



Visual Basic.NET Using Procedures



ROAD MAP

- Introduction
- Modular Coding
- Subroutines
- Functions
- Arguments
- Argument-Passing Mechanisms



INTRODUCTION



- Application is made up of
 - Small, self-contained segments
- Code is made up of
 - Small segments called **Procedures**
 - one procedure at a time
- Two procedures:
 - Subroutines
 - functions



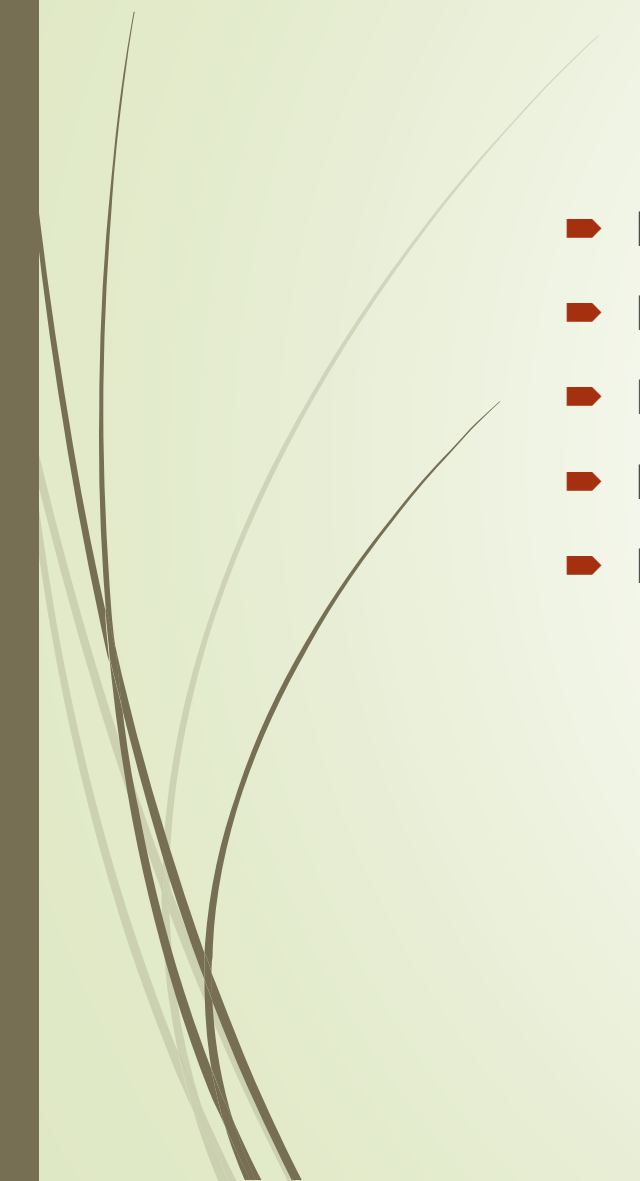
Modular Coding

Example:

```
RetrievePOLines(productID)
Sum1 = SumQtyPrice()
Qty1 = SumQuantities()
RetrieveInvoiceLines(productID)
Sum2 = SumQtyPrice()
Net = (Sum2 - Sum1) / Qty1
```



Modular Coding

- Divide-and-conquer approach
 - Broken down into simple tasks
 - Read almost like English
 - Programmers focus on different parts of application
 - Both functions and subroutines accept arguments (later in the section)
- 



Subroutines

- Subroutines
 - A block of segments that carries out a well-defined task
 - Sub... End Sub – invoke by a name
 - All variables are local within a subroutine
 - Do not return results
- Example:

```
Sub ShowDate()  
    MsgBox(Now())  
End Sub
```



Subroutines

- End Sub
 - control returns to the calling program
- Exit Sub
 - Exit a subroutine prematurely
- Arguments
 - Example:

```
Sub ShowDate(ByVal birthDate As Date)
    MsgBox(birthDate)
End Sub
```




