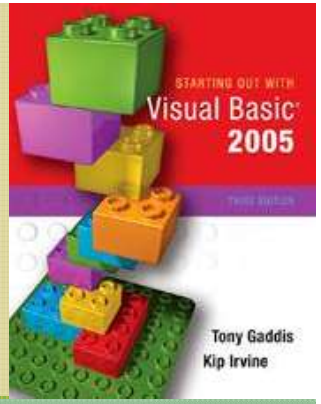
Chapter

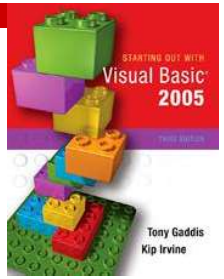


6

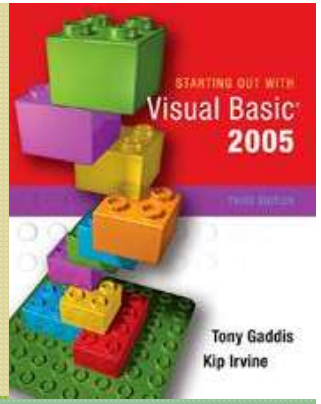
Sub Procedures And Functions

A procedure is a collection of statements that performs a task.

Introduction



- A *Sub* procedure is a collection of statements that performs a task
  - An abbreviation of the older term *subroutine*
  - Event procedures are Sub procedures
- A *Function* procedure is a collection of statements that performs a task *and* returns a value to the VB statement that executed it
  - Function procedures work like intrinsic functions, such as Val and IsNumeric
- A *method* is a procedure declared in a class

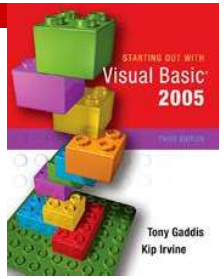


6.1 Sub Procedures

You Can Write Your Own General Purpose Sub Procedures That Perform Specific Tasks

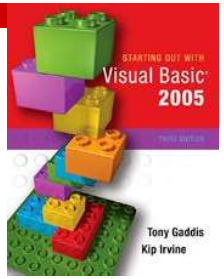
General Purpose Sub Procedures Are Not Triggered by Events but Called From Statements in Other Procedures

Sub Procedure Uses



- May handle events such as a click event
- Also used to simplify a program by
  - Breaking it into small, manageable pieces ***or***
  - Performing a task that is needed repeatedly
- Sub procedures help to *modularize* code
  - Divides a program into a set of logical tasks

Sample Sub Procedure, Tutorial 6-1



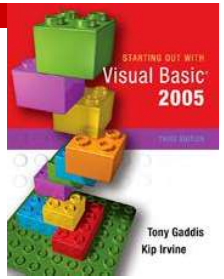

```
Private Sub btnGo_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnGo.Click
        ' This procedure calls the DisplayMessage procedure.
        lstOutput.Items.Add("Hello from btnGo_Click procedure.")
        lstOutput.Items.Add("Calling the DisplayMessage " & _
              "procedure.")
        DisplayMessage()
        lstOutput.Items.Add("Now I am back in the btnGo_Click
              procedure.")
End Sub
```

Returns to btnGo_Click

Calls DisplayMessage procedure

```
  Sub DisplayMessage()
        'A Sub procedure that displays a message.
        lstOutput.Items.Add("")
        lstOutput.Items.Add("Hello from DisplayMessage.")
        lstOutput.Items.Add("")
  End Sub
```

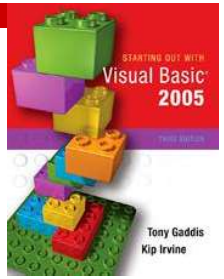
Declaring a Sub Procedure




```
[AccessSpecifier] Sub ProcedureName ([ParameterList])
        [Statement(s)]
End Sub
```

