

# Chapter 5 – Subroutines and Functions

## 5.1 What Are Subroutines and Functions?

As your applications grow, you need to break them into separate logical units.

The code associated with accomplishing each task is separated from the code the accomplishes other tasks.

These actions are referred to as events and are one way of breaking up code into smaller, more logical units.

Another way to break up an application is by using either **functions** or **subroutines**.

Programs are made more readable by breaking large amounts of code into smaller, more concise parts.

By breaking code into functions and subroutines, code can be written once and reused often.

This reduces the size of the application and debugging time.

Each time you repeat a calculation, you waste space and increase the likelihood of a typographical error and therefore cause your application to execute improperly.

Functions and subroutines operate similarly but have one key difference.

A function is used when a value is returned to the calling routine, while a subroutine is used when a desired task is needed, but no value is returned.

# Chapter 5 – Subroutines and Functions

## Invoking a Subroutine

A subroutine is used when a series of steps are required but no value is returned to the routine that called the subroutine. Subroutines are invoked using a subroutine name:

```
SubroutineName (ParameterList)
```

Invoking a subroutine can occur with parameters:

```
OutputMin(intValue1, intValue2)
```

Invoking a subroutine can also occur without parameters:

```
Message ()
```

# Chapter 5 – Subroutines and Functions

## Invoking a Function Call

A function by definition has a **return value**. Therefore, a function call must be assigned to a variable of the type that the function returns:

```
VariableName = FunctionName(ParameterList)
```

The following code calls a `UCase` function to return an uppercase representation of a `String` that is passed as a parameter:

```
strVariableName = UCase("please uppercase this")
```



# Chapter 5 – Subroutines and Functions

## 5.2 Built-In Functions

Visual Basic .NET provides many built-in functions to assist your coding or applications.

By using built-in functions you save time in coding and debugging work that has already been provided for you.

# Chapter 5 – Subroutines and Functions

## String Functions

**Function Name:** UCase

**Function Description:** Returns the String that is passed in all uppercase letters.

**Common Uses:** UCase can be used when the desired output is required to be in all uppercase letters. It is also commonly used when you wish to validate data entered by a user against a given string.

**Syntax:** String = UCase(String)

**Examples:**

Function call	Return Value
UCase("Input String")	"INPUT STRING"
UCase("all lowercase")	"ALL LOWERCASE"
UCase("ALL UPPERCASE")	"ALL UPPERCASE"
UCase("UpPeP AnD lOwErCaSE")	"UPPER AND LOWERCASE"

**Previous Way of Coding Validation:**

```
If (txtVote.Text = "Bush" Or txtVote.Text = "BUSH" Or _  
    txtVote.Text = "bush" Then ...
```

**Better Way of Coding Validation:**

```
If (UCase(txtVote.Text) = "BUSH") Then ...
```

# Chapter 5 – Subroutines and Functions

**Function Name:** LCase

**Function Description:** Returns the `String` that is passed in all lowercase letters.

**Common Uses:** LCase is very similar in use to UCase.

**Syntax:** `String = LCase(String)`

**Examples:**

Function call	Return Value
<code>UCase("Input String")</code>	"input string"
<code>UCase("all lowercase")</code>	"all lowercase"
<code>UCase("ALL UPPERCASE")</code>	"all uppercase"
<code>UCase("UpPeP AnD lOwErCaSE")</code>	"upper and lowercase"

# Chapter 5 – Subroutines and Functions

**Function Name:** `Trim`

**Function Description:** Returns a `String` with the same content, except the leading and trailing spaces are removed.

**Common Uses:** Often when data is gathered, additional spaces may exist before the first noncharacter or after the last nonblank character.

It is good practice to remove these so that data may be presented cleanly.

**Syntax:** `String = Trim(String)`

**Examples:**

Function call	Return Value
<code>Trim("  InputString")</code>	"InputString"
<code>Trim("InputString  ")</code>	"InputString"
<code>Trim("  InputString  ")</code>	"InputString"
<code>Trim("  Input String  ")</code>	"Input String"

# Chapter 5 – Subroutines and Functions

## Function Name: Trim (continued)

The following code will initialize two Strings.

One will contain a String that has the leading and trailing spaces removed by the Trim function.

