DIDATTICA DELLE SCELTE ECONOMICHE E FINANZIARIE

MODULO 2

L'ambiente VBA e la programmazione

S.S.I.S. Puglia

Classe 48/A - Indirizzo Fis.-Inf.-Mat.

Prof. Crescenzio Gallo

C.GALLO@UNIFG.IT

Sommario

1. L'ambiente di sviluppo VBA
2. Gli elementi di base della sintassi VBA
3. Il flusso algoritmico di esecuzione
4. Dall'algoritmo al programma

1. L'ambiente di sviluppo VBA

Visualizzare il codice della Macro

È didatticamente molto utile esaminare il codice di una macro ed eseguirne passo-passo le istruzioni: facciamolo sulla macro **Formatta**, premendo il pulsante **Esegui istruzione**.

1. L'ambiente di sviluppo VBA

Visualizzare il codice della Macro

Si aprirà la finestra dell'ambiente VBA con il codice della macro visibile in una finestra.

L'istruzione che sta per essere eseguita è evidenziata in giallo e segnalata da una freccia gialla sulla sinistra; essa è la prima della macro, quella cioè che contiene il nome della macro stessa: **Sub Formatta()**

1. L'ambiente di sviluppo VBA

Visualizzare il codice della Macro

Premendo **F8** si evidenzia la prima istruzione del corpo della macro:

Range("B15").Select

che, una volta eseguita (alla successiva pressione del tasto **F8**), produce la selezione della cella **B15**.

Cap1.xls – Modulo1 (codice)

(generale) | Formatta

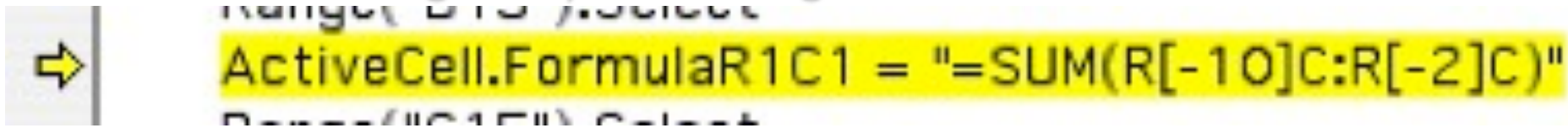
```
Sub Formatta()
'
' Formatta Macro
' Macro registrata il 15-01-2005 da Crescenzio Gallo
'

'
    Range("B15").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-10]C:R[-2]C)"
    Range("C15").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-10]C:R[-2]C)"
    Rows("3:3").Select
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Grassetto"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    Rows("15:15").Select
    With Selection.Font
```

Visualizzare il codice della Macro

Premuto F8 sul foglio la cella B15 diverrà la *cella attiva*.

La successiva istruzione (correntemente selezionata ma ancora da eseguire) è la seguente:



Questa istruzione inserisce nella cella attiva la somma del contenuto delle celle della stessa colonna della cella attiva che vanno dalla riga dieci posizioni sopra la riga della cella attiva alla riga che si trova due posizioni sopra quest'ultima (indirizzamento relativo).

Visualizzare il codice della Macro

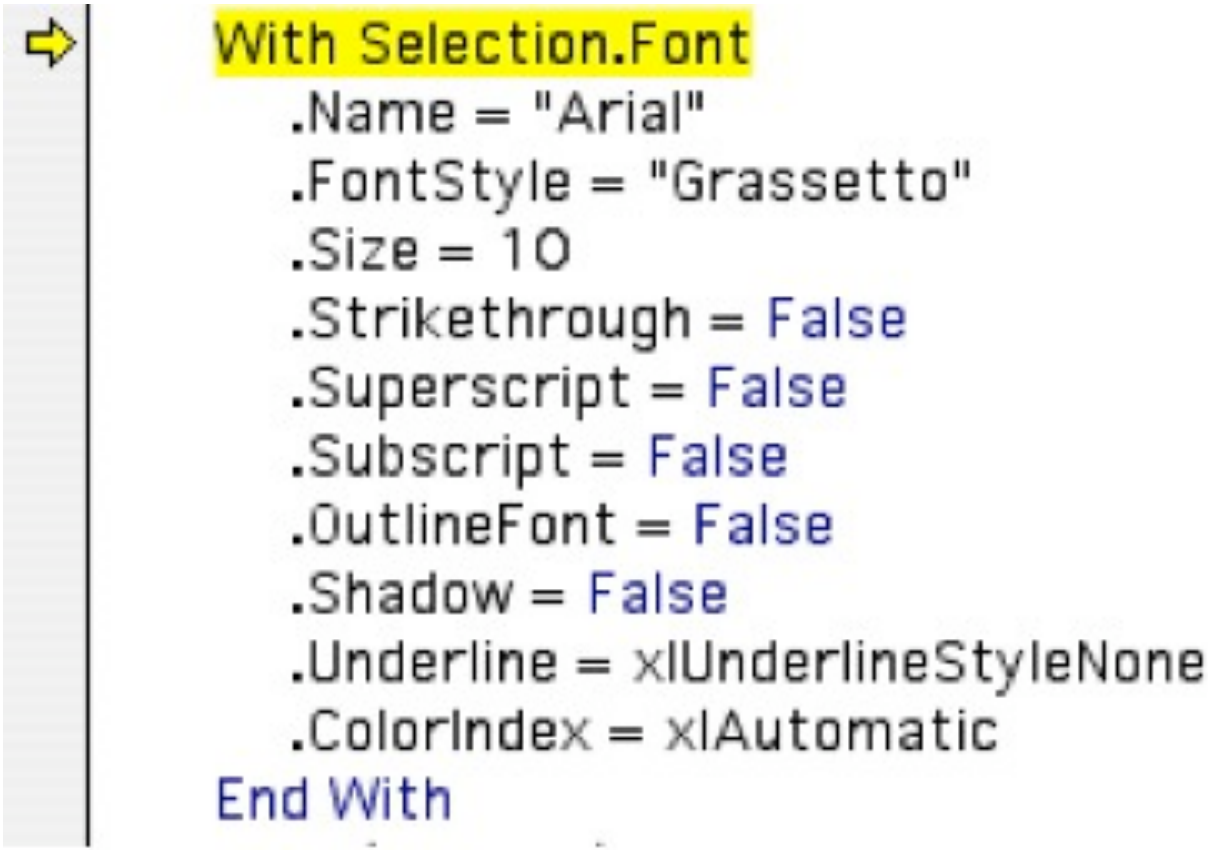
Proseguendo, le successive due istruzioni effettuano la stessa operazione sulla cella **C15**.

Premendo **F8** ripetutamente fermiamoci alla riga che contiene il comando: ➪ `Rows("3:3").Select`

che produrrà la selezione della terza riga del foglio attivo.

Visualizzare il codice della Macro

Premendo F8 otterremo la selezione della terza riga ed il cursore passerà sul blocco di istruzioni seguente:



nel quale le istruzioni comprese tra le parole chiave **With** e **End With** rappresentano diverse impostazioni di proprietà per il font del testo contenuto nelle celle.

Visualizzare il codice della Macro

Semplifichiamo il tutto cancellando le proprietà superlfue e scrivendo:

```
With Selection.Font
   .FontStyle = "Grassetto"
End With
```

O, più semplicemente:

```
Selection.Font.FontStyle = "Grassetto"
```

Visualizzare il codice della Macro

Le stesse operazioni vengono eseguite sulla riga 15, giungendo a:

```
Range("B5:B15").Select
Selection.NumberFormat = "[$€-410] #,##0.00"
Range("C5:C15").Select
Selection.NumberFormat = "[$ITL] #,##0"
```

Il cui significato è facilmente intuibile.

