare la matrice per ogni elemento come viene aggiunto, ma questo metodo è molto inefficiente e piuttosto disordinato, quindi, possiamo utilizzare una Collection in questo modo.

```
Dim coll As New Collection
coll.Add "Gino"
coll.Add "Mauro"
coll.Remove 1
```

Quando si aggiunge o si rimuove un elemento in una collezione, VBA fa tutto il ridimensionamento da solo, **non è necessario specificare le dimensioni o allocare nuovi spazi**, tutto quello che si deve fare è aggiungere un elemento o rimuoverlo. Le raccolte sono molto più facili da usare rispetto agli array soprattutto per chi non ha molta pratica nella programmazione, il più delle volte si fanno tre cose con le collezioni: Si creano, si aggiungono elementi e si leggono le voci

Si deve però tener presente nella scelta del metodo da adottare che le Collezioni hanno anche uno svantaggio, infatti **sono procedure in sola lettura**, pertanto si può aggiungere o rimuovere un elemento, ma non è possibile modificarne il valore. Se si ha la necessità di cambiare i valori in un gruppo di voci, allora è meglio utilizzare un array. Ora che sappiamo quando e perché utilizzare una collezione diamo un'occhiata a come usarle. È possibile dichiarare e creare in una sola riga del codice una collezione in questo modo

```
Dim coll As New Collection
```

Come si può vedere non è necessario specificare le dimensioni e una volta che la collezione è stata creata è possibile aggiungere facilmente elementi, inoltre è possibile dichiarare e quindi creare la collezione, quando se ne ha bisogno.

```
'Dichiarazione
Dim coll As Collection
'crea la raccolta
Set coll = New Collection
```

La differenza tra questi metodi è che nel primo la raccolta viene sempre creata, mentre nel secondo metodo viene creata solo quando si raggiunge la linea **Set**, così si potrebbe impostare il codice per creare solo la raccolta se una certa condizione viene soddisfatta

```
'Dichiarazione
Dim coll As Collection
'crea la raccolta se viene trovato un file
If filefound = True Then
Set coll = New Collection
End If
```

Il vantaggio di questo metodo è minimo, l'allocazione di memoria era un parametro importante tanti anni fa, quando la memoria del computer era limitata, a meno che non si stia creando un enorme numero di collezioni su un PC lento non si noterà alcun beneficio. Utilizzando la parola chiave **Set**, la raccolta si comporterà in modo diverso rispetto a quando si imposta la raccolta a Nothing, questa istruzione viene usata per rimuovere tutti gli elementi

```
Set coll = Nothing
```

Un punto importante da capire è che ciò che fa una collezione dipende da come è stata creata, come abbiamo visto è possibile creare una collezione dichiarandola utilizzando **New** o utilizzando **Set** e **New**. Diamo ora un'occhiata a entrambi i tipi

Creare una raccolta utilizzando New

Quando si aggiunge un nuovo elemento VBA imposta automaticamente la variabile Collection a una raccolta valida, in altre parole se si imposta la raccolta a **Nothing** vengono svuotati tutti gli elementi e se poi si aggiunge un elemento si avrà una collezione con una sola voce. Ciò rende più semplice per svuotare una raccolta, il codice seguente illustra questo aspetto

```
Sub Test ()
'crea la raccolta e aggiungi degli elementi
Dim coll As New Collection
Set coll = Nothing
Coll.Add "Pippo"
End Sub
```

Creare una raccolta utilizzando Set e New

Quando si utilizza la parola chiave **Set** per creare una collezione è necessario creare nuovamente la raccolta, se successivamente viene impostata a Nothing, nel codice che segue, dopo aver impostato la collezione a **Nothing**, si deve poi impostarla di nuovo utilizzando la parola chiave **New**, se non si esegue questa operazione si ottiene l'errore: "Variabile o Oggetto non impostato".

```
Sub Test ()
'crea la collezione
Dim coll As Collection
Set coll = New Collection
Coll.Add "Pippo"

Set coll = Nothing
Set coll = New Collection
Coll.Add "Franco"
End Sub
```

Rimuovere tutta la Collezione - un metodo alternativo

Il seguente metodo rimuove tutti gli elementi di una collezione, ma è molto lento, il vantaggio è che funziona indipendentemente dal modo in cui si crea la collezione.

```
Sub elimina (ByRef coll As Collection)
Dim k As Long
For k = coll.Count To 1 Step -1
Coll.Remove K
Next k
End Sub
```

Aggiunta di elementi a una raccolta

Per aggiungere elementi a una Collezione è possibile farlo utilizzando la proprietà **Add** seguita dal valore che si desidera aggiungere.

```
coll.Add "Pera"
coll.Add "Mela"
```

Quando si aggiungono elementi in questo modo vengono aggiunti al successivo indice disponibile, nell'esempio sopra riportato, Pera si aggiunge alla posizione 1 e Mela alla posizione 2.

