Introduzione al linguaggio e panoramica sulle basi della programmazione

Introduzione a Visual Basic

Visual Basic è un linguaggio di programmazione sviluppato da Microsoft, basato su un ambiente di sviluppo integrato (IDE) che facilita la creazione di applicazioni Windows. È noto per la sua semplicità e per la possibilità di creare rapidamente interfacce grafiche.

Caratteristiche di Visual Basic: Un linguaggio versatile e facile da usare

1. Interfaccia Visuale Intuitiva

1. Le funzioni di progettazione dell'interfaccia sono completamente visuali.
2. All'avvio di Visual Basic, nell'area centrale dell'ambiente di sviluppo, è presente una finestra (il form) che rappresenta l'applicazione. Per inserire elementi come pulsanti, caselle di testo e etichette, è sufficiente selezionarli dalla casella degli strumenti e trascinarli direttamente sul form. Questo consente di posizionare i controlli esattamente dove si desidera. È altrettanto facile modificare la loro posizione e dimensione utilizzando il mouse.

2. Linguaggio Event-Driven

2. Visual Basic è caratterizzato anche come linguaggio event-driven.
3. Questo significa che il fulcro del linguaggio sono gli eventi, come il clic dell'utente su un pulsante, la digitazione in una casella di testo o la selezione di un comando dal menu.
4. Gli oggetti inseriti in un form di Visual Basic sono in grado di rilevare automaticamente i più comuni eventi, senza che il programmatore debba gestire esplicitamente quando accadono. Questa caratteristica rende Visual Basic facile da usare ma allo stesso tempo potente e flessibile.

I tipi di dati

Visual Basic supporta diversi tipi di dati come:

- **Integer**: Numeri interi senza parte decimale.

  VB -> Dim age As Integer = 30

- **Double:** Numeri decimali a precisazione doppia, fino a 14 cifre

  VB -> Dim price As Double=19,99

- **String:** Sequenze di caratteri testuali.

  VB -> Dim name As String = "Raffaella"

- **Boolean:** Valori logici (vero/falso).

  VB -> Dim isActive As Boolean = True

- **Date:**Valori di data e ora.

  VB -> Dim today As Date = Date.Today

- **Single :** Numeri in virgola mobile a precisazione singola, fino a 6 cifreoccupa meno memoria rispetto al tipo Double.

  VB -> Dim temperatura As Single = 36.6

- **Object:** Tipo di dato generico per oggetti.

  VB -> Dim obj As Object = New MyClass()

Ogni tipo di dato ha un ruolo specifico nella gestione delle variabili e delle operazioni.

Dichiarazioni delle Variabili

Dim *Nome* [As *Tipo*]

Nella dichiarazione delle variabili si usa la parola chiave **DIM** per indicare al compilatore che quella che si sta per definire è una variabile. Anche per le variabili il parametro **As** è opzionale se non viene specificato, la variabile verrà dichiarata di tipo **Variant,** un particolare tipo che può contenere dati di tutti i tipi È sconsigliabile definire variabili di tipo **Variant** se non espressamente necessario .Questo tipo di dato occupa molta memoria.

Esempi:

Dim Utenti As Integer

Dim Nome As String, Cognome As String

Riepilogo dei principali tipi di dato

- **Integer** da -32.768 a 32.767
- **Single** da -3.402823E38 a -1.401298E45 (4 byte)
- **Double** da -4.94065645841247E-324 a 4.94065645841247E-324
- **Boolean** True o False
- **String** Da 0 a, circa, 2 miliardi

Dichiarazioni delle costanti

In Visual Basic, le costanti sono definite con la parola chiave **Const.** Una costante è un valore che non può essere modificato durante l'esecuzione del programma.

Const Nome **As Tipo** = valore

1. **Nome**: È il nome che viene assegnato alla costante.(Es. PI, nome)
2. **Tipo**: Indica il tipo di dato della costante (opzionale). Se non specificato, il compilatore lo determina automaticamente.
3. **Valore:** È il valore vero e proprio della costante, che non può essere cambiato dopo la sua assegnazione.

Regole per i nomi delle costanti:

- Non possono essere più lunghi di 40 caratteri.
- Non possono iniziare con un numero.
- Non possono contenere spazi o caratteri speciali come &, #, @, ecc.
- I nomi non fanno distinzione tra maiuscole e minuscole (non sono case sensitive).

Esempi:

Const PI **As Double** = 3.14

Const nome **As String** = "Raffaella"

Principali Operatori in Visual Basic

Operatore	Descrizione	Esempio	risultato
+	addizione	3 + 2	5
-	sottrazionecrea una slide su i principali operatori in visual basic	3 - 2	1
*	moltiplicazione	3 * 2	6
/	divisione	3 / 2	1.5
^	potenza	3^2	9
Mod	Resto della divisione	3 Mod 2	1
\	Divisione intera	3 \ 2	1

Casella degli Strumenti di Visual Basic

- **Button (Pulsante)**

  Esegue un'azione quando cliccato.

  Esempio: Button1_Click

- **TextBox (Casella di testo)**

  Consente all'utente di inserire o visualizzare del testo.

  Esempio: TextBox1.Text

- **Label (Etichetta)**

  Visualizza un testo che non può essere modificato dall'utente.

  Esempio: Label1.Text

- **ComboBox (Elenco a discesa)**

  Fornisce un menu a discesa con una lista di opzioni.

  Esempio: ComboBox1.SelectedItem

- **CheckBox (Casella di controllo)**

  Consente all'utente di selezionare o deselezionare un'opzione.

  Esempio: CheckBox1.Checked

- **RadioButton (Pulsante di opzione)**

  Permette di scegliere tra opzioni mutualmente esclusive.

  Esempio: RadioButton1.Checked

- **ListBox (Casella di selezione)**

  Visualizza una lista di elementi che l'utente può selezionare.

  Esempio: ListBox1.SelectedIndex

- **PictureBox (Casella immagine)**

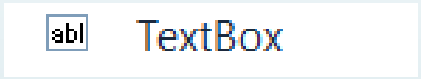
  Consente di visualizzare immagini all'interno di un'applicazione.

  Esempio: PictureBox1.Image = Image.FromFile("path_to_image")

Casella degli Strumenti di Visual Basic

Label(etichetta) 

Consente di visualizzare sul form un breve testo. Viene usato per inserire un'intestazione generale (descrizione dell'applicazione) oppure la descrizione delle funzioni di un altro controllo. Il testo visualizzato da un'etichetta non può essere modificato in fase di runtime.

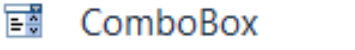
TextBox (casella di testo) 

Viene utilizzata per inserire o visualizzare dati numerici o testuali. La TextBox utilizza solo dati testuali e per questo motivo che ci avvaliamo di alcune funzioni, per le trasformazioni di dati testuali in valori numerici e viceversa.

CommandButton (pulsante di comando) 

Viene utilizzato per avviare, interrompere o concludere un'elaborazione. Solitamente attivato con un clic del mouse.

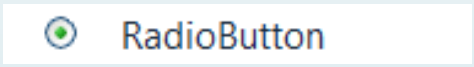
Casella degli Strumenti di Visual Basic

ComboBox (Elenco a Discesa) 

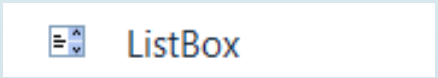
Fornisce un elenco a discesa con un insieme di opzioni per l'utente. Permette di selezionare un elemento da una lista o inserire un nuovo valore.

CheckBox (Casella di Controllo) 

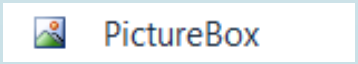
Consente all'utente di selezionare o deselezionare un'opzione. Utilizzato per scelte multiple o per attivare/disattivare una funzionalità.

RadioButton (Pulsante di Opzione) 

Permette di scegliere tra opzioni mutualmente esclusive (solo una opzione può essere selezionata in un gruppo di RadioButton).

ListBox (Casella di Selezione) 

Visualizza una lista di elementi tra cui l'utente può fare una selezione. Può essere configurato per permettere una selezione singola o multipla.

PictureBox (Casella Immagine) 

Visualizza un'immagine all'interno del form. È utile per visualizzare immagini dinamiche o statiche.

Oggetto textBox – Applicazion delle funzioni Val e CStr

Esempio di utilizzo di TextBox (casella di testo)

1. **Assegnazione di un valore alla variabile:**
   - **Nome = Txt_nome.Text**
     La variabile Nome (di tipo stringa) assume il valore digitato nella TextBox Txt_nome.Text.
2. **Conversione di un valore in numerico:**
   - **N = Val(Txt_nome.Text)**
     La variabile N (di tipo numerico) assume il valore digitato nella TextBox di nome Txt_nome.Text, trasformato (dalla funzione Val) in numerico.
3. **Conversione di un numero in stringa:** **Cstr (**Convert to String**)**
   - **Txt_N.Text = CStr(N)**
     La variabile N (di tipo numerico) è trasformata in una stringa (dalla funzione CStr) per poter essere visualizzata nella casella della TextBox di nome Txt_N.Text.

Proprietà Enable e Focus

1. **Proprietà Enable:**
   - Enable (True/False) permette di **attivare/disattivare** la scrittura in una TextBox.
     - **Txt_Nome.Enabled = False**
       Questa istruzione **disabilita** la possibilità di scrivere nella TextBox Txt_Nome.Text.
       Per default, questa proprietà è settata su True.
2. **Proprietà Focus:**
   - Focus posiziona il cursore in una casella TextBox specificata.
     - **Txt_Nome.Focus()**
       Questa istruzione **posiziona** il cursore nella casella di testo Txt_Nome.Text.

