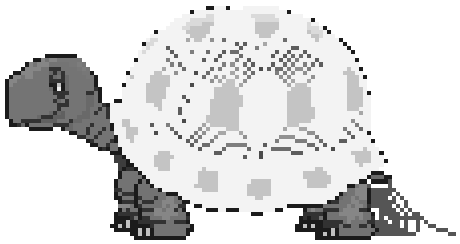
PROGRAMMAZIONE VBA-EXCEL E FINANZA

ANTONIO GRANDE



Una guida per i principianti

Antonio Grande
Programmazione VBA-Excel e finanza
Soluzioni VBA di problemi di finanza classica e moderna

ISBN: 978-88-907402-0-6

La versione digitale dell'opera è disponibile all'indirizzo:
https://www.memotef.uniroma1.it/node/7035

Email antonio.grande@uniroma1.it

a Pièveloce.
1955 – 2006

ABSTRACT

The VBA (Visual Basic for Application) language has among it's characteristics the capacity to interact with spreadsheet data, that has a very simple structure.
This study, after having examined the language basic notions, suggests a methodology to find out automatic solutions for classic and modern finance problems such as: the (pseudo)casual numbers generation, the price of the Asian and European Call/Put with Black-Scholes and Montecarlo methods, the efficient frontier with the Markowitz model, the download of stock prices from Internet.

ABSTRACT

Il linguaggio VBA (Visual Basic for Application) ha tra le sue caratteristiche quella di interagire con i dati del foglio elettronico che utilizza una struttura di dati concettualmente molto semplice.
Il libro, passati in rassegna i fondamentali dei linguaggio, propone le soluzioni automatiche di problemi di finanza classica e moderna tra cui citiamo: la generazione di numeri (pseudo)casuali, il calcolo di una put/call sia per le opzioni europee che per quelle asiatiche con Black-Scholes e Montecarlo, la frontiera efficiente con il modello di Markowitz, lo scaricamento delle quotazioni di titoli da Internet.

RINGRAZIAMENTI

L'anno scorso, durante un brevissimo viaggio in treno alla volta di una sede distaccata dell'università, la professoressa Giusy Bruno ed il sottoscritto, discutemmo sul progetto di una serie di appunti per la soluzione di problemi di finanza moderna con un linguaggio di programmazione.

L'idea del libro nasce da lì. Voglio ringraziare in modo particolare la professoressa Bruno per i consigli sulla scaletta degli argomenti e per la pazienza che ha avuto nelle occasioni in cui le ho chiesto chiarimenti e quant'altro.

La responsabilità di quanto è scritto è soltanto dell'autore.

Un altro sentito ringraziamento lo devo alle persone che si occupano di sviluppare e diffondere LaTeX, il software utilizzato per scrivere il libro, tra i quali ricordo: (mitico) Donald Erwin Knuth, **pantieri:2012**, **miede:2012** (cui si deve lo stile tipografico di questo libro), **bring:2008**, **pantieri:2012** nonché la comunità LaTeX.

Ultimo ringraziamento, non per questo di minore importanza, a tutti gli studenti dei corsi di Strumenti Informatici per la Finanza senza i quali, certamente, la qualità di quanto scritto sarebbe stata inferiore. In particolare vorrei ringraziare Claudio Trischitta ed Enkeleda Kertalli.

Antonio Grande

Roma, maggio 2012

INDICE

ELENCO DELLE FIGURE

Parte I

FONDAMENTALI VBA

INTRODUZIONE

CONTENUTO

Visual Basic for Application (VBA) è il linguaggio di programmazione Microsoft per la suite Office (Word, Excel, Power Point, Access).

Questo libro si occupa della soluzione di alcuni problemi di finanza classica e moderna con la programmazione in VBA del foglio elettronico Excel. Non è un manuale di Visual Basic.

Nella parte iniziale, capitoli 1-8, vengono illustrate la caratteristiche principali del linguaggio corredate da numerosi esempi. Al termine di ciascun capitolo è presente un paragrafo di esercizi relativi agli argomenti trattati.

I capitoli 9 e 10 sono dedicati a due argomenti di rilievo: le variabili con indice e le operazioni con le matrici.

Dal capitolo 11 in poi sono presentate le soluzioni di alcuni tra i principali problemi di finanza. Queste le soluzioni sono improntate alla generalizzazione del problema.

Alcune importanti precisazioni che riguardano lo "stile" con cui sono stati scritti i programmi. Per motivi legati alle finalità del libro, di carattere essenzialmente didattico, si tenga presente che:

- le istruzioni non sempre cominciano a partire dalla prima colonna ma sono disposte secondo una tabulazione (allineamento orizzontale) particolare. Questa tecnica, riconducibile alla "programmazione strutturata" , si utilizza per facilitare la leggibilità del programma ma non è assolutamente vincolante;

- sempre al fine di miglioramento della leggibilità, abbiamo evitato di ricorrere ad istruzioni o artifici di programmazione. In altre parole si è preferito scrivere i programmi in uno stile orientato alla facilità di comprensione più che alla velocità di esecuzione. Si tenga presente che questo approccio porta, di solito, ad un aumento del numero di istruzioni.

VERSIONE DEL PROGRAMMA

Tutto il codice del libro è stato verificato con il programma Microsoft Office 2003. La versione più recente di questo programma è la 2010.

SCARICAMENTO DEI PROGRAMMI

La versione PDF del libro ed il codice dei programmi sono disponibili all'indirizzo: http://antoniogrande.uniroma1.it

CONVENZIONI TIPOGRAFICHE

Nel testo del libro sono state adottate le seguenti convenzioni tipografiche:

- quello che segue è lo stile impiegato per riferirsi ad una cartella `Cartel1.xls`;
- quella riportata di seguito è un esempio della sequenza con cui nel libro ci riferiamo all'esecuzione di due comandi del menu comando1 → comando2;
- in tutti gli esempi di programmi del libro, le parole chiave del linguaggio, ossia i comandi VBA, sono riportati con un carattere più chiaro, mentre le parti restanti, decise in modo arbitrario dall'utente, sono scritte con un carattere nero.

  A titolo di esempio si consideri il codice 1.8. Le parole chiave in grigio se leggete su carta, in azzurro se leggete a video, sono "Sub", "End Sub", "MsgBox". Le parti restanti sono nomi di fantasia.

1

L'AMBIENTE VBA

Il linguaggio *Visual Basic (for) Application*, VBA nel seguito, è una evoluzione del Basic, il linguaggio in dotazione ai personal computer di prima generazione (inizi degli anni '80). VBA è integrato nel programma Microsoft Office che, come sappiamo, comprende i programmi Word, Excel, Power Point e Access. Tra le sue caratteristiche ricordiamo anche quelle di ammettere la programmazione *ad oggetti* (l'argomento sarà discusso in un apposito capitolo).

1.1 L'IDE DI VBA

L'ambiente che utilizzeremo per la scrittura dei programmi si presenta sotto forma di interfaccia grafica a finestre di tipo IDE o *Integrated Development Environment* (ambiente di sviluppo integrato). In una interfaccia di questo tipo, si dispone di un insieme *intregrato* di strumenti atti a facilitare il programmatore nella scrittura dei programmi. Gli strumenti dell'IDE VBA sono:

- l'editore di testi ovvero un ambiente che facilita la scrittura delle istruzioni;
- il debugger ossia lo strumento che facilita il controllo delle istruzioni;
- un sistema per la navigazione tra i componenti del linguaggio.

La visualizzazione di questa finestra, chiamata anche finestra VBA, si ottiene con i tasti Alt+F11 durante una sessione Excel. La vediamo riprodotta nella figura 1. Da qui, come sarà chiarito meglio nel prosieguo, potremo accedere a comandi, icone e quant'altro sia utile alla memorizzazione, la scrittura, l'esecuzione ed il controllo (debugging) dei programmi. Può essere suddivisa idealmente in due parti: una superiore ed una inferiore e ciascuna di queste è divisibile in ulteriori parti.

Nella zona superiore si trova la barra del menu e, sotto di questa, la barra degli strumenti. Nella parte inferiore, suddivisa in tre sottofinestre ridimensionabili e fluttanti, possiamo individuare:[1]

- la finestra di gestione dei progetti (il suo titolo è Progetto - VBAProject);

1 se queste finestre non fossero visibili eseguire, per ciascuna di esse i comandi del menu: Visualizza → Gestione progetti per la prima, Visualizza → Finestra proprietà per la seconda, Visualizza → Codice per la terza.

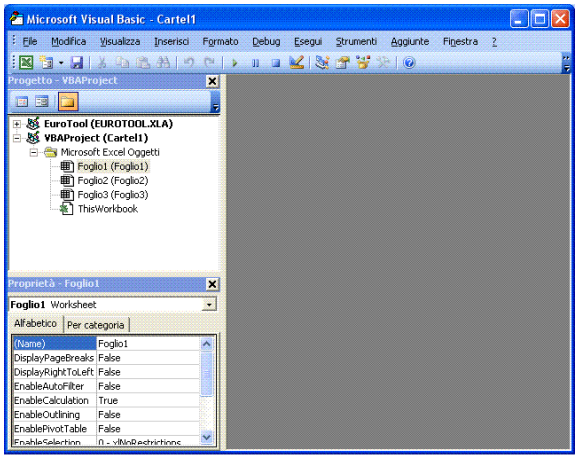


Figura 1: La finestra Visual Basic

- la finestra delle proprietà (il suo titolo è Proprietà - Foglio1);
- la finestra del codice (non visibile nella figura occupa normalmente la parte in grigio).

1.2 I COMANDI DEL MENU

La barra dei menu è composta da un insieme di comandi che, al momento attuale, non è il caso di conoscere in modo approfondito. La funzione dei comandi che la compongono sarà chiarita al momento in cui se ne presenterà l'occasione.

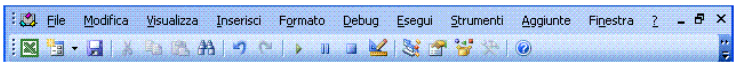


Figura 2: La barra dei menu (sopra) e la barra degli strumenti (sotto)

1.3 LA BARRA DEGLI STRUMENTI

La barra degli strumenti si trova sotto quella dei comandi. E' chiamata Standard in quanto esistono altre barre di strumenti che per semplicità, sono visualizzate a richiesta (si tratta delle barre Modifica, Debug e User Form). Al passaggio del mouse sopra le icone che la compongono, si visualizza il titolo corrispondente. Presentiamo una elencazione di quelli che necessitano di approfondimento in base all'ordine di visualizzazione.

Visualizza Microsoft Excel visualizza la finestra con la cartella Excel.

Inserisci User Form la sua funzione verrà commentata nel capitolo appositamente dedicato.

Salva salva la cartella di lavoro insieme a tutto cio' che si trova nella finestra dell'editor. Si tenga presente, se siamo partiti da una cartella vuota, che alla cartella di lavoro viene attibuito in automatico un nome costituito dal prefisso Cartel seguito da un numero intero progressivo. Se vogliamo attribuire un nome mnemonico alla cartella corrente si consiglia di non usare questa icona per il salvataggio. In questo caso è necessario ritornare nell'ambiente Excel e ricorrere ai comandi del menu File → Salva con nome.

Taglia

Copia

Incolla

Trova

Impossibile annullare

Impossibile ripetere

Esegui Sub/User Form si usa per eseguire un programma VBA; se ne parlerà nel prossimo paragrafo.

Interrompi sospende l'esecuzione del programma.

Ripristina si utilizza quando l'esecuzione del programma si arresta a causa di un errore. In questo caso viene evidenziata l'istruzione che lo ha generato e l'utente premendo questo pulsante riporta il programma allo stato iniziale (ovvero nello stato in cui si trovava prima dell'esecuzione).

Modalità progettazione

Gestione progetti visualizza la finestra corrispondente.

Finestra Proprietà visualizza la finestra corrispondente.

Visualizzatore Oggetti visualizza la finestra corrispondente.

1.4 LA FINESTRA PROGETTO

Si trova nella parte alta sinistra e visualizza un elenco dei progetti[2] sotto forma di diagramma ad albero collassabile (le cartelle attualmente aperte). Si tratta di una finestra che all'occorrenza può essere chiusa utilizzando l'icona a destra della sua barra del titolo. Per visualizzarla, oltreché nel modo descritto alla fine del paragrafo 1.1, si può ricorrere al pulsante Gestione progetti che si trova sulla barra degli strumenti di figura 2.

2 in prima approssimazione possiamo dire che un progetto coincide con una cartella ovvero un file di Excel.

Sotto la barra del titolo troviamo tre pulsanti. Del primo ci occuperemo ampiamente più avanti. Il secondo visualizza la cartella di lavoro (il foglio elettronico) mentre il terzo espande/comprime le cartelle che compaiono nella finestra del progetto.

1.5 LA FINESTRA PROPRIETÀ

La finestra Proprietà elenca tutte le proprietà dell'oggetto selezionato che di regola è il foglio corrente. Anche questa finestra, come la precedente, può essere chiusa. Per riaprirla basta utilizzare l'apposito pulsante che si trova a destra del pulsante Gestione progetti.