Subroutines & Events Handlers

➤ Event Handlers

- a segment of code that is executed each time an external condition triggers the event
- Perform all actions you want
- Separate from rest of codes
- Every Application is made up of event handlers
- No need to return results
- Implemented as subroutines



Functions

- Functions
 - Returns results
 - Have types
 - All variables are local within a subroutines
- Return value
 - The value you pass back to the calling program from a function
 - Value type = Function type (Must Match)
- Accept arguments



Functions

- Functions... End Function

- Example:

```
Function NextDay() As Date
    Dim theNextDay As Date
    theNextDay = DateAdd(DateInterval.Day, 1, Now())
    Return(theNextDay)
End Function
```

- Custom function has unique name

- Modify default scope of a function with keywords:

- Public / Private / Protected / Friend / Protected Friend

Build-In Functions

- .NET Framework provides a large number of functions
- Abs() function
- Example :

```
Function Abs(X As Double) As Double
    If X >= 0 Then
        Return(X)
    Else
        Return(-X)
    End If
End Function
```

Custom Functions

- called from several places in the application
- ISBNCheckDigit() function
- Example:

```
Function ISBNCheckDigit(ByVal ISBN As String) As String
    Dim i As Integer, chksum, chkDigit As Integer
    For i = 0 To 8
        chkSum = chkSum + (10 - i) * ISBN.Substring(i, 1)
    Next
    chkDigit = 11 - (chkSum Mod 11)
    If chkDigit = 10 Then
        Return ("X")
    Else
        Return (chkDigit.ToString)
    End If
End Function
```

Calling Functions and Subroutines

- To call a procedure: enter name followed by arguments

```
Dim chDigit As String  
chDigit = ISBNCheckDigit("078212283")
```

- The values of the arguments must match their declared type
- Functions are called by name, and a list of arguments follows the name in parentheses

```
Degrees = Fahrenheit(Temperature)
```

- Functions can be called from within expressions

```
MsgBox("40 degrees Celsius are " & Fahrenheit(40).ToString & _  
      " degrees Fahrenheit")
```

Arguments

- Recall:

- Arguments – a value you pass to the procedure and on which the procedure usually acts

- Functions accept argument

- - Min()

```
Function Min(ByVal a As Single, ByVal b As Single) As Single  
    Min = IIf(a < b, a, b)  
End Function
```

- IIf()

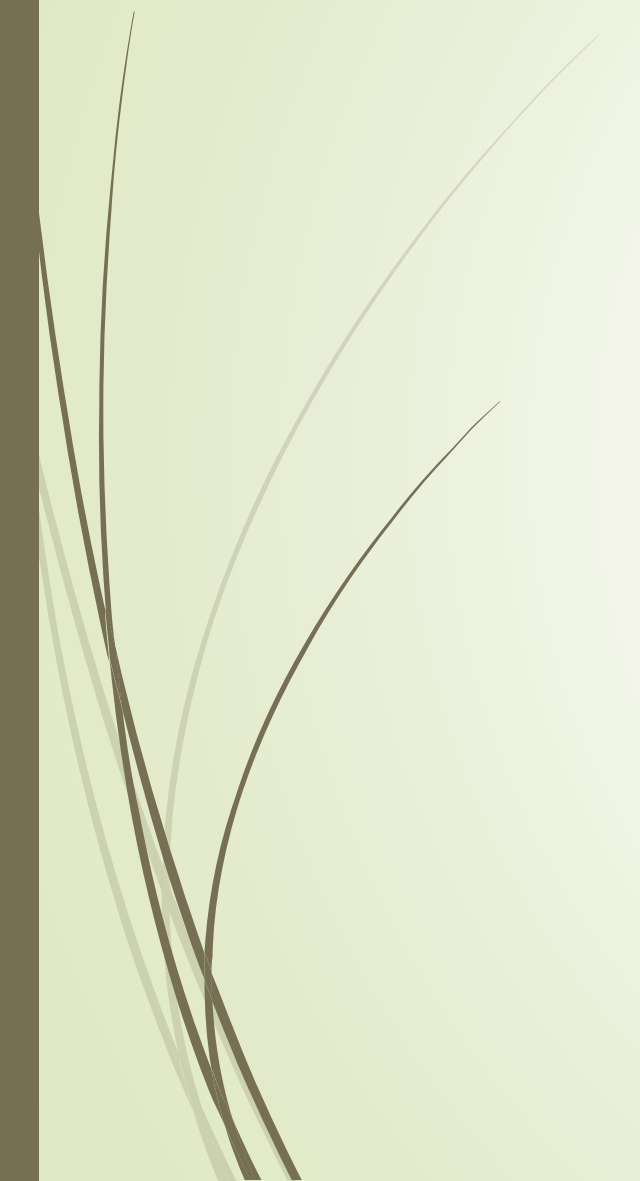
- built-in function that evaluates the first argument, which is a logical expression\
- If expression = TRUE, it returns second argument; otherwise, it returns third argument