Oggetto textBox

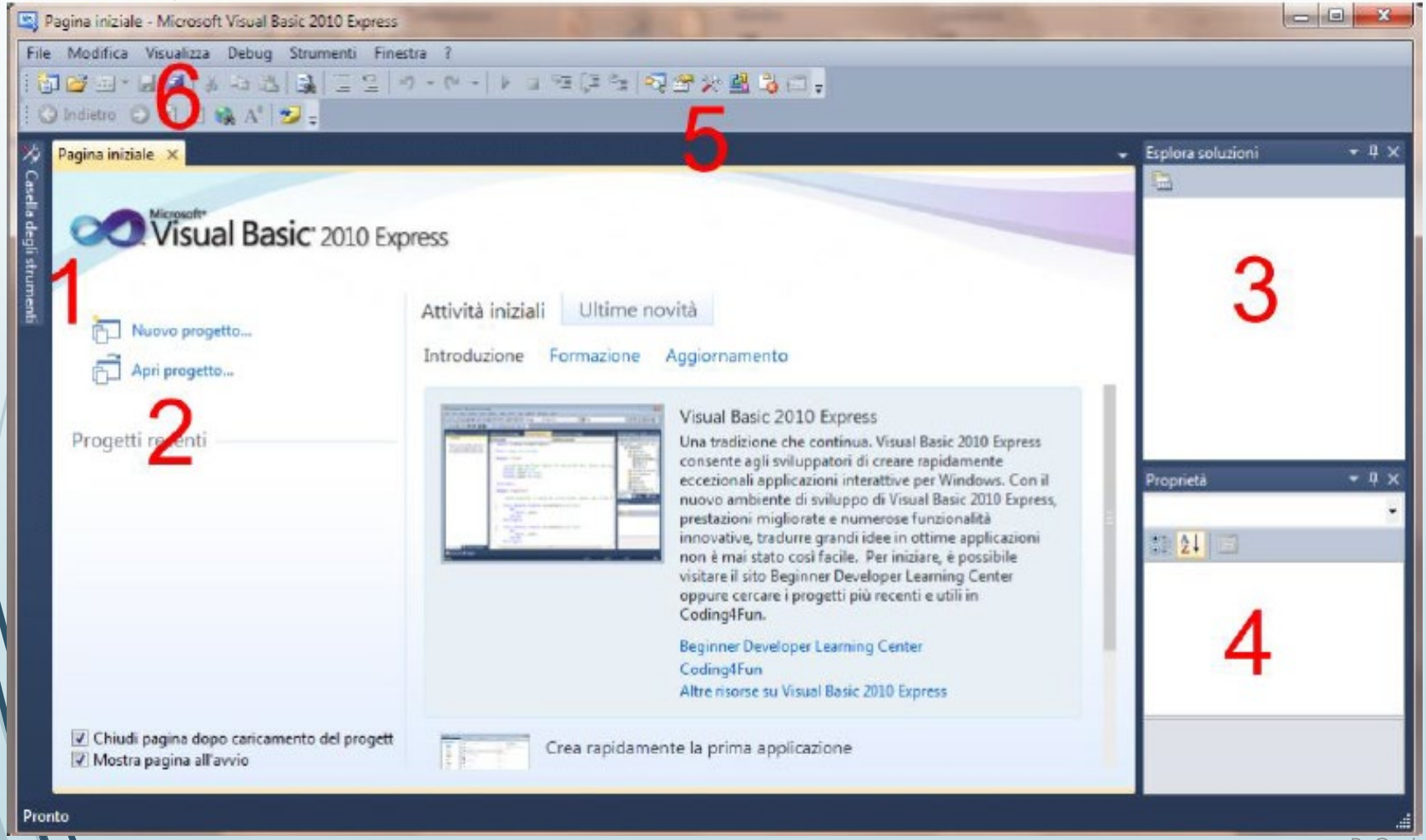
Proprietà Clear e Visible

1. **Proprietà Clear**
   - **Funzione**: Pulisce una casella TextBox specificata.
     - **Esempio**: Txt_Nome.Clear()
       Questa istruzione pulisce il contenuto della casella di testo Txt_Nome.Text.

2. **Proprietà Visible**
   - **Funzione**: Rende visibile (TRUE) o meno (FALSE) una casella TextBox specificata.
     - **Esempio**: Txt_Nome.Visible = (TRUE/FALSE)
       Questa istruzione rende visibile o nascosta la casella di testo Txt_Nome.Text a seconda del valore passato (TRUE o FALSE).

AMBIENTE DI PROGETTAZIONE DI VB

Per prima cosa per iniziare a progettare in VB apriamo **Microsoft Visual Basic 2010 Express**. Compare così la pagina iniziale dell'ambiente di progettazione di VB. In termini tecnici, questo è l'**IDE** di VB (**Integrated Development Environment,** ambiente integrato di sviluppo). Ci troviamo nella **Pagina iniziale**.



AMBIENTE DI PROGETTAZIONE DI VB

1. **Casella degli Strumenti**

   La Casella degli Strumenti è una casella a scomparsa che si apre quando ci si passa sopra con il mouse e si chiude quando il mouse si allontana. È possibile fissarla per renderla sempre visibile premendo l'icona con la puntina. La casella contiene gli strumenti necessari per progettare, ma per ora è vuota poiché non sono stati avviati progetti.

2. **Link per creare o aprire un progetto**

   Sulla destra della Casella degli Strumenti, troviamo due link: uno per creare un nuovo progetto e l'altro per aprire un progetto preesistente. Nella sezione "Progetti recenti" vengono elencati i progetti avviati di recente.

3. **Esplora Soluzioni e Finestra Proprietà**

   - **Esplora Soluzioni**: Questa finestra mostra tutte le parti del progetto in corso, permettendo di accedere e modificare ciascuna di esse singolarmente.
   - **Finestra Proprietà**: Finestra di uso frequente che visualizza le proprietà di ogni controllo (pulsanti, caselle di testo, immagini, ecc.) presenti nel progetto. Da questa finestra è possibile modificare le proprietà degli oggetti.

4. **Striscia dei Pulsanti**

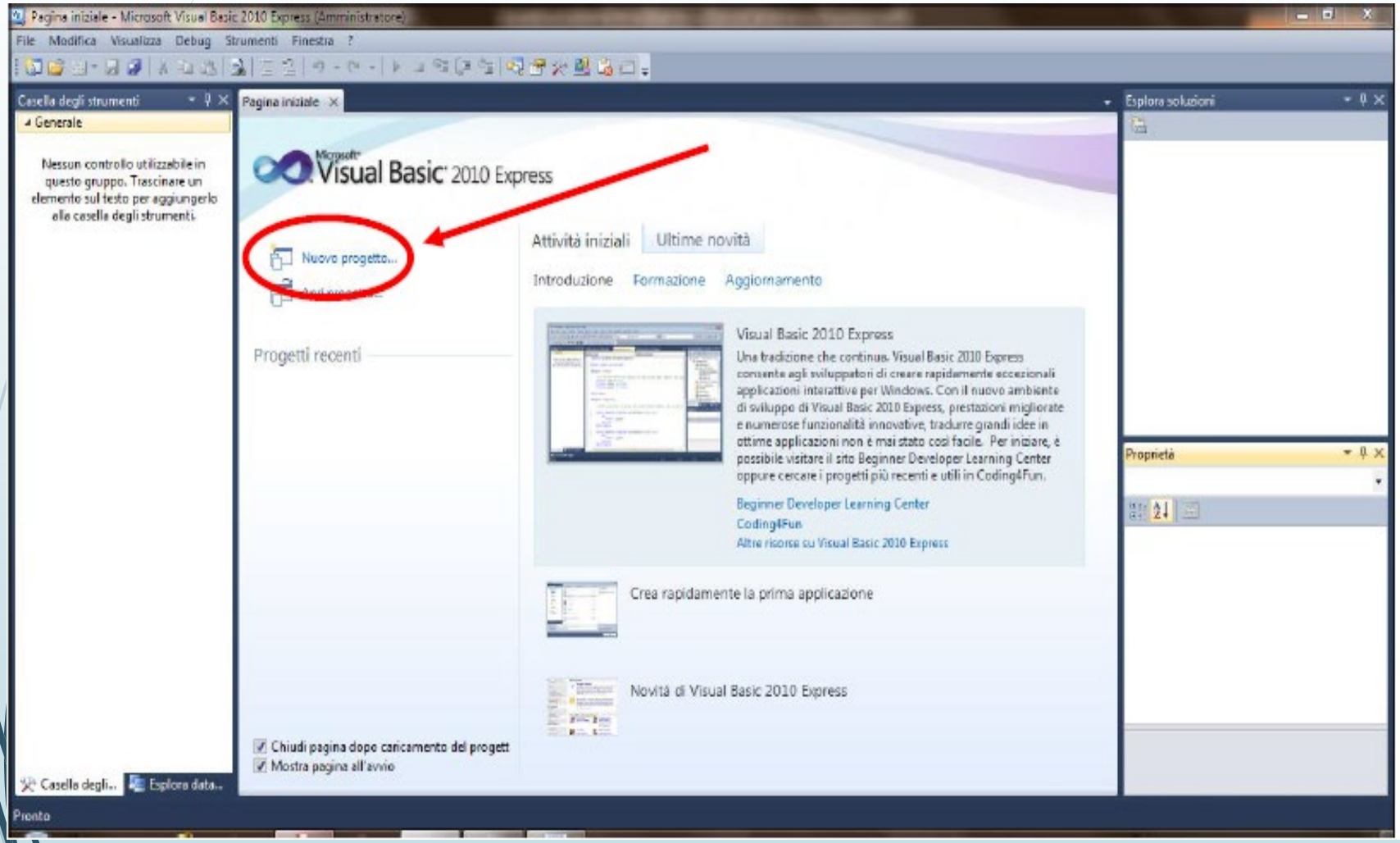
   Nella parte superiore della finestra si trovano tre icone per aprire rispettivamente la Casella degli Strumenti, la finestra Esplora Soluzioni e la Finestra Proprietà.

5. **Menu Visualizza**

   Cliccando sul menu "Visualizza" e successivamente su "Altre finestre", è possibile accedere ad altre finestre necessarie durante la progettazione, come la Casella degli Strumenti, la Finestra Esplora Soluzioni e la Finestra Proprietà. Inoltre, la Finestra Proprietà è accessibile premendo il tasto F4.