It is displayed between two vertical bars so that it will be obvious that the spaces have been removed.

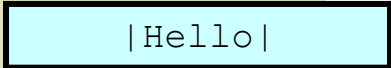
The second String will be created in a similar manner; however, the spaces will not be removed.

```
Dim strTest As String
Dim strWithBlanks As String
Dim strBorder As String
Dim strTrimmedOutput As String
Dim strUnTrimmedOutput As String
```


```
strTest = " Hello " 'Two spaces before and after
strBorder = "|"
```

```
strTrimmedOutput = strBorder & Trim(strTest) & strBorder
strUnTrimmedOutput = strBorder & strTest & strBorder
```

```
MsgBox(strTrimmedOutput)
MsgBox(strUnTrimmedOutput)
```



|Hello|



| Hello |



# Chapter 5 – Subroutines and Functions

**Function Name:** `Space`

**Function Description:** Returns a `String` containing the number of spaces indicated by the parameter.

**Common Uses:** Often you wish to add spaces to set the total length of a `String` to an exact size.

This is often used when working with fixed-width data files.

**Syntax:** `String = Space(Integer)`

**Examples:**

Function call	Return Value
<code>Space(5)</code>	" "
<code>Space(10)</code>	" "
<code>Space(0)</code>	""
<code>"Hello" &amp; Space(10) &amp; "Goodbye"</code>	"Hello        Goodbye"

# Chapter 5 – Subroutines and Functions

**Function Name:** Len

**Function Description:** Returns the number of characters contained in a String

**Common Uses:** Len is used to determine the size of a String.

**Syntax:** Integer = Len(String)

**Examples:**

Function call	Return Value
Len("Inconceivable")	13
Len("Iocaine Powder")	14
Len("Hello, my name is Inigo Montoya. You killed my father. Prepare to die.")	70
Len("")	0

# Chapter 5 – Subroutines and Functions

**Function Name:** `Left`

**Function Description:** Returns the first N characters of a `String` where N is an `Integer` parameter indicating the number of characters to return.

If N is greater than the number of characters in the `String`, then the `String` is returned.

No extra spaces are added.

**Common Uses:** Often you are only concerned with the first few characters of a `String`. `Left` is a great way to look at only the beginning of a `String`.

**Syntax:** `String = Microsoft.VisualBasic.Left(String, Integer)`

**Examples:**

Function call	Return Value
<code>Microsoft.VisualBasic.Left("Beginning of String", 5)</code>	"Begin"
<code>Microsoft.VisualBasic.Left("Beginning of String", 2)</code>	"Be"
<code>Microsoft.VisualBasic.Left("Beginning of String", 0)</code>	""
<code>Microsoft.VisualBasic.Left("Beginning of String", 20)</code>	"Beginning of String"

# Chapter 5 – Subroutines and Functions

## Function Name: Left (continued)

The following code shows how you might use Left to determine if a person's full name belongs to either a man or a woman.

```
Dim strPerson1 As String
Dim strPerson2 As String
Dim strPerson3 As String
Dim strPerson4 As String
strPerson1 = "Mr. Jeff Salvage"
strPerson2 = "Ms. Charlene Nolan"
strPerson3 = "Mrs. Karen Charles"
strPerson4 = "Miss Lynn Bosko"

'Process Person1
If ("Mr." = Microsoft.VisualBasic.Left(strPerson1, 3)) Then
    MsgBox "Person 1 is a Man"
ElseIf ("Miss" = Microsoft.VisualBasic.Left(strPerson1, 4) Or _
        "Ms." = Microsoft.VisualBasic.Left(strPerson1, 3) Or _
        "Mrs." = Microsoft.VisualBasic.Left(strPerson1, 4)) Then
    MsgBox "Person 1 is a Woman"
Else
    MsgBox "Is Person 1 an Alien?"
EndIf
'Process Person2
If ("Mr." = Microsoft.VisualBasic.Left(strPerson2, 3)) Then
    MsgBox "Person 2 is a Man"
ElseIf ("Miss" = Microsoft.VisualBasic.Left(strPerson2, 4) Or _
        "Ms." = Microsoft.VisualBasic.Left(strPerson2, 3) Or _
        "Mrs." = Microsoft.VisualBasic.Left(strPerson2, 4)) Then
    MsgBox "Person 2 is a Woman"
Else
    MsgBox "Is Person 2 an Alien?"
EndIf
'Person3 and Person4 code could follow
```

# Chapter 5 – Subroutines and Functions

**Function Name:** `Right`

**Function Description:** Returns the last N characters of a `String` where N is an `Integer` parameter indicating the number of characters to return.