- Min() function with the Strict option

```
Function Min(ByVal a As Object, ByVal b As Object) As Object  
    Min = IIf(Val(a) < Val(b), a, b)  
End Function
```



Argument-passing Mechanisms

- Passing Arguments by Value
 - Passing Arguments by Reference
 - Returning Multiple Values
 - Passing Objects as Arguments
- 

Argument-passing By Value

- a copy of the argument
- No permanent
- Argument values are isolated from the procedure
- Default: only the code segment can change values
- ByVal keyword
 - Default: insert automatically if omit

```
Function Degrees(ByVal Celsius as Single) As Single  
    Degrees = (9 / 5) * Celsius + 32  
End Function
```

Argument-passing By Value

- When change the value:

```
Function Degrees(ByVal Celsius as Single) As Single  
    Degrees = (9 / 5) * Celsius + 32  
    Celsius = 0  
End Function
```

```
CTemp = InputBox("Enter temperature in degrees Celsius")  
MsgBox(CTemp.ToString & " degrees Celsius are " & Degrees((CTemp)) & _  
    " degrees Fahrenheit")
```

- Result:

```
32 degrees Celsius are 89.6 degrees Fahrenheit
```



Argument-passing By Value



Note:

NOTE When you pass arguments to a procedure by reference, you're actually passing the variable itself. Any changes made to the argument by the procedure will be permanent. When you pass arguments by value, the procedure gets a copy of the variable, which is discarded when the procedure ends. Any changes made to the argument by the procedure won't affect the variable of the calling program.

NOTE When you pass an array as argument to a procedure, the array is always passed by reference—even if you specify the `ByVal` keyword. The reason for this is that it would take the machine some time to create a copy of the array. Since the copy of the array must also live in memory, passing too many arrays back and forth by value would deplete your system's memory.

Argument-passing By Reference

- Gives the procedure access to the actual variable
- Change value permanently

```
Function Add(ByRef num1 As Integer, ByRef num2 As Integer) As Integer
    Add = num1 + num2
    num1 = 0
    num2 = 0
End Function
```

Argument-passing By Reference

➤ Next...

```
Dim A As Integer, B As Integer
A = 10
B = 2
Dim Sum As Integer
Sum = Add(A, B)
Console.WriteLine(A)
Console.WriteLine(B)
Console.WriteLine(Sum)
```

Argument-passing By Reference

- Output window:

```
0  
0  
12
```

- Changes remain in effect only in the function.