1.6 LA FINESTRA CODICE

La terza delle sottofinestre è quella c.d. del codice. In questa finestra verranno scritte le istruzioni del linguaggio VBA. Per visualizzarla si può ricorrere al tasto F7 nonchè alla sequenza di comandi del menu Visualizza → Codice.

Siamo ora in grado di scrivere il nostro primo programma che si compone delle istruzioni che riportiamo di seguito. Se facciamo attenzione a cosa accade dopo la scrittura della prima riga, ci accorgeremo che non appena avremo terminato la sua scrittura comparirà automaticamente l'istruzione di chiusura `End Sub`.

Listing 1: Una semplice procedura

```
1 Sub maschio()
     MsgBox "ciao bello!", ,"Titolo della finestra"
  End Sub
```

Si tratta di un semplicissimo programma (procedura in gergo VBA) chiamata `maschio`[3] che si compone di tre righe.

La prima e la terza riga contengono le istruzioni di inizio/fine programma. Le parentesi che seguono il nome della procedura vengono inserite automaticamente perché obbligatorie.

La seconda riga contiene il comando MsgBox che serve a visualizzare un messaggio all'interno di una finestra. Dopo il messaggio, racchiuso tra i primi doppi apici, vediamo due virgole separate da uno spazio (in seguito chiariremo il significato di questo) ed il titolo della finestra. Controllato di aver scritto tutto correttamente, possiamo eseguire questo programma.

L'esecuzione di una procedura si ottiene in uno dei seguenti modi[4]:

- premendo il tasto Esegui Sub/User Form della Barra degli Strumenti;

3 il nome di una procedura deve iniziare con un carattere alfabetico e può contenere soltanto lettere o numeri.

4 il cursore lampeggiante deve trovarsi sempre all'interno della procedura da eseguire.

- premendo il tasto F5;
- con i comandi del menu Esegui → Esegui Sub/UserForm.

Se non abbiamo commesso errori, ci apparirà la consueta finestra del foglio elettronico e, al suo interno, la finestra definita con l'istruzione 2 del codice. Il pulsante OK terminerà l'esecuzione ritornando alla finestra VBA.

1.7 GLI ERRORI DEL PROGRAMMA

Nel caso avessimo sbagliato qualcosa nella scrittura delle istruzioni è necessario cercare la causa dell'errore. Questa può dipendere sostanzialmente da due differenti motivi.

Per simulare il primo di questi cancelliamo una parte della seconda riga del programma (p.e. quella che segue la seconda virgola). In questo caso il comando manca di una sua parte essenziale, ovvero dell'intestazione della finestra. Non appena spostiamo il cursore dalla riga in cui ci troviamo viene visualizzata una finestra che riporta un messaggio simile a quello mostrato nella figura sottostante. Si nota

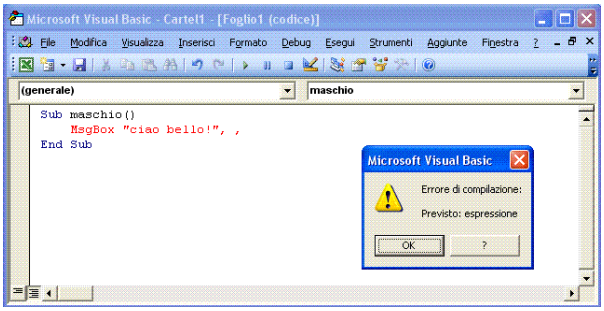


Figura 3: errore formale

come il comando che ha causato l'errore sia evidenziato con il colore rosso. Premuto il tasto OK possiamo ritornare alla finestra del codice per effettuare la correzione.

Diverso è il caso di istruzioni solo *formalmente* corrette. Per simulare questo, cambiamo la parola `MsgBox` in `MsgBux` e procediamo di nuovo all'esecuzione del programma.

La finestra dell'errore, simile alla precedente, riporta questa volta il messaggio Sub o Function non definita. Premuto il tasto OK ritorneremo alla finestra del codice come mostrato nella figura 4. Il programma è ancora in esecuzione ma è sospeso in attesa della correzione (si veda la dicitura [interruzione] che compare nella barra del titolo). Nella finestra sono evidenziate, in due diversi colori, la procedura in errore in giallo, ed il nome sconosciuto in blu. Una volta individuato l'errore, procediamo alla correzione e selezioniamo nuovamente l'icona di esecuzione.

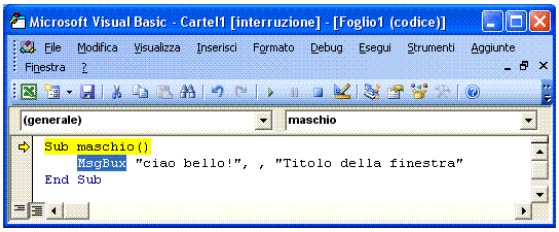


Figura 4: errore in esecuzione

1.8 L'ESECUZIONE DI UNA MACRO

Una volta passati in rassegna i componenti fondamentali dell'interfaccia con l'ambiente di programmazione, possiamo approfondire meglio il funzionamento delle procedure.

Chiudiamo la cartella di lavoro senza preoccuparci di salvare alcunché. Apriamo quindi la cartella di lavoro `ciccio1.xls` ed utilizzando la combinazione di tasti Alt F11 visualizziamo la finestra del codice.

Il codice di ciccio1.xls.

```
Sub Main()
    femmina
End Sub
Sub maschio()
   MsgBox "ciao bello!", , "Titolo della Finestra"
End Sub
Sub femmina()
   MsgBox "ciao bella!", , "Titolo della Finestra"
End Sub
```

Il codice è organizzato in tre procedure: `Main, maschio, femmina` (anche qui tre nomi di fantasia).

La procedura `Main`, al suo interno contiene una richiamo alla procedura `femmina`. Le procedure `maschio` e `femmina` contengono l'oramai noto comando che visualizza una finestra con la stessa intestazione ("Titolo della finestra") ma ciascuna con un differente messaggio. In presenza di questa organizzazione, costituita dal programma chiamante `Main` e due programmi, ciascuno a sé stante, valgono le seguenti regole:

- in caso di richiesta di esecuzione di una procedura (che si ottiene con un clic sull'opportuno tasto della barra degli strumenti) verrà eseguita solo quella in cui si trova il cursore lampeggiante;
- nel caso in cui venga eseguita una procedura che contiene una chiamata ad un'altra verrà eseguita anche quest'ultima laddove richiesto dal chiamante.

Per verificare quanto sopra possiamo eseguire una delle tre procedure spostando opportunamente il cursore lampeggiante. In alterna-

tiva possiamo sostituire il nome della procedura chiamata da `Main` scrivendo `maschio` al posto di `femmina` etc.

ESERCIZI

1. A partire da un foglio di lavoro vuoto, scrivere nel modo ritenuto più idoneo una procedura con le seguenti istruzioni:

```
1 Sub primo()
        MsgBox "a", , "b"
  End Sub()
```

   avendo l'accortezza di sostituire al posto di *a* un messaggio che riporta il giorno di nascita ed al posto di *b* un titolo per questa finestra.

2. Scrivere un programma che visualizza consecutivamente due finestre. La prima con il nome del vostro animale preferito, la seconda con la vostra data di nascita. Ciascuna di queste finestre dovrà avere il titolo corrispondente all'informazione cui si riferisce.

2 DATI ED ESPRESSIONI

Ci occuperemo ora delle tipologie di dati che può trattare un programma VBA nonché dello spazio che ciascuno di questi dati occupa in memoria. Allo scopo di chiarire meglio di cosa stiamo parlando, faremo ora alcune considerazioni preliminari che ci permetteranno esporre l'argomento oggetto di questo capitolo.

2.1 TIPOLOGIE DI DATI

Consideriamo la sequenza di caratteri "12042000". Apparentemente sembrerebbe trattarsi di un numero, ma a ben guardare potrebbe trattarsi anche di una data e precisamente del 12 aprile dell'anno 2000 o ancora del numero di telefono 12 04 20 00. Come facciamo, allora a determinare di cosa si tratta? La risposta è semplice (o difficile fate voi) e dipende dall'impiego che dovremo fare del dato in questione.

Se, per esempio, si tratta dell'importo di una fattura, probabilmente lo utilizzeremo per eseguire dei calcoli, conseguentemente si tratta di un numero (sarebbe meglio dire dato numerico).

Se invece è una data, la utilizzeremo in modo particolare ed infatti sommando a questa un valore, supponiamo 30, vorremmo ottenere 12052000 cioè il 12 maggio dello stesso anno, non 12042030 come se si trattasse di un numero.

La situazione si complica nel caso di un numero telefonico perchè esso, pur essendo costituito da caratteri numerici, non sarà mai utilizzato a fini di calcolo!. Ecco allora la necessità di dover sapere sempre la natura o l'impiego di un dato in un programma[1].

2.2 COSTANTI E VARIABILI

I dati all'interno di un programma si classificano come variabili o costanti. Un dato di tipo variabile è riconoscibile perché identificato attraverso un nome, un dato di tipo costante non è identificato attraverso un nome ma con il valore corrispondente.

Nella tabella 1, sono riportati alcuni esempi di costanti e di variabili. Si noti la differenza tra `Uffa` nome di variabile e `"Uffa"` costante. La seconda delle due è riconoscibile come tale, per essere racchiusa tra i doppi apici.

Riguardo al nome di una variabile possiamo dire che questo può essere costituito da una sequenza qualsiasi di caratteri (fino ad un

1 Un numero di telefono, pur essendo costituito da caratteri numerici, va considerato alla stregua di un nome.

massimo di 255). L'iniziale del nome deve essere necessariamente una lettera dell'alfabeto (maiuscola o minuscola). Si consiglia, ma non è obbligatorio, di scegliere un nome di variabile che ricordi, mnemonicamente, la funzione che svolge all'interno del programma.

Tabella 1: costanti e variabili

Costanti	Variabili
123, "123" , "Uffa!123"	Uffa, X123, Abc

2.3 LA DICHIARAZIONE DELLE VARIABILI

Abbimo visto in apertura quanto sia importante conoscere l'impiego cui è destinato una dato. Questo principio trova in VBA una immediata soluzione costituita dalle istruzioni di dichiarazione delle variabili[2]. L'istruzione di dichiarazione si mette di solito all'inizio della procedura e si utilizza per associare a ciascuna variabile il tipo di dato che questa dovrà contenere.

La cartella di lavoro `dichiarazione_delle_variabili.xls` riporta parecchi esempi di istruzioni di dichiarazione di dati. Nel codice 2 sono visibili le prime righe della procedura della cartella di lavoro.

Listing 2: Le dichiarazioni delle variabili

```
Dim CiccioRiccio, Caterina, _
      Tio    ' dichiarazioni senza tipo (viene assunto il tipo
          Variant)
             ' si esemplifica l'uso del carattere di
                continuazione
Dim Nome As String * 10 ' variabile stringa a lunghezza fissa
       . . .
```

Con la prima istruzione `Dim` si dichiarano tre variabili `CiccioRiccio, Caterina, Tio`. Il carattere finale della prima riga "_" è un carattere di continuazione dell'istruzione alla riga successiva.

Il carattere "'" (apice singolo) che si trova dopo `Tio`, rappresenta un commento. Tutto ciò che segue questo carattere non ha influenza sull'esecuzione del programma.

Dopo il nome o i nomi viene la descrizione del tipo di dato che deve essere preceduta dalla parola `As`. I tipi di variabile, lo spazio di memoria che occupano in bit[3], i numeri che si possono rappresentare

2 La dichiarazione delle variabili non è obbligatoria ma la sua mancanza comporta una riduzione nella velocità di esecuzione. Noi la effettueremo in tutti i programmi del libro.

3 Un bit è un dispositivo fisico in grado di memorizzare una informazione binaria come per esempio 0 oppure 1. Si ricorda che 1 byte = 8 bit.

	A	B	C	D
1				
2	TIPO	LUNGHEZZA IN BYTE	ESTREMI/VALORI AMMESSI	
3				
4	Array	a seconda di quanto dichiarato		
5	Byte	8 bit	0	255
6	Boolean	16 bit	True	False
7	Currency	64 bit	(~)-922.000.000.000.000,5808	(~)922.000.000.000.000,5808
8				
9	Date	64 bit	1 gennaio 100(00:00:00)	31 12 9999(23:59:59)
10	Double	64 bit	-1.(14cifre)E308	4.(14cifre)E-324
11			-4(14cifre)E-324	1.(14cifre)E308

Figura 5: Tipo, bytes e valori corrispondenti

con quel tipo di dato (se si tratta di variabile numerica), si trovano nelle due colonne successive del foglio Excel e sono parzialmente riprodotti nella figura 5. In base a quanto vediamo, possiamo dire per esempio che se una variabile è dichiarata di tipo *Byte*, non potrà essere utilizzata in calcoli che eccedano l'intervallo $0, 255$.