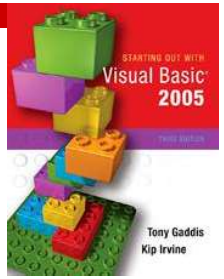
- *AccessSpecifier* is optional and establishes accessibility to the program
- *Sub* and *End* are keywords
- *ProcedureName* used to refer to procedure
  - Use *Pascal casing*, capitalize 1st character of the name and each new word in the name
- *ParameterList* is a list of variables or values being passed to the sub procedure

More on Access Specifier

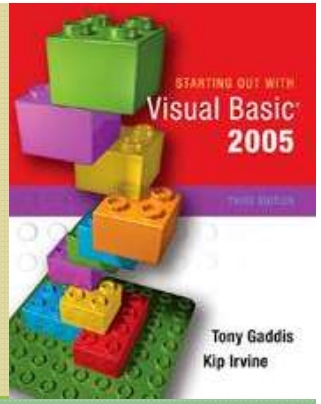


- *Private* allows use only from that form or class
- *Public* allows use from other forms or classes
- If not specified, default is Public
- Additional access specifiers:
  - *Protected*
  - *Friend*
  - *Protected Friend*
  - These will be discussed in later chapters
- Access specifiers won't be used for now
- Practice writing procedures in Tutorial 6-2

Procedures and Static Variables



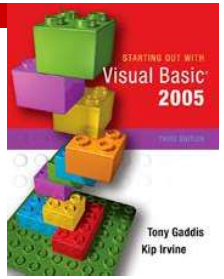
- Variables needed only in a Sub procedure, should be declared *within* the Sub procedure
  - Creates a *local variable* with scope only within the sub procedure where declared
  - Local variable values are not saved from one sub procedure call to the next
- To save value between procedure calls, use *Static* keyword to create a *static local variable*
  - `Static VariableName As DataType`
  - Scope is only within the procedure
  - But variable exists for lifetime of procedure



6.2 Passing Arguments to a Sub Procedure

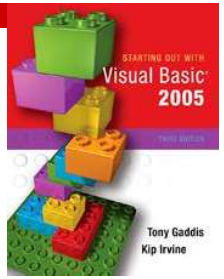
When calling a procedure, you can pass it values known as arguments

Arguments



- *Argument* – a value passed to a procedure
- We've already done this with functions
  - `Value = CInt(txtInput.Text)`
  - Calls CInt function and passes txtInput.Text
- A Sub must be declared so it accepts an argument

Passing Arguments By Value

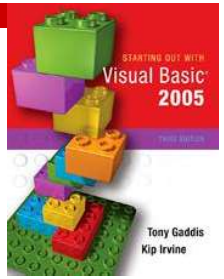



```
DisplayValue(5)                 'calls DisplayValue procedure
```

```
Sub DisplayValue(ByVal number As Integer)
        ' This procedure displays a value in a message box.
       MessageBox.Show(number.ToString)
End Sub
```

- *Number* declared as an integer argument
- Storage location *number* created
- A value, 5 in this case, must be supplied and is copied into the storage location for *number*
- DisplayValue then executes
- Tutorial 6-3 demonstrates passing arguments

Passing Multiple Arguments

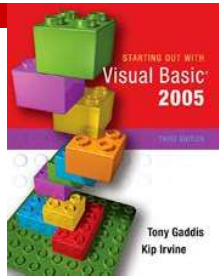



```
ShowSum(5, 10)                  'calls ShowSum procedure


Sub ShowSum(ByVal num1 As Integer, ByVal num2 As Integer)
        ' This procedure accepts two arguments, and prints
        ' their sum on the form.
        Dim sum As Integer
        sum = num1 + num2
        MessageBox.Show("The sum is " & sum.ToString)
End Sub
```

- Multiple arguments separated by commas
- Value of first argument is copied to first
- Second to second, etc.

Passing Arguments ByVal or ByRef



- Arguments are usually passed *ByVal*
  - New storage location created for procedure
  - Storage location gets a copy of the value
  - Any changes in value are made to the copy
  - Calling procedure won't "see" the changes
- Arguments can also be passed *ByRef*
  - Procedure points to (references) argument's original storage location
  - Any changes are made to the original value
  - Calling procedure "sees" the changes
- Tutorial 6-4 demonstrates this difference