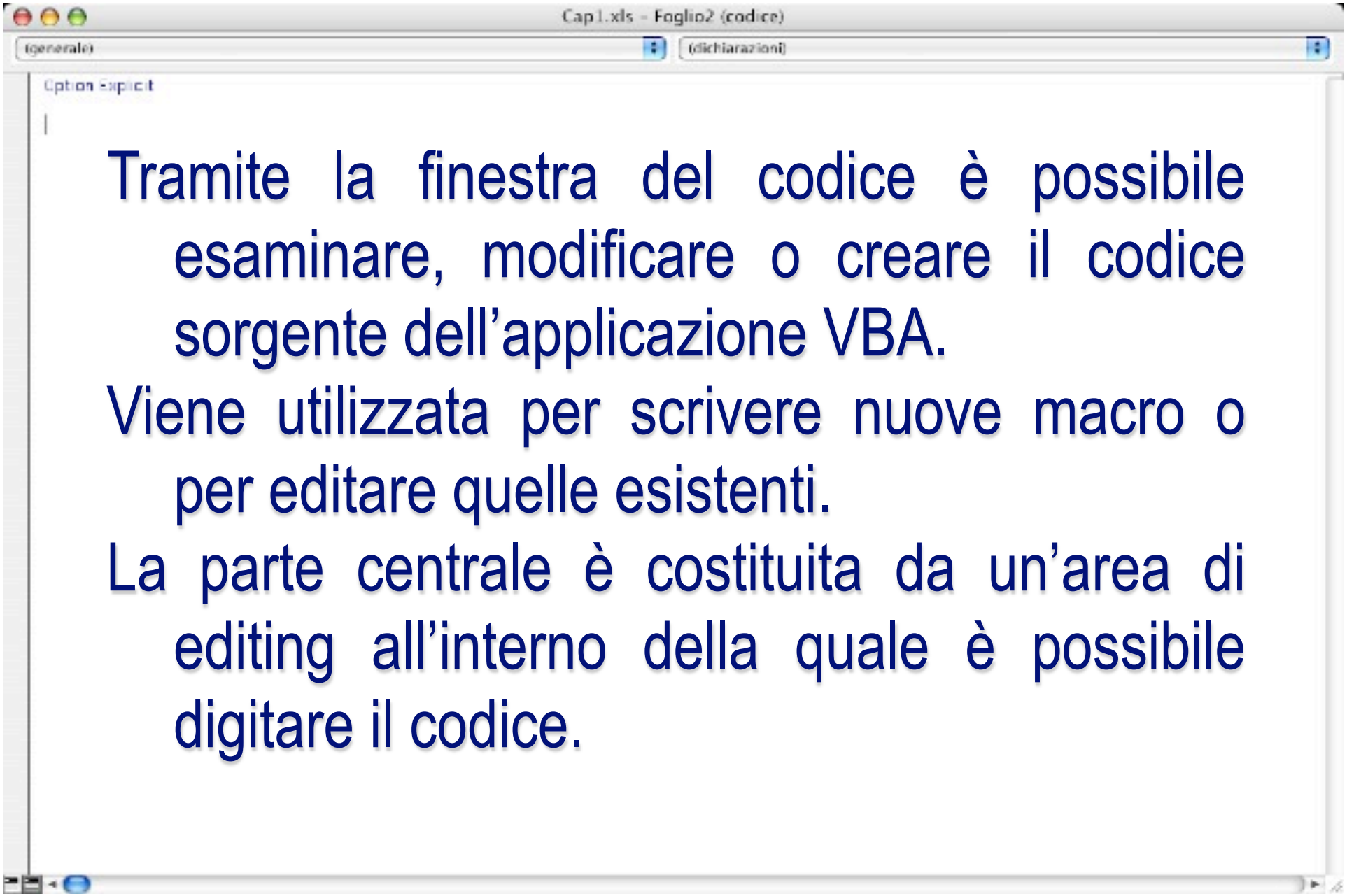
L'Editor VBA

L'ambiente di sviluppo Visual Basic for Applications (VBA) è un **ambiente integrato** col quale è possibile sviluppare e testare il codice, composto da una barra di *menu*, una barra di *pulsanti* ed una serie di *finestre*, quali:

- codice;
- progetto;
- proprietà;
- variabili locali;
- immediata;
- stack delle chiamate;
- espressioni di controllo.

Esaminiamo in dettaglio le prime tre.

L'Editor VBA - Finestra del codice

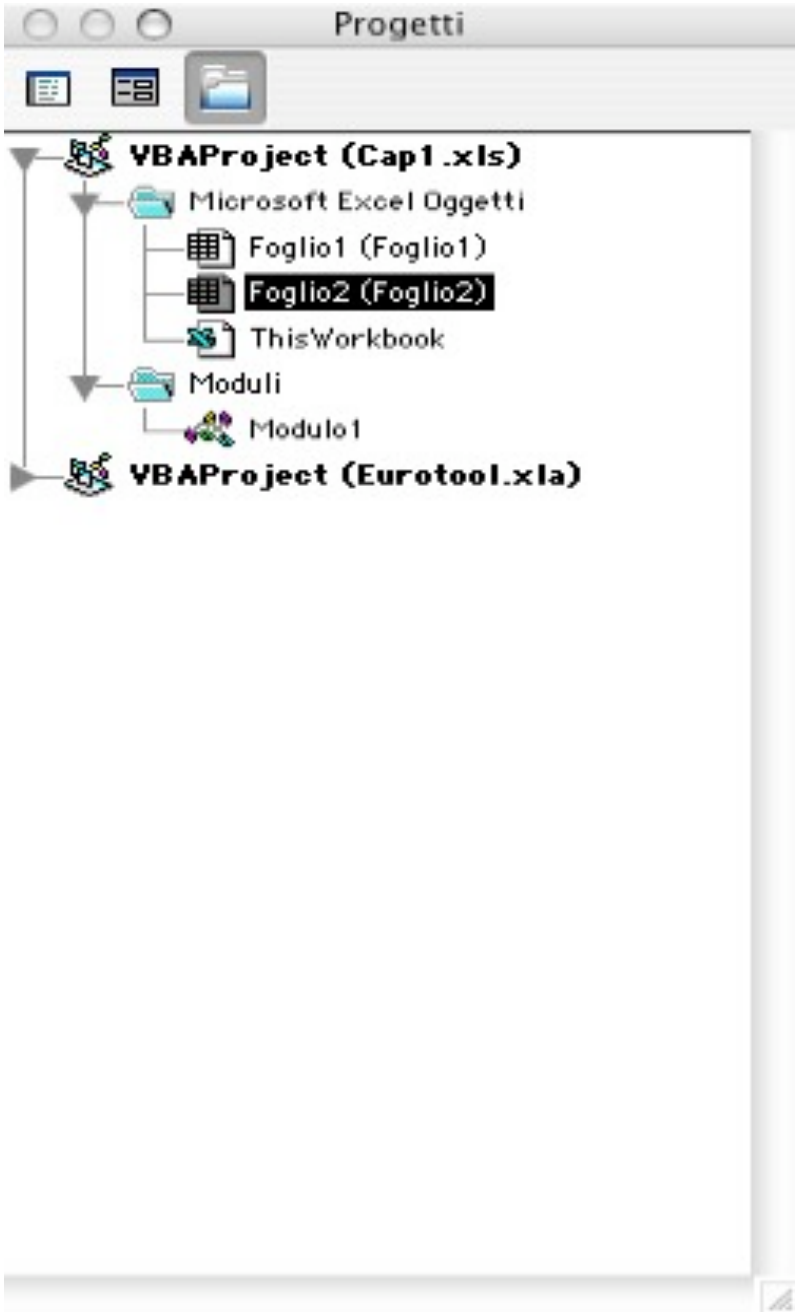


Tramite la finestra del codice è possibile esaminare, modificare o creare il codice sorgente dell'applicazione VBA.

Viene utilizzata per scrivere nuove macro o per editare quelle esistenti.

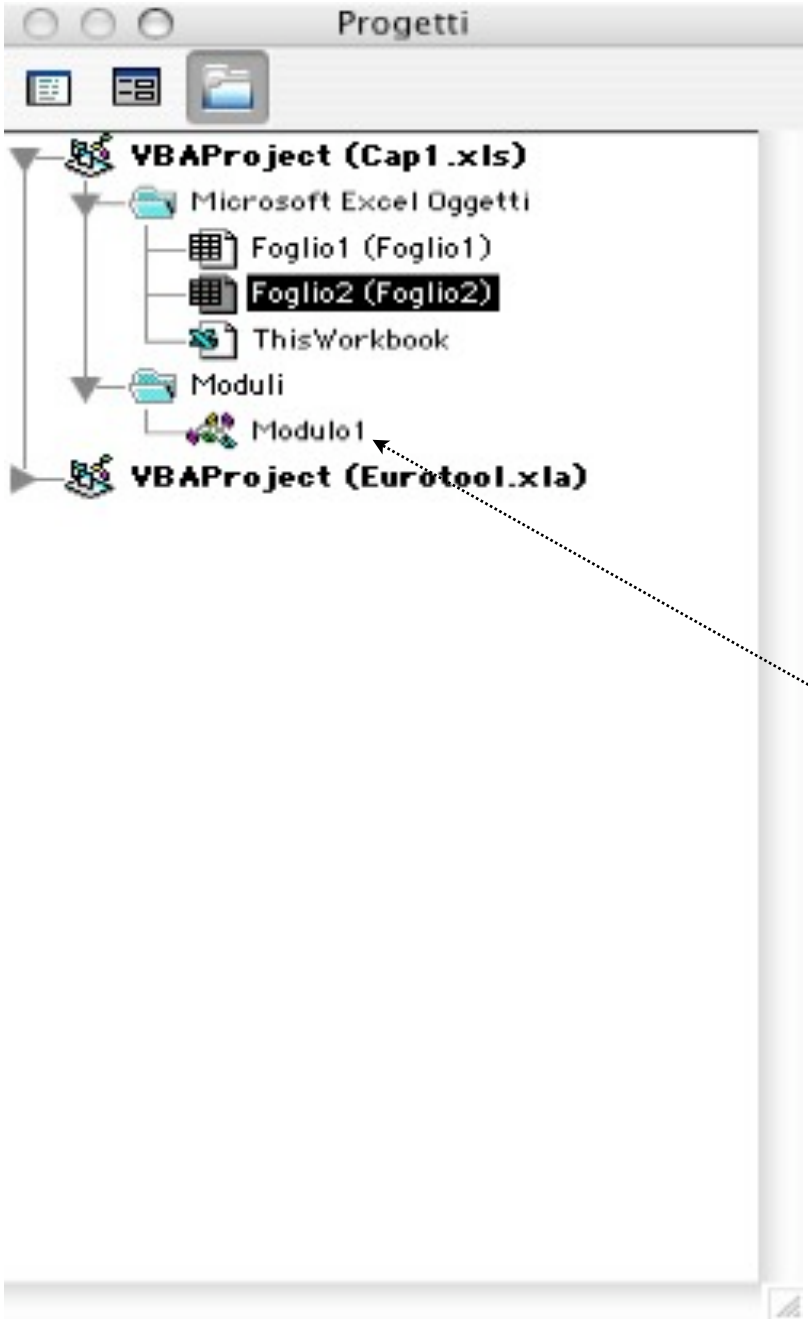
La parte centrale è costituita da un'area di editing all'interno della quale è possibile digitare il codice.