Returning Multiple Values

- ▶ pass additional arguments by reference
- ▶ & set their values from within the function's code
- ▶ Example:
 - ▶ Stats() function

Returning Multiple Values

- Stats() function: return Avg and StDev

```
Function Stats(ByRef Data() As Double, ByRef Avg As Double, _  
               ByRef StDev As Double) As Integer  
    Dim i As Integer, sum As Double, sumSqr As Double, points As Integer  
    points = Data.Length  
    For i = 0 To points - 1  
        sum = sum + Data(i)  
        sumSqr = sumSqr + Data(i) ^ 2  
    Next  
    Avg = sum / points  
    StDev = System.Math.Sqrt(sumSqr / points - Avg ^ 2)  
    Return(points)  
End Function
```

Returning Multiple Values

- Call Stats() function:

```
Dim Values(100) As Double
' Statements to populate the data set
Dim average, deviation As Double
Dim points As Integer
points = Stats(Values, average, deviation)
Console.WriteLine points & " values processed."
Console.WriteLine "The average is " & average & " and"
Console.WriteLine "the standard deviation is " & deviation
```

Passing Objects as Arguments

- Passed by reference, even if you have specified the ByVal keyword

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
                          ByVal e As System.EventArgs) Handles Button1.Click  
    Dim aList As New ArrayList()  
    PopulateList(aList)  
    Console.WriteLine(aList(0).ToString)  
    Console.WriteLine(aList(1).ToString)  
    Console.WriteLine(aList(2).ToString)  
End Sub  
Sub PopulateList(ByVal list As ArrayList)  
    list.Add("1")  
    list.Add("2")  
    list.Add("3")  
End Sub
```

Event-Handler Arguments

- Event-Handler:
 - Never return results
 - Implemented as subroutines
 - Accept two arguments:
 - Sender
 - e

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

Event-Handler Arguments

- Sender...
 - conveys information about the object that initiated the event
 - find out the type of the object that raised the event

```
Console.WriteLine(sender.ToString)
    System.Windows.Forms.Button, Text: Button1
Console.WriteLine(sender.GetType)
    System.Windows.Forms.Button
```

Event-Handler Arguments

➤ e...

- An object that exposes some properties, which vary depending on the type of the event and the control that raised the event
- The e argument passed to the Click event handler has no special properties.
- Examine members of this argument:
 - The Mouse Event
 - Button
 - Clicks
 - Delta
 - X,Y
 - The Key Events
 - Alt, Control, Shift
 - KeyCode
 - KeyData
 - KeyValue



The Mouse Event

- A series of events is triggered when click the mouse
- A single click:
 - A MouseDown Event
 - A Click Event
 - A MouseUp Event



Button

- ▶ Return the button that was pressed
 - ▶ MouseButton enumeration:
 - ▶ Left, Middle, None, Right, XButton1, and XButton2
- ▶ The e argument of the Click and DoubleClick events doesn't provide a Button property
- ▶ These two events can only be triggered with the left button.



Clicks

- Return the number of times the mouse button was pressed and released
- A single click:
 - 1
- Double-click:
 - 2
- You can't click a control three times




Delta

- Used with wheel mice
- Read the number of detents (that is, notches or stops) that the mouse wheel was rotated

X,Y

- Return the coordinates of the pointer at the moment the mouse button was pressed (in the MouseDown event) or released (in the MouseUp event)
- If you click a Button control at the very first pixel (its top-left corner), the X and Y properties will be 0.
- Example:

```
Private Sub Button1_MouseDown(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.MouseEventArgs) _  
    Handles Button1.MouseDown  
    Console.WriteLine("Button pressed at " & e.X & ", " & e.Y)  
End Sub  
Private Sub Button1_MouseUp(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.MouseEventArgs) _  
    Handles Button1.MouseUp  
    Console.WriteLine("Button released at " & e.X & ", " & e.Y)  
End Sub
```



X,Y

- If you press and release the mouse at a single point, both handlers will report the same point
- If you move the pointer before releasing the button, you will see four values like the following

Button pressed at 63, 16

Button released at -107, -68

The Key Events

- One of the most important events in programming the TextBox
- Raised when a key is pressed, while the control has the focus
- The KeyDown event handler's definition:

```
Private Sub TextBox1_KeyDown(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.KeyEventArgs) _  
    Handles TextBox1.KeyDown
```



Alt, Control, Shift

- These three properties return a True/False value indicating whether one or more of the control keys were down when the key was pressed.
- 




Key Code

- Return the code of the key that was pressed
- Its value in the keys enumeration
- Keys enumeration:
 - A member of all keys: mouse keys
 - Each key has its own code
 - Usually corresponding to 2 different characters




Key Data

- Return a value that identifies the key pressed
 - Distinguish the character or symbol on the key
- 



Key Value

- Return the keyboard value for the key that was pressed
- 



Passing An Unknown Number of Arguments

- ▶ When you Do NOT know how many arguments will be passed to the procedure...
- ▶ Visual Basic supports the ParamArray keyword, which allows you to pass a variable number of arguments to a procedure
- ▶ Example:
 - ▶ populate a ListBox control

```
ListBox1.Items.Add("new item")
```

Passing An Unknown Number of Arguments

- Adds a variable number of arguments

```
Sub AddNamesToList(ParamArray ByVal NamesArray() As Object)
    Dim x As Object
    For Each x In NamesArray
        ListBox1.Items.Add(x)
    Next x
End Sub
```

- To add items to the list

```
AddNamesToList("Robert", "Manny", "Renee", "Charles", "Madonna")
```

- To know the number of arguments

```
NamesArray.Length
```

Passing An Unknown Number of Arguments

- Go through all the elements of the *NamesArray*

```
Dim i As Integer
For i = 0 to NamesArray.GetUpperBound(0)
    ListBox1.Items.Add(NamesArray(i))
Next i
```

- To use the array's Length property

```
Dim i As Integer
For i = 0 to NamesArray.Length - 1
    ListBox1.Items.Add(NamesArray(i))
Next i
```

- To accept multiple arguments and omit some

```
ProcName(arg1, , arg3, arg4)
```



Named Arguments

- Main limitation is the order of arguments
- By default: Visual Basic matches the values passed to a procedure to the declared arguments by their order
 - So far: positional arguments
- Named Arguments:
 - supply arguments in any order
 - Recognized by name not order

Named Arguments

➤ Example:

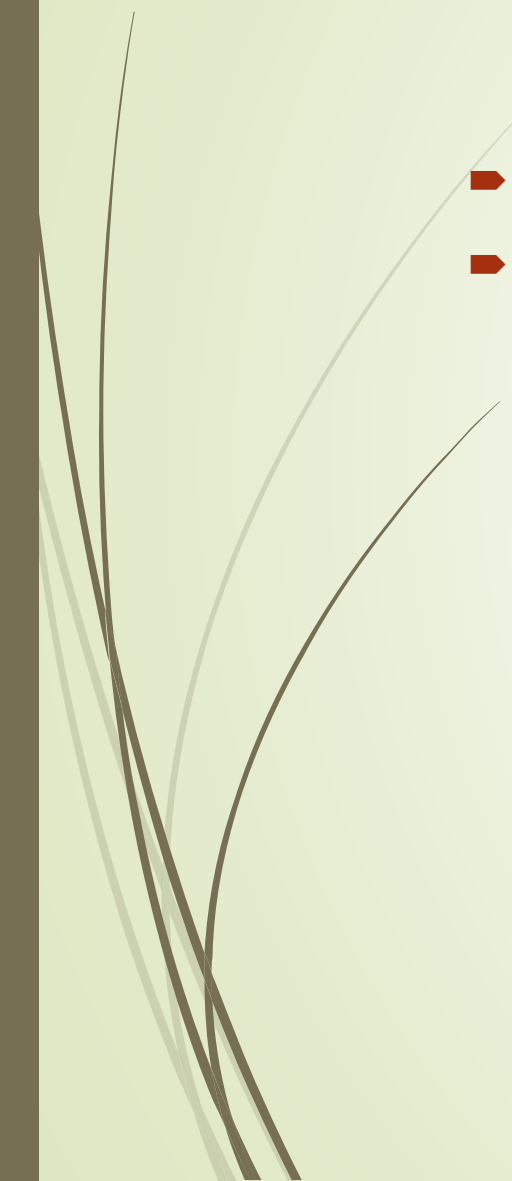
```
Function Contact(Name As String, Address As String, EMail As String)
```