2.4 ISTRUZIONI DI ASSEGNAZIONE

Nella parte di codice che segue le dichiarazione delle variabili sono riportate, a titolo d'esempio, diverse istruzioni di assegnazione.

La forma generale di una istruzione di questo tipo è:

$$nome\_di\_variabile = espressione\_qualsiasi$$

in cui *espressione_qualsiasi* è qualunque combinazione di nomi di variabili con simboli aritmetici, mentre *nome_di_variabile* è il nome di una variabile.

L'istruzione di assegnazione si utilizza per attribuire il risultato di un calcolo (a destra del segno di uguaglianza), ad una variabile (a sinistra del segno di uguaglianza). Non deve essere inteso come una espressione matematica! Prima di approfondire (si veda la sezione appositamente dedicata di seguito), commentiamo gli esempi proposti nel codice sottostante.

dichiarazione_delle_variabili.xls

```
        . . .
Dim Nome As String*10
                . . .
        DataNascita = #9/26/1953#
' calcoliamo i giorni passati dalla data di nascita ad oggi
' Now calcola la data odierna
        GiorniVissuti = Now - DataNascita
' alcuni esempi di istruzioni di assegnazione e di uso di
  operatori:
'   assegnazione di una costante alfabetica ad una variabile
        Nome = "Antonio"
' - effettua il calcolo di una espressione ed assegna il
  risultato ad una variabile
        Gio = 1 + 1.51
' - assegna una variabile ad una cella (scrive nella cella A21)
```

```
        Cells(21, 1).Value = Gio
' - legge il contenuto della cella C21,
'     somma a questro la costante numerica 3,
'     scrive il risultato nella cella A23)
        Cells(23, 1).Value = Cells(21, 3).Value + 3
        . . .
```

La prima istruzione dopo `Dim` è un esempio di assegnazione di una data in formato inglese (mm/gg/aaaa) ad una variabile. A destra dell'uguale abbiamo una costante data riconoscibile perché racchiusa tra doppi cancelletti.

La seconda istruzione calcola la differenza, espressa in giorni, tra la data odierna (rappresentata da `Now`) ed il contenuto della variabile `DataNascita` (attualmente questa variabile contiene il dato 26 settembre 1953).

Il significato delle istruzioni successive è facilmente comprensibile dai commenti inseriti nel codice.

2.4.1 *L'incremento di una variabile*

Una conseguenza notevole della logica sottesa all'istruzione di assegnazione riguarda il concetto di incremento di una variabile. Con incremento di una variabile si intende la somma o la sottrazione di una certa quantità ad una variabile, riassegnando poi così ottenuto alla stessa variabile. Per poter fare questo dovranno essere definite almeno due funzionalità:

- una istruzione iniziale del valore 0 alla variabile (di 1 nel caso della produttoria);
- l'istruzione di assegnazione per l'incremento/decremento.

Un esempio di questo tipo è visibile nel codice seguente che si trova scritto nella parte finale della procedura:

```
        . . .
        VariabileModulo = 0
        VariabileModulo = VariabileModulo + 2
        . . .
```

Il significato della seconda riga di codice, riprendendo quanto detto all'inizio del paragrafo, è da intendersi:
il contenuto di `VariabileModulo`, pari a 0, deve essere sommato a 2 ed il risultato dell'operazione deve essere assegnato a `VariabileModulo`.

2.4.2 *Le espressioni logiche*

Oltre ai valori visti nei paragrafi precedenti, esiste un altro tipo di dato, e quindi una variabile, chiamato *logico* o *booleano*. Esso può valere "Vero" oppure "Falso". Utilizzando opportuni operatori, chia-

mati operatori logici, è possibile costruire espressioni logiche il cui risultato può essere assegnato a variabili dello stesso tipo.

Listing 3: Una espressione logica

```
1         . . .
  Dim Verro As Boolean
          . . .
          Verro = 4 > 5
          . . .
```

Nel listato vediamo un esempio di espressione logica il cui risultato viene assegnato ad una variabile. Si noti l'uso dell'operatore logico ">" che definisce il tipo di espressione. Il dato contenuto in Verro è Falso in quanto 4 è minore di 5. Come già accennato nel paragrafo 2.4 abbiamo proceduto alla dichiarazione della variabile.

ESERCIZI

1. Nel codice VBA esaminato in questo capitolo, sostituire la data di nascita già presente con la vostra data di nascita. Visualizzare in una MsgBox il risultato ottenuto.

2. Scrivere una procedura che visualizza in una MsgBox il vostro cognome e nome separati dal carattere spazio. Si ipotizza che le celle A1 e B1 contengano rispettivamente il vostro nome e cognome. Nel codice, a parte le dichiarazioni di inizio/fine procedura, devono essere utilizzati solamente:
   - comando MsgBox;
   - l'operatore di concatenazione di stringhe &;
   - Cells(1,1).Value e Cells(1,2).Value;
   - la costante spazio " ".

3. Come nell'esercizio 2 ma assegnando il contenuto delle celle A1 e B1 ad altrettante variabili dichiarate in modo opportuno. Di conseguenza la MsgBox dovrà contenere i nomi delle variabili e non il riferimento alle due celle del foglio Cells(1,1).Value, Cells(1,2).Value.

4. Tre celle contengono tre numeri interi. Calcolare la somma e scrivere il risultato in un'altra cella.

5. Ricordando che:

   $$5 : 3 = 1 \quad (\mathrm{resto2})$$

   e che:

   $$\mathrm{quoto} \quad \mathrm{x} \quad \mathrm{quoziente} + \mathrm{resto} = \mathrm{dividendo}$$

   calcolare il resto e il quoziente della divisione tra la somma calcolata nell'esercizio precedente con il numero 3 e scrivere questi due risultati in due celle scelte a piacere.

6. Calcolare il risulato della moltiplicazione tra la variabile ciccio e 5 assegnando nuovamente il risultato del calcolo alla variabile ciccio. Il valore di ciccio prima della moltiplicazione deve essere opportunamente inizializzato.

3 LE FUNZIONI

Qualunque linguaggio di programmazione è in grado di eseguire una delle quattro operazioni aritmetiche. Ognuno poi, in relazione all'ambiente cui è orientato, dispone di un insieme aggiuntivo di operatori, simili ai quattro operatori aritmetici e di uso frequente, che hanno lo scopo di facilitare l'utente nella soluzione di problemi. Questo insieme di operatori prende il nome di *funzioni*.

Il lettore pratico del programma Excel ne dovrebbe conoscere qualcuna, per averla utilizzata nella scrittura di formule in celle del foglio. Vedremo adesso come si possono calcolare le funzioni in VBA.

3.1 LE FUNZIONI EXCEL IN VBA

Nel programma VBA è ammesso il riferimento alle funzioni di Excel ma bisogna tenere presente alcune considerazioni.

La prima è relativa al fatto che le funzioni VBA e le funzioni di Excel, facendo parte di due applicazioni diverse, vanno riferite in modo diverso. Per questo motivo in VBA il riferimento ad una funzione Excel deve premettere il termine Application, ovvero il nome del programma Excel, al nome della funzione separando le due quantità con il carattere ". ".

Per esempio se vogliamo calcolare il massimo di quattro valori dovremo scrivere: Application.Max(3, 5, 123, 18).

Si nota che il nome della funzione Excel deve utilizzare il suo corrispondente in lingua inglese[1] [2].

Una seconda avvertenza riguarda la specificazione di zone di celle. VBA non segue la sintassi di Excel nel riferimento a zone di celle. Per esempio, in relazione alla funzione di somma, non sarà possibile scrivere `Application.Sum(A1:G1)`! Il problema, che richiede conoscenze supplementari, verrà trattato e risolto nel paragrafo 7.7.

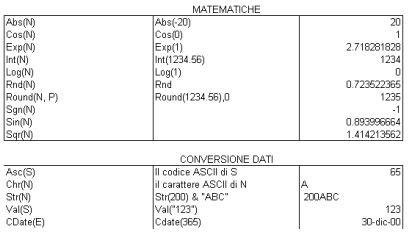
3.2 LE FUNZIONI VBA

Il programma VBA è dotato di un insieme di funzioni proprie. Nel foglio di lavoro `funzioni.xls` abbiamo elencate le principali, raggruppandole in base al tipo di risultato che producono. Nella colonna A

1 Per un elenco con i nomi delle funzioni inglesi, relativi a quelli italiani, si consulti l'indirizzo http://support.microsoft.com/kb/638465/it, oppure si inserisca in un motore di ricerca il termine *Cross Reference Italiano-Inglese funzioni del foglio*.

2 Nella fattispecie Max ha lo stesso nome sia in inglese che in italiano di conseguenza non si trova nell'elenco suddetto.

si trova il nome della funzione e gli argomenti richiesti (N=numero, S=stringa, D=data, E=espressione qualsiasi), nella colonna B si trova una istruzione di esempio, nella colonna C (quando ritenuto significativo), il risultato ottenuto. Le istruzioni VBA, utilizzate per il calcolo delle funzioni, sono contenute nella procedura Funzioni visibile nella finestra del codice (Alt+F11). I risultati sono riportati nella colonna C.



MATEMATICHE		
Abs(N)	Abs(-20)	20
Cos(N)	Cos(0)	1
Exp(N)	Exp(1)	2.718281828
Int(N)	Int(1234.56)	1234
Log(N)	Log(1)	0
Rnd(N)	Rnd	0.723522365
Round(N, P)	Round(1234.56),0	1235
Sgn(N)		-1
Sin(N)		0.893996664
Sqr(N)		1.414213562
CONVERSIONE DATI		
Asc(S)	Il codice ASCII di S	65
Chr(N)	il carattere ASCII di N	A
Str(N)	Str(200) & "ABC"	200ABC
Val(S)	Val("123")	123
CDate(E)	Cdate(365)	30-dic-00

Figura 6: Le principali funzioni VBA

Di seguito riepiloghiamo i principali punti da tenere presenti riguardo ad una funzione:

- calcola un solo valore;
- si identifica con un nome (solitamente) seguito da uno o più argomenti racchiusi tra parentesi tonde;
- può trovarsi in una espressione purché abbia un risultato compatibile con il dato di quell'espressione;
- i suoi argomenti possono essere sostituiti con altre espressioni purché compatibili.

Se per esempio, una funzione richiede come argomento un numero intero, al posto di questo potranno essere specificati: un numero intero, un'espressione aritmetica il cui risultato è un numero intero, l'indirizzo di una cella contenente un numero intero o una espressione numerica intera[3].

3.3 LA GUIDA DELLE FUNZIONI

Nel caso volessimo ottenere un aiuto maggiore, si può ricorrere ad una procedura che visualizza una descrizione di una determinata funzione (nonché di altri componenti del linguaggio) con alcuni esempi relativi al suo uso. I passaggi per l'attivazione di questa guida sono costituiti da:

1. selezione del comando Visualizza → Visualizzatore oggetti nel menu dei comandi;

3 La sostituibilità dei dati è molto utile nella programmazione perché rappresenta il modo di rendere "generalizzabile " un programma. Vedremo in seguito numerosi esempi di applicazione di tale principio.

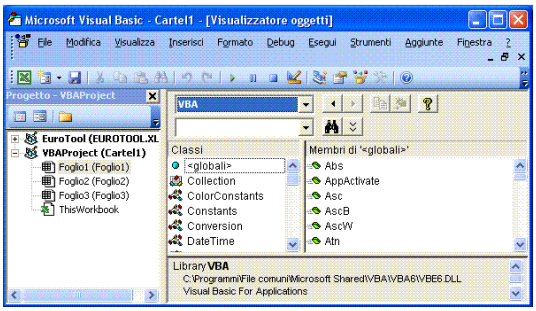


Figura 7: Il visualizzatore di oggetti

2. selezione del termine VBA nel primo menu a tendina in alto a sinistra visualizzato nella nella sottofinestra del codice;

3. (facoltativo) nella finestra di scorrimento intitolata Classi, subito sottostante quella del menu a tendina VBA, selezione della famiglia cui appartiene la funzione che cerchiamo (Conversion, DateTime, FileSystem, Financial etc.);

4. nel menu di destra compare l'elenco di tutte le funzioni di quella famiglia;

5. selezione della funzione;

6. premere il tasto "?" che si trova sopra l'elenco delle funzioni che stiamo consultando. A questo punto viene visualizzata la guida relativa alla funzione che ci interessa[4].

Una prima alternativa molto più rapida consiste nell'utilizzo del tasto F2 quando ci troviamo in ambiente VBE. In questo modo otterremo immediatamente la comparsa della finestra Visualizzatore oggetti.

Una seconda opzione consiste nel fare clic con il tasto sinistro del mouse in corrispondenza di uno qualunque dei caratteri del nome della funzione scritta nel codice VBA. A questo punto si preme il tasto F1 che aprirà la finestra della guida di quella funzione.