If N is greater than the number of characters in the `String`, then the `String` is returned.

No extra spaces are added.

**Common Uses:** Often you are only concerned with the last few characters of a `String`. `Right` is a great way to look at only the end of a `String`.

**Syntax:** `String = Microsoft.VisualBasic.Right(String, Integer)`

**Examples:**

Function call	Return Value
<code>Microsoft.VisualBasic.Right("Ending of String", 5)</code>	"tring"
<code>Microsoft.VisualBasic.Right("Ending of String", 2)</code>	"ng"
<code>Microsoft.VisualBasic.Right("Ending of String", 0)</code>	""
<code>Microsoft.VisualBasic.Right("Ending of String", 20)</code>	"Ending of String"

# Chapter 5 – Subroutines and Functions

## Function Name: Right (continued)

The following code shows how you might use Right to determine a person's suffix, as in, Jr., Sr., or Ph.D.

```
Dim strPerson1 As String
Dim strPerson2 As String
Dim strPerson3 As String

strPerson1 = "Nira Herrmann, Ph.D."
strPerson2 = "John Cunningham, Sr."
strPerson3 = "Bob Bruno, Jr."

'Process Person1
If ("Ph.D." = Microsoft.VisualBasic.Right(strPerson1, 5)) Then
    MsgBox "Person 1 has a doctorate degree."
ElseIf ("Sr." = Microsoft.VisualBasic.Right(strPerson1, 3)) Then
    MsgBox "Person 1 has a kid with the same name."
ElseIf ("Jr." = Microsoft.VisualBasic.Right(strPerson1, 3)) Then
    MsgBox "Person 1 has a father with the same name."
EndIf

'Person2 and Person3 code could follow
```

# Chapter 5 – Subroutines and Functions

**Function Name:** `Mid`

**Function Description:** Returns a specific number of characters of a `String` allowing the developer to indicate where to start and how many characters to return.

The first parameter is the source `String`.

The second is an `Integer` indicating the starting position to copy from.

The third parameter is optional and indicates the number of characters to copy.

If the third parameter is left out, all characters from the starting position are returned.

**Common Uses:** Often you wish to extract a portion of a `String` to use separately from the rest of the `String`. This is often the case when working with fixed-width data files.

**Syntax:** `String = Mid(String, Starting Position, Optional Length)`

**Examples:**

Function call	Return Value
<code>Mid("This is the String", 6, 2)</code>	"is"
<code>Mid("This is the String", 9, 3)</code>	"the"
<code>Mid("This is the String", 13, 4)</code>	"Stri"
<code>Mid("This is the String", 8)</code>	" the String"

# Chapter 5 – Subroutines and Functions

**Function Name:** `InStr`

**Function Description:** Returns the position of the first occurrence of a substring that is searched for in the `String` passed.

**Common Uses:** `InStr` can be used to tell us if a `String` has a certain substring contained within it. It operates much like searching a document for a word.

**Syntax:** `Long = InStr(String to be Searched, Search String)`

**Examples:**

Function call	Return Value
<code>InStr("This is a very", "is")</code>	3
<code>InStr("ab ab ab", "ab")</code>	1
<code>InStr("ab ab ab", "a")</code>	1
<code>InStr("ab ab ab", "c")</code>	0



# Chapter 5 – Subroutines and Functions

## Drill 5.1

What is the output of the following code?

```
MsgBox UCase("What is the output?")
```

Answer: "WHAT IS THE OUTPUT?"

# Chapter 5 – Subroutines and Functions

## Drill 5.2

What is the output of the following code?

```
MsgBox Microsoft.VisualBasic.Left("What is the output?", 4)
```

Answer: "What"

# Chapter 5 – Subroutines and Functions

## Drill 5.3

What is the output of the following code?

```
MsgBox Microsoft.VisualBasic.Right("What is the output?", 4)
```

Answer: "put?"