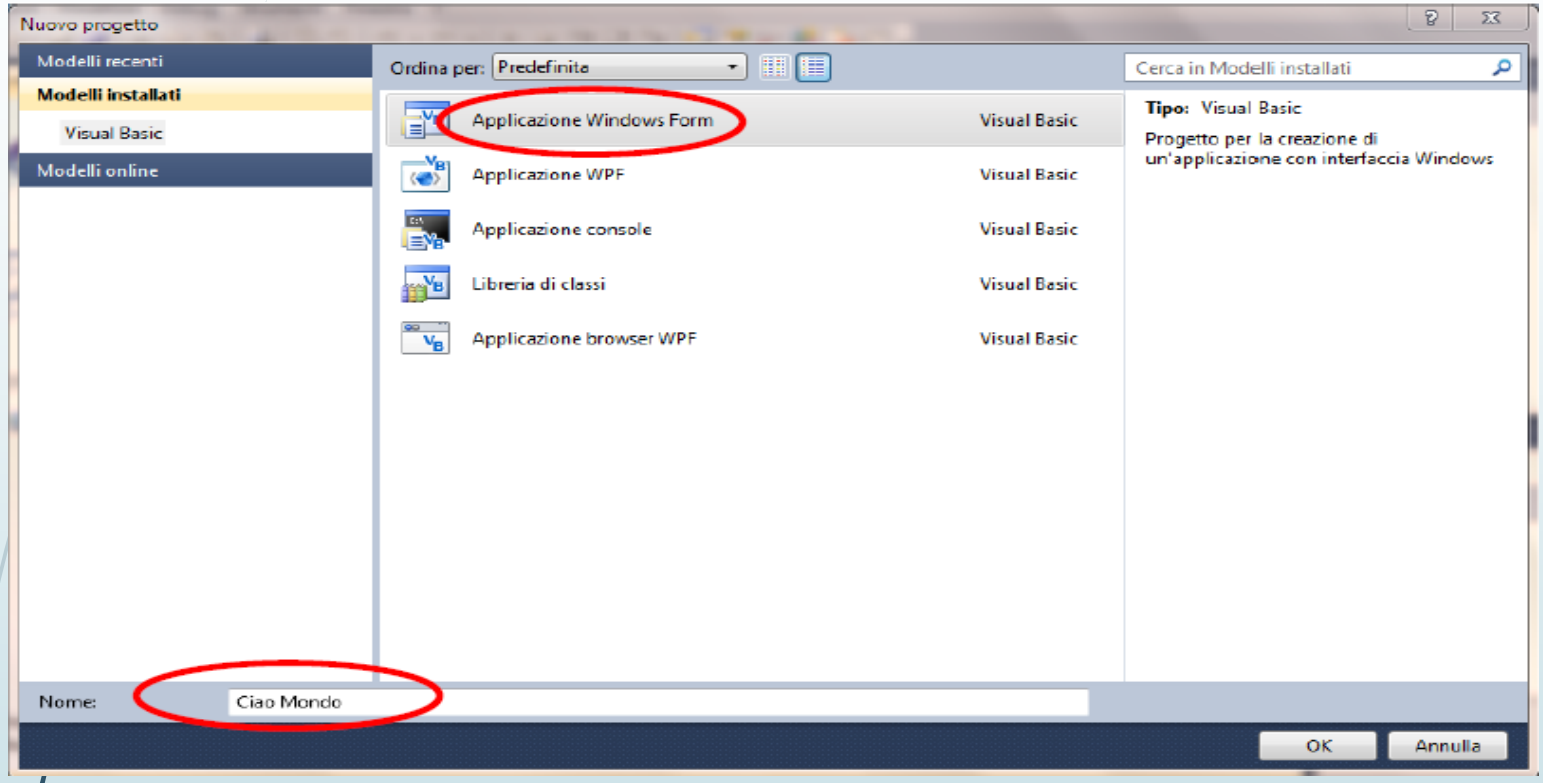
AMBIENTE DI PROGETTAZIONE DI VB

Per prendere confidenza con l'ambiente di progettazione e le sue finestre, avviamo la creazione di un **Nuovo progetto**:



AMBIENTE DI PROGETTAZIONE DI VB

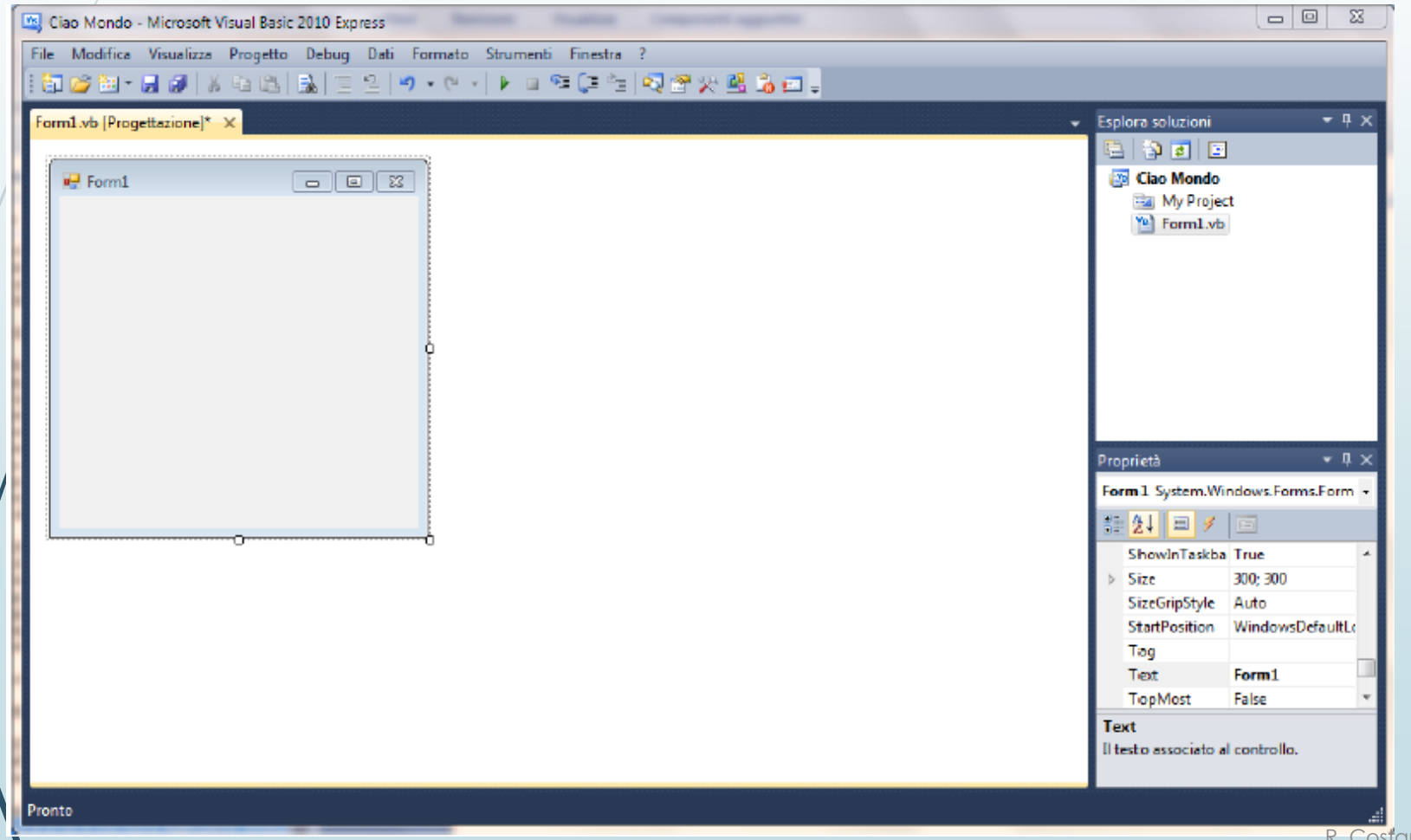
Compare la finestra **Nuovo progetto**, visibile nella prossima figura, in cui ci viene chiesto di specificare quale tipo di progetto intendiamo avviare.



Come si vede nella colonna centrale di questa finestra, con VB si possono creare progetti di diverso tipo: programmi destinati a internet, programmi destinati a operare dietro le quinte e a rimanere invisibili, oppure classi di oggetti da utilizzare in altri programmi. Una **Applicazione Windows Form**, prima opzione della lista, è il tipo di programma famigliare a ognuno di noi: è un programma che opera usando una finestra sul monitor e oggetti ormai famigliari come pulsanti, testi, immagini e audio.

AMBIENTE DI PROGETTAZIONE DI VB

Non clicchiamo subito la voce **Applicazione Windows Form**. Prima, nella linea in basso, scriviamo il nome del progetto che intendiamo realizzare: lo chiamiamo **Ciao Mondo**, e vedremo tra poco perché. Ora possiamo fare *clic* su OK. In questo modo passiamo al linguaggio VB l'informazione che vogliamo creare una nuova **Applicazione Windows Form** che si chiamerà **Ciao Mondo.**



AMBIENTE DI PROGETTAZIONE DI VB

Facciamo un doppio *clic* sull'oggetto **Button**: vediamo che un pulsante compare all'interno del nostro Form1, dove è possibile cliccarlo per collocarlo dove preferiamo. Collochiamo successivamente un oggetto **Label**, come nella figura a lato.

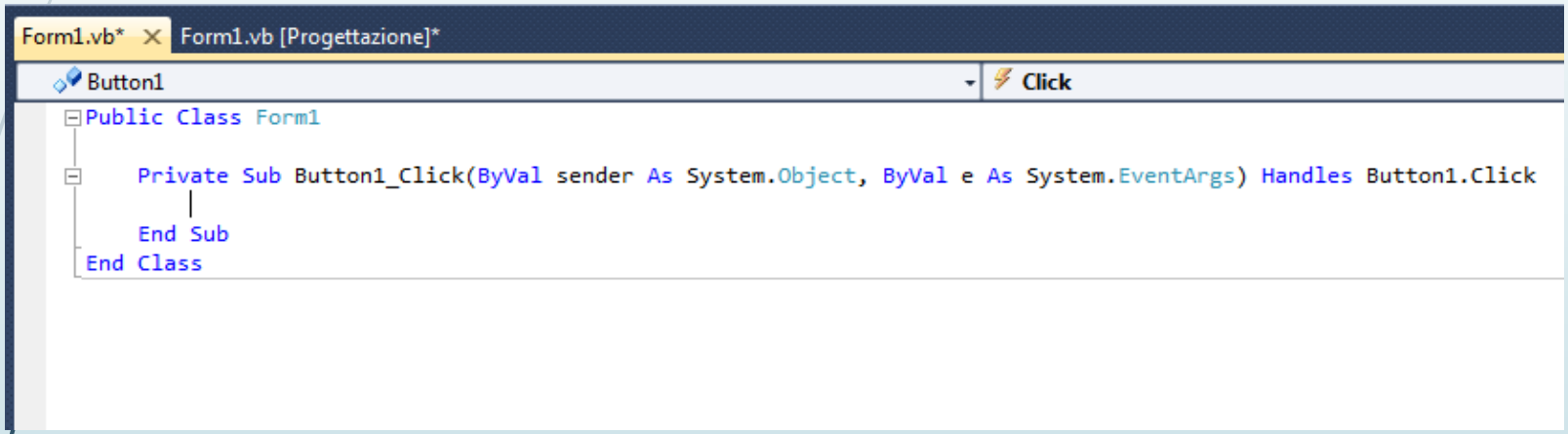
Con queste operazioni abbiamo terminato la creazione dell'interfaccia grafica del progetto; mancano ora le istruzioni per fare funzionare i due oggetti contenuti nel Form.