3.4 LA MODIFICA DELL'ORDINE DEGLI ARGOMENTI

Quando si vuole modificare l'ordine degli argomenti di una funzione si deve ricorrere ad una sintassi alternativa nella quale sia presente l'associazione tra nome e valore. La procedura finale presente nella cartella riferita in apertura del paragrafo 3.2 costituisce un esempio applicato alla funzione MsgBox. Le sue istruzioni sono visibili nel codice 4.

Ricordando che il primo ed il terzo argomento della funzione sono chiamati rispettivamente Prompt e Title, vediamo come come la

4 Deve essere installata la *Guida in linea di Microsoft Visual Basic*, diversamente la finestra apparirà vuota. Questo componente può essere installato in un momento successivo.

sintassi alternativa, presente nella seconda MsgBox, non abbia necessità di specificare il secondo argomento. Questo si spiega con il fatto che l'associazione argomento-valore viene fatta solamente per gli argomenti presenti. Si noti anche il carattere di separazione ":=" tra il nome di un argomento ed il suo valore[5].

Listing 4: La modifica degli argomenti della funzione

```
Sub messageBox()
    MsgBox "Suggerimento in posizione a priori", , "Titolo"
' sintassi alternativa che specifica i parametri della funzione
' secondo un ordine diverso da quello stabilito a priori
    MsgBox Title:="Titolo", Prompt:="Suggerimento specificato in
        posizione diversa"
End Sub
```

ESERCIZI

1. In A1 è scritto il tuo codice fiscale. Riporta in A2 il codice del mese in cui sei nato (il mese è codificato nel nono carattere).

2. La cella A1 contiene la costante stringa "12,2" (ovviamente nella cella non ci sono i doppi apici). Si deve scrivere in B1 il numero 12 e in C1 il numero 2.

3. La cella A1 contiene una costante stringa simile alla precedente in cui le parti che precedono e seguono la virgola hanno un numero di cifre non conosciuto a priori (mai però maggiore di 4). Si deve scrivere in B1 il numero che precede la virgola e in C1 il numero che segue la virgola.

4. Utilizzando lo strumento ritenuto più opportuno, chiarire la differenza tra le funzioni `Str` e `CStr`.

5. La cella A1 contiene il tuo indirizzo di posta elettronica. Scrivere la procedura che calcola in B1 ed in C1 rispettivamente il prefisso ed il suffisso (stringa che precede e stringa che segue) il carattere "@ " in altrettante celle a tuo piacimento. Al termine visualizza con un comando `MsgBox` il risultato separando le due parti con un "a capo".

6. In A1 è scritto il tuo codice fiscale. Sapendo che il carattere 11 e 12 contengono il giorno della data di nascita, che nel caso di femmine alla data di nascita viene aggiunta la costante 40, si chiede il programma che scrive in una cella scelta a piacere il sesso corrispondente al codice fiscale.

5 In una procedura, terminata la scrittura di una funzione, dopo aver inserito la parentesi aperta, compare una finestra con il nome di quella funzione seguita dall'elenco degli argomenti.

7. La cella A1 contiene un codice fiscale. Spacchettarlo nei suoi componenti fondamentali: cognome, nome, anno di nascita, codice del mese di nascita, giorno di nascita, codice Istat del comune, carattere di controllo (ovvero inserire queste parti in altrettante variabili applicando una opportuna funzione. Fatto questo riscrivere il codice fiscale intero, quello ottenuto dallo spacchettamento, nella cella B1.

4 LE ISTRUZIONI CONDIZIONALI

Negli esempi visti finora, le istruzioni del programma venivano eseguite comunque. Questa modalità non è adatta a risolvere le situazioni in cui è richiesta una diversificazione nei comportamenti del programma in relazione a determinati eventi.

Quando questo accade è possibile utilizzare un insieme di istruzioni, chiamate "condizionali", che definiscono percorsi alternativi in relazione al codice da eseguire. Vediamo di cosa si tratta.

4.1 IF MONOISTRUZIONE

La prima e più semplice istruzione condizionale ha la seguente sintassi:

```
If |*espressione_condizionale*| Then |*istruzioni*|
```

in cui *espressione_condizionale* è una qualunque espressione logica, come quella del codice 3, e *istruzioni* rappresenta una o più istruzioni che devono trovarsi nella stessa riga dell'`If`. Qualora le istruzioni siano più di una, deve essere utilizzato il carattere ":" a separare l'una dall'altra[1].

A titolo di esempio, cui ci riferiremo in tutto il capitolo, supponiamo di voler calcolare le soluzioni di una equazione di secondo grado, dati i tre coefficienti A, B, C[2].

Nella cartella di lavoro `ifthenelse.xls` la procedura `IfThenElse1` contiene il primo esempio di `If` monoistruzione. Con questo tipo di istruzione condizionale la capacità elaborativa è molto bassa in relazione al problema che dobbiamo risolvere. L'unica possibilità che abbiamo è quella di scrivere una stringa relativa al tipo di soluzioni (reali/immaginarie). Nel codice 5 ne vediamo un esempio: il risultato del calcolo di Δ, viene riportato in A1.

Listing 5: If monoistruzione

```
Sub IfThenElse1()
    Dim A As Single, B As Single, C As Single, Delta As Single
    A = 2
    B = 5
    C = 2
```

1 Se ne sconsiglia l'uso per motivi di leggibilità. Si veda nel paragrafo successivo come risolvere questo problema.

2 L'assegnazione dei tre valori numerici alle corrispondenti variabili è stata fatta con istruzioni di assegnazione. Questa scelta comporta una perdita di generalità nella soluzione ma si giustifica per motivi di semplicità.

```
    Delta = (B ^ 2) - 4 * A * C
    Cells(1, 1).Value = "Due soluzioni reali(distinte/non)"
    If Delta < 0 Then Cells(1, 1).Value = "Due soluzioni
        immaginarie"
End Sub
```

Cambiando i dati assegnati alle tre variabili A, B, C, sarà possibile effettuare altri calcoli però le potenzialità rimarranno comunque le stesse[3].

Fortunatamente l'istruzione `If` dispone di altre e più complesse forme di condizioni. Vediamo quali sono.

4.2 IF SEMPLICE

La soluzione migliore quando si debbano eseguire più istruzioni in subordine ad una condizione, è quella dell'`If` semplice. La sua sintassi è:

```
If |*espressione_condizionale*| Then
        . . .
   |*istruzioni*|
        . . .
  End If
```

Come visto in precedenza questa soluzione non si presta ad aumentare le nostre capacità risolutive in relazione al problema da risolvere.

4.3 IF .. THEN .. ELSE

Quando si debbano eseguire insiemi di istruzioni mutuamente esclusivi, si impiega questo tipo di istruzione. La sintassi è:

```
If |*espressione_condizionale*| Then
        . . .
   |*istruzioni (blocco 1)*|
        . . .
  Else
        . . .
   |*istruzioni (blocco 2)*|
        . . .
  End If
```

In questo caso l'insieme di istruzioni che si trovano prima di `Else` viene eseguito se la condizione è vera mentre l'insieme compreso tra `Else` e `End If` viene eseguito se la condizione è falsa. Nel codice riprodotto di seguito è visibile un esempio di questo tipo.

```
Sub IfThenElse2()
    Dim A As Single, B As Single, C As Single, Delta As Single
```

3 Ricorrendo a qualche forzatura, si potrebbe aggirare l'ostacolo, ma si tratta di artifici che dati i nostri obbiettivi non è il caso di approfondire. Si veda il prossimo paragrafo.

```
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
    If Delta < 0 Then
                Cells(1, 1).Value = "Due soluzioni immaginarie"
            Else
                Cells(1, 1).Value = "Due soluzioni reali(distinte
                    /non)"
            End If
```

La soluzione, dal punto di vista della leggibilità, è senz'altro migliore rispetto al caso precedente ma la capacità del programma, in relazione al problema da risolvere, rimane ancora scarsa.

4.4 IF NIDIFICATI

Nulla vieta che all'interno dei blocchi di istruzioni visti nel caso precedente, si trovino ulteriori istruzioni condizionali. Si parla in questo caso di If "nidificati". Questa soluzione permette di aumentare notevolmente le nostre capacità di calcolo ma la leggibilità risulta abbastanza penalizzata nonostante la differente tabulazione. Il codice che mostriamo si riferisce alla procedura `IfThenElse3`.

```
Sub IfThenElse3()
       . . .
    Delta = (B ^ 2) - 4 * A * C
    If Delta < 0 Then
                Cells(1, 1).Value = "Due soluzioni immaginarie"
            Else
                If Delta = 0 Then
                        X1 = -B / (2 * A)
                        Cells(1, 1).Value = "X1=X2= " & Str(X1)
                    Else
                        X1 = (-B - Sqr(Delta)) / (2 * A)
                        X2 = (-B + Sqr(Delta)) / (2 * A)
                        Cells(1, 1).Value = "X1= " & Str(X1)
                        Cells(2, 1).Value = "X2= " & Str(X2)
                    End If
            End If
End Sub
```

4.5 SELECT CASE

Questa soluzione permette di evitare i problemi accennati alla fine del paragrafo precedente. Essa è consigliabile quando le alternative su cui scegliere sono maggiori di due e i blocchi sono costituiti da svariate istruzioni. La sua sintassi è costituita da:

```
Select Case  |*espressione*|
     Case  |*elenco condizioni1*|
```

```
                    . . .
                    . . .
        Case  |*elenco condizioni2*|
                    . . .
                    . . .
        [Case Else
                    . . .
                    . . .]
        End Select
```

in cui:

- *espressione* è una qualunque espressione numerica o stringa;
- *elenco condizioni1, elenco condizioni2*, possono essere espressioni oppure espressioni condizionali;
- *istruzioni* sono una o più istruzioni.

Il comando calcola il risultato di espressione e lo confronta con gli elenchi di condizioni di ciascun `Case`. Quando una di queste condizioni è soddisfatta, vengono eseguite le istruzioni corrispondenti al `Case`. Il controllo del programma, fatto questo, passa alla prima istruzione seguente `End Select`. Viceversa, se nessuna delle condizioni dei vari elenchi di condizioni viene soddisfatta ed è presente `Case Else`, vengono eseguite le istruzioni che si trovano al suo interno[4].

Allo scopo di dimostrare il funzionamento del comando `Case` si consideri la procedura `IfThenElse4` ottenuto dalla sostituzione in `IfThenElse3` di `If` con `Case` (lasciamo al lettore un giudizio sulla leggibilità tra i due).

```
Sub IfThenElse4()
    Dim A As Single, B As Single, C As Single, Delta As Single, X
        1 As Single, X2 As Single
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
    Select Case Delta
        Case Is < 0          ' Delta < 0
            Cells(1, 1).Value = "Due soluzioni immaginarie"
        Case Is = 0          ' Delta = 0
            X1 = -B / (2 * A)
            Cells(1, 1).Value = "X1=X2= " & Str(X1)
        Case Is > 0          ' si puo' scrivere anche Case Else
            X1 = (-B - Sqr(Delta)) / (2 * A)
            X2 = (-B + Sqr(Delta)) / (2 * A)
            Cells(1, 1).Value = "X1= " & Str(X1)
            Cells(2, 1).Value = "X2= " & Str(X2)
        End Select
End Sub
```

4 Le parentesi quadre evidenziano che si tratta di una parte che non è obbligatoria.

Qualora *elenco condizioni1, elenco condizioni2, etc.*, contengano più di una condizione, bisogna utilizzare la virgola come separatore. Inoltre se si devono imporre condizioni di maggiore, minore, uguale, si deve usare la parola chiave `Is` seguita dall'operatore di condizione e poi da un'altra espressione. Il codice seguente, adattato al nostro problema, ne mostra un esempio[5].

```
Sub case_regredito()
    Dim A As Single, B As Single, C As Single, Delta As Single
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
' qui scrivere espressione numerica o stringa
    Select Case Delta
' specificare Is nel caso si faccia uso di operatori di confronto
     (<, =, >, etc.)
        Case Is > 0, Is = 0
             Cells(1, 1).Value = "Due soluzioni reali(distinte/
                 non)"
        Case Else
             Cells(1, 1).Value = "Due soluzioni immaginarie"
        End Select
End Sub
```

La condizione, qualora debba riferirsi ad intervalli numerici, si può esprimere nella forma *espressione1 To espressione2*. Nel codice seguente ne vediamo un esempio assieme a forme condizionali viste in precedenza.

```
Sub case_senza_is()
        Dim I As Integer
        I = 8
        Select Case I
' nel caso si tratti di un intervallo finito
            Case 1 To 3
                MsgBox "I compreso tra 1 e 3"
' nel caso si tratti di una lista di variabili/espressioni
            Case 3, 5, 7, 9
                MsgBox "I uguale a 3/5/7/9"
            Case 4, 6, 8
                MsgBox "I uguale a 4/6/8"
            Case Is < 1, Is > 9
                MsgBox "I > 9 oppure I < 1"
            End Select
End Sub
```

ESERCIZI

1. In relazione al codice 5 sostituire la scrittura del risultato nella cella A1 con un opportuno messaggio.

5 Il nome attribuito a questa procedura non è casuale.

2. La cella A1 contiene un codice fiscale (CF). Riportare in A2 il giorno di nascita relatvo a quel CF sotto forma di numero.