# Chapter 5 – Subroutines and Functions

## Drill 5.4

What is the output of the following code?

```
MsgBox UCase(Microsoft.VisualBasic.Left("What is the output?", 4))
```

Answer: "WHAT"

# Chapter 5 – Subroutines and Functions

## Drill 5.5

What is the output of the following code?

```
MsgBox Microsfot.VisualBasic.Left("What is the output?", 4) & _  
    Space(5) & Trim(" ? ")
```

Answer: "What ?"

# Chapter 5 – Subroutines and Functions

## Conversion Functions

**Function Name:** `Str`

**Function Description:** Returns a `String` representation of the numeric value passed to it.

By default it will place a single space in front of the first numeric character.

**Common Uses:** Visual Basic .NET will not allow you to assign a value of one data type to a variable of another data type.

Avoid future problems by manually converting numeric values to `Strings` before assigning them to either a variable of type `String` or a `Text` attribute in a control.

**Syntax:** `String = Str(Numeric Value)`

**Examples:**

```
'Proper conversion
Dim strDestination As String
Dim intSource As Integer
intSource = 1
strDestination = Str(intSource)
```

# Chapter 5 – Subroutines and Functions

**Function Name:** Str (continued)

Here are two demonstrations of an improper conversion.

```
'Improper conversion
Dim strDestination As String
Dim intSource As Integer
intSource = 1
strDestination = intSource
```

```
'Run-time Error
Dim strDestination As String
Dim strSource As String
strSource = "Source"
strDestination = Str(strSource)
```

# Chapter 5 – Subroutines and Functions

**Function Name:** Val

**Function Description:** Returns a numeric representation of the String value passed to it.

Val will convert a String to a numeric until it reaches a character that is not a numeric value, a decimal point, or a white-space character.

Once an unrecognizable character is read, conversion stops at that point.

**Common Uses:** Val is used much in the same manner as Str, except in the opposite direction.

**Syntax:** Numeric Value = Val(String)

**Examples:**

Function call	Return Value
Val("199.11")	199.11
Val(" 199.11 ")	199.11
Val(" 1 99.1 1")	199.11
Val(" 199 ")	199
Val("\$199.11")	0
Val("1,199.11")	1
Val(" ")	0
Val("123abc")	123
Val("abc123")	0



# Chapter 5 – Subroutines and Functions

## Conversion Functions

**Function Name:** CDate

**Function Description:** Returns a Date representation of the String value passed to it.

**Common Uses:** CDate is used when a Date representation is needed.

Often a date can be stored in a String when it is gathered from a fixed-width or comma-delimited file.

If proper operations are going to be performed on the date, then it is necessary to store it in its native format.

**Syntax:** Date = CDate(String)

**Examples:**

```
Dim dteToday As Date
Dim strToday As String
Dim strTomorrow As String
Dim dteTomorrow As Date
strToday = "September 30, 2001"
dteToday = CDate(strToday)
'See Coach's Tip for explanation of the DateAdd function
dteTomorrow = DateAdd(DateInterval.Day, 1, dteToday)
strTomorrow = CStr(dteTomorrow)
MsgBox(strTomorrow)
```

# Chapter 5 – Subroutines and Functions

## Drill 5.6

What is the output of the following code?

```
Dim sngValue1 As Single
Dim strValue2 As String

sngValue1 = 1.1
strValue2 = Str(sngValue1)

MsgBox(strValue2)
```

Answer: "1.1"

# Chapter 5 – Subroutines and Functions

## Drill 5.7

What is the output of the following code?

```
Dim strValue As String  
strValue = Str("A")  
MsgBox(strValue)
```

Answer: no output

# Chapter 5 – Subroutines and Functions

## Mathematical Functions

**Function Name:** `Int`, `Fix`

**Function Description:** Returns the `Integer` portion of the numerical value passed to it.

**Common Uses:** `Int` or `Fix` are used when you wish to convert a numerical value to an integer without regard for the decimal value. It performs a truncation of the number.