L'Editor VBA - Finestra progetti



La finestra di gestione progetti contiene un diagramma ad albero delle cartelle di lavoro aperte e degli oggetti Excel in esse contenuti (fogli, moduli, riferimenti, form, la stessa cartella, etc.) e serve per muoversi tra di essi.

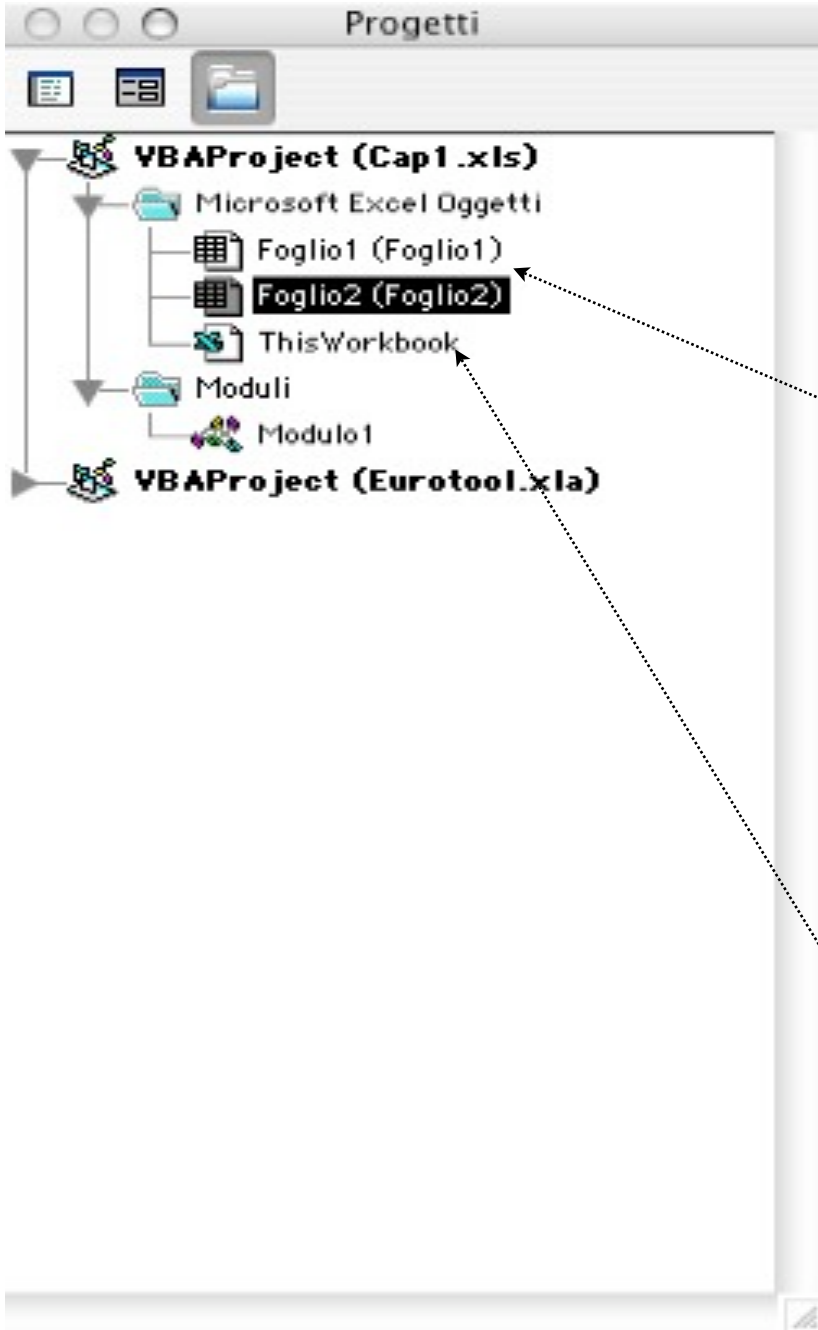
L'Editor VBA - Finestra progetti



Vi sono due gruppi di elementi: *oggetti* e *moduli*, questi ultimi utilizzati per raggruppare logicamente le funzioni (ad es. routine finanziarie, routine di accesso ai dati, etc.) e consentire una maggiore leggibilità e manutenibilità del codice.

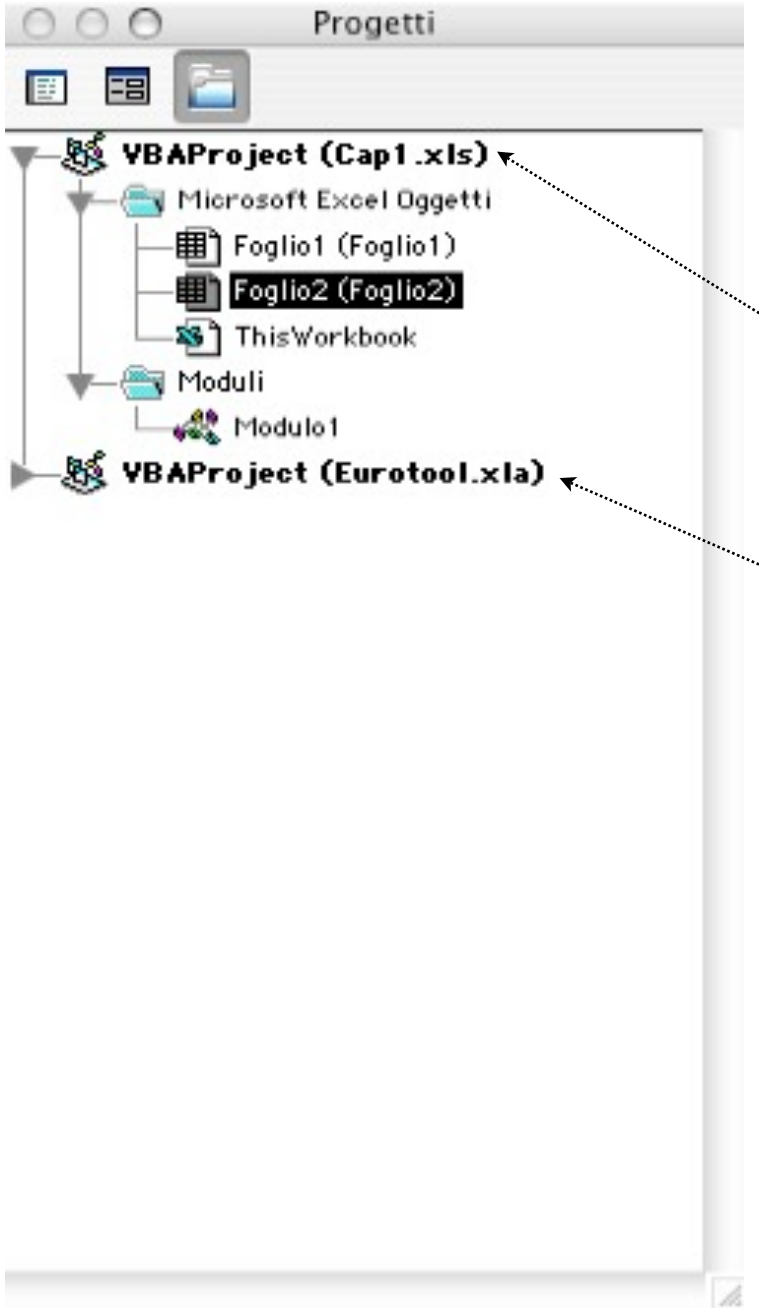
Nel nostro caso la macro **Formatta** è contenuta nel modulo **Modulo1**.