I Parametri Before e After

È possibile utilizzare i parametri **Before** e **After** per specificare dove si desidera posizionare l'elemento della collezione.

```
coll.Add "Pera"
coll.Add "Mela"
'Aggiungere Limone come prima voce
coll.Add "Limone", Before: =1
```

Dopo questo codice la collezione è nell'ordine
1. Limone
2. Pera
3. Mela

```
coll.Add "Pera"
coll.Add "Mela"
'Aggiungere Limone dopo la prima voce
coll.Add "Limone", After: =1
```

Dopo questo codice la collezione è nell'ordine
1. Pera
2. Limone
3. Mela

Accesso agli elementi di una collezione

Per accedere alle voci di una raccolta è sufficiente utilizzare l'indice, che come abbiamo già visto, è la posizione della voce nella raccolta in base all'ordine che sono stati aggiunti. L'ordine può anche essere impostato utilizzando il parametro Before o After.

```
Sub Test ()
Dim coll As New Collection
coll.Add "Pera"
coll.Add "Mela"
' stampare Pera nella finestra Immediata
Debug.Print coll (1)
' Aggiungere fragola come prima voce
coll.Add "Fragola", Before: = 1
' stampare Fragola nella finestra Immediata
Debug.Print coll (1)
' stampare Pera nella finestra Immediata che ora è nella seconda posizione
Debug.Print coll (2)
End Sub
```

È inoltre possibile utilizzare la proprietà **Item** per accedere a un elemento della collezione, che è il metodo predefinito della raccolta in modo che le linee di codice siano equivalenti

```
Debug.Print coll (1)
Debug.Print coll.Item (1)
```

Abbiamo detto che non è possibile modificare il valore di un elemento in una collezione, perché quando si accede a un elemento di una raccolta è di sola lettura, per cui se si tenta di scrivere una voce della raccolta si ottiene un errore. Il seguente codice produce un errore "Necessario oggetto"

```
Sub Test ()
Dim coll As New Collection
Coll.Add "Mela"
' questa riga genera un errore
Coll (1) = "Pera"
End Sub
```

Aggiungere tipi di dati diversi

È inoltre possibile aggiungere diversi tipi di oggetti alla collezione.

```
coll.Add "Mela"
coll.Add 45
coll.Add N° 12/12/2015 #
```

Il codice seguente visualizza il tipo e il nome di tutti i fogli della cartella di lavoro corrente.

Nota: *Per accedere a tipi di dati diversi è necessario che la variabile sia dichiarata come Variant o avrete un errore.*

Si utilizza un ciclo For Each...Next quando si desidera ripetere un set di istruzioni per ciascun elemento di una raccolta o di una matrice.

```
Sub Test ()
Dim sh As Variant
For Each sh In ThisWorkbook.Sheets
' stampa nella finestra immediate il nome del foglio e il tipo di foglio
Debug.Print TypeName (sh), sh.Name
Next sh
End Sub
```

Aggiunta di elementi utilizzando una chiave (Key)

È inoltre possibile aggiungere elementi utilizzando una chiave come l'esempio sotto riportato

```
Coll.Add Item:=45, Key:=”Gino”
Debug.Print "Hai inserito: ",coll("Bill")
```

Ho incluso i nomi dei parametri per rendere l'esempio più chiaro, tuttavia non è necessario, basta ricordare la chiave, che è il secondo parametro, e deve essere una stringa univoca. Il codice seguente mostra un secondo esempio di utilizzo di chiavi

```
Sub Test()
Dim coll1 As New Collection
    coll1.Add 45, "Gino"
    coll1.Add 67, "Franco"
    coll1.Add 12, "Laura"
    coll1.Add 89, "Bruno"
    Debug.Print coll1("Franco")
    Debug.Print coll1("Bruno")
End Sub
```

Utilizzando la chiave si hanno tre vantaggi:

- Se l'ordine cambia il codice può accedere lo stesso alla voce corretta
- È possibile accedere direttamente alla voce senza leggere l'intera collezione
- Il codice è più leggibile

Nella collezione VBA delle cartelle di lavoro è molto meglio accedere alla cartella di lavoro con la chiave (il nome) in quanto l'indice è poco affidabile perché dipende da quando sono stati aperti e quindi è molto casuale.

