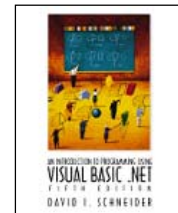


Chapter 4: Procedures

- ✚ Sub Procedures, Part I (4.1)
- ✚ Sub Procedures, Part II (4.2)
- ✚ Function (4.3)
- ✚ Modular Design (4.4)



David I. Schneider, *An Introduction to Programming using Visual Basic .NET, 5th Edition*, Prentice Hall, 2002.

Sub Procedures, Part I (4.1)

- ✚ Sub Procedures
- ✚ Calling Other Sub Procedures

Sub Procedures, Part I (4.1) (cont.)

🔧 Devices for modularity

– VB.NET has two devices for breaking problems into smaller pieces:

- Sub procedures
- Function procedures

Sub Procedures, Part I (4.1) (cont.)

🔧 Sub Procedures

- Performs one or more related tasks
- General syntax

```
Sub ProcedureName()  
    statements  
End Sub
```

Sub Procedures, Part I (4.1) (cont.)

🔦 Calling a Sub procedure

- The statement that invokes a Sub procedure is also referred to as a call statement
- A call statement looks like this:

ProcedureName()

Sub Procedures, Part I (4.1) (cont.)

🔦 Naming Sub procedures

- The rules for naming Sub procedures are the same as the rules for naming variables.

Sub Procedures, Part I (4.1) (cont.)

🔦 Example

```
IstResult.Items.Clear()
```

```
ExplainPurpose()  
IstResult.Items.Add("")
```



```
Sub ExplainPurpose()
```

```
    IstResult.Items.Add("This program displays a sentence")
```

```
    IstResult.Items.Add("identifying two numbers and their sum.")
```

```
End Sub
```



Sub Procedures, Part I (4.1) (cont.)

🔦 Passing

– You can send items to a Sub procedure

```
Sum(2, 3)
```

```
Sub Sum(num1 As Double, num2 As Double)
```

– In the Sum Sub procedure, 2 will be stored in num1
and 3 will be stored in num2

Sub Procedures, Part I (4.1) (cont.)

🔧 Parameters and Arguments

```
CalculateDensity("Alaska", 627000, 591000)
```

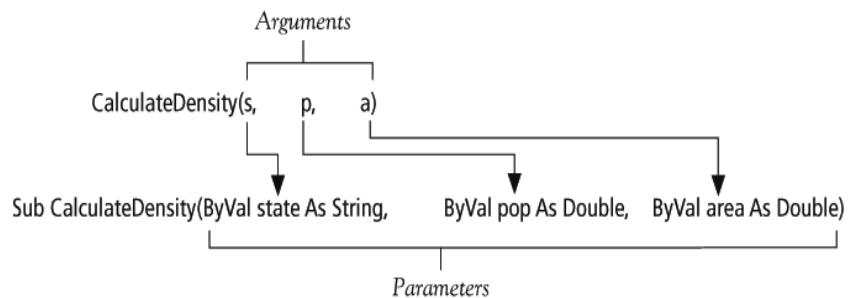
Arguments – what you send to a Sub procedure

```
Sub CalculateDensity(ByVal state As String, _  
                    ByVal pop As Double, _  
                    ByVal area As Double)
```

Parameters – place holders for what the sub procedure receives

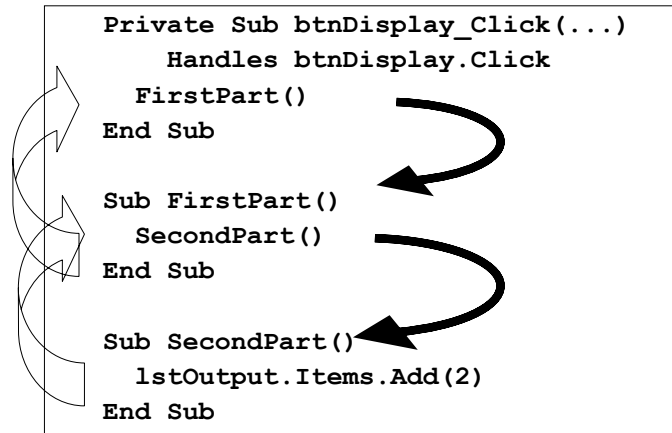
Sub Procedures, Part I (4.1) (cont.)

🔧 Figure 4.2



Sub Procedures, Part I (4.1) (cont.)