   Si ricorda che il giorno di nascita è scritto nei caratteri 10-11 di CF. Inoltre in CF, la codifica del giorno di nascita è maggiore di 31 quando si tratta di una femmina. In tal caso il giorno di nascita effettivo si ottiene sottraendo 40 al numero corrispondente in CF.

   Se per esempio, la cella A1 contenesse `GRNNTN53P26H501S`, dopo aver eseguito il programma, la cella A2 dovrà contenere il numero 26 (il codice è di un maschio). Se A1 contenesse `GRNNTN53P46H501S`, dopo aver eseguito il programma, A2 dovrà contenere comunque il dato 26. In questo caso il sesso del soggetto intestatario del CF è femminile.

3. Riscrivere il codice di 5 tenendo presente che i tre coefficienti sono scritti nelle celle A1, B1, C1.

4. Se nel codice 5 modifichiamo la condizione `Delta < 0` in `Delta >= 0` come deve essere modificato il resto del codice per avere una risposta corretta?

5. Se nel codice discusso nel paragrafo 4.4 modifichiamo la condizione `Delta < 0` in `Delta > 0` come deve essere riscritto il codice per ottenere una risposta corretta?.

5 I CICLI

Il ciclo o loop costituisce uno dei costrutti fondamentali nella programmazione in quanto permette di gestire con facilità la ripetizione di una o più istruzioni. Gli strumenti disponibili alla gestione della ripetizione sono molteplici e dipendono dalla condizione che ne presiede il funzionamento. Nei paragrafi che seguono passeremo in rassegna i diversi modi di eseguire un loop.

5.1 IL CICLO FOR NEXT

La forma più semplice di ciclo VBA ha la seguente sintassi:

```
For |*nome_di_variabile*| = |*espressione1*| To |*espressione2*|
        . . .
       istruzioni
        . . .
       Next |*nome_di_variabile*|
```

in cui *nome_di_variabile* è il nome di una variabile numerica qualsiasi chiamata variabile di controllo del loop. In genere è una variabile intera (ma questo non è obbligatorio), *espressione1*, *espressione2* sono espressioni numeriche qualsiasi come visto nel paragrafo 2.4.

Quando viene eseguita per la prima volta l'istruzione `For`, vengono svolti i seguenti automatismi:

1. il programma assegna alla variabile di controllo il valore di *espressione1* e lo confronta con quello di *espressione2*. Se il valore assegnato alla variabile di controllo è minore o uguale a quello di *espressione2* vengono eseguite le istruzioni comprese tra `For` e `Next`, altrimenti si passa ad eseguire la prima istruzione successiva a `Next`;

2. il valore della variabile di controllo viene incrementato di uno[1] e si procede a quanto descritto nel punto 1.

Nella cartella di lavoro `cicli.xls` si trova il codice VBA che discutiamo di seguito. Sotto il commento intitolato `CICLO1` è riportato il primo e più semplice esempio di loop. Questo, a parte le istruzioni `For/Next`, si compone di una sola istruzione costituita dalla visualizzazione di un messaggio a video. La variabile di controllo è `M`. I valori di inizio e fine loop sono rispettivamente 1 e 3. Come si dimostra con la prima istruzione successiva al ciclo, terminata la ripetizione, la variabile di controllo vale 4.

1 Vedremo in seguito una deroga a questo comportamento.

Un semplice loop

```
. . .
For M = 1 To 3
        MsgBox "Il mattino ha l'oro in bocca", , "Titolo
            della Finestra"
        Next M
MsgBox "La variabile di controllo contiene" & Str(M)
. . .
```

Nell'esempio mostrato, eseguita per la prima volta l'istruzione `For`, la variabile di controllo `M` ha valore 1; si confronta questa con 3. Essendo 1<3, vengono eseguite le istruzioni interne al ciclo.

Fatto questo si incrementa di uno il valore della variabile di controllo - pari a 2 - quindi si confronta nuovamente con quello di fine ciclo uguale a 3. Essendo il primo ancora inferiore al secondo, si eseguono le istruzioni... e così via per tre volte fino a quando la variabile di controllo non diventa pari a 4.

La variabile di controllo a questo punto, contiene un valore superiore a quello stabilito, il ciclo termina ed il programma passa ad eseguire la prima istruzione successiva.

5.1.1 *La sommatoria*

Sotto il commento `CICLO2a` è riportato un esempio in cui i valori assunti dalla variabile di controllo nel corso del loop (1, 2, 3, 4, ..., 10) vengono utilizzati all'interno delle istruzioni da ripetere per eseguire la somma dei primi 10 numeri[2].

Listing 6: La sommatoria

```
. . .
Totale = 0
For M = 1 To 10
        Totale = Totale + M
        Next M
Cells(1, 3).Value = "La somma dei primi 10 numeri e' " &
    Str(Totale)
. . .
```

Si nota, trattandosi della sommatoria di un insieme numeri, l'istruzione di azzeramento ad inizio loop `Totale = 0` e, all'interno del ciclo, l'istruzione che calcola la sommatoria `Totale = Totale + M`. Si ricordi quanto discusso nel paragrafo 2.4.

2 ATTENZIONE: all'interno di un ciclo è vietato assegnare un valore alla variabile di controllo (p.e. scrivere un'istruzione come M = 15). Questo perchè la gestione della variabile del ciclo è appannaggio del ciclo e la sua modifica dall'esterno può causare malfunzionamenti o errori nel programma.

Sotto i commenti `CICLO2b` e `CICLO2c` sono riportati cicli equivalenti all'esempio precedente[3].

Cicli equivalenti

```
        Totale = 0
        For M = 11 To 2 Step -1
3               Totale = Totale + M - 1
                Next M
        Cells(1, 3).Value = "La somma dei primi 10 numeri e' " &
            Str(Totale)

    '============================================ CICLO2c
8
        Totale = 0
        For M = 2 To 20 Step 2
                Totale = Totale + M \ 2
                Next M
13      Cells(1, 3).Value = "La somma dei primi 10 numeri e' " &
            Str(Totale)
```

Si nota l'uso della parola `Step` necessaria a definire incrementi della variabile di controllo diversi da 1. Oltre questo si possono notare le modifiche necessarie alle istruzioni interne al ciclo per correggere l'effetto dovuto ai valori assunti da M.

Nel primo ciclo i valori assunti da M nel corso del ciclo sono 11, 10, ..., 3, 2. Nel secondo 2, 4, ..., 18, 20.

Le istruzioni sotto `CICLO3` si caratterizzano per l'uso della variabile di controllo del ciclo in veste di indirizzo di colonna della cella in cui scrivere. A questo proposito si rammenti quanto detto nel capitolo sulle funzioni riguardo la sostituibilità degli argomenti.

```
        . . .
2       For M = 1 To 10
                Cells(30, M).Value = M
                Next M
        . . .
```

5.2 IL CICLO DO WHILE

Questo ciclo si usa quando la ripetizione deve essere eseguita in base a condizioni non numeriche.

3 Può capitare di voler "simulare" il comportamento di un programma. Una tecnica che consigliamo di seguire è quella di seguire passo passo le istruzioni che vogliamo controllare. Dotati di matita e gomma, disegnamo le variabili con un quadratino ed scriviamo vicono a questo il nome della variabile corrispondente. Poi incominciamo a leggere le istruzioni annotando nei quadratini i valori che il programma genera all'esecuzione delle istruzioni eventualmente cancellando o sovrascrivendo i contenuti precedenti. Questa tecnica risulta particolarmente utile per capire bene il funzionamento dei cicli, magari utilizzando un minor numero di ripetizioni.

Figura 8: Il calcolo delle celle piene

Consideriamo il seguente problema: abbiamo una colonna di celle piene consecutive. La sequenza si interrompe con una cella vuota. Ipotizzando di conoscere l'inizio della sequenza, vogliamo sapere quante sono le celle piene. La soluzione è legata alla possibilità di sapere quando una cella è piena o vuota. In questo caso è sufficiente spostarsi sulla sequenza di celle fintantoché non ci troviamo in presenza di una cella vuota. A questo punto in ciclo si interrompe[4].

Vedremo più avanti come risolvere questo problema. Per adesso consideriamo la forma generale di un ciclo costituita da:

```
Do While | Until |*espressione_condizionale*|
. . .
Loop
```

L'impiego di `Do While` oppure di `Do Until` (il carattere "`|`" indica che si tratta di due parole mutuamente esclusive), è basato sul risultato dell'espressione condizionale. Se il ciclo deve essere ripetuto finché l'espressione condizionale è vera si usa `Do While`. Se il ciclo va ripetuto finquando la condizione è falsa, si utilizza `Do Until`. Nel codice 7 che mostriamo di seguito sono riportati due esempi che ottengono lo stesso risultato con le due diverse modalità espressive di ripetizione. Il codice fa riferimento al foglio della figura 8.

4 Nel capitolo 7 vedremo come questo problema sia facilmente risolvibile più semplicemente ed in modo completamente diverso. Per ora ipotizzeremo di non avere nessun'altra risorsa che questa.

Listing 7: Do While/Until

```
        . . .
' CICLO4
    M = 1
' premesso che Cells(riga, colonna).Value = "" significa cella
    vuota
' possiamo dire che
' il ciclo viene eseguito fintantoche' la condizione e' Vera
    Do While Cells(M, 1).Value <> ""
        M = M + 1
        Loop
    Cells(2, 3).Value = "Le celle occupate nella colonna A sono "
         & CStr(M - 1)
'
    M = 1
' il ciclo viene eseguito fintantoche' la condizione e' Falsa
    Do Until Cells(M, 1).Value = ""
        M = M + 1
        Cells(1, 8).Value = M
        Loop
    Cells(3, 3).Value = "Le celle occupate nella colonna A sono "
         & CStr(M - 1)
        . . .
```

5.3 IL CICLO DO . . LOOP WHILE/UNTIL

Nei precedenti paragrafi abbiamo visto come la condizione che determina l'esecuzione del ciclo viene testata a monte delle istruzioni da ripetere. Può capitare invece che le istruzioni interne al ciclo debbano essere eseguite almeno una volta. In questo caso la condizione che ne presiede il funzionamento deve trovarsi alla fine e non all'inizio del ciclo e si usa `Do .. Loop While`.

Si vuole contare in modo ordinale la posizione di una lettera dell'alfabeto (la lettera "g" nel nostro esempio). Per risolvere il problema mettiamo in due colonne del foglio in corrispondenza biunivoca ciascun carattere dell'alfabeto con il corrispondente valore ordinale. Se

Figura 9: La ricerca di un dato

eseguiamo un confronto tra la lettera che cerchiamo e tutti i caratteri dell'alfabeto, potremo sfruttare il contenuto di una variabile (M nel nostro esempio) che tiene conto dei confronti effettuati. Quando si verifica l'uguaglianza tra la lettera che cerchiamo ed uno dei caratteri, terminiamo il ciclo. A questo punto M, numero dei confronti effettuati fino a quel momento, sarà il dato che ci permette di indirizzare correttamente la cella con l'ordinale corrispondente.

```
Sub cicli4()
   Dim Carattere As String
   Dim M As Byte
   Carattere = "g"
   M = 0
' in questo esempio la condizione di terminazione del ciclo viene
     controllata alla fine
   Do
        M = M + 1
        Loop While Carattere <> Cells(M, 10).Value
   Cells(7, 3).Value = "La lettera " & Carattere & " e' " & _
               Cells(M, 11).Value & " carattere dell'alfabeto"
End Sub
```

5.4 COME INTERROMPERE UN CICLO

L'interruzione di un ciclo può essere intenzionale o dovuta ad errori di logica. Negli esempi precedenti le condizioni di terminazione del ciclo erano stabilite direttamente nei comandi che lo eseguivano sia nel caso del `For` che nelle svariate forme `Do`.

Alcuni problemi richiedono però una soluzione diversa. Si consideri, a titolo d'esempio, di voler cercare una parola nella sequenza di celle di figura 8. Se, per semplicità, ipotizziamo di conoscere il numero di celle piene della colonna, sarà conveniente ricorrere ad un ciclo `For`. All'interno del ciclo inseriamo una istruzione che, qualora la ricerca abbia esito positivo, provveda all'uscita. Il codice corrispondente sarà:

```
       Dim M As Byte, parolaDaCercare As String
' si provi con una parola presente oppure mancante
       parolaDaCercare = "ciccio"
       'parolaDaCercare = "pirata"
       For M = 1 To 15
              If parolaDaCercare =  Cells(M, 1).Value Then Exit
                   For
              Next M
       If M>15 Then
                     MsgBox "Eureka"
              Else
                     MsgBox "Buuu"
              End If
End Sub
```

Si nota il comando `Exit For` che provoca l'uscita dal ciclo quando la stringa da ricercare viene trovata. L'istruzione condizionale dopo il ciclo permette di sapere se la ricerca ha avuto esito positivo o negativo a causa della terminazione anticipata.