Quando utilizzare le chiavi

Un esempio di quando usare le chiavi è il seguente: Immaginate di avere una collezione di nomi di 10.000 studenti con i loro ID (indici), si potrebbe aggiungere i 10.000 studenti ad una collezione utilizzando il loro ID studente come chiave. Quando si legge un ID dal foglio di lavoro è possibile accedere direttamente al nome dello studente.

Problemi ad usare le chiavi nelle collezioni

Ci sono tre problemi con l'utilizzo delle chiavi nelle collezioni

- Non è possibile controllare se la chiave esiste
- Non è possibile modificare la chiave
- Non è possibile recuperare la chiave

VBA contiene una classe simile alla Collezione chiamata dizionario, che permette di utilizzare sempre le chiavi per aggiungere un elemento. Il dizionario fornisce ulteriori funzionalità per lavorare con le chiavi e se avete bisogno di più funzionalità con le chiavi, si potrebbe trovare il Dizionario molto utile. Tornando alle Collezioni, se si ha bisogno di accedere direttamente a un singolo elemento possono essere molto utili, in caso contrario, non è conveniente usarle.

Accedere a tutti gli elementi di una collezione

Per accedere a tutti gli elementi di una collezione è possibile utilizzare un ciclo For o un For Each. Diamo un'occhiata a questi singolarmente.

Utilizzo del ciclo For

Con un normale ciclo For, si utilizza l'indice per accedere a ciascuna voce. L'esempio seguente stampa il nome di tutte le cartelle di lavoro aperte

```
Sub Test()
Dim k As Long
For k = 1 To Workbooks.Count
Debug.Print Workbooks(k).Name
Next k
End Sub
```

Come si può vedere nel codice abbiamo usato come indice 1 con Workbooks.Count, in quanto il primo elemento è sempre in posizione 1 e l'ultimo elemento è sempre nella posizione specificata dalla proprietà Count dell'insieme. Il prossimo esempio stampa tutti gli elementi di una collezione creati dagli utenti.

```
Sub Test()
'Dichiarare e creare la raccolta
    Dim coll2 As New Collection
    ' aggiungere gli articoli
    coll2.Add "Mela"
    coll2.Add "Pera"
    coll2.Add "Fragola"
    ' stampare tutte le voci
    Dim i As Long
    For i = 1 To coll2.Count
        Debug.Print coll2(i)
    Next i
End Sub
```

Utilizzo del ciclo For Each

Il ciclo For Each non usa l'indice ed il formato è mostrato nel seguente esempio

```
Sub Test()
Dim sh As Variant
    For Each sh In Workbooks
        Debug.Print sh.Name
    Next
End Sub
```

Il formato del **ciclo For** è:
For i = 1 To coll2.Count
Next i
dove i è una variabile Long e coll2 è una collezione. Il formato del **ciclo For Each** è:
For Each var In coll2
Next
dove var è una variabile Variant e coll2 è una raccolta. È importante comprendere la differenza tra i due cicli, il ciclo For Each:

- E' più veloce
- E' più ordinato da scrivere
- Ha un solo ordine, dal più basso al più alto

Mentre invece Il Ciclo For

- E' più lento
- E' meno ordinato da scrivere
- Si può accedervi con un ordine diverso

Se mettiamo a confronto i due cicli il For Each è considerato più veloce rispetto al ciclo For ed è più ordinato da scrivere, soprattutto se si utilizzano cicli annidati e si ha meno probabilità di avere errori. L'ordine del ciclo For Each è sempre dall' indice più basso al più alto e se si vuole ottenere un ordine diverso, allora si deve utilizzare il ciclo For, dove l'ordine può essere modificato. Inoltre è possibile leggere le voci in senso inverso, leggere una sezione degli articoli o è possibile leggere ogni due fogli.

```
Sub Test()
Dim i As Long 'scorrere i fogli da destra a sinistra
For i = ThisWorkbook.Worksheets.Count To 1 Step -1
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
For i = 1 To 3 'scorrere i primi 3 fogli
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
For i = 1 To ThisWorkbook.Worksheets.Count Step 2 'scorrere ogni due fogli
Debug.Print ThisWorkbook.Worksheets(i).Name
Next i
End Sub
```

Le collezioni sono una parte molto utile di VBA, sono più facili da usare rispetto agli Array e sono molto utili quando il numero di elementi cambia. Hanno solo quattro proprietà: **Add, Remove, Count e Item** e questo li rende molto facile da padroneggiare.