🔦 Sub Procedures Calling Other Sub Procedures



Sub Procedures, Part II (4.2)

- 🔦 Passing by Value
- 🔦 Passing by Reference
- 🔦 Local Variables
- 🔦 Class-Level Variables
- 🔦 Debugging

Sub Procedures, Part II (4.2) (cont.)

👉 Passing by Value

- ByVal stands for “By Value”
- ByVal parameters retain their original value after Sub procedure terminates

Sub Procedures, Part II (4.2) (cont.)

👉 Passing by Reference

- ByRef stands for “By Reference”
- ByRef parameters can be changed by the Sub procedure and retain the new value after the Sub procedure terminates

Sub Procedures, Part II (4.2) (cont.)

🔦 Local Variables

- Variables declared inside a Sub procedure with a Dim statement
- Space reserved in memory for that variable until the End Sub – then the variable ceases to exist

Sub Procedures, Part II (4.2) (cont.)

🔦 Class-Level Variables

- Visible to every procedure in a form's code without being passed
- Dim statements for Class-Level variables are placed
 - Outside all procedures
 - At the top of the program region

Sub Procedures, Part II (4.2) (cont.)

🔦 Scope

- Class-level variables have class-level scope and are available to all procedures in the class
- Variables declared inside a procedure have local scope and are only available to the procedure in which they are declared

Sub Procedures, Part II (4.2) (cont.)

🔦 Debugging

- Programs with Sub procedures are easier to debug
- Each Sub procedure can be checked individually before being placed into the program

Functions (4.3)

- ✚ User-Defined Functions Having Several Parameters
- ✚ Comparing Function Procedures with Sub Procedures
- ✚ Collapsing a Procedure with a Region Directive

Functions (4.3) (cont.)

✚ User-Defined Functions

- Functions always return one value
- Syntax:

```
Function FunctionName (ByVal var1 As Type1, _  
                        ByVal var2 As Type2, _  
                        ...) As dataType  
  
    statement(s)  
    Return expression  
End Function
```

Functions (4.3) (cont.)

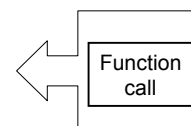
🔗 Some Built-In Functions

Function	Example	Input	Output
Int	Int(2.6) is 2	number	number
Chr	Chr(65) is "A"	number	string
Asc	Asc("Apple") is 65	string	number
FormatNumber	FormatNumber (12345.628, 1) is 12,345.6	number, number	string

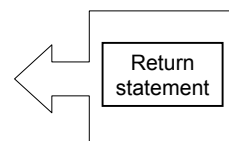
Functions (4.3) (cont.)

🔗 Sample

```
Private Sub btnDetermine_Click(...)
    Handles btnDetermine.Click
    Dim name As String
    name = txtFullName.Text
    txtFirstname.Text = FirstName(name)
End Sub
```



```
Function FirstName(ByVal name As String) As String
    Dim firstSpace As Integer
    firstSpace = name.IndexOf(" ")
    Return name.Substring(0, firstSpace)
End Function
```



Functions (4.3) (cont.)

🔧 Having Several Parameters

```
Private Sub btnCalculate_Click(...)
    Handles btnCalculate.Click
    Dim a, b As Double
    a = Cdbl(txtSideOne.Text)
    b = Cdbl(txtSideTwo.Text)
    txtHyp.Text = CStr(Hypotenuse(a, b))
End Sub

Function Hypotenuse(ByVal a As Double, _
                    ByVal b As Double) As Double
    Return Math.Sqrt(a ^ 2 + b ^ 2)
End Function
```

Functions (4.3) (cont.)

🔧 User-Defined Functions Having No Parameters

```
Private Sub btnDisplay_Click(...) _
    Handles btnDisplay.Click
    txtBox.Text = Saying()
End Sub

Function Saying() As String
    Return InputBox("What is your" _
                    & " favorite saying?")
End Function
```

Functions (4.3) (cont.)

🔗 Comparing Function Procedures with Sub Procedures

- Subs are accessed using a call statement
- Functions are called where you would expect to find a literal or expression
- For example:
 - `Result = functionCall`
 - `IstBox.Items.Add (functionCall)`

Functions (4.3) (cont.)

🔗 Functions vs. Procedures

- Both can perform similar tasks
- Both can call other subs and functions
- Use a function when you want to return one and only one value

Functions (4.3) (cont.)