L'Editor VBA - Finestra progetti



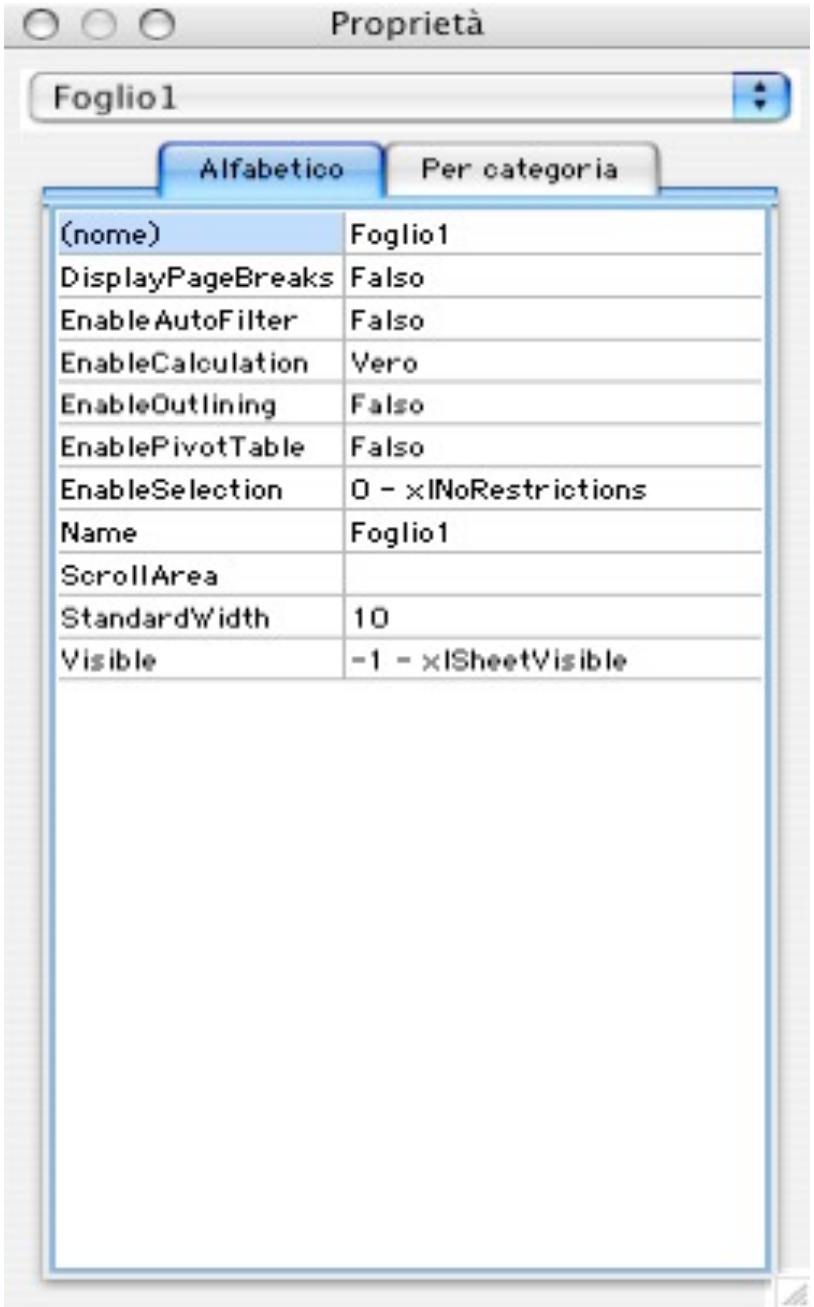
In un qualunque progetto esistono sempre almeno due oggetti: un foglio (ad es. **Foglio1**, **Foglio2**, …) che identifica un oggetto di tipo foglio di lavoro il cui nome in Excel è quello riportato tra parentesi (in questo esempio i nomi coincidono), e la cartella corrente (**ThisWorkbook**).

L'Editor VBA - Finestra progetti



Nella finestra oltre al progetto corrente, che si chiama **VBAProject** (il nome Cap1.xls tra parentesi è il nome del file in cui è memorizzato) sono di solito presenti altri progetti, come il progetto **Eurotool.xla** in figura (xla = Excel add-in, non modificabile dall'utente).

L'Editor VBA - Finestra proprietà



Questa finestra elenca tutte le proprietà dell'oggetto (**Foglio1**, nella figura) attualmente selezionato.

Le proprietà definiscono le caratteristiche dell'oggetto (come il suo nome, il colore, la posizione sullo schermo) o lo stato.

L'Editor VBA - Finestra proprietà

Per visualizzare le proprietà di un oggetto è sufficiente selezionarlo nella finestra di progetto; per modificare il valore di una proprietà è sufficiente selezionarla nella colonna a sinistra e modificarne il valore nella colonna a destra (digitandolo o selezionandolo da un elenco a discesa).

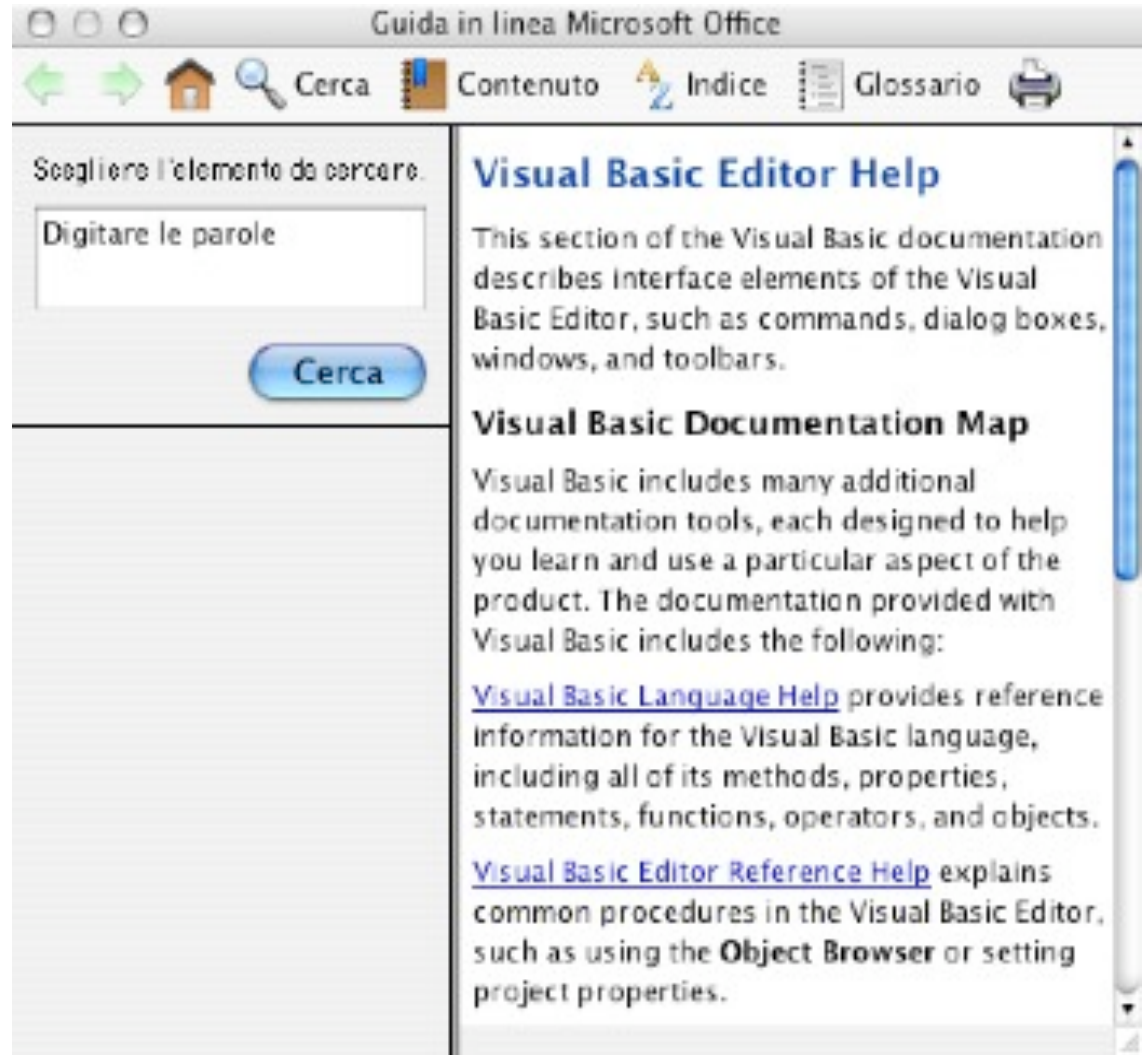
2. La sintassi VBA

Esamineremo ora una sintesi dei principali **elementi sintattici** del VBA, con lo scopo di offrire una base di lavoro per comprendere le varie applicazioni che saranno sviluppate.

Sarà in tal modo agevole estendere le nozioni di VBA presentando di volta in volta esempi specifici per l'apprendimento sia delle nozioni sintattiche che di quelle funzionali.

2. La sintassi VBA

Importante è comunque l'uso della **guida in linea** nonché il ricorso a manuali specifici del linguaggio per la padronanza degli strumenti di sviluppo presentati in questo corso.



Le variabili

Tramite le variabili è possibile (in qualunque linguaggio, VBA compreso) memorizzare dei valori (di testo o numerici) in "**contenitori**" accessibili tramite un nome loro assegnato (hanno una funzione analoga alle memorie temporanee di una calcolatrice).

Il loro contenuto (il valore della variabile) può variare, ma il nome resta inalterato, analogamente anche all'uso che si fa di una variabile in un'equazione matematica.

Le variabili

Una variabile è detta *locale* se definita all'interno di una routine (procedura o funzione), attraverso la dichiarazione (obbligatoria, se si usa **Option Explicit**):

Dim *nome* [**As** *tipo*]

in cui *nome* è il nome assegnato alla variabile; il *tipo* può non essere specificato, ed in tal caso la variabile è di tipo **variant**, cioè può accogliere valori di tipi diversi (anche se questa pratica è fortemente sconsigliata, sia per la maggiore occupazione di memoria che per i potenziali problemi derivanti dall'ambiguità del valore presente nella variabile stessa).

Le variabili

Le variabili non definite localmente ad una procedura o funzione (esamineremo la differenza in seguito) si dicono *globali*, e possono essere utilizzate in qualsiasi punto del codice essendo "visibili" ovunque (e perciò pericolose!), a differenza di quelle locali, che "esistono" solo nella routine in cui sono dichiarate.

Le variabili

Il **nome di una variabile**:

- deve iniziare con una lettera dell'alfabeto;
- non può contenere spazi, punti o simboli speciali (operatori matematici, …);
- non può superare i 255 caratteri (!);
- non deve corrispondere ad una delle parole chiave del VBA.

Le variabili

Il VBA non distingue fra lettere minuscole e maiuscole nei nomi degli "oggetti", ed è consigliabile assegnare nomi attinenti all'uso: ad es. **Ipotenusa** piuttosto che **y**, **Stipendio_base** piuttosto che **Pippo**, etc.