Diverso è il caso di interruzione forzata dovuta ad un errore di logica non prevsto nel programma. Il seguente codice "simula" una situazione di questo tipo nella quale il ciclo si ripete fino ad esaurire

le righe della colonna. In queste condizioni, non avendo di meglio, si ricorre alla combinazione di tasti Ctrl+Interr.

```
Sub cicloInfinito()
    Dim I As Long
    I = 0
    Do
        I = I + 1
        MsgBox "Sono a riga " & Str(I)
        Loop While Cells(I, 2) = ""
End Sub
```

5.5 I CICLI NIDIFICATI

Così come visto nel caso dell'istruzione `If` al 4.4, anche le istruzioni di ripetizione possono essere nidificate.

Classico esempio di impiego di cicli nidificati è rappresentato dal calcolo della somma di un insieme di celle del foglio organizzate in righe e colonne.

Figura 10: Totali di riga e colonna

Supponiamo di aver rilevato la suddivisione di studenti in base al paese di provenienza ed al sesso come mostrato, suddivisi in maschi e femmine, come riportato nella figura 10. Vogliamo calcolare il totale di ciascuna riga e di ciascuna colonna nonché il totale generale (quello della cella G11).

La soluzione di un problema di questo tipo è costituita da un doppio ciclo nidificato che letto il contenuto di ciascuna cella della zona D7.E9, lo somma al totale di competenza. Il codice corrispondente è riportato di seguito.

Cicli nidificati

```
Sub ciclonidificato()
    Dim I As Byte, J A Byte
    For I = 1 To 3
        For J = 1 To 2
                        'totale di riga
            Cells(6+I, 7).Value = Cells(6+I, 7).Value + _
                                  Cells(6+I, 3+J).Value
                        'totale di colonna
            Cells(11, 3+J).Value = Cells(11, 3+J).Value + _
                                   Cells(6+I, 3+J).Value
                        'totale generale
            Cells(11, 7).Value = Cells(11, 7).Value + _
                                 Cells(6+I, 3+J).Value
            Next J
        Next I
End Sub
```

Nel codice il riferimento al contenuto da sommare nei totali è individuato con Cells(6+I, 3+J).Value. Se a tale riferimento applichiamo la sequenza di valori generata dai cicli I=1 To 3 e J=1 To 2, otteniamo per l'appunto tutti i valori da sommare. D'altro canto, in riferimento al totale di riga, Cells(6+I, 7).Value individua, per I=1 To 3, il totale di ciascuna riga ed allo stesso modo, cambiando quello che c'è da cambiare, avviene per il totale di colonna. Nella figura 10 abbiamo riportato delle linee che visualizzano l'effetto delle tre istruzioni interne al doppio ciclo per quanto riguarda la totalizzazione della frequenza relativa al numero di soggetti "Femmine" provenienti da paesi "Extracomunitari".

Il totale generale, individuato dal riferimento fisso Cells(11, 7).Value, si riferisce ovviamente al totale generale.

ESERCIZI

1. La colonna di celle da A1 ad A12, contiene 12 numeri qualsiasi. Calcolare il massimo dei valori della zona senza adoperare alcuna funzione. Per la soluzione tradurre nelle corrispondenti istruzioni VBA le seguenti istruzioni scritte in linguaggio naturale:
   a) massimo=Cella(1,1)
   b) ciclo in I da 2 a 12

      se massimo < Cella(I,1) massimo = Cella(I,1)

      fine ciclo
   c) Visualizza massimo

2. Nella zona di celle A1.A13, sono presenti un insieme di 13 numeri uno per ogni cella. Scrivere il programma che calcola la somma e la media di questi numeri. Scrivere poi questi due risultati in altrettante celle senza ricorrere ad alcuna funzione Excel/VBA.

3. Riempire alcune celle del foglio con stringhe di caratteri. Calcolare, riportando il risultato in una MsgBox, la lunghezza media $\overline{x}$ delle stringhe che riempiono le celle in base alla relazione:

$$\overline{x} = \frac{totale\_dei\_caratteri}{numero\_di\_celle\_piene}$$

4. Premesso che la funzione VBA `IsEmpty` restituisce "True/Vero" se l'espressione al suo interno è vuota, scrivere i programmi equivalenti a quelli del paragrafo 5.2.

5. Risolvere il problema del paragrafo 5.2 ipotizzando che la riga da cui incomincia la sequenza di celle piene possa essere una qualunque della colonna A.

3

6 LE FUNZIONI DEFINITE DALL'UTENTE

Nel capitolo 3 avevamo illustrato, in maniera dettagliata, le funzioni del programma VBA. Qui parleremo di funzioni definite dall'utente. L'esempio cui ci riferiremo in questo capitolo è costituito dalla richiesta di inserimento di un dato all'utente e dall'eventuale controllo di quanto immesso[1].

6.1 LA FUNZIONE INPUTBOX

Nella cartella di lavoro del paragrafo 3.2 abbiamo elencato, senza discuterne, la funzione `InputBox`. Essa visualizza una finestra che richiede l'inserimento di un dato in un apposito spazio chamato *campo*. Ne vediamo un esempio nella figura 11.

La funzione ha cinque argomenti costituiti da: la richiesta relativa al dato da inserire, il titolo della finestra, la risposta predefinita che viene visualizzata nel campo di input, le coordinate x,y, espresse in twips[2] dell'angolo alto sinistro della finestra.

Le istruzioni che producono la finestra della figura sono contenute nella procedura che fa parte della cartella di lavoro inpututente.xls.

Listing 8: La funzione InputBox

```
Sub interazione_con_utente()
    Dim inputUtente As String
    Dim InputUtente1 As Integer
    inputUtente = InputBox("Io voto per ", "SCHEDA ELETTORALE", _
                           , 0, 0)
    MsgBox inputUtente
End Sub
```

Si nota la dichiarazione della variabile `inputUtente` che, coerentemente con il risultato calcolato dalla funzione, è di tipo String.

Una prima utile applicazione di questa funzione è contenuta nella procedura `InserisceInputQualsiasi`, parzialmente visibile nel codice 9. Essa trasferisce il dato immesso nel campo di InputBox nelle celle della colonna A, partendo dalla prima riga. L'azione è controllata da

1 Se la richiesta è relativa ad un dato numerico sarà necessario accertarsi che non siano presenti caratteri estranei.

2 1 pixel = 15 twips

Figura 11: La funzione InputBox

un ciclo che si ripete fintantoché si inserisce nel campo almeno un carattere. Diversamente la ripetizione viene conclusa.

6.2 LA FUNZIONE ISNUMERIC

Quando si inserisce un dato può essere utile verificare se esso è coerente con quanto richiesto. La funzione `IsNumeric` si rivela di grande utilità a questo fine perché controlla che nel suo argomento siano presenti soltanto caratteri numerici. Il risultato di questa funzione è una variabile booleana (ne avevamo parlato al termine del paragrafo 2.4) e sarà Vero qualora il test sulla presenza di sole cifre sia stato favorevole.

Listing 9: Ciclo di inserimento con InputBox

```
  . . .
Do              ' ciclo per la richiesta di un dato
3          inputUtente = InputBox("Inserisci un dato", _
                                  "FINESTRA DI RICHIESTA DATO",
                                     _
                                  "", 500, 800)
           If inputUtente = "" Then Exit Do
           Cells(I, 1).Value = inputUtente
8          I = I + 1
           Loop Until inputUtente = ""
  . . .
```

La procedura `prova_isnumeric` parzialmente visibile nel codice 10, calcola il risultato della funzione in relazione a diversi argomenti. Nei primi due esempi, in cui l'argomento è una stringa numerica o un numero, la funzione restituisce Vero[3].

Listing 10: Risultato di IsNumeric

```
' risultato Vero
MsgBox "Il risultato di IsNumeric(123) e' " & IsNumeric(123)
' risultato Vero
MsgBox "Il risultato di IsNumeric(""123"") e' " & IsNumeric
    ("123")
5 ' risultato Falso
MsgBox "Il risultato di IsNumeric("""") e' " & IsNumeric("")
' risultato Vero
MsgBox "Il risultato di IsNumeric su una cella vuota e' " &
    IsNumeric(Cells(1, 1).Value)
a = Cells(1, 1).Value
10 MsgBox "Il valore di a dopo l'assegnazione di una cella vuota e'
    " _
                & Val(a)   ' questo spiega IsNumeric(cellavuota)=
                    True
```

3 In questo esempio e nel successivo si noti come, per utilizzare il riferimento ad una stringa all'interno di una costante stringa, debba essere scritta una sequenza di doppi apici.

Nel terzo esempio la funzione `IsNumeric`, calcolata su una stringa vuota, ha come risultato Falso.

La sequenza di istruzioni dalla quarta riga in poi, vuole dimostrare cosa accade quando applichiamo la funzione ad una cella vuota. In questo caso la funzione restituisce True, risultato che si spiega con il fatto che per Excel una cella vuota contiene un valore pari a zero. A dimostrazione di questo, si considerino le righe successive in cui il contenuto della cella viene assegnato alla variabile `a`.

Sulla base di queste premesse, abbiamo realizzato la procedura `InserisceInputNumerico` che si occupa di controllare che il dato inserito sia numerico. In caso negativo si visualizza un messaggio di errore altrimenti si inserisce il dato in una cella a partire da ... Con l'inserimento di un dato "vuoto" si intende terminata l'acquisizione.

Listing 11: Acquisizione e controllo dell'input

```
Sub InserisceInputNumerico()            ' inserisce una sequenza
    di dati numerici
    Dim inputUtente As String           ' a partire da A1
    Dim ok As Boolean
    Dim I As Integer
    I = 1
    Do
        inputUtente = InputBox("Inserisci un dato numerico." &
            Chr(13) & _
                                    "N.B. per terminare non
                                        inserire niente e
                                        premere OK", _
                                           "ACQUISISCE INPUT
                                               NUMERICO DA
                                               TASTIERA", _
                                           "", 500, 800)
        If IsNumeric(inputUtente) = False Then
' se accedo qui e' x 2 motivi:
'               a. dato non numerico
'               b. dato vuoto (lunghezza = 0) (questo caso non lo
     considero ma terminera' il ciclo)
                If Len(inputUtente) > 0 Then MsgBox _
                                "Il dato inserito " & inputUtente
                                     & " non e' numerico"
                    Else
                       Cells(I, 1).Value = CStr(inputUtente)
                           ' input pieno, inserisce il dato
                       I = I + 1
                       End If
          Loop While inputUtente <> ""
End Sub
```

La procedura `InserisceInputNumerico_varianteSelect` che si trova di seguito a quella appena vista, è la variante con l'uso del comando Select.

6.3 LA FUNZIONE AD UN RISULTATO

Nel capitolo dedicato alle funzioni abbiamo discusso delle modalità di richiamo delle funzioni sia di Excel che di VBA. In questo paragrafo e nei successivi parleremo di funzioni definite dall'utente. Consideriamo il seguente codice:

Una funzione definita dall'utente e ..

```
Function ciccioriccio() As String
    ciccioriccio = "123sftgrtg"
End Function
```

Nell'esempio `ciccioriccio`, è il nome di una funzione di tipo stringa. Nella seconda riga abbiamo un'istruzione di assegnazione di una costante stringa ("123sftgrtg") ad una variabile. Si nota, fatto non casuale, che questa variabile ha un nome identico a quello della funzione. Dopo aver scritto il codice che richiama questa funzione e proceduto alla sua esecuzione non dovrebbe essere difficile comprenderne il funzionamento.

.. la chiamata alla funzione.

```
Sub prova_function()
    MsgBox ciccioriccio
End Sub
```

Alla luce di quanto abbiamo appena visto è possibile dire che:

- l'utente può definire funzioni personalizzate con o senza argomenti (nell'esempio la funzione era senza argomenti);
- all'interno della funzione `Function` deve trovarsi un'assegnazione ad una variabile che ha lo stesso nome e tipo della funzione;
- la funzione definita dall'utente si comporta come una qualsiasi funzione VBA;
- la funzione restituisce un solo risultato.

Nel caso di una `Function` con argomenti la variante consiste nella dichiarazione di questi tra le parentesi. Si consideri il seguente codice:

Una funzione ad un solo risultato e la sua chiamata

```
Function richiesta_dato_numerico_Cargomento(domanda As String) As
     String
    Dim inputUtente As String
    Do
        inputUtente = InputBox(domanda, "ACQUISISCE DATO NUMERICO
            ", _
                        "", 500, 800)
```

```
        If IsNumeric(inputUtente) = False Then
7                   MsgBox "Il dato richiesto e' mancante o
                        sbagliato", vbCritical
        ' potrebbe trattarsi anche di un input
        ' che contiene caratteri non numerici
        ' per garantire la prosecuzione del ciclo
        ' devo impostare a "vuoto" il contenuto della variabile
            inputUtente
12                      inputUtente = ""
                        End If
        Loop While inputUtente = ""
    richiesta_dato_numerico_Cargomento = inputUtente
End Function
17 ' ********************** la chiamata alla funzione
Sub prova_function_rdm1()
    Dim domanda As String
    domanda = "Quanti anni hai?"
    MsgBox richiesta_dato_numerico_Cargomento(domanda)
22 End Sub
```

Nell'esempio la funzione consta di un solo argomento. Al suo interno è definito un ciclo che si ripete fino a quando non è stato inserito un valore numerico[4].