🔧 Collapsing a Procedure with a Region Directive

- A procedure can be collapsed behind a captioned rectangle
- This task is carried out with a Region directive.
- To specify a region, precede the code to be collapsed with a line of the form

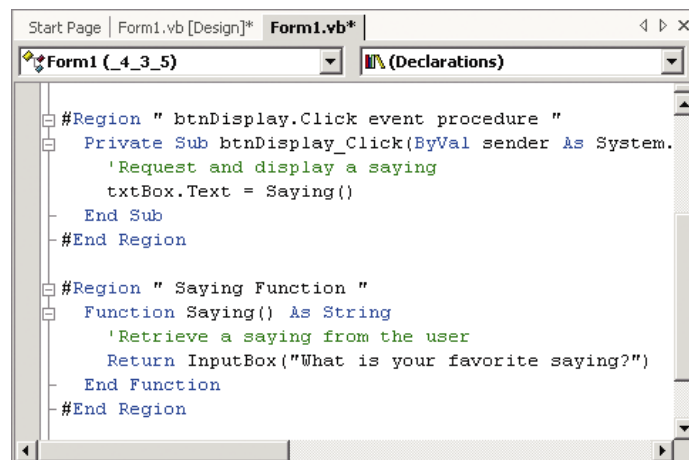
```
#Region "Text to be displayed in the box."
```

- and follow the code with the line

```
#End Region
```

Functions (4.3) (cont.)

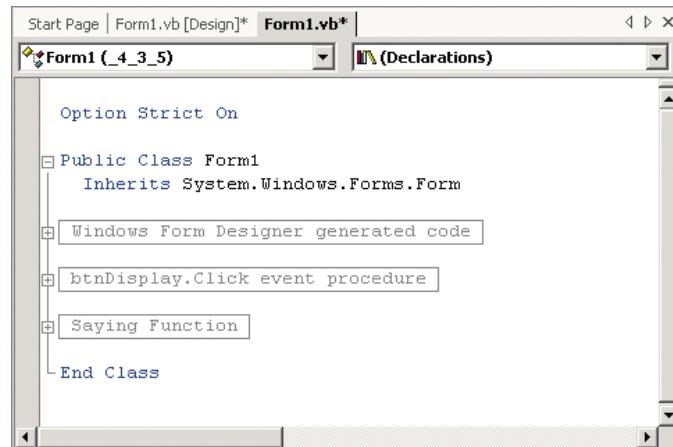
🔧 Region Directives



```
Start Page | Form1.vb [Design]* | Form1.vb*  
Form1 (_4_3_5) | (Declarations)  
#Region " btnDisplay.Click event procedure "  
Private Sub btnDisplay_Click(ByVal sender As System.  
    'Request and display a saying  
    txtBox.Text = Saying()  
End Sub  
#End Region  
  
#Region " Saying Function "  
Function Saying() As String  
    'Retrieve a saying from the user  
    Return InputBox("What is your favorite saying?")  
End Function  
#End Region
```

Functions (4.3) (cont.)

🔧 Collapsed Regions



The screenshot shows a Visual Studio window with a code editor. The code is in VB.NET and includes a class declaration for Form1, a comment for Windows Form Designer generated code, a click event procedure for btnDisplay, and a function named Saying. Several regions of the code are collapsed, indicated by minus signs in the left margin. The collapsed regions are: 'Windows Form Designer generated code', 'btnDisplay.Click event procedure', and 'Saying Function'. The code is as follows:

```
Option Strict On

Public Class Form1
    Inherits System.Windows.Forms.Form

    [Collapsed] Windows Form Designer generated code

    [Collapsed] btnDisplay.Click event procedure

    [Collapsed] Saying Function

End Class
```

Modular Design (4.4)

🔧 Top-Down Design

🔧 Structured Programming

🔧 Advantages of Structured Programming

Modular Design (4.4) (cont.)

✎ Design Terminology

- Large programs can be broken down into smaller problems
- "divide-and-conquer" approach called "stepwise refinement"
- Stepwise refinement is part of top-down design methodology

Modular Design (4.4) (cont.)

✎ Top-Down Design

- General problems are at the top of the design
- Specific tasks are near the end of the design
- Top-down design and structured programming are techniques to enhance programmers' productivity

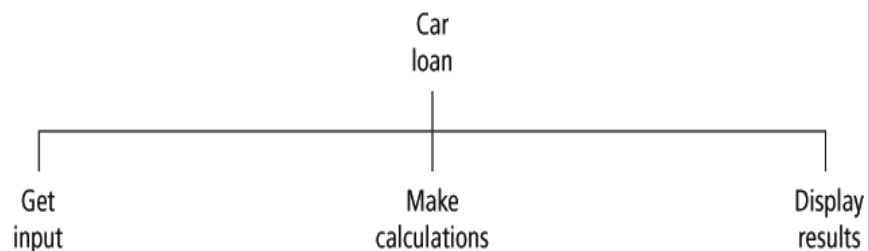
Modular Design (4.4) (cont.)