**Syntax:** `Integer = Int(Numerical Value)`

`Integer = Fix(Numerical Value)`

**Examples:**

Function call	Return Value
<code>Int(199.11)</code>	199
<code>Int(0.1)</code>	0
<code>Int(1.5)</code>	1
<code>Int(0.99999)</code>	0
<code>Fix(199.11)</code>	199
<code>Fix(0.1)</code>	0

# Chapter 5 – Subroutines and Functions

## Drill 5.8

What is the output of the following code?

```
MsgBox (Str(Int(-9.9)))
```

Answer: -10

# Chapter 5 – Subroutines and Functions

## Drill 5.9

What is the output of the following code?

```
MsgBox (Str(Fix(-9.9)))
```

Answer: -9

# Chapter 5 – Subroutines and Functions

## Miscellaneous Functions

**Function Name:** `IsNumeric`

**Function Description:** Returns `True` if the value passed to it evaluates to a numeric data type, otherwise it returns `False`.

**Common Uses:** `IsNumeric` can be used to verify that a value can be evaluated as a number. Instead of possibly getting either inaccurate results or a run-time error, by using `IsNumeric`, a proactive approach to error handling can be achieved.

**Syntax:** `Boolean = IsNumeric(Expression)`

**Examples:**

Function call	Return Value
<code>IsNumeric("199.11")</code>	<code>True</code>
<code>IsNumeric(199.11)</code>	<code>True</code>
<code>IsNumeric("ABC")</code>	<code>False</code>
<code>IsNumeric("123ABC")</code>	<code>False</code>
<code>IsNumeric("1,999")</code>	<code>True</code>
<code>IsNumeric("\$1,1999")</code>	<code>True</code>
<code>IsNumeric("50%")</code>	<code>False</code>
<code>IsNumeric("One")</code>	<code>False</code>

# Chapter 5 – Subroutines and Functions

**Function Name:** `IsDate`

**Function Description:** Returns `True` if the value passed to it evaluates to a valid `Date`, otherwise it returns `False`.

**Common Uses:** `IsDate` can be used when you are about to convert a value to a `Date` representation.

Instead of possibly getting either inaccurate results or a run-time error, by using `IsDate`, a proactive approach to error handling can be achieved.

**Syntax:** `Boolean = IsDate(Expression)`

**Examples:**

Function call	Return Value
<code>IsDate("January 1, 2001")</code>	<code>True</code>
<code>IsDate("1/1/2001")</code>	<code>True</code>
<code>IsDate("1/1/01")</code>	<code>True</code>
<code>IsDate(#1/1/01#)</code>	<code>True</code>
<code>IsDate(1/1/2001)</code>	<code>False</code>
<code>IsDate("Today")</code>	<code>False</code>



# Chapter 5 – Subroutines and Functions

**Function Name:** `Today`

**Function Description:** Returns the current system date.

**Common Uses:** `Today` can be used anytime the developer wishes to access the system date.

While it is returned as a variant, it can be used as a native date format or converted to a `String` representation.

**Syntax:** `Date = Today()`

**Examples:**

```
'Code to display Yesterday's Date in a MessageBox
Dim dteToday As Date
Dim dteYesterday As Date
Dim strYesterday As String

dteToday = Today()
dteYesterday = DateAdd(DateInterval.Day, -1, dteToday)
strYesterday = CStr(dteYesterday)
MsgBox(strYesterday)
```

# Chapter 5 – Subroutines and Functions

**Function Name:** `TimeOfDay`

**Function Description:** Returns the current system time.

**Common Uses:** `TimeOfDay` can be used anytime the developer wishes to access the system time.

While it is returned as a variant, it can be used as a native date format or converted to a `String` representation.

You can store a `Time` in the `DateTime` data type.

**Syntax:** `DateTime = TimeOfDay()`

**Examples:**

```
'Code to display the time now in a MessageBox
Dim dteExactTime As DateTime
dteExactTime = TimeOfDay()
MsgBox(dteExactTime.ToString())
```

# Chapter 5 – Subroutines and Functions

**Function Name:** Rnd

**Function Description:** Returns a pseudo random decimal number that is greater than or equal to 0 and less than 1.

By passing Rnd an optional numeric parameter, you can further control the type of random number you generate.