L'assegnazione di un valore ad una variabile avviene mediante l'operatore =. Ad es.:

Ipotenusa = 5.4

Tipi di dati VBA

Si definisce ***tipo di dato*** l'insieme dei valori e delle possibili operazioni che si possono compiere su un dato di una certa natura.

Tipi di dati VBA

In VBA i tipi di dati **numerici** (e relative occupazioni di memoria) sono:

- **Byte** (1 byte), da 0 a 255;
- **Integer** (2 byte), da -32.768 a 32.767;
- **Long** (4 byte), da -2.147.483.648 a 2.147.483.647;
- **Single** (4 byte), da $-3{,}402823 \times 10^{38}$ a $-1{,}401298 \times 10^{-45}$ per valori negativi, e da $1{,}401298 \times 10^{-45}$ a $3{,}402823 \times 10^{38}$ per valori positivi;
- **Double** (8 byte), da $-1{,}7976931348623 \times 10^{308}$ a $-4{,}9406564584124 \times 10^{-324}$ per valori negativi, e da $4{,}9406564584124 \times 10^{-324}$ a $1{,}7976931348623 \times 10^{308}$ per valori positivi;
- **Currency** (8 byte), da -922.337.203.685.477,5808 a -922.337.203.685.477,5807;
- **Decimal** (14 byte), ±79.228.162.514.264.337.593.543.950.335 (da 0 a 28 decimali; può essere definito solo per conversione CDec di un tipo *variant*).

Operatori matematici

Gli **operatori matematici** più comuni sui tipi numerici sono:

`+` : addizione

`-` : sottrazione

`*` : moltiplicazione

`/` : divisione floating (risultato completo della parte decimale)

`\` : divisione intera (solo il quoziente)

`^` : elevamento a potenza

`mod` : resto della divisione intera

Operatori matematici

Esempi:

`x+1`	`3.5+2.7`	`a-3`	`4.7-1`
`y*2`	`x*y`	`x/2.0`	`7/5`
`b\c`	`7\5`	`x^2`	`2^3`
`x mod 2`	`7 mod 5`		

Operatori di confronto

Gli **operatori relazionali** (o di confronto) fra variabili numeriche sono:

< : minore

<= : minore o uguale (≤)

= : uguale

<> : diverso (≠)

> : maggiore

>= : maggiore o uguale (≥)

Operatori di confronto

Esempi:

a<b (minore)

X<=0 (minore o uguale)

A=B (uguale)

y<>1 (diverso)

Raggio>3.14 (maggiore)

c>=2 (maggiore o uguale)

Altri tipi di dati VBA

Gli altri tipi di dati (e relative occupazioni di memoria) sono:

- **Date** (8 byte), dall'1/1/100 al 31/12/9999;
- **Object** (4 byte), qualsiasi riferimento ad un oggetto;
- **String** (10 byte+lunghezza stringa), da zero a circa 2 miliardi di caratteri;
- **Variant** (16 byte per qualsiasi tipo numerico);
- **Variant** (22 byte+lunghezza stringa per il tipo *string*).

Altri tipi di dati VBA

Le **stringhe di testo** sono di due tipi: a lunghezza variabile (il tipo *string*, da zero a circa 2 miliardi di caratteri) ed a lunghezza fissa (il tipo *string*n*); ad es.:

```
Dim sv As String  'fino a 2 miliardi di caratteri
sv = "Stringa a lunghezza variabile"
Dim sf As String*20 'contiene sempre 20 caratteri
sf = "stringa fissa" 'altri 7 spazi in coda…
```

Altri tipi di dati VBA

Il tipo **Date**

Memorizza la data e l'ora; il valore va racchiuso tra #.

Ad esempio:

```
Dim Capodanno As Date
Dim Timbratura As Date
Capodanno = #31/12/2004#
Timbratura = #15/01/2005 08:33:00#
```

Altri tipi di dati VBA

Il tipo **Boolean**

Rappresenta un valore "logico", cioè vero (True) o falso (False). Ad esempio:

```
Dim SaldoPositivo As Boolean
SaldoPositivo = Entrate - Uscite > 0
```

Altri tipi di dati VBA

Il tipo **Variant**

È il tipo di dati in cui vengono convertite tutte le variabili non esplicitamente "tipizzate"; richiede più memoria degli altri tipi e le operazioni su di esso sono più lente, specie quelle sui numeri. Ad esempio:

```
Dim C  'può essere qualsiasi cosa…
C = 3.5  'contiene un dato numerico di tipo double
C = "pippo"  'contiene un dato di tipo string
```

Tipi di dati VBA - Esempi

Facciamo un esempio pratico di quanto sinora esposto. Creiamo una nuova macro di nome **Test_Variabili**, facendo attenzione alla presenza, nel modulo, della dichiarazione **Option Explicit** (se non compare, selezioniamo in Windows: *Strumenti→Opzioni→Editor→Dichiarazione di variabili obbligatoria* oppure per il MacOSX: *Excel→Preferenze→Redattore→Dichiarazione di variabili obbligatoria*).

Tipi di dati VBA - Esempi

```
Sub Test_Variabili()
Dim VariabileIntera As Integer
Dim VariabileReale As Single
Dim VariabileTesto As String
Dim RisultatoConfronto As Boolean

VariabileIntera = 7
VariabileReale = 8.5

VariabileTesto = "Questa è la macro "
VariabileTesto = VariabileTesto & "Test_Variabili"

RisultatoConfronto = VariabileIntera > VariabileReale
RisultatoConfronto = VariabileIntera < VariabileReale
End Sub
```

Tipi di dati VBA - Esempi

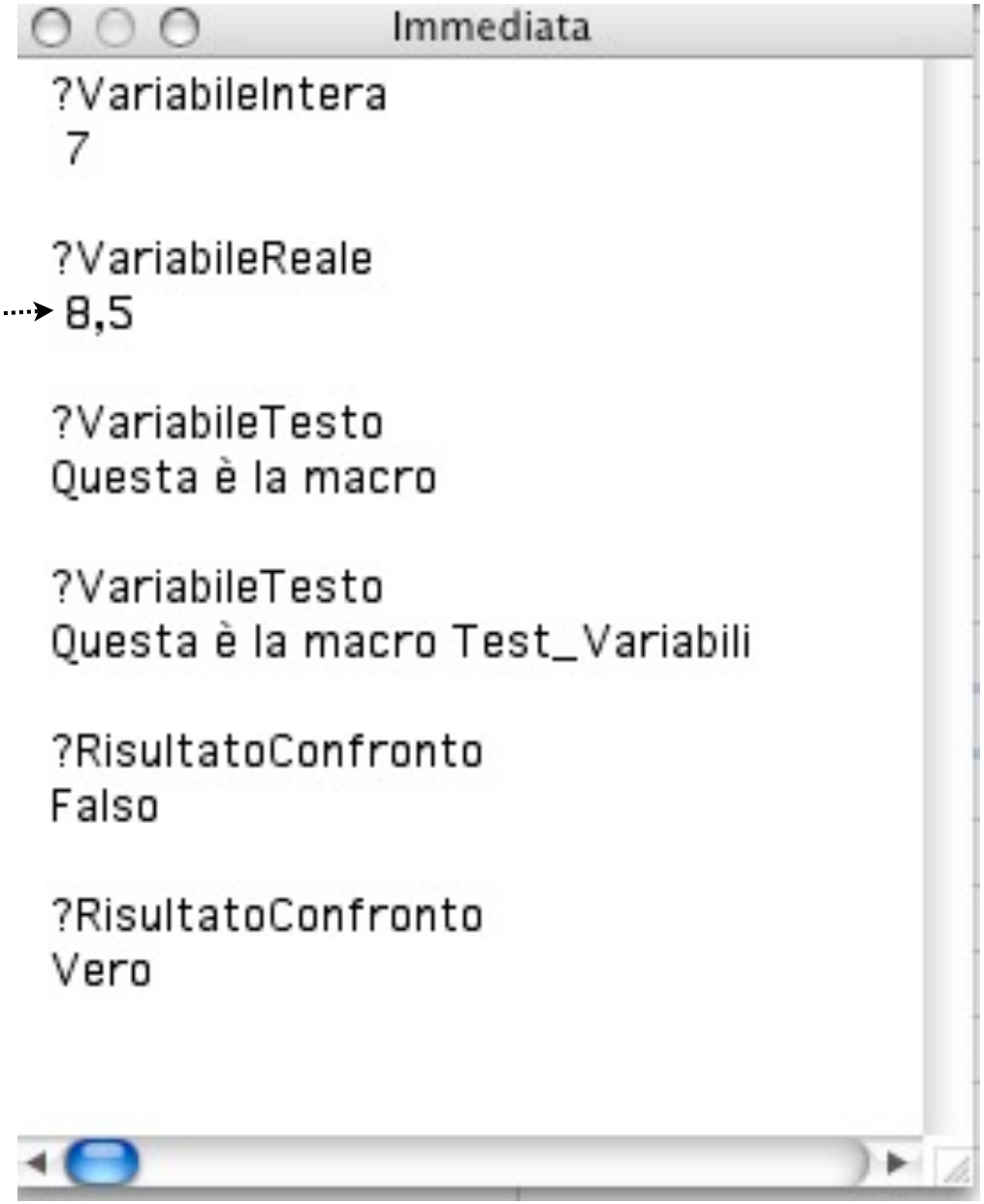
Eseguiamo la macro passo-passo (premendo il tasto F8).

Inizialmente, le variabili numeriche sono a zero, mentre quelle stringa sono di lunghezza nulla e quelle logiche hanno valore falso.