A questo scopo, dobbiamo inserire nel progetto queste due istruzioni:

- quando avverti il *clic* del mouse sull'oggetto Button1
- cambia il testo della Label1 in **"Ciao 4H".**

AMBIENTE DI PROGETTAZIONE DI VB

Queste istruzioni si scrivono in una nuova finestra, che non abbiamo ancora visto: è la **Finestra del Codice**. Vi sono vari modi per accedervi; per ora adottiamo il modo più diretto: facciamo un doppio *clic* sull'oggetto Button1.
Abbiamo così accesso alla **Finestra del Codice**, che è di importanza fondamentale perché è qui che vengono scritte le istruzioni per il programma (cioè cosa deve fare il computer quando accade un evento nel programma in esecuzione).



Notiamo le due scritte che si vedono nella linea della intestazione di questa finestra:

- Form1.vb* e
- Form1.vb [Progettazione]*

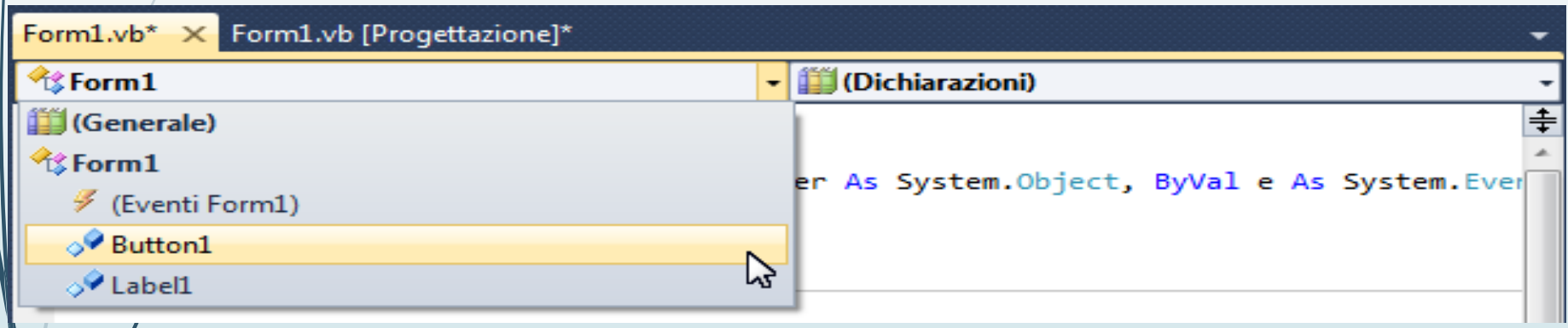
Si tratta dei titoli di due schede diverse:

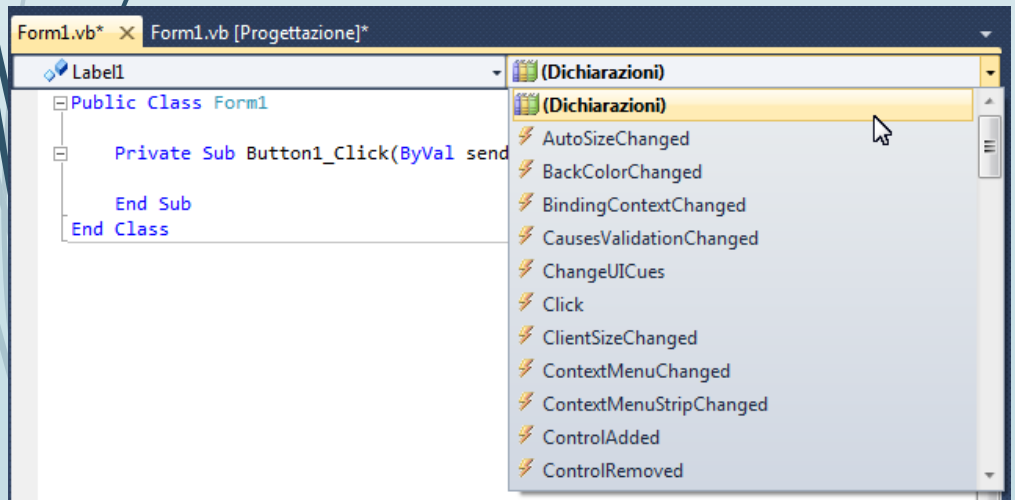
- una scheda in cui si vede il codice del progetto;
- una scheda in cui si vede l'interfaccia grafica del progetto.

AMBIENTE DI PROGETTAZIONE DI VB

Ora, nella scheda intitolata **Form1.vb**, cioè nella **Finestra del Codice**, sotto la linea della intestazione notiamo due menu che si aprono a tendina:

- Nel menu di **sinistra** vediamo l'elenco degli **oggetti** presenti nel Form1, e ritroviamo qui gli oggetti Button1 e Label1.
- Nel menu di **destra** vediamo l'elenco degli **eventi** che ognuno di questi oggetti è in grado di riconoscere e di gestire.





Notiamo che ogni procedura inizia con le parole Private Sub… e termina con la parole End Sub. Tra **Private Sub**… e **End Sub** vanno scritti i comandi che il programmatore vuole fare eseguire al computer.

AMBIENTE DI PROGETTAZIONE DI VB

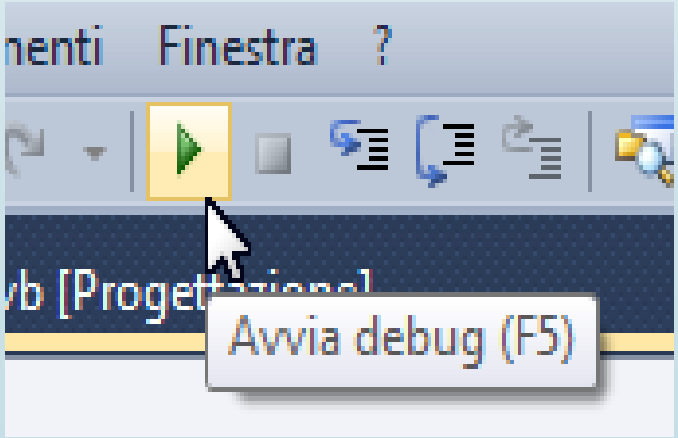
Riprendiamo lo schema delle istruzioni di questo programma:

- quando avverti il *clic* del mouse sull'oggetto Button1
- cambia il testo della Label1 in "Ciao Mondo".

Inseriamo in questa procedura, dedicata all'evento *clic* del mouse sul Button1, le istruzioni affinché nella Label1 compaia la scritta "Ciao Mondo":

```
Private Sub Button1_Click(sender As System.Object, e As
System.EventArgs) Handles Button1.Click

    Label1.Text = "Ciao Mondo"

End Sub
```

Mandiamo in esecuzione il nostro progetto cliccando il tasto **F5**, oppure cliccando l'icona con la freccina verde nella striscia dei pulsanti, oppure ancora cliccando, in alto nella striscia dei menu, il menu **Debug / Avvia debug**.

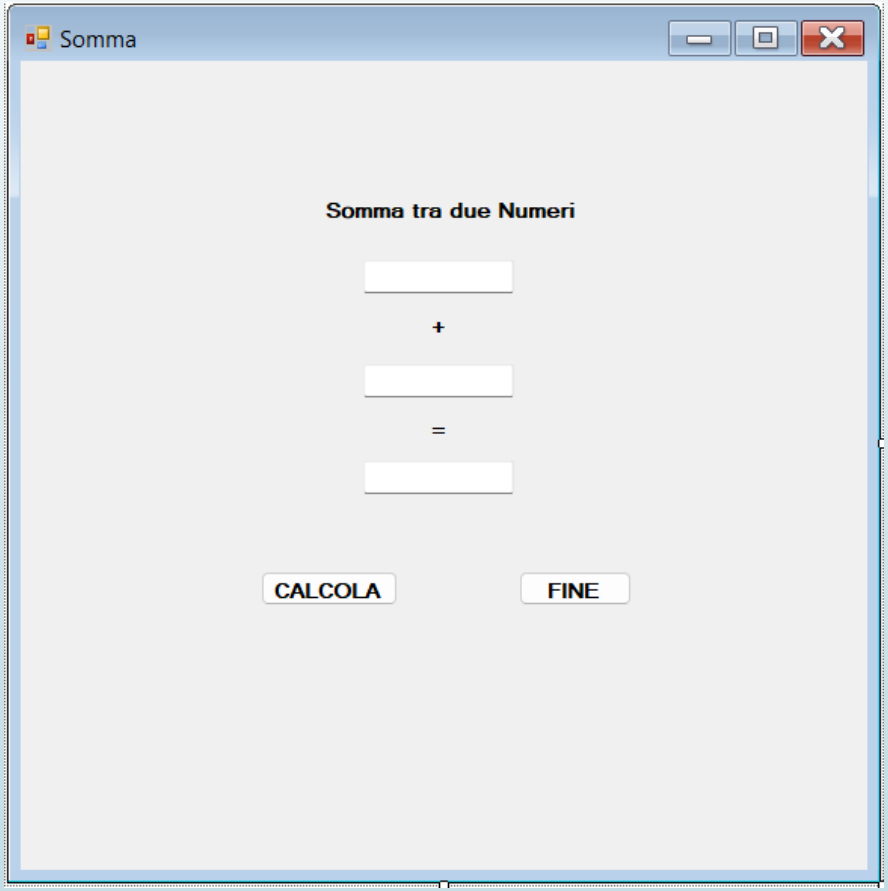


Visual Basic

Applicazione delle funzioni Val e CStr all'oggetto TextBox, un primo esempio:

Somma tra due numeri interi

IL FORM



Applicazione delle funzioni **Val** e **CStr(**Convert to String**)** all'oggetto TextBox, un primo esempio:

IL CODICE

```
Public Class Form1

    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label1.Click

    End Sub

    Private Sub Label3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label3.Click

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles calcola.Click
        Dim a As Integer
        Dim b As Integer
        Dim totale As Integer
        a = Val(testo1.Text)
        b = Val(testo2.Text)
        totale = a + b
        ' visualizza risultato con TextBox
        testo3.Text = CStr(totale)
    End Sub

    Private Sub esci_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles esci.Click
        End
    End Sub
End Class
```

Visual Basic

Operatori di confronto e connettivi logici

Di seguito sono riportati gli operatori di confronto ed i connettivi logici utilizzati da Visual Basic

OPERATORI DI CONFRONTO

= <= >= <>

CONNETTIVI LOGICI

And Or Not

Operatore logico	Esempio	Traduzione
And	If x > y AND x > z	Se la prima comparazione è vera E se la seconda comparazione è vera...
Or	If x > y OR x > z	Se la prima comparazione è vera O se la seconda comparazione è vera...
XOR		Se la prima comparazione è diversa dalla seconda comparazione...

Visual Basic

Il costrutto SELEZIONE (unaria)

IF - THEN

La **Pseudocodifica** di una struttura di selezione unaria ha la seguente forma:

```
...
SE (<condizione>)
    ALLORA
        <istruzioni>
FINE SE
```

Nel linguaggio **Visual Basic** la struttura di selezione sarà tradotta come segue:

```
...
If <condizione>  Then
    <istruzioni>
End If
```

Visual Basic

il costrutto SELEZIONE (binaria)

IF – THEN - ELSE

La **Pseudocodifica** di una struttura di selezione completa (o binaria) ha la seguente forma:

```
...
SE (<condizione>)
    ALLORA
        <istruzioni>
    ALTRIMENTI
        <istruzioni>
FINE SE
```

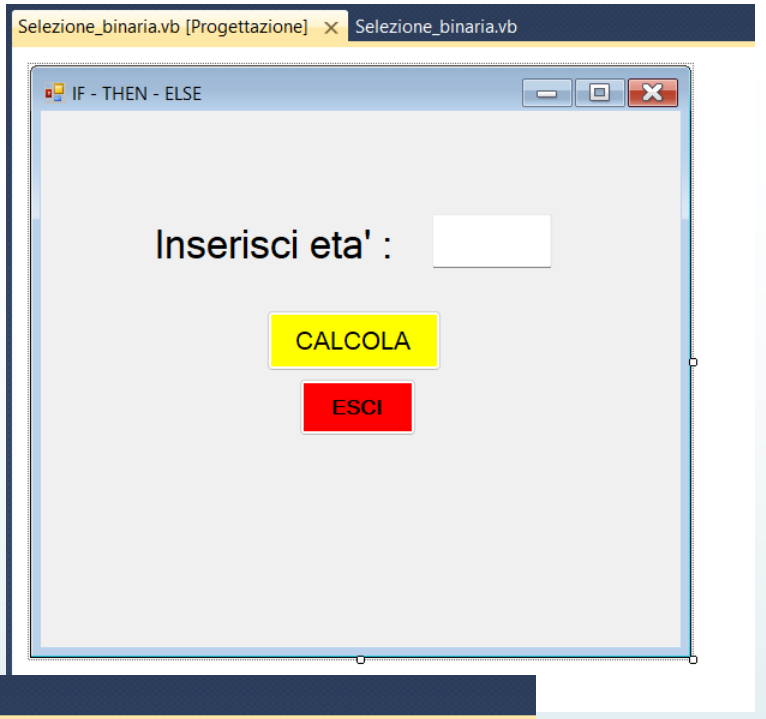
Nel linguaggio **Visual Basic** la struttura di selezione sarà tradotta come segue:

```
...
If <condizione> Then
    <istruzioni>
Else
    <istruzioni>
End If
```

Il costrutto SELEZIONE (binaria)
IF – THEN - ELSE

Dato età in input, visualizzare se si tratta di minorenne o maggiorenne!

FORM



Form1.vb* | Form1.vb [Progettazione]*

CALCOLA | Click

```
Public Class Form1

    Private Sub ESCI_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ESCI.Click
        End
    End Sub

    Private Sub CALCOLA_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CALCOLA.Click
        Dim eta As Integer
        eta = Val(TextBox1.Text)
        If (eta >= 18) Then
            MsgBox("Sei Maggiorenne", , "Risultato")
        Else
            MsgBox("Sei Minorenne", , "Risultato")

        End If
        'Primo parametro: il messaggio da mostrare
        'Secondo parametro (facoltativo): il tipo di pulsanti e icone
        'Terzo parametro: il titolo della finestra di dialogo
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        TextBox1.Clear()
    End Sub
End Class
```

Visual Basic

il costrutto selezione multipla

SELECT CASE

La **Pseudocodifica** di una struttura di selezione multipla ha la seguente forma:

...
NEL CASO CHE (<var>|<espressione>) *SIA*
< valore_1> : <istruzioni>
< valore_2> : <istruzioni>
.....
< valore_n> : <istruzioni>
[**ALTRIMENTI** : <istruzioni>]
FINE CASO

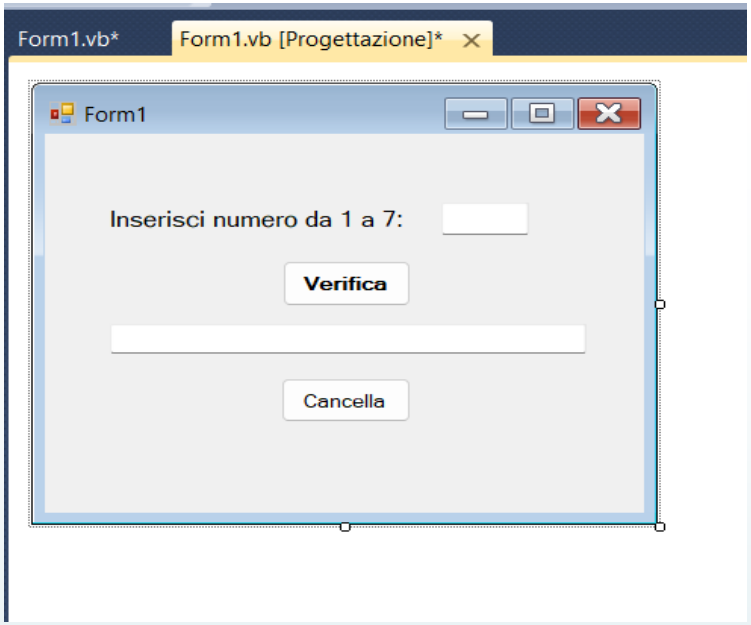
Nel linguaggio **Visual Basic** la struttura di selezione multipla sarà tradotta come segue:

...
Select Case <var>|<espressione>
Case *<valore_1>:*
<istruzioni>
.....
Case *<valore_n>:*
<istruzioni>
[Case Else
<istruzioni da eseguire se tutti i confronti falliscono >]
End Select

il costrutto selezione multipla

SELECT CASE

Determinare il giorno della settimana in base a un numero inserito dall'utente.



Form1.vb* | Form1.vb [Progettazione]*

Form1 | (Dichiarazioni)

```
Public Class Form1
    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label1.Click

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim numero As Integer
        numero = Val(TextBox1.Text)

        Select Case numero
            Case 1
                TextBox2.Text = "Lunedì"
            Case 2
                TextBox2.Text = "Martedì"
            Case 3
                TextBox2.Text = "Mercoledì"
            Case 4
                TextBox2.Text = "Giovedì"
            Case 5
                TextBox2.Text = "Venerdì"
            Case 6
                TextBox2.Text = "Sabato"
            Case 7
                TextBox2.Text = "Domenica"
            Case Else
                TextBox2.Text = "Numero non valido. Inserisci un numero da 1 a 7."
        End Select

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        TextBox1.Clear()
        TextBox2.Clear()
    End Sub
```

83 %

Il ciclo Do…..Loop

15 Il ciclo *Do... Loop*

Oltre al costrutto iterativo *For... Next*, Visual Basic dispone di un altro tipo di ciclo che termina in base ad una specifica condizione dettagliatamente imposta: il ciclo **Do... Loop**. Questo tipo di struttura consente di non specificare a priori il numero di volte che dovrà essere ripetuta la sequenza di istruzioni contenuta al suo interno (il corpo del ciclo).

Il ciclo *Do... Loop* presenta quattro sintassi diverse:

```
1) Do
       <Corpo del ciclo>
   Loop Until <Condizione>
2) Do
       <Corpo del ciclo>
   Loop While <Condizione>
3) Do Until <Condizione>
       <Corpo del ciclo>
   Loop
4) Do While <Condizione>
       <Corpo del ciclo>
   Loop
```

Analizziamo le differenze. La prima cosa che si nota è che le strutture (1) e (2) verificano la condizione *alla fine* del ciclo (**iterazioni postcondizionali**), mentre le strutture (3) e (4) verificano la condizione *all'ingresso*, cioè prima di eseguire le istruzioni presenti nel corpo del ciclo (**iterazioni precondizionali**). Ciò vuol dire che i cicli relativi alle strutture (3) e (4) potrebbero anche non essere mai eseguiti se la condizione testata non risultasse valida.

Le strutture (1) e (2), invece, differiscono unicamente per la presenza delle istruzioni **Until** e **While**, che si traducono, rispettivamente, con *fino a che* e con *mentre*. Ciò significa che il ciclo (1) sarà eseguito almeno una volta e, comunque, fintanto che la condizione si mantiene falsa. Nel momento in cui la condizione diventa vera, il controllo passa all'istruzione successiva all'istruzione **Loop**.