6.4 LA FUNZIONE A PIÚ RISULTATI

Quando una funzione deve calcolare più di un risultato si usa il comando Sub al posto di Function.

Nell'esempio seguente risolveremo il problema del calcolo delle radici di una equazione di secondo grado. Per semplificare abbiamo scritto i tre coefficienti a, b, c, tra gli argomenti del programma chiamante e li abbiamo definiti come numeri interi.

La chiamata e . .

```
Sub chiama_equazione_IIgrado()
    Dim a As Integer, b As Integer, c As Integer
3   Dim x1 As Single, x2 As Single
    Dim esito As Boolean
        equazione_2grado_sub x1, x2, 2, 4, 1, esito
        If esito = False Then
            MsgBox "Due soluzioni immaginarie"
8           Exit Sub
            End If
        MsgBox "X1= " & CStr(x1) & Chr\$(13) & "X2= " & CStr(x2)
End Sub
```

Nel programma chiamante si nota, a differenza di quanto avviene con la Function, la mancanza di parentesi. Si noti anche la presenza,

4 Al momento, per semplicità, escluderemo che l'utente inserisca il separatore dei decimali. Il problema verrà affontato in uno dei prossimi capitoli.

come ultimo argomento di una variabile booleana utile, dopo la chiamata del sottoprogramma, a conoscere nel programma chiamante l'esito dei calcoli: due soluzioni immaginarie/due soluzioni (distinte o coincidenti). In questo modo sapremo se procedere o meno al calcolo delle due radici.

Il sottoprogramma è abbastanza simile alla funzione. Per ovvi motivi il sottoprogramma non dichiara alcun tipo di dato. Si noti la conversione del tipo della variabile `delta` in quanto la funzione Sqr richiede un argomento Double[5].

.. la funzione a diversi risultati

```
Sub equazione_2grado_sub(x1 As Single, x2 As Single, a As Integer
    , _
        b As Integer, c As Integer, sw_ok As Boolean)
' eventuale dichiarazione di variabili non incluse tra gli
    argomenti
' decido di non trasferire al chiamante delta dunque qui scrivero
    '
    Dim delta As Integer
' calcolo delta
    delta = b ^ 2 - 4 * a * c
    sw_ok = True
    If delta < 0 Then
                sw_ok = False
        Else
' calcolo di x1 e x2 ATTENZIONE alle conversioni
                x1 = (-CSng(b) - CSng(Sqr(CDbl(delta)))) / CSng(2
                     * a)
                x2 = (-CSng(b) + CSng(Sqr(CDbl(delta)))) / CSng(2
                     * a)
        End If
End Sub
```

ESERCIZI

1. Un programma deve chiedere l'inserimento di tre dati numerici in tre celle del foglio scelte a piacere. Le istruzioni dovranno prevedere un ciclo che si ripete fintantoché tutti e tre i numeri non sono stati acquisiti.

2. Scrivere la Function che calcola le soluzioni di una equazione di II^ grado. Si tenga presente che i tre coefficienti a, b, c, si trovano in altrettante celle del foglio scelte a piacere. Oviamente, considerato di non poter calcolare i valori x1 e x2 come visto in precedenza, la funzione restituirà il risultato sotto forma di messaggio come "due soluzioni reali e distinte", oppure "due soluzioni immaginarie" etc.

5 Sarebbe stato più semplice definire a, b, c come variabili di tipo `Single`. La scelta è dovuta a motivi di coerenza con quanto fatto nel paragrafo 4.1.

3. Come nel problema n.2 tenendo presente che i tre coefficienti a, b, c, inseriti nelle celle sono dichiarate Single.

7 LA PROGRAMMAZIONE AD OGGETTI IN VBA

Le prime idee relative al concetto di "classi di oggetti" risalgono ad Aristotele (300 a.C) quando il filosofo parlava di "classe dei pesci e degli uccelli". Il fatto che degli oggetti siano unici ed allo stesso tempo parte di un insieme di elementi aventi caratteristiche e comportamenti comuni, è il concetto su cui si basano i linguaggi di programmazione ad oggetti.

L'idea di linguaggi di programmazione ad oggetti (Object Oriented Programming ovvero OOP) nasce intorno agli anni '80 e si basa sul fatto che un programma, così come il mondo circostante, si compone di oggetti che hanno proprietà e qualità particolari. In questo modo, i dati di un programma (anch'essi oggetti), hanno le loro proprietà e qualità che, per effetto della loro trasmissibilità, chiamata "ereditarietà" , facilitano notevolmente la programmazione[1].

7.1 PROPRIETÀ E METODI

Nella terminologia OOP, con il termine "proprietà", intendiamo riferirci alle caratteristiche di un oggetto equiparabili, nel linguaggio naturale, agli aggettivi ovvero alle qualità.

Tutti noi conosciamo l'oggetto *Radio*. Le sue qualità sono costituite dal modo di funzionare (analogica o digitale), oppure dal suo colore (bianco o verde), oppure dal suo peso. Proseguendo nell'analogia possiamo anche capire come alcune proprietà siano modificabili mentre altre non lo sono: posso cambiare il colore della radio, ma non posso modificare l'insieme delle frequenze che riceve.

Sugli oggetti possiamo poi eseguire determinate azioni che, nella terminologia OOP, si chiamano "metodi". Sempre in riferimento all'oggetto *Radio* possiamo eseguire un'azione costituita dall'accendersi, lo spegnersi, il sintonizzarsi, l'aumento o la diminuzione del volume.

Il vantaggio della programmazione ad oggetti è che il programma non si preoccupa di come fare ad eseguire l'azione, esso deve semplicemente chiedere di eseguirla. Dunque, se voglio accendere/spegnere la radio, eseguirò il metodo "spegni" oppure "accendi" senza preoccuparmi di come questo sia realizzato fisicamente nell'oggetto.

1 Se nel nostro programma creiamo un oggetto, copiandolo da uno preesistente, esso erediterà tutte le qualita/proprietà da quello originale senza bisogno di altro.

7.2 COME SI PROGRAMMA UN OGGETTO

Nell'OOP, quindi anche in VBA, la programmazione di un oggetto si espleta attraverso una delle seguenti attività:

- leggere/scrivere una proprietà;
- eseguire un metodo.

La sintassi per modificare la proprietà o applicare un metodo ad un oggetto è costituita da:

nome_dell'oggetto.nome_di_proprietà | *nome_di_metodo*

in cui dopo il punto, necessario a distinguere il nome dell'oggetto dal resto, bisogna specificare un nome valido di proprietà oppure di metodo.

7.3 LE PROPRIETÀ IN VBA

Le proprietà di un oggetto possono essere lette oppure assegnate. A questo proposito, l'esempio che ormai ci dovrebbe essere familiare, è quello che assegna un valore ad una proprietà come nel caso dell'istruzione:

```
  . . .
Cells(21, 1).Value = Gio
  . . .
```

Nell'esempio `Cells(21,1)` rappresenta l'oggetto; `Value` è la proprietà costituita dal contenuto (il valore); `Gio` è un'espressione nella forma più semplice (il nome di una variabile).

In generale, nel caso di un'assegnazione, la sintassi sarà del tipo:

Oggetto.Proprietà = espressione

mentre per la lettura, il riferimento all'oggetto ed alla proprietà, potrà essere ottenuto riferito in vari modi (si veda il codice poco oltre).

Si tenga presente che *espressione* è una qualsiasi espressione che abbia come risultato un valore compatibile con la proprietà (`Value` nel caso del nostro esempio) riferita all'oggetto `Cells` in base al criterio della compatibilità. Infine `Proprietà` è una qualsiasi proprietà valida per l'oggetto `Cells`.

Nella cartella di lavoro `proprieta_metodi.xls` sono presenti diverse procedure ricche di esempi significativi. I commenti a fianco delle istruzioni dovrebbero bastare a chiarirne il significato.

La prima di queste, denominata `proprieta`, visualizza con una funzione MsgBox, le proprietà di altrettanti oggetti del foglio. In tutti questi esempi il carattere "." separa l'oggetto (a sinistra) dalla proprietà (a destra).

Le proprietà di alcuni oggetti

```
        . . .
2 '                  la proprieta' Name di ActiveSheet
    MsgBox ActiveSheet.Name, , "Il foglio attivo"
  '                  l'oggetto e' la cartella di lavoro
    MsgBox ThisWorkbook.Name, , "La cartella di lavoro"
    MsgBox ThisWorkbook.Path, , "Unita' e sottodir."
7   MsgBox ThisWorkbook.FullName, , "Unita',sottodir., cartella
        di lavoro"
  '                  l'oggetto e' la cella corrente
    MsgBox ActiveCell.Address, , "L'indirizzo della cella
        corrente"
    MsgBox ActiveCell.Row, , "Il numero di riga della cella
        corrente"
    MsgBox ActiveCell.Column, , "Il numero di colonna della cella
         corrente"
12  MsgBox Cells(1, 1).Address, , "L'indirizzo della cella A1"
    MsgBox Range("A3").Column, , "Il numero di colonna di una
        cella"
' esempio di comando di assegnazione di un valore alla proprieta'
    Formula dell'oggetto Cells
' si noti
    Cells(10, 3).Formula = "=B3*2"
17      . . .
```

Solo una notazione sulla scrittura di una formula in una cella. Nell'ultima istruzione dell'esempio si nota che la formula deve essere racchiusa tra doppi apici (come se si trattasse di una stringa).

7.4 I METODI IN VBA

La sintassi per l'esecuzione di un metodo si esprime generalmente nella forma:

Oggetto.Metodo

Nel codice della procedura metodi_selezione_celleZone, abbiamo riportato diversi esempi del metodo di selezione di una zona di celle Tutti i metodi della procedura ipotizzano che sia attivo il foglio di lavoro cui si riferiscono le zone di celle. Diversamente occorrerà specificare esplicitamente quest'ultimo tenendo presente che la gerarchia dell'oggetto `Range` è:

```
Application(..).Workbook(..).Worksheet(..).Range(..)
```

in cui le parentesi si riferiscono agli eventuali argomenti richiesti da ciascun elemento della gerarchia.

La selezione di Celle/Zone

La programmazione degli oggetti definisce un grande numero di metodi per la selezione di celle o zone del foglio[2]. In questo primo esempio vediamo come eseguire l'azione di selezione in relazione all'oggetto zona di celle compreso nell'intervallo C2 D10.

```
Sub metodi_selezione_celleZone()
        . . .
    Range("C2:D10").Select
' se levo il commento al comando successivo,
' notero' la zona evidenziata nel foglio Excel
'    Exit Sub
```

Per capire bene il suo effetto sul foglio Excel è consigliabile eliminare il carattere di commento che abbiamo inserito davanti all'istruzione di uscita dalla procedura per evitare di eseguire le istruzioni successive.

La sintassi definita all'inizio del paragrafo relativa all' esecuzione di un metodo non sempre è così "lineare". Nel prossimo esempio la zona di selezione, costituita da una sola cella, è individuata in base ad uno spostamento rispetto alla cella corrente.

```
        . . .

' si usa il riferimento denominato R1C1
' dunque 5 righe in avanti, 4 colonne indietro
' rispetto alla cella F7
    Range("F7").Offset(5, -4).Select
    Exit Sub
```

Il commento presente davanti al comando rende inutile ogni ulteriore considerazione. Si tenga presente che `Offset(5, 4)` è una proprietà che restituisce un oggetto. La riprova di questo si può avere consultando la finestra che visualizza gli oggetti del foglio con la procedura che descriviamo:

- aprire il Visualizzatore degli oggetti del foglio (tasto F2);
- selezionare Excel nel menu a discesa che si trova in alto a sinistra;
- nel pannello di sinistra, intitolato Classi, selezionare Range;
- selezionare Offset nella parte destra. Visualizzando il pannello inferiore (la finestra deve essere massimizzata) si vedrà che Offset restituisce un oggetto Range.

Il prossimo esempio è particolarmente importante perché seleziona la zona di celle piene contigue a partire da una cella predefinita.

2 Per una trattazione più approfondita, si può consultare il seguente link http://support.microsoft.com/kb/291308/it in cui sono descritti numerosi metodi relativi alla selezione di zone di celle. Qualora il link non funzionasse correttamente, si inserisca il termine "*How to select cells/ranges by using Visual Basic procedures in Excel*" in un motore di ricerca per ottenere lo stesso risultato.

```
        . . .
        ActiveSheet.Range("A1").CurrentRegion.Select
3       . . .
```

Particolarmente importante, per ovvi motivi, è la combinazione di comandi che effettua il Copia/Incolla di una zona di celle. A questo proposito consideriamo il codice 12.