L'esecuzione progressiva porta all'attribuzione dei valori indicati nella macro, per cui dopo la prima istruzione la variabile intera ha valore **7** (come si vede dalla finestra Immediata).

Tipi di dati VBA - Esempi

Successivamente, la variabile reale assume il valore **8,5** (si noti che viene visualizzata la virgola come separatore decimale, ma nel codice viene utilizzato il punto).



Tipi di dati VBA - Esempi

Quindi la variabile di testo contiene prima il valore "**Questa è la macro** " e poi, concatenandosi con la stringa "**Test_Variabili**", diventa "**Questa è la macro Test_Variabili**".

Tipi di dati VBA - Esempi

Infine, la variabile logica assume il valore **Falso** a seguito del primo confronto (infatti non è 7>8,5), e poi il valore **Vero** a seguito del secondo confronto (infatti è 7<8,5).

Nell'ambiente Windows è anche possibile fare uso della finestra "Variabili locali", *che mostra l'evoluzione dei valori per le variabili in uso.*

Strutture di controllo

L'istruzione condizionale If

Il **controllo del flusso** di esecuzione all'interno di una routine può variare a seconda del verificarsi di date condizioni, in conseguenza delle quali occorre prendere delle decisioni operative.

Strutture di controllo

L'istruzione condizionale If

Il "Visual Basic for Applications" consente di fare questo per mezzo dell'istruzione **If**, così strutturata:

If <*condizione*> **Then**
  <*istruzioni da eseguire se la condizione è vera*>
[**Else**
  <*istruzioni da eseguire se la condizione è falsa*>]
End If

Notiamo che la parte **Else** è opzionale, mentre **End If** chiude la struttura condizionale.

Strutture di controllo

Esempi di istruzioni If

Questo esempio "cicla" sulle celle A1:D10 sul Foglio1. Se una delle celle ha un valore < 0,001, il codice sostituisce il valore con 0 (zero):

```
For Each c in _
  Worksheets("Sheet1").Range("A1:D10")
     If c.Value < .001 Then
          c.Value = 0
     End If
Next c
```

Strutture di controllo

Esempi di istruzioni If

Questo esempio "cicla" sulla zona denominata "TestRange" e visualizza quindi il numero di celle vuote nel range:

```
numBlanks = 0
For Each c In Range("TestRange")
    If c.Value = "" Then
        numBlanks = numBlanks + 1
    End If
Next c
MsgBox "Ci sono " & numBlanks & _
  " celle vuote in questa zona."
```

Strutture di controllo

Esempi di istruzioni If

Questo esempio imposta il font standard a Geneva (sul Macintosh) o Arial (in Windows):

```
If Application.OperatingSystem Like
"*Macintosh*" Then
    Application.StandardFont = "Geneva"
Else
    Application.StandardFont = "Arial"
End If
```

Strutture di controllo

L'istruzione condizionale Select Case

La presenza di **condizioni multiple** pone seri problemi di scrittura e leggibilità se viene utilizzata l'istruzione **If**.

Infatti, supponiamo di voler calcolare la provvigione sulla base dell'importo fatturato; con l'istruzione **If** dovremmo scrivere qualcosa del genere:

Strutture di controllo

L'istruzione condizionale Select Case

```
If Importo<10000 Then
  Provvigione = Importo * .05 '5%
Else
  If Importo<50000 Then
    Provvigione = Importo * .08 '8%
  Else
    Provvigione = Importo * .1 '10%
  End If
End If
```

Strutture di controllo

L'istruzione condizionale Select Case

Possiamo invece fare uso della *struttura a selezione multipla* **Select Case**, molto più sintetica e leggibile dell'istruzione **If** ed avente la seguente sintassi (nella forma più semplice):

Strutture di controllo

L'istruzione condizionale Select Case

```
Select Case <variabile/espressione>
   Case <valore 1>
      <blocco istruzioni 1>
   Case <valore 2>
      <blocco istruzioni 2>
    ...
   Case <valore n>
      <blocco istruzioni n>
   Case Else
      <blocco da eseguire se tutti i confronti falliscono>
End Select
```

Strutture di controllo

L'istruzione condizionale Select Case

Applicandola all'esempio precedente, scriveremo:

```
Select Case Importo
  Case Is < 10000 Then
    Provvigione = Importo * .05 '5%
  Case Is < 50000
    Provvigione = Importo * .08 '8%
  Case Else
    Provvigione = Importo * .1 '10%
End Select
```

Strutture di controllo

L'istruzione condizionale Select Case

Se la <*variabile/espressione*> corrisponde ad uno qualsiasi dei valori elencati (<*valore 1*>, <*valore 2*>, …, <*valore n*>), vengono eseguite le istruzioni del blocco corrispondente sino alla successiva clausola **Case** o, per l'ultima clausola, sino a **End Select**.

Il controllo passa quindi all'istruzione successiva a **End Select**.

Strutture di controllo

L'istruzione condizionale Select Case

Se la <*variabile/espressione*> corrisponde a più di un valore nelle clausole **Case**, vengono eseguite solamente le istruzioni successive alla prima corrispondenza.

Strutture di controllo

L'istruzione condizionale Select Case

La clausola **Case Else** viene utilizzata per indicare le istruzioni alternative da eseguire qualora non si verifichi alcuna corrispondenza.

Sebbene non strettamente necessario, è buona norma inserire una clausola **Case Else** in ogni blocco **Select Case** per gestire valori non previsti.

Strutture di controllo

L'istruzione condizionale Select Case

Se non si verifica nessuna corrispondenza tra i valori delle clausole **Case** e la <*variabile/espressione*> e manca la clausola **Case Else**, l'esecuzione prosegue con l'istruzione successiva a **End Select**.

Strutture di controllo

L'istruzione condizionale Select Case

È possibile utilizzare espressioni multiple o intervalli di valori in ciascuna clausola **Case**.

Ad esempio, è lecito scrivere:

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

Strutture di controllo

L'istruzione condizionale Select Case

È anche possibile specificare intervalli di valori ed espressioni multiple per stringhe di caratteri.

Nell'esempio seguente, la clausola **Case** è vera per stringhe che sono esattamente uguali a "tutto", stringhe comprese alfabeticamente tra "alfa" e "omega" e per il valore corrente della variabile **ProvaTest**:

```
Case "tutto", "alfa" To "omega",
ProvaTest
```

Strutture di controllo

L'istruzione condizionale Select Case

Le istruzioni **Select Case** possono essere "nidificate"; a ciascuna clausola **Select Case** deve corrispondere una **End Select**.

Strutture di controllo

L'operatore And

La presenza di condizioni multiple può essere più semplicemente valutata ricorrendo all'operatore **And** (congiunzione logica). Ad esempio, invece di scrivere:

```
If Numero>=1 Then
  If Numero<=10 Then
    MsgBox "Il numero è compreso
tra 1 e 10"
  End If
End If
```

Strutture di controllo

L'operatore And

potremmo più semplicemente scrivere:

```
If (Numero>=1) And (Numero<=10) Then
  MsgBox "Il numero è compreso tra 1 e 10"
End If
```

Strutture di controllo

L'operatore Or

Più condizioni indipendenti tra di loro possono essere sintetizzate ricorrendo all'operatore **Or** (disgiunzione logica). Ad esempio, invece di scrivere:

```
If Numero<1 Then
   MsgBox "Il numero è errato!"
End If
If Numero>10 Then
   MsgBox "Il numero è errato!"
End If
```

Strutture di controllo

L'operatore Or

potremmo più semplicemente scrivere:

```
If (Numero<1) Or (Numero>10) Then
  MsgBox "Il numero è errato!"
End If
```

Strutture di controllo

Il ciclo For

Volendo ad es. calcolare il fattoriale, è necessario ripetere più volte la stessa operazione (la moltiplicazione).

In tal caso, risulta più agevole l'utilizzo di una nuova struttura, detta "**iterativa**" (di tipo *enumerativo*) così definita:

Strutture di controllo

Il ciclo For

```
For <contatore> = <inizio> To <fine> [Step <passo>]
    <istruzione 1>
    ...
    <istruzione n>
Next [<contatore>]
```

Il <*contatore*> è una variabile intera che assume il valore <*inizio*>, viene incrementata (algebricamente) del <*passo*> (di solito 1, e quindi può essere omesso) ad ogni iterazione, fino al raggiungimento del valore <*fine*>; le istruzioni comprese tra le parole chiave For e Next costituiscono il cosiddetto "corpo" del ciclo.

Strutture di controllo

Il ciclo For

Il calcolo del fattoriale di 10 sarebbe quindi effettuato come segue:

```
Dim F As Long, i As Integer
F = 1
For i = 1 To 10 Step 1
    F = F * i
Next i
MsgBox "Il fattoriale di 10 è " & F
```

Osserviamo che è possibile omettere il passo (Step 1) e la variabile dopo Next.

Strutture di controllo

Il ciclo While

Un tipo diverso di iterazione è costituito dal ciclo While, che permette la ripetizione delle istruzioni contenute nel corpo (delimitate dalle parole chiave Do While…Loop) fintantoché risulta vera la <*condizione*>:

```
Do While <condizione>
    <istruzione 1>
    ...
    <istruzione n>
Loop
```

Strutture di controllo

Il ciclo While

Il ciclo termina ed il flusso di esecuzione prosegue dopo la parola chiave Loop quando la condizione diventa falsa.