**Common Uses:** Random numbers are required for a great many reasons.

By combining the output of the Rnd function with some basic mathematics, random numbers can be generated within a given range.

**Syntax:** Variant = Rnd()

**Examples:**

Function call	Return Value
<code>Int (Rnd () *3)</code>	Generates a random number from 0 to 2
<code>Int (Rnd () *3)+1</code>	Generates a random number from 1 to 3
<code>Int (Rnd () *6)+1</code>	Generates a random number from 1 to 6
<code>Int (Rnd () *6) +1) + (Int (Rnd () *6) +1)</code>	Generates a random number from 2 to 12 similar to rolling a pair of dice

# Chapter 5 – Subroutines and Functions

**Function Name:** `Format`

**Function Description:** Returns a `String` representation of the expression passed formatted according to instructions passed to it in the second parameter. `Format` may be used to improve the appearance of numbers, dates, times, or string values.

**Common Uses:** `Format` is used anytime the user needs to beautify the output of a value.

**Syntax:** `String = Format(Expression, String)`

**Second Parameter:**

Standard Format	Description
Currency	Displays the number as a monetary value. A dollar sign precedes the number that is formatted to two decimal places. If a number requires it, a comma is placed to the left of every three digits (except for the two for the decimal places). If the number is negative, it is enclosed in parenthesis.
Date	There are many “Standard” date formats. “General Date” will format a date/time as mm/dd/yyyy and hh:mm:ss if there is a time. Other formats include: “Long Date”, “Medium Date”, “Short Date”
Fixed	Displays the number with two decimal places and at least one digit to the left of the decimal place.
Percent	Displays the number as a percentage. It multiplies the number passed by 100 (with two decimal places) and places a percent sign to the right.
Standard	Displays the number with two decimal places. If the number requires it, a comma is placed to the left of every three digits. If the number is negative, a negative sign is displayed to the left of the number.
Time	There are many “Standard” time formats: “Long Time”, “Medium Time”, “Short Time”

# Chapter 5 – Subroutines and Functions

**Function Name:** Format (continued)

**Examples:**

Function call	Return Value
Format(123.1, "Currency")	\$123.10
Format(#03/03/2001#, "Short Date")	3/3/2001
Format(123.1, "Fixed")	123.10
Format(.1231, "Percent")	12.31%
Format(123.1, "Standard")	123.10
Format(#10:30:01#, "Long Time")	10:30:01 AM

# Chapter 5 – Subroutines and Functions

## Function Name: `Format` (continued)

Custom format strings can be made by combining different predefined format characters.

Format Character	Description
0	A 0 in the format string is replaced by the digit from the expression that belongs in that space. If there is no digit for that space, the 0 is displayed.
#	When a format string specifies a #, if the expression has a digit in the position where the # is, the digit is displayed, otherwise nothing is displayed.
Decimal point	Forces a decimal place to be displayed. Does not necessarily force a digit to be displayed to the right.
Comma	Places a comma in the number. Usually used in the standard position, every three digits to the left of the decimal place.

# Chapter 5 – Subroutines and Functions

**Function Name:** Format (continued)

**Examples:**

Function call	Return Value
<code>Format(123.1, "00000.00")</code>	00123.10
<code>Format(0, "00000.00")</code>	00000.00
<code>Format(123.1, "00000")</code>	00123
<code>Format(123.1, "#####.##")</code>	123.1
<code>Format(123.1, "0###.##")</code>	0123.1
<code>Format(123.1, "0, ###.##")</code>	0,123.1

# Chapter 5 – Subroutines and Functions

## Drill 5.10

What is the range of values produced by the following code?

```
MsgBox Str(Int(Rnd()*100))
```

Answer: Integers from 0 to 100



# Chapter 5 – Subroutines and Functions

## Drill 5.11

What is the range of values produced by the following code?

```
MsgBox Str(Int(Rnd()*10+3))
```

Answer: Integers from 3 to 12

# Chapter 5 – Subroutines and Functions

## Drill 5.12

What is the String produced by the following code to Format?

```
Format(1111.1111, "Standard")
```

Answer: "1,111.11"

# Chapter 5 – Subroutines and Functions

## Drill 5.13

What is the `String` produced by the following code to `Format`?

```
Format(452.23, "0000.000")
```

Answer: "0452.230"

# Chapter 5 – Subroutines and Functions

## Drill 5.14

What is the `String` produced by the following code to `Format`?

```
Format(#03/01/2001#, "Date")
```

Answer: Date is not a standard format.