Nella prima riga abbiamo eseguito la selezione di un zona di celle. In questo modo viene generato un oggetto Selection che è quello cui viene applicato il metodo di copia utilizzato nella seconda riga. Il terzo comando incolla a partire dalla cella E5. Il quarto e quinto comando servono rispettivamente ad eliminare la selezione della zona di output[3], ed a svuotare gli Appunti (tasto Esc in Excel).

Listing 12: Copia e incolla

```
          . . .
2     Range("C2:D10").Select
      Selection.Copy
      Range("E5").PasteSpecial
      Range("A6").Select
      Application.CutCopyMode = False
7           . . .
```

Se i comandi necessari, ben 5 istruzioni, sono troppi, si può ricorrere ad una forma molto più sintetica che ottiene lo stesso risultato:

```
      . . .
      Range("C2:D10").Copy Destination:=Range("E5")
3     . . .
```

In questo caso `Destination` serve a definire il valore di un parametro del metodo Copy (si noti l'uso di ":=" , una forma sintattica analizzata nel paragrafo 3.4).

Una corretta valutazione dei due metodi, deve però tenere presente che il secondo di questi non è utilizzabile qualora si debbano effettuare copie multiple della stessa zona.

7.5 LA CANCELLAZIONE DI UNA ZONA

Un altro metodo utile è quello che cancella il contenuto di una zona. Il codice seguente ne dimostra un esempio: Più interessante, rispetto

```
Range("F7:E5").Clear
```

alla generalizzazione del metodo, è il codice della prossima procedura. Se l'argomento di Range è una stringa di caratteri, è possibile

3 La tecnica di selezione di una cella dovrà essere utilizzata in tutti i casi in cui si vuole "deselezionare" una zona come vedremo nel caso di cancellazione del contenuto di celle

esprimere la zona sotto forma di variabile. Di conseguenza sarà possibile il riutilizzo di questa procedura, opportunamente modificata, per essere generalizzata.

La cancellazione di una zona

```
Sub metodo_cancellaZona2()
    Dim numero_di_riga As Byte
    Dim ciccio As String
    numero_di_riga = 7
    ciccio = "E5:F" & CStr(numero_di_riga)
    Range(ciccio).Clear
End Sub
```

7.6 LE COLLEZIONI DI OGGETTI

All'inizio del capitolo abbiamo accennato alle classi di oggetti. Un concetto notevole su cui è necessario soffermarsi riguarda le c.d. "collezioni". Le collezioni sono insiemi di oggetti accomunati dalle stesse caratteristiche (metodi e proprietà). Esse stanno agli oggetti che le compongono come le variabili con indici stanno ai loro elementi. Nel caso del foglio Excel le collezioni predefinite[4] più importanti sono quelle che si riferiscono ad insiemi di:

- righe/colonne chiamati rispettivamente Rows e Columns;
- fogli chiamati Sheets;
- grafici chiamati Charts.

Nel primo esempio viene calcolata la proprietà Count che applicata alle righe di una zona (questa è la collezione) ne conteggia il numero. Si noti come nella stessa riga di codice sia possibile prima riferire l'oggetto zona di celle (a destra dell'istruzione di assegnazione) e contestalmente crearne uno nuovo con il comando Set

```
' questa e' la dichiarazione dell'oggetto
   Dim esempio As Range
' creo l'oggetto esempio e gli assegno un range.
' A questo punto esempio
' e' l'oggetto "zona di celle A1:C4"
   Set esempio = Range("A1:C4")
' per il principio dell'EREDITARIETA',
' all'oggetto "esempio"
' si possono applicare le proprieta' (ed i metodi)
' validi per l'oggetto di cui e' copia
```

4 Come vedremo tra poco è possibile creare oggetti e quindi anche collezioni. Agli oggetti creati dall'utente, a differenza di quelli predefiniti, possono essere assegnate proprietà a piacimento.

```
    MsgBox esempio.Rows.Count              ' ritorna 4
    MsgBox esempio.Columns.Count           ' ritorna 3
```

Le collezioni dispongono di strumenti in grado di gestire gli elementi che le compongono uno per uno. Nel seguente esempio si effettua un ciclo su tutti i fogli della cartella di lavoro corrente; quelli con un prefisso del nome diverso da "Foglio" vengono eliminati.

Nella fattispecie `ActiveWorkbook.Sheets` è un oggetto costituito dalla collezione di tutti i fogli della cartella attiva, `aSheet.Name` si riferisce al nome di ogni foglio (proprietà), `Sheets(aSheet.Name)` è un oggetto della collezione cui viene eventualmente applicato il metodo di cancellazione. Si noti anche la novità costituita dal modo di esprimere un ciclo applicato a tutti gli elementi di una collezione.

Il ciclo su una collezione

```
Dim aSheet As Object
    For Each aSheet In ActiveWorkbook.Sheets
3 '       eliminiamo tutti i worksheet con nome differente da
    Foglio(n)
        If Left(aSheet.Name, 6) <> "Foglio" Then
                aSheet.Delete
            End If
        Next
```

7.7 L'OGGETTO EXCEL

In alcuni casi, nel codice VBA, è necessario riferirsi al programma Excel come, p. e., quando dobbiamo utilizzare una sua funzione che non ha corrispondenti in VBA. Il problema era stato posto, senza che venisse data una spiegazione, nel paragrafo 3.1. Abbiamo ora la possibilità di risolvere il problema con il riferimento all'oggetto `Application` che in VBA è il nome del programma Excel.

Così la somma di una zona di celle sarà calcolata con la funzione Sum che richiede come argomento l'oggetto `Range`. Nella stessa procedura viene impiegata un'altra interessante funzione del foglio che calcola il numero di celle piene di una zona. Si ricorda che il nome della funzione Excel deve essere specificato in inglese.

```
Sub sintassi_funzione_conZona()
    Dim totCelle As Integer, conta_piene As Integer
3 ' Sum e' l'equivalente inglese della funzione SOMMA
    totCelle = Application.Sum(Range("A1:B3"))
    MsgBox "la somma dei numeri in A1:B3 e' " & CStr(totCelle)
' CountA e' l'equivalente inglese della funzione CONTAVALORI
    conta_piene = Application.CountA(Range("A1:C4"))
8   MsgBox "le celle piene in A1:C4 sono " & CStr(conta_piene)
End Sub
```

Il riferimento allo stesso oggetto, il programma Excel, viene utilizzato nell'esempio seguente in cui vediamo i comandi che chiudono una

cartella di lavoro aperta. Si tratta dell'equivalente dei comandi del menu Excel File → Esci.

La prima riga della procedura ha la funzione di evitare la richiesta di salvare il file nel caso in cui fossero intervenute delle modifiche. Per capire meglio si provi a commentare tale riga senza salvare e ad eseguire la macro.

```
    . . .
ActiveWorkbook.Saved = True
Application.Quit
    . . .
```

7.8 LA GESTIONE DEI FOGLI

Nella programmazione VBA risulta di notevole importanza la gestione dei fogli di una cartella di lavoro. Negli esempi che vedremo, relativi a questo argomento, è consigliabile partire da una cartella di lavoro vuota. É consigliabile inoltre, per comprendere bene cosa accade all'esecuzione del codice, tenere aperta la finestra VBA in modo da vedere, in secondo piano, la finestra Excel nella parte che visualizza le linguette corrispondenti ai fogli come evidenziato nella figura seguente.

Figura 12: La finestra VBA e, in secondo piano, la finestra Excel

Come tutti sappiamo il programma Excel visualizza tre fogli di lavoro denominati, nella versione italiana Foglio1, Foglio2, Foglio3. Le istruzioni VBA per inserire un nuovo foglio nella cartella di lavoro sono:

```
    . . .
Dim WS As Worksheet
Set WS = Sheets.Add
    . . .
```

Nel codice, dopo la dichiarazione di WS come foglio di lavoro, procediamo al riferimento del foglio appena creato con il nome WS (in realtà il comando Set assegna il riferimento ad un oggetto). Eseguita questa procedura vediamo che è stato creato un nuovo foglio chiamato Foglio4 che si trova davanti a quello attivo prima della sua esecuzione[5] . Volendo definire un comportamento diverso, dobbiamo ricorrere ad una variante del codice precedente costituita da:

```
Sheets.Add after:=Sheets(Sheets.Count)
```

5 In tutti gli esempi relativi all'inserimento di nuovi fogli, è fondamentale avere presente qual'è il foglio di lavoro attivo prima dell'esecuzione del codice, diversamente potrebbe risultare difficoltosa la comprensione di quanto accade.

che si differenzia dal precedente oltreché per la definizione della posizione in cui effettuare l'inserimento, anche per la mancanza di qualunque riferimento a nomi di foglio[6].

Con i comandi visti fin'ora, ai nuovi fogli veniva dato un nome costituito dal prefisso Foglio seguito da un numero progressivo. Volendo attribuire un nome diverso possiamo agire sulla proprietà costituita dal nome dell'oggetto attribuendogli un nome diverso. Il codice per fare questo è dato da:

```
   Worksheets.Add
' si ipotizza che non esista un foglio con lo stesso nome
' diversamente si verifichera' una condizione di errore
4    ActiveSheet.Name = "ciccio"
```

Tutti gli esempi appena visti, a condizione di essere partiti da una cartella di lavoro vuota, dovrebbero aver generato una situazione che può riassumersi nella figura 13 . Si notino le differenze tra la finestra

Figura 13: I fogli aggiunti

del progetto e quella di Excel di cui sono visibili le linguette. Nella finestra del progetto i fogli sono ordinati in un qualche modo, in quella di Excel i fogli sono disposti in riferimento al foglio attivo prima del loro inserimento.

Per chiarire meglio questo comportamento, è bene precisare che ogni foglio di una cartella ha tre identificatori chiamati rispettivamente *Code Name, Tab Name, Index Number*.

Il *Code Name* è quello che nella finestra del progetto si presenta per primo accanto al simbolo del foglio a sinistra. Nella figura **??** l'ultimo *Code Name* è Foglio6.

Il *Tab Name* si riferisce alla linguetta visibile nel foglio Excel. La posizione di questa è correlata a quella del foglio attivo prima del suo inserimento. Il suo valore può essere gestito tramite il codice VBA.

L'*Index Number* è un indice numerico corrispondente all'ordinale di quel foglio nella finestra del progetto. Dunque 1 per il primo foglio, 2 per il secondo, ..., e così via. La sua gestione è a totale appannaggio del programma Excel. Resta il fatto che l'utente può conoscerne il contenuto. Si consideri il seguente esempio:

```
1    MsgBox ActiveWorkbook.Sheets("ciccio").Index
     Sheets(3).Select
```

La prima istruzione visualizza l'Index Number, pari a 6, del foglio `ciccio`. La seconda costituisce un esempio di attivazione di un foglio grazie a codesto indice.

6 Per ognuno dei nuovi modi di inserire un foglio, si raccomanda di commentare le istruzioni eseguite in precedenza se fanno parte della stessa procedura o, in alternativa, di definire una procedura diversa. La ragione di questo sarà evidente nel seguito.

7.9 MISCELLANEA

La trattazione completa di tutti i metodi e le proprietà di VBA non è tra gli obiettivi di questo libro. In questo paragrafo ci occuperemo di illustrare altri metodi e proprietà che verranno impiegati nei prossimi capitoli.

Tra le funzionalità più utilizzate nella gestione delle celle del foglio possiamo annoverare quelle che si occupano di formato colore ed allineamento dei dati, occultamento o visualizzazione di righe o colonne, etc. Nel codice 13, della cartella di lavoro `proprieta_range.xls` sono scritti i comandi che definiscono alcune di queste proprietà.

Listing 13: L'aspetto delle celle

```
    . . .
Range("B3:C2").NumberFormat = "0"
Range("B2").Font.Bold = True
Range("C2").Font.Italic = True
Range("D2").Font.Underline = xlUnderlineStyleSingle
Range("B4").HorizontalAlignment = xlCenter
    . . .
```

Qui vediamo i vari comandi di formattazione del contenuto di una zona di celle. Potremmo poi aver bisogno di definire colore di fondo o del carattere. Ricordando che nello scrivere il nome di un oggetto, premuto il carattere ".", compare la finestra che visualizza quelli disponibili in relazione all'oggetto, consideriamo il seguente un esempio:

```
    . . .
Range("B2:C4").Interior.Color = vbYellow
    . . .
```

Si tratta del codice che definisce il colore di fondo di una zona. Volendo inserire dei bordi per le celle della zona dobbiamo utilizzare il codice sottostante:

```
    . . .
Range("B2:C4").Select
With Selection.Borders
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = xlAutomatic
End With
Range("A1").Select
    . . .
```

ESERCIZI

1. Assegnato il risultato di `Cells(5,2).Address` ad una opportuna variabile, si scriva la funzione che trasforma tale risultato da riferimento assoluto a riferimento relativo. Si tenga presente il risultato di `MsgBox Cells(3,3).Address` visto dianzi.

2. La colonna A, a partire da una riga qualsiasi, contiene una sequenza di celle piene. Contare le celle piene della zona utilizzando 2 differenti modalità.

Parte II

APPENDIX