```
Contact("Peter Evans", "2020 Palm Ave., Santa Barbara, CA 90000", _  
        "PeterEvans@example.com")
```

```
Contact(Address:="2020 Palm Ave., Santa Barbara, CA 90000", _  
        EMail:="PeterEvans@example.com", Name:="Peter Evans")
```



More Types of Function Return Values

- Functions Returning Structures
 - Functions Returning Arrays
- 

Functions Returning Values

- create a custom data type (a structure) and write a function that returns a variable of this type
- Example:

```
Structure CustBalance  
    Dim BalSavings As Decimal  
    Dim BalChecking As Decimal  
End Structure
```

Functions Returning Arrays

- Return arrays
 - write functions that return not only multiple values, but also an unknown number of values.
- Example: Revise Stats() to return Arrays

```
Function Stats(ByRef DataArray() As Double) As Double()
```

- 1) Specify a type for the function's return value, and add a pair of parentheses after the type's name.
- 2) Declare an array of the same type and specify its dimension

```
Dim Results(3) As Double
```

Functions Returning Arrays

- To return the *Results* array, simply use it as argument to the Return statement

```
Return(Results)
```

- In the calling procedure, you must declare an array of the same type without dimensions

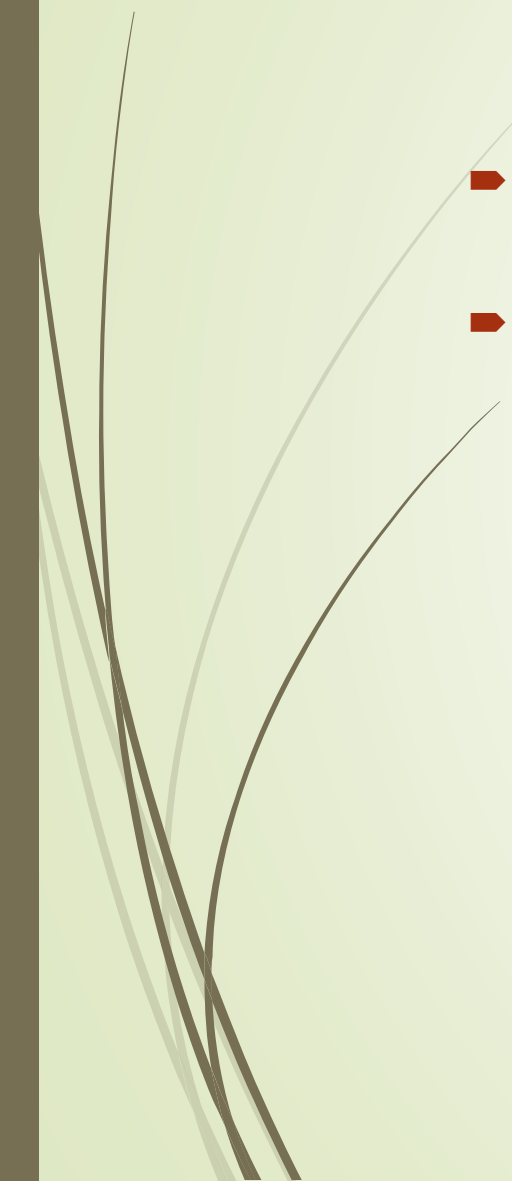
```
Dim Stats() As Double
```

- Call the function and assign its return value to this array

```
Stats() = Stats(DataSet())
```



Overloading Functions

- Have multiple implementations of the same function, each with a different set of arguments and a different return value
 - Share the same name
- 



Exercise

- (1) What is the format of subroutines?
- (2) Use build-in function `Abs()` to code:
 - If $X \geq 5$, return $X+5$
 - If $X < 5$, return $X-5$