Diversamente, il ciclo (2) sarà eseguito fintanto che la condizione si mantiene vera. Nel momento in cui la condizione diventa falsa, il procedimento di calcolo prosegue con le istruzioni successive all'istruzione *Loop*.

Visual Basic

iterazioni PRE- condizionali

Do – While - Loop

La **Pseudocodifica** di una struttura di iterazione PRE-condizionale **MENTRE** ha la seguente forma:

```
...
MENTRE (<condizione>) ESEGUI
    <istruzioni>
FINE MENTRE
```

Nel linguaggio Visual Basic la struttura di iterazione PRE-condizionale **MENTRE** sarà tradotta come segue:

```
...
Do While <condizione>
    <istruzioni>
Loop
```

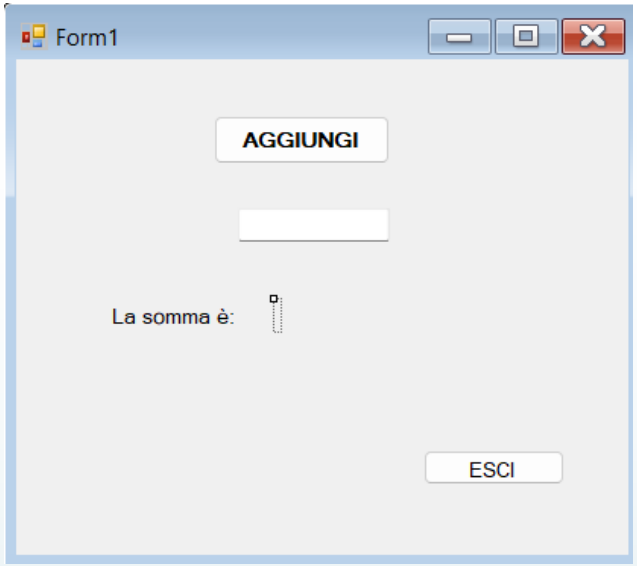
Il processo di iterazione:

- avviene per condizione **VERA**
- si arresta quando la condizione iniziale diventa **FALSA**.

Iterazioni PRE- condizionali

Do – While - Loop

Calcolare la somma di numeri inseriti dall'utente. Il programma chiederà all'utente di inserire numeri e sommarli fino a quando l'utente inserisce 0 (che farà terminare il ciclo).




```
Public Class Form1
    Dim somma As Integer = 0

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

       Dim numero As Integer

        ' Ottieni il numero dall'utente
        numero = Val(TextBox1.Text)

        ' Esegui il ciclo finché numero non è 0
        Do While numero <> 0
            ' Aggiungi il numero alla somma
            somma = somma + numero

            ' Pulisce la TextBox per il nuovo inserimento
            TextBox1.Clear()
            TextBox1.Focus()

            ' Con Exit Sub,l'esecuzione del codice viene interrotta immediatamente nel momento
            'in cui raggiungi quella istruzione. Dopo aver letto il numero e aggiunto alla somma,
            'il metodo viene interrotto con Exit Sub.
            'Questo consente al ciclo di "fermarsi temporaneamente", dando all'utente il controllo
            'per inserire un nuovo numero nella TextBox e cliccare di nuovo il pulsante.
            Exit Sub
        Loop

        ' Se l'utente inserisce 0, visualizza la somma
        If numero = 0 Then
            Label2.Text = somma
            somma = 0 ' Resetta la somma per nuovi calcoli
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class
```

Visual Basic

Iterazioni PRE-condizionali

Do – Until- Loop

I cicli di comandi retti da Do Until… Loop (= esegui fino a quando… ripeti) sono una variante dei cicli Do While… Loop, dai quali si differenziano per queste caratteristiche:

- un ciclo Do While… Loop si ripete **MENTRE** esiste una determinata condizione;
- un ciclo Do Until… Loop si ripete **FINCHE'** è raggiunta una determinata condizione.

Nel linguaggio Visual Basic la struttura di iterazione sarà tradotta come segue:

```
...
Do Until <condizione>
    <istruzioni>
Loop
```

Il processo di iterazione:

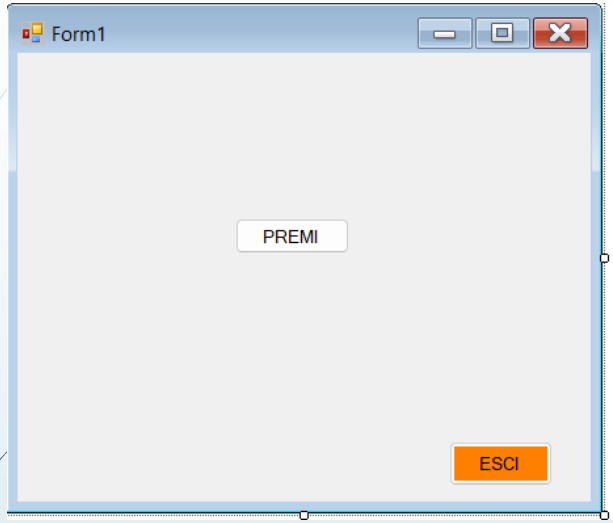
-avviene per condizione **FALSA**

-si arresta quando la condizione iniziale diventa **VERA**.

Visual Basic

iterazioni PRE-condizionali

Do – Until- Loop



rm1.vb | Form1.vb [Progettazione]

Form1 | (Dichiarazioni)

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        'crea la variabile Numero destinata a contenere numeri interi e le assegna il valore 1:
        Dim Numero As Integer = 1
        ' inizia il ciclo di comandi retto da Do Until:
        Do Until (Numero = 10)
            ' esegui questi comandi sino a quando la variabile Numero è uguale a 10:
            ' visualizza un messaggio con il valore della variabile Numero:
            MsgBox("Devo contare fino a 10. Sono arrivato a " & Numero)
            ' aumenta la variabile Numero di una unità:
            Numero = Numero + 1
            ' ripeti il ciclo:
        Loop
        ' quando il ciclo è finito, visualizza questo messaggio:
        MsgBox("Dovevo contare fino a 10. Ho finito!")
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class
```

Il programma visualizza in un messaggio un numero progressivo, e continua a ripetere il comando finché il numero arriva a 10.

Visual Basic
iterazioni POST-condizionali
Do – Loop Until

La **pseudocodifica** di una struttura di iterazione postcondizionale FINCHE' ha la seguente forma:

```
...
RIPETI
        <istruzioni>
FINCHE' (<condizione>)
```

Nel linguaggio **Visual Basic** la struttura di iterazione post condizionale sarà tradotta come segue:

```
...
Do
        <istruzioni>
Loop Until <condizione>
```

Il processo di iterazione:

-avviene per condizione **FALSA**

-si arresta quando la condizione iniziale diventa **VERA**.

Visual Basic

iterazioni enumerative For…Next

La pseudocodifica di una struttura di iterazione enumerativa PER ha la seguente forma:

…

PER <indice> ← <inizio> [INDIETRO] A <fine> ESEGUI
<istruzioni>
INCREMENTA <indice> [DECREMENTA <indice>]
FINE PER

Nel linguaggio Visual Basic la struttura di iterazione enumerativa sarà tradotta come segue:

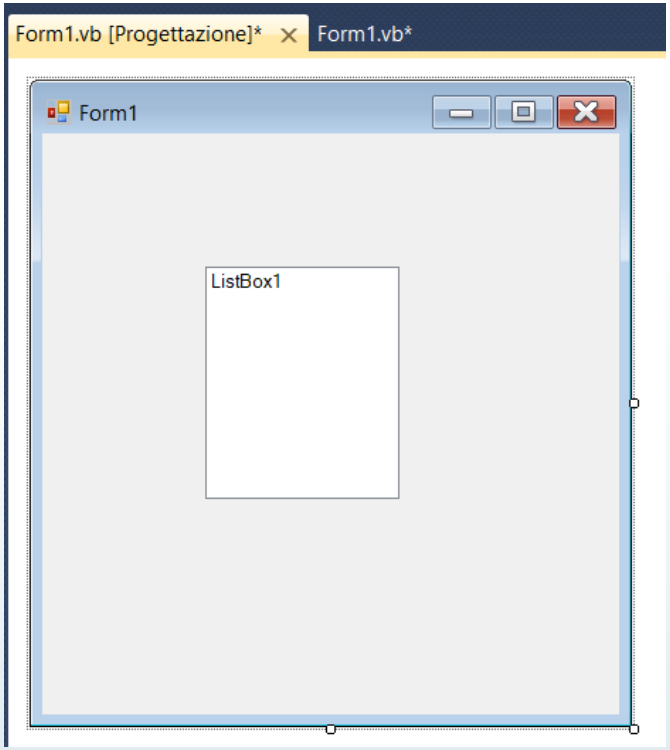
…

For <indice> = <inizio> To <fine>
<istruzioni>
[Step incremento]
Next <indice>

La variabile incremento può contenere valori sia positivi che negativi.

I cicli retti da **For… Next** (= per… vai al prossimo), si basano su un contatore, che è una variabile creata dal programmatore, che aumenta (o diminuisce) a ogni ripetizione del ciclo, sino a raggiungere il limite stabilito dal programmatore.
I comandi scritti tra la riga iniziale del ciclo For… e la riga finale Next vengono ripetuti a ogni passaggio, sino alla conclusione del ciclo. Quando il contatore ha raggiunto il limite stabilito dal programmatore, il ciclo termina e il programma prosegue per la sua strada.

Visual Basic
iterazioni enumerativa For….Next



Visualizza all'interno di una ListBox i numeri da 1 a 100.

Form1.vb [Progettazione]* | Form1.vb*

(Generale) | (Dichiarazioni)

```
Public Class Form1

    Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

        Dim contatore As Integer
        For contatore = 1 To 100
            ListBox1.Items.Add(contatore)
        Next contatore
    End Sub

End Class
```

Visual Basic

Funzioni di carattere generale

IsNumeric <espressione>

Restituisce un valore Boolean che indica se un'espressione può essere valutata come un numero.

Esempio IsNumeric:

```
Dim n As Integer  'variabile per numeri interi
…
If (IsNumeric (Txt_n.Text)) Then
        n = Val (Txt_n.Text)
Else
        MsgBox ("n -inserire un valore numerico")
End If
```

If (IsNumeric(Txt_n.Text)) Then:

Questo è un controllo che verifica se il contenuto di un controllo di input, in questo caso Txt_n.Text (presumibilmente una TextBox chiamata Txt_n), è un valore numerico. La funzione IsNumeric() restituisce True se il contenuto della TextBox può essere interpretato come un numero (sia intero che decimale).n = Val(Txt_n.Text):Se il valore nella TextBox è numerico, la funzione Val() converte il testo della TextBox in un numero. Il numero risultante viene assegnato alla variabile n.

Else:

Se il contenuto della TextBox non è numerico (cioè IsNumeric restituisce False), allora viene eseguita l'istruzione nell'Else.

Visual Basic
funzioni di carattere generale
MessageBox

La funzione MsgBox() ci permette di produrre una finestra di dialogo che dà la possibilità all'utente di interagire con la nostra applicazione come in quest'esempio:

Nella finestra "Esempio MsgBox " è visualizzato un messaggio che chiede all'utente se vuole continuare con la stampa del documento mettendo a disposizione due pulsanti attivi OK o Annulla che danno la possibilità all'utente di continuare o interrompere il processo in corso. Il codice che ci ha permesso di ottenere questa finestra è il seguente:

```
Dim risp As Integer
risp = MsgBox ("Vuoi stampare il documento?", 1, "Esempio MsgBox ")
```

dove risp è una variabile integer che coglie il valore della risposta dell'utente alla finestra. La variabile **risp** può essere omessa nel caso non necessita rilevare il valore della risposta.

Visual Basic
funzioni di carattere generale

MessageBox()

Come abbiamo visto nell'esempio MsgBox ha all'interno delle parentesi diversi argomenti che servono per dare la forma alla nostra finestra di dialogo.

MsgBox(prompt[,button][,title][helpfile, context])

Prompt :indica il messaggio che sarà visualizzato nelle finestra di dialogo E' l'unico argomento obbligatorio per la funzione e va scritto fra virgolette

Buttons: indica il valore numerico dei pulsanti da visualizzare nel nella finestra di dialogo

Title: Indica il titolo della finestra di dialogo e va scritto fra virgolette

helpfile e context: sono relativi alla guida dell'applicazione quindi ancora non li tratteremo per evitare confusione anche perché non sono indispensabili per la funzionalità delle nostra finestra di dialogo.

La tabella seguente indica i valori da attribuire a **buttons** affinché vengano visualizzati i pulsanti:

Valore numerico	Pulsante visualizzato
0	Pulsante OK
1	Pulsanti OK e Annulla
2	Pulsanti Termina, Riprova e Ignora
3	Pulsanti Sì, No, Annulla
4	Pulsanti Sì e No
5	Pulsanti Riprova e Annulla

Ognuno di questi pulsanti ha un valore numerico che ci permette di richiamarli in eventuali routine che controllano il programma

Valore	Pulsante cliccato dall'utente
1	OK
2	Annulla
3	Termina
4	Riprova
5	Ignora
6	Sì
7	No

MessageBox()

Sommando anche il valore di visualizzazione delle icone a quello dei buttons può essere inserita anche un'icona nella finestra di dialogo così:

I valori da inserire sono elencati nella seguente tabella:

Valore	Icona
16	
32	
48	
64	

Per sommare i valori da attribuire a buttons basta scriverli separati da una +, in modo da poterli visualizzare sullo schermo insieme come nell'immagine dell'esempio precedente, nella quale vi erano presenti sia i pulsanti di comando che un'icona.

MessageBox() e Inputbox()

La differenza principale tra MsgBox e InputBox in Visual Basic riguarda il loro utilizzo e il tipo di interazione che offrono all'utente:

MsgBox: Mostra un messaggio all'utente con uno o più pulsanti per rispondere (ad esempio, "OK", "Annulla", "Sì", "No", ecc.). Viene utilizzata per visualizzare informazioni, avvisi o messaggi di errore. Non richiede alcun input dall'utente, se non una risposta alla domanda o alla richiesta. Ritorna un valore numerico che rappresenta il pulsante premuto dall'utente

Esempio:

```
MsgBox("Operazione completata con successo!", Successo")
```

InputBox: Mostra una finestra di dialogo che consente all'utente di inserire un valore (ad esempio, un numero o una stringa). Viene utilizzata per chiedere all'utente di fornire un input (ad esempio, un nome, una quantità, una password, ecc.). Ritorna una stringa contenente il valore inserito dall'utente. Se l'utente preme "Annulla" o chiude la finestra, viene restituita una stringa vuota.

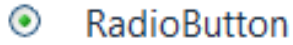
Esempio:

```
Dim nome As String
nome = InputBox("Inserisci il tuo nome:", "Richiesta dati")
```

Riepilogo:

MsgBox è usato per visualizzare un messaggio all'utente senza chiedere alcun input.
InputBox è usato per chiedere all'utente di inserire un valore in una finestra di dialogo.

RadioButton

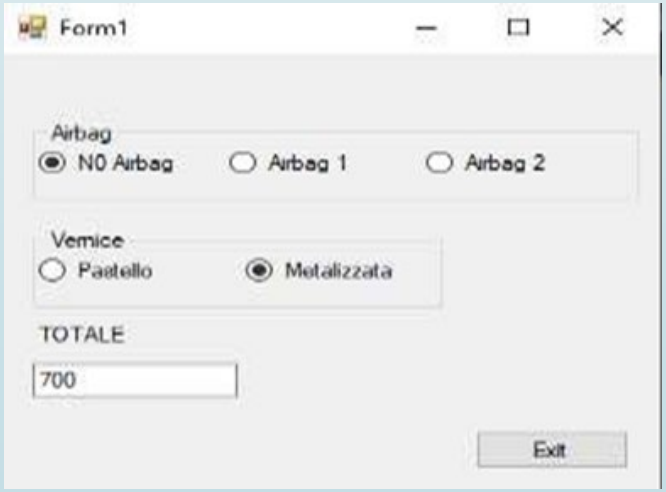


RadioButton sono progettati per fornire agli utenti una scelta tra due o più impostazioni, di cui solo una può essere assegnata a una procedura o a un oggetto. È pertanto possibile selezionare un solo **RadioButton** alla volta, anche se fa parte di un gruppo funzionale.

Per raggruppare i pulsanti di opzione, è possibile disegnarli all'interno di un contenitore, ad esempio un controllo **GroupBox.** Tutti i pulsanti di opzione aggiunti direttamente a un modulo diventano un gruppo. Per aggiungere gruppi distinti, è necessario inserirli nei pannelli o nelle caselle di gruppo.

Nell'esempio, di seguito riportato, sono stati inseriti due gruppi di opzioni (GroupBox), Group 1 e Group 2, contenenti ognuno una possibile scelta su tre opzioni.

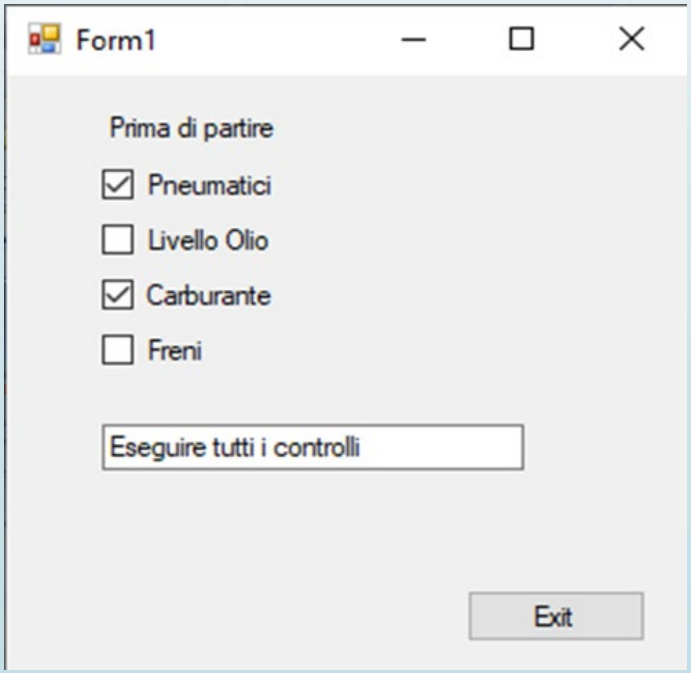
RadioButton restituisce il valore "True o False" in funzione del suo stato.



CheckBox

I controlli **CheckBox** e **RadioButton** hanno una funzione simile: consentono all'utente di scegliere da un elenco di opzioni. **CheckBox** consente all'utente di scegliere una combinazione di opzioni. Al contrario, i controlli RadioButton consentono a un utente di scegliere tra opzioni che si escludono a vicenda.

Nell'esempio, di seguito riportato, la possibilità di scelta (una o più) tra quattro opzioni. CheckBox restituisce il valore "True o False" in funzione del suo stato.



Visual Basic

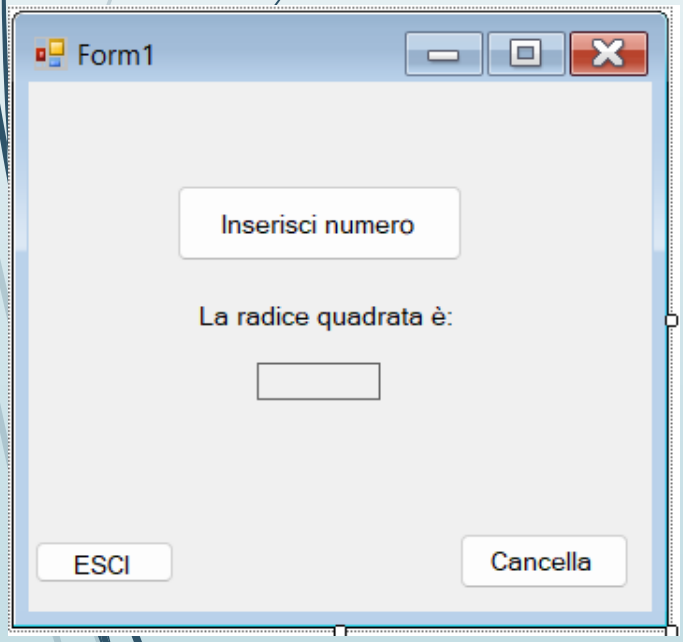
Funzioni di carattere generale

Sqr <numero> **sqr= square root**

Restituisce un valore Double che specifica la radice quadrata di un numero. In Visual Basic (come in molti altri linguaggi di programmazione), la funzione Sqrt fa parte della classe Math, quindi per utilizzarla è necessario precederla con il nome della classe Math.

Esempio Sqr:

```
Dim n As Integer
Dim r As Double
...
r = Math.Sqrt(n)
...
```




```
Public Class Form1

    Private Sub calcola_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles calcola.Click
        Dim num As Double
        Dim radice As Double
        num = InputBox("Numero:")
        If num >= 0 Then
            radice = Math.Sqrt(num)
        Else
            MsgBox("Impossibile calcolare la radice quadrata di un numero negativo!")
        End If
        Label1.Text = radice
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Label1.Text = ""
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class
```

Visual Basic
I vettori – array monodimensionali

L'istruzione per definire un vettore ha il seguente formato:

Dim nomevettore(dimensione) As Integer

Dove **nome vettore** è il nome collettivo dei componenti del vettore, **dimensione** indica il massimo numero di valori che il vettore può contenere.

Esempio dichiarazione vettori

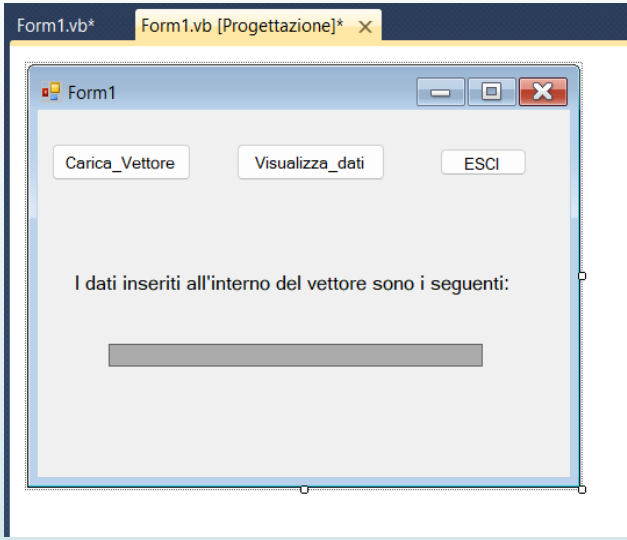
```
Const N 5
Dim vettore (N) As Integer
Dim vettore (5) As Double
```

Gli indici, del vettore in esempio, assumeranno i valori da "0" a "4".

Visual Basic
I vettori – array monodimensionali

Applicazione per memorizzare dei numeri all'interno del vettore e successivamente visualizzarli in una label.

```
Public Class Form1
    Const numeri = 30
    Dim i, N As Integer
    Dim vett() As Integer

    Private Sub carica_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles carica.Click
        ' richiesta dimensone vettore
        Do
            N = InputBox("Quanti numeri desideri inserire?", "Richiesta dati")
        Loop Until N < numeri

        ' Inizializzazione del vettore con la dimensione corretta
        ReDim vett(N)
        'caricamento vettore

        For i = 0 To N - 1
            vett(i) = InputBox("Inserisci un valore", "Inserimento dati")

        Next i


    End Sub

    Private Sub esci_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles esci.Click
        End
    End Sub

    Private Sub visualizza_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles visualizza.Click
        'viasualizzazione dei numeri all'interno della label

        For i = 0 To N - 1
            Label2.Text = Label2.Text & vett(i) & " "
        Next i
    End Sub
```



Visual Basic

Funzioni e procedure (Function e Sub)

Le procedure sono dei blocchi di codice che permettono di eseguire una serie di istruzioni quando vengono richiamate.

Le **Function** oltre ad eseguire il loro codice hanno un valore di ritorno.

Le **Sub** sono delle procedure che quando terminano la loro esecuzione, non restituiscono un valore di ritorno.

```
' Funzione che calcola l'area di un rettangolo dati base e altezza
Function CalcolaArea(base As Double, altezza As Double) As Double
    Return base * altezza
End Function
```

```
' Procedura che stampa un messaggio di saluto
Sub Saluta(nome As String)
    MsgBox("Ciao, " & nome & "!")
End Sub
```

Visual Basic

Sub

Vediamo come si dichiara e si richiama una procedura Sub:

Private Sub NomeProcedura (Param1 As Tipo, Param2 As Tipo)
Dichiarazioni locali ed istruzioni
End Sub

L'istruzione che attiva (chiama) il sottoprogramma nel programma chiamante ha la seguente sintassi:

NomeProcedura (Valore1, Valore2)

Per l'insieme dei dati *Valore1*, *Valore2* possono essere specificate sia variabili che costanti, e che
vi deve essere esatta corrispondenza rispetto all'insieme *Param1, Param2* sia per quanto riguarda il tipo di dati, sia per quanto riguarda l'ordine in cui sono specificati.

Visual Basic
Function

Vediamo come si dichiara e si richiama una procedura Function:

Function *NomeFunction* (Param1 As Tipo, Param2 As Tipo) As Tipo
Dichiarazioni locali ed istruzioni
NomeFunction = espressione
End Function

L'istruzione che attiva (chiama) la funzione nel programma chiamante ha la seguente sintassi:

Var = NomeFunction (Valore1, Valore2)

La variabile ***Var*** deve essere dello stesso tipo di ***NomeFunction.***

Per l'insieme dei dati *Valore1*, *Valore2* possono essere specificate sia variabili che costanti, e che
vi deve essere esatta corrispondenza rispetto all'insieme *Param1, Param2* sia per quanto riguarda il tipo di dati, sia per quanto riguarda l'ordine in cui sono specificati.

Visual Basic

Passaggio dei parametri

Esistono due modi per passare una variabile a una procedura (funzione):

-Per **valore**
-Per **referenza o riferimento**

Quando si effettua il passaggio par **valore**, il parametro presente nell'istruzione che richiama la procedura conserva il suo valore anche se, nella procedura chiamata, il valore del parametro corrispondente si modifica.

Quando si passa una variabile per **referenza o riferimento** (indirizzo), qualsiasi modifica effettuata sul parametro della procedura si ripercuote sul valore passato alla procedura.

Le due modalità di passaggio vengono così indicate:

ByVal
ByRef

Esempio:

```
Sub Calcola (ByVal x As Single, ByRef y As Double)
```

Visual Basic
Function

```
' Funzione che somma due numeri
Function SommaNumeri(num1 As Integer, num2 As Integer) As Integer
    ' Calcola la somma e la restituisce
    SommaNumeri = num1 + num2
End Function

' Programma che richiama la funzione SommaNumeri
' Dichiarazione delle variabili
Dim risultato As Integer
Dim valore1 As Integer = 5
Dim valore2 As Integer = 10

' Richiamo della funzione SommaNumeri
risultato = SommaNumeri(valore1, valore2)

' Visualizza il risultato nella Label della Form
Label1.Text = "La somma di " & valore1 & " e " & valore2 & " è: " & risultato
```

Visual Basic

Operazioni sui FORM

Un progetto può contenere più FORM, con la possibilità di passare il controllo su uno di essi.

Form1.Show() passa il controllo al Form1.

Form1.Hide() Nasconde il Form1, che resta in esecuzione.

Non utilizzando questa funzione il Form richiamato viene sovrapposto al Form1.

Form1.Close() Termina l'esecuzione del Form.

N.B.

Il nome del Form può essere sostituito con **Me,** in questo caso l'azione richiesta viene effettuata sul Form stesso.

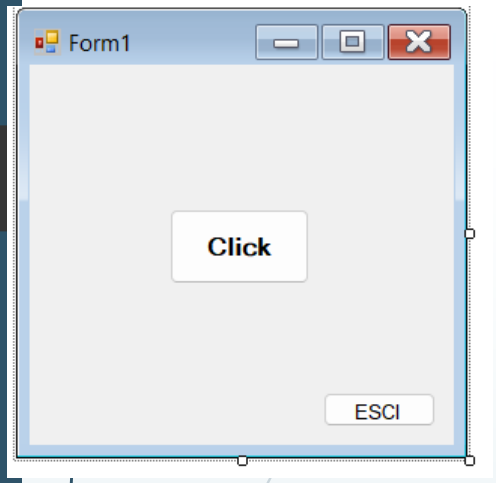
Esempio: supponiamo di voler passare da un Form PRIMO ad un Form SECONDO e viceversa, utilizzando un CommandButton(Btn_Secondo):

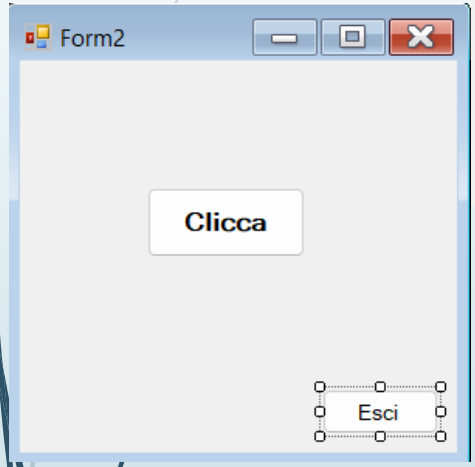
FORM PRIMO

```
Private Sub Btn_Secondo_Click(senderAs System.Object, e As System.EventArgs)
        HandlesBtn_Secondo.Click
        Secondo.Show()
        Me.Hide()
End Sub
```

Visual Basic

Operazioni sui FORM





Form1.vb | Form2.vb | Form1.vb [Progettazione] | Form2.vb [Progettazione]

(Generale) | (Dichiarazioni)

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim nuovoForm As New Form2()
        nuovoForm.Show() ' mostra Form2
        Me.Hide()        ' nasconde Form1 (oppure usa Me.Close() per chiuderlo)
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class
```

Nel menu in alto, clicca su "Progetto" → "Aggiungi Windows Form " .Dai un nome al nuovo form (es: Form2.vb) e clicca su " Aggiungi". VB aggiungerà il file Form2.vb al tuo progetto.

```
Public Class Form2

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim form1 As New Form1()
        form1.Show()
        Me.Close()
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        End
    End Sub
End Class
```