Bisogna porre molta attenzione alla formulazione della <*condizione*>, poiché si potrebbe incappare in un "ciclo infinito" (se la condizione stessa non diventa mai falsa!).

Strutture di controllo

Il ciclo Until

Questa struttura è simile al ciclo While, ma ripete le istruzioni contenute nel corpo (delimitate dalle parole chiave Do Until…Loop) fintantoché risulta falsa la <*condizione*>:

```
Do Until <condizione>
    <istruzione 1>
    ...
    <istruzione n>
Loop
```

Strutture di controllo

Il ciclo Until

Il ciclo termina ed il flusso di esecuzione prosegue dopo la parola chiave Loop quando la condizione diventa vera.

Bisogna porre molta attenzione alla formulazione della <*condizione*>, poiché si potrebbe incappare in un "ciclo infinito" (se la condizione stessa non diventa mai vera!).

Strutture di controllo

I cicli

Il ciclo "For…Next" è detto **enumerativo** poiché consente di "contare" il numero di iterazioni che si intende realizzare (al limite nessuna).

Strutture di controllo

I cicli

Entrambe le strutture iterative "Do While" e "Do Until" viste hanno la caratteristica di eseguire il controllo della condizione prima dell'eventuale esecuzione del corpo.

Ciò significa che il corpo potrebbe anche non essere affatto eseguito, qualora la condizione sia già inizialmente falsa (per il ciclo Do While) o vera (per il ciclo Do Until).

Strutture di controllo

I cicli

Qualora si intenda garantire almeno una iterazione, la formulazione sintattica dovrebbe essere la seguente:

```
Do
    <istruzione 1>
        ...
    <istruzione n>
Loop While/Until <condizione>
```

3. Il flusso algoritmico di esecuzione

La conoscenza della **sintassi** di un linguaggio di programmazione è ovviamente solo il primo passo verso la risoluzione di problemi.

3. Il flusso algoritmico di esecuzione

Quando si intende eseguire un'elaborazione, è necessario aver prima studiato il "**metodo risolutivo**", da descrivere in maniera estremamente rigorosa e precisa.

3. Il flusso algoritmico di esecuzione

Si deve arrivare a formulare il cosiddetto *algoritmo* (dal nome del matematico arabo al-Khowaritzmi, 825 a.C.), da descrivere in modo rigoroso, secondo passi elementari logicamente e temporalmente connessi tra di loro.

Un algoritmo possiede le seguenti **caratteristiche**:

Caratteristiche di un algoritmo

1. È costituito da una successione finita di azioni elementari che determinano la soluzione di un problema.
2. Le azioni specificate devono essere "non ambigue", cioè univocamente interpretabili dall'esecutore.
3. A parità di "input", l'algoritmo deve produrre gli stessi "output" (*deterministico*).
4. Deve essere quanto possibile *generale*, cioè deve risolvere una classe di problemi analoghi.

Un algoritmo può essere rappresentato: graficamente (tramite flow-chart), in forma discorsiva, mediante un linguaggio di programmazione.

Esempio di algoritmo

Descrivere in forma discorsiva e successivamente mediante flow-chart l'algoritmo di Newton-Raphson per la determinazione dello zero di una funzione $y = f(x)$.

Data una funzione $f(x)$ (derivabile) il problema consiste nel determinare il valore di $\underline{x}$ tale che $f(\underline{x})=0$. Partendo da una stima iniziale x_0 della soluzione si genera una successione di valori $\{x_k\}$ approssimando, per ogni k, la curva $y=f(x)$ con la tangente nel punto $(x_k, f(x_k))$ e calcolando x_{k+1} come l'intersezione della tangente con l'asse delle x.

All'equazione $f(x)=0$ si sostituisce così l'equazione della retta tangente:

$$f(x_k) + (x - x_k)f'(x_k) = 0$$

da cui:

$$x = x_{k+1} = x_k - f(x_k)/f'(x_k)$$

Esempio di algoritmo

Ma, dato per scontato che la successione di valori converga verso una soluzione finale (il che non è sempre vero…), il problema consiste nel sapere "quando fermarsi", cioè nel determinare il valore di x_{k+1} "accettabile".

Si pone cioè il problema (fondamentale nella scrittura di un algoritmo non banale) di definire delle condizioni di controllo del flusso elaborativo che garantiscano in ogni caso il raggiungimento della soluzione finale in un numero finito di passi.

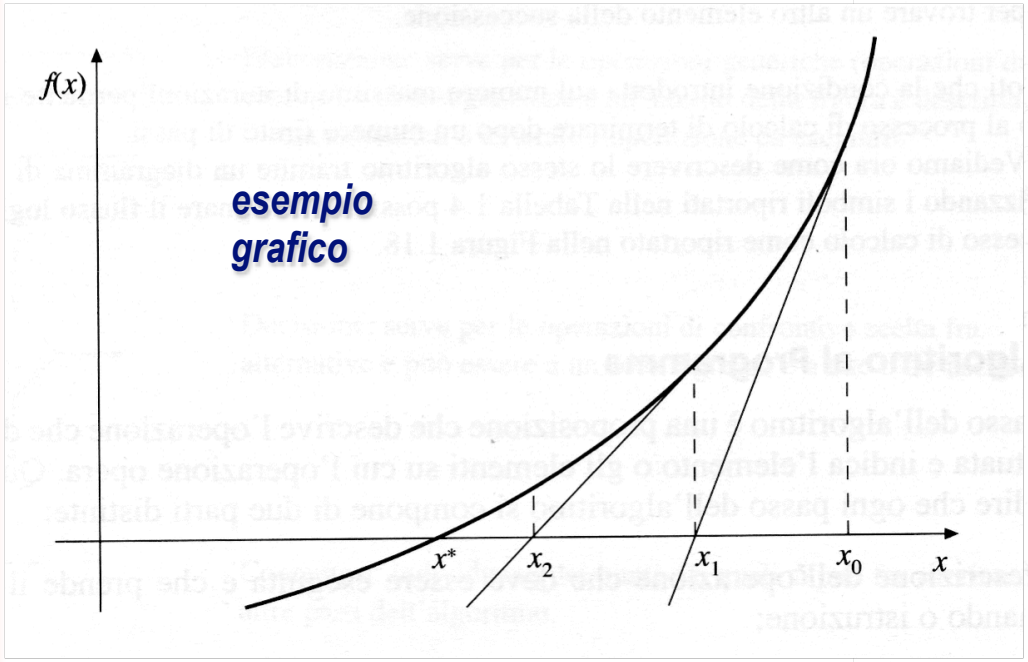
In questo caso si potrebbe terminare l'algoritmo quando lo scostamento tra due valori successivi è minore di una tolleranza ε prefissata: $|x_{k+1} - x_k| < \varepsilon$

Esempio di algoritmo

Vediamo allora come descrivere in forma discorsiva un algoritmo che implementi il metodo di Newton-Raphson:

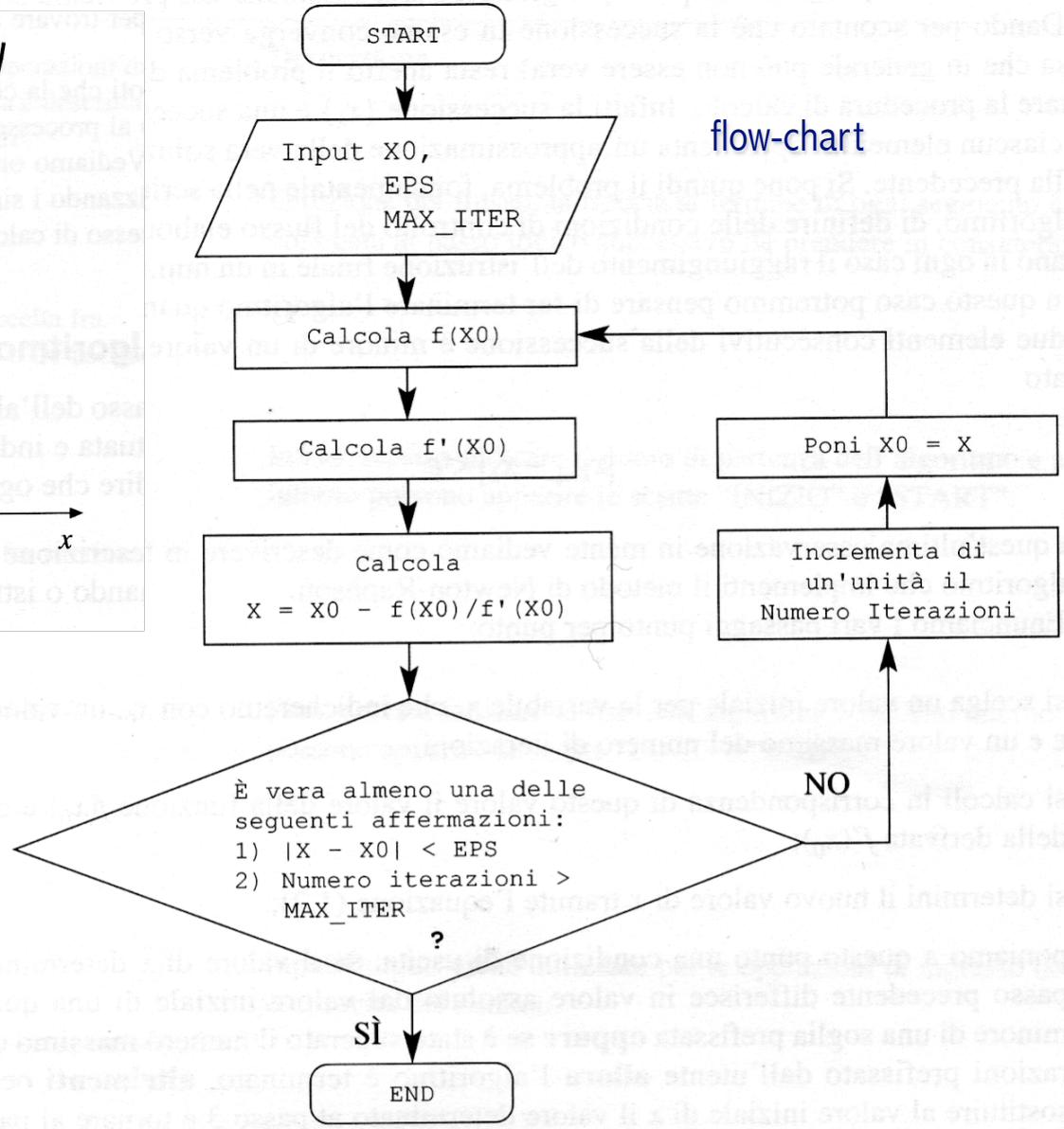
1. *Scegli un valore iniziale x_0, un valore per ε ed un numero massimo di iterazioni possibili, e sia $k=0$.*
2. *Calcola il valore iniziale $f(x_k)$ e quello di $f'(x_k)$.*
3. *Determina il nuovo valore $x = x_{k+1} = x_k - f(x_k)/f'(x_k)$.*
4. ***Se** $|x_{k+1} - x_k| < \varepsilon$ **oppure se** è stato raggiunto il numero massimo di iterazioni inizialmente prefissato **allora** l'algoritmo è terminato, **altrimenti** poni $x_k = x_{k+1}$ e torna al punto 2.*