🔦 Top-Down Design Criteria

1. The design should be easily readable and emphasize small module size.
2. Modules proceed from general to specific as you read down the chart.
3. The modules, as much as possible, should be single minded. That is, they should only perform a single well-defined task.
4. Modules should be as independent of each other as possible, and any relationships among modules should be specified.

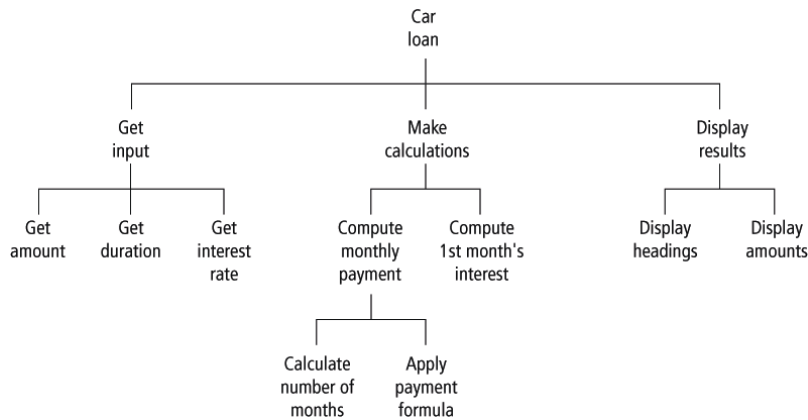
Modular Design (4.4) (cont.)

🔦 Top-Level Design HIPO Chart



Modular Design (4.4) (cont.)

🔗 Detailed HIPO Chart



Modular Design (4.4) (cont.)

🔗 Structured Programming

- Control structures in structured programming:
- **Sequences:** Statements are executed one after another.
- **Decisions:** One of two blocks of program code is executed based on a test for some condition.
- **Loops (iteration):** One or more statements are executed repeatedly as long as a specified condition is true.

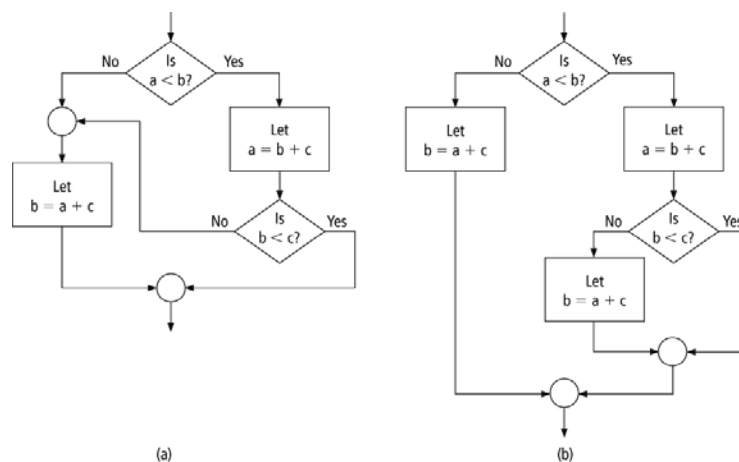
Modular Design (4.4) (cont.)

🔧 Advantages of Structured Programming

- Goal to create correct programs that are easier to
 - write
 - understand
 - Modify
- "GOTO -less" programming

Modular Design (4.4) (cont.)

🔧 Comparison of Flow Charts



Modular Design (4.4) (cont.)

👉 Easy to Write

- Allows programmer to first focus on the big picture and take care of the details later
- Several programmers can work on the same program at the same time
- Code that can be used in many programs is said to be reusable

Modular Design (4.4) (cont.)

👉 Easy to Debug

- Procedures can be checked individually
- A driver program can be set up to test modules individually before the complete program is ready
- Using a driver program to test modules (or stubs) is known as stub testing

Modular Design (4.4) (cont.)

✦ Easy to Understand

- Interconnections of the procedures reveal the modular design of the program.
- The meaningful procedure names, along with relevant comments, identify the tasks performed by the modules.
- The meaningful variable names help the programmer to recall the purpose of each variable.

Modular Design (4.4) (cont.)

✦ Easy to Change

- Because a structured program is self-documenting, it can easily be deciphered by another programmer.

Modular Design (4.4) (cont.)

👉 Object-Oriented Programming

- an encapsulation of data and code that operates on the data
- objects have properties, respond to methods, and raise events.