Esempio di algoritmo



esempio grafico

flow-chart



4. Dall'algoritmo al programma

Ogni passo dell'algoritmo si compone di due parti distinte:

1. la descrizione dell'operazione che deve essere eseguita (*istruzione*);
2. uno o più elementi su cui opera l'istruzione (*operandi o dati*).

A seconda dell'azione che l'istruzione deve compiere, essa può essere:

- di tipo *operativo* (opera sui dati e produce risultati);
- di *controllo* del flusso elaborativo;
- di *assegnazione* di valori alle variabili.

Può essere quindi utile esaminare la versione VBA dell'algoritmo di Newton-Raphson, ipotizzando come funzione $y(x)=e^x-5$ (e quindi $y'(x)=e^x$), il cui zero è banalmente $x = \ln(5)$.

Esaminiamo ora il codice corrispondente.

Esempio di codice VBA: il programma Newton-Raphson

```
Option Explicit          'Obbliga alla dichiarazione delle variabili
' Dichiarazione delle variabili
Dim F As Single          ' Valore della funzione
Dim DF As Single         ' Valore della derivata della funzione
Dim Eps As Single        ' Valore della soglia di precisione
Dim Xk As Single         ' Valore di X al passo k-esimo
Dim X As Single          ' Valore di X al passo k+1-esimo
Dim Zero As Single       ' Valore finale calcolato
Const MAX_ITER = 1000     ' Massimo numero di iterazioni
Dim Nro_Iterazioni As Integer  ' Contatore n.ro di iterazioni
```

Esempio di codice VBA: il programma Newton-Raphson

```
' Assegno il valore iniziale x0 alla variabile X
X = 1.5
' Assegno il valore alla tolleranza ε
Eps = 0.0001
' Ciclo di calcolo: ad ogni iterazione viene incrementata la variabile
' Nro_Iterazioni
Nro_Iterazioni = 0
Do
    Xk = X
    F = Exp(Xk) - 5
    DF = Exp(Xk)
    X = Xk - F/DF
    Nro_Iterazioni = Nro_Iterazioni + 1
Loop Until Abs(X-Xk)<Eps Or Nro_Iterazioni>MAX_ITER
Zero = X
```

La stesura di un programma

Sulla falsariga di quanto già fatto in precedenza, si provi a scrivere una **macro** per il calcolo dell'algoritmo di Newton-Raphson digitando il codice appena visto ed avviando l'esecuzione passo-passo per verificarne la corretta applicazione.

La stesura di un programma

Da quest'esempio si deduce che la stesura di un programma è un'attività che richiede un'attenta pianificazione; in particolare occorre:

1. identificare con precisione i **dati** di **input** e quelli di **output** (tipo e natura);
2. sviluppare il procedimento descrivendone con chiarezza i vari passaggi;
3. controllare la soluzione con valori noti e provare i "casi particolari";
4. non dimenticare mai di commentare il codice e accompagnare sempre lo svolgimento del programma con una descrizione chiara ed esauriente del processo logico seguito per arrivare alla soluzione.

La stesura di un programma

E, soprattutto, pensare prima di scrivere un programma: è tutto tempo guadagnato!

Modularità dei programmi

Nelle applicazioni reali, i programmi sono strutture molto più complesse dei semplici esempi visti sinora: per gestire tale complessità si fa ricorso alla **modularità**, cioè alla *strutturazione interna delle singole parti del programma* ed al modo in cui cooperano tra di loro per conseguire l'obiettivo dell'elaborazione.

Modularità dei programmi

*Normalmente si fa uso della "**programmazione strutturata**" mediante la quale si scompone il programma complessivo in un insieme di moduli, individuando con precisione la funzione di ciascuno e la modalità di interazione.*

Modularità dei programmi

Le "routine" (Sub e Function) sono lo strumento pratico VBA per realizzare la modularità, rendendo invisibili i dettagli implementativi: in tal modo il programma nel suo complesso diventa più chiaro e ne viene notevolmente facilitata la manutenzione.

Modularità dei programmi

Ad esempio, per generalizzare l'algoritmo di Newton-Raphson rendendolo applicabile ad una generica funzione, possiamo modificare il codice introducendo le routine seguenti:

```
Function Funzione(x As Single) As Single
  Funzione = Exp(x) - 5
End Function

Function FunzioneDerivata(x As Single) As Single
  FunzioneDerivata = Exp(x)
End Function
```

Modularità dei programmi

Il ciclo di calcolo diventa:

```
...
Do
        Xk = X
        F = Funzione (Xk)
        DF = FunzioneDerivata(Xk)
        X = Xk - F/DF
        Nro_Iterazioni = Nro_Iterazioni + 1
Loop Until Abs(X-Xk)<Eps Or Nro_Iterazioni>MAX_ITER
...
```

Come si può osservare, aver suddiviso il programma in procedure distinte, oltre a garantire una maggiore manutenibilità del codice, lo rende anche estremamente più comprensibile.

Modularità dei programmi

Funzioni di interazione con l'utente

Alcune particolari funzioni VBA hanno lo scopo di consentire all'utente di **inserire** o **ricevere** dati dal programma attraverso una finestra di dialogo; si tratta delle funzioni InputBox e MsgBox, la cui sintassi base è la seguente:

```
Variabile_input = InputBox(Messaggio, Titolo, Default)
```

```
MsgBox Messaggio, Pulsanti, Titolo
```

Modularità dei programmi

Funzioni di interazione con l'utente

Messaggio è una qualsiasi stringa, che viene visualizzata nella finestra di dialogo, avente nella barra superiore la stringa Titolo.

Il valore iniziale Default viene presentato all'utente; il contenuto digitato nella finestra di dialogo viene quindi restituito nella Variabile_input.

Il parametro Pulsanti può assumere uno o più combinazioni (somma) di valori, corrispondenti ad opportuni "comportamenti" dell'interfaccia utente, quali:

Modularità dei programmi

Funzioni di interazione con l'utente

Costante	Valore	Descrizione
vbOkOnly	0	Visualizza solo il pulsante **OK**
vbOkCancel	1	Visualizza i pulsanti **OK** e **Annulla**
vbAbortRetryIgnore	2	Visualizza i pulsanti **Termina**, **Riprova** e **Ignora**
vbYesNoCancel	3	Visualizza i pulsanti **Sì**, **No** e **Annulla**
vbYesNo	4	Visualizza i pulsanti **Sì** e **No**
vbRetryCancel	5	Visualizza i pulsanti **Riprova** e **Annulla**
vbCritical	16	Visualizza l'icona di messaggio critico
vbQuestion	32	Visualizza una richiesta di avviso
vbExclamation	48	Visualizza un messaggio di avviso
vbInformation	64	Visualizza un messaggio informativo

Modularità dei programmi

Funzioni di interazione con l'utente

Modifichiamo allora il programma Newton-Raphson in modo da permettere all'utente di definire in input un valore iniziale per la variabile X e ricevere il valore di output in una finestra di dialogo di tipo MsgBox.

Modularità dei programmi

Funzioni di interazione con l'utente

Per ottenere il risultato desiderato, occorre eliminare l'istruzione di assegnazione **X** = 1.5 e sostituirla con:

```
X = InputBox("Valore iniziale di X?",
"Procedura Newton-Raphson", 1.5)
```

Modularità dei programmi

Funzioni di interazione con l'utente

Inoltre, immediatamente prima della fine (End Sub) della procedura principale, occorre inserire l'istruzione:

```
MsgBox "Lo zero della funzione è " & Str(Zero),
    vbOkOnly + vbInformation, "Procedura Newton-
    Raphson"
```

A questo punto abbiamo tutti gli elementi per scrivere e provare il funzionamento della macro NewtonRaphson!