# Chapter 5 – Subroutines and Functions

## 5.3 Writing Functions and Subroutines

The functions built into Visual Basic .NET are useful, but they are not all-inclusive.

There are countless functions and subroutines that developers wish were built into Visual Basic .NET, but they are not.

Because it would be impossible to predict all the routines developers require, you have the ability to create your own.

The syntax for creating your own function is as follows:

```
Scope Function FunctionName (ParameterList) As ReturnType  
    BodyOfFunction()  
End Function
```

# Chapter 5 – Subroutines and Functions

## Coding a Function

A function is declared with a scope of either `Private` or `Public`.

`Private` indicates that they are usable only in the form that they are coded.

If you specify a `Public` scope, then the function would be visible to other forms.

Using the `Function` keyword makes the distinction between a function and subroutine.

The `FunctionName` is used to differentiate this function from any others. Naming a function follows the same rules as naming a variable.

The `ParameterList` is a list of variables that will be passed to the function.

A parameter list is specified by specifying the parameter name, the keyword `As`, and then the data type of the parameter.

The `ReturnType` is the type of value that is returned from the function. When a function is written, it must return a value of the same data type as specified by the `ReturnType`. A value is returned using the following syntax:

```
Return ReturnValue
```

The `BodyOfFunction` is the code that accomplishes the function's task.

# Chapter 5 – Subroutines and Functions

## Example: Max Function

This is an example of a function that returns the maximum of two `Integer`s.

It compares the first parameter, `intValue1`, to the second parameter, `intValue2`.

If the first parameter is greater than the second parameter, then the first parameter is the value returned from the function. Otherwise, the second parameter is returned.

While not explicitly tested for, when the first parameter equals the second parameter, `Max` returns the second parameter.

```
Private Function Max(ByVal intValue1 As Integer, ByVal intValue2_  
                    As Integer) As Integer  
    If (intValue1 > intValue2) Then  
        Return intValue1  
    Else  
        Return intValue2  
    End If  
End Function
```

# Chapter 5 – Subroutines and Functions

## Example: PayRate Function

This is an example of a function that returns the pay rate of a department. The function accepts the department as a String parameter and returns the pay rate associated with the department. It assumes a valid department, otherwise it returns 0.

```
'Declare constants for entire application, place in declarations
areas of code
Const sngSalesPayRate As Single = 25
Const sngProcessingPayRate As Single = 15
Const sngManagementPayRate As Single = 75
Const sngPhonePayRate As Single = 10

'Declare PayRate function
Private Function PayRate(ByVal strDepartment As String) As Single
    Select Case strDepartment
        Case "Sales"
            Return sngSalesPayRate
        Case "Processing"
            Return sngProcessingPayRate
        Case "Management"
            Return sngManagementPayRate
        Case "Phone"
            Return sngPhonePayRate
        Case Else
            Return 0
    End Select
End Function
```



# Chapter 5 – Subroutines and Functions

## Coding a Subroutine

The coding of a subroutine does not vary much from the coding of a function.

The only difference is that a subroutine cannot directly return a resulting value.

The syntax for subroutine is as follows:

```
Scope Sub SubroutineName(ParameterList)  
    Body of Subroutine  
End Sub
```

# Chapter 5 – Subroutines and Functions

## Example: ThankYouMessage Subroutine

The `ThankYouMessage` subroutine outputs a message in a message box.

This particular subroutine does not contain any parameters.

A subroutine does not require parameters, they are optional.

```
Private Sub ThankYouMessage()  
    MsgBox("Thank You Pat Croche for a Great Basketball Team!")  
End Sub
```

# Chapter 5 – Subroutines and Functions

## Example: DisplayGrade Subroutine

Here is a subroutine that accepts an `Integer` parameter that indicates a person's average and displays a `MessageBox` containing the letter grade associated with the person's average.

```
Private Sub DisplayGrade(ByVal intStudentAverage As Integer)
    Select Case intStudentAverage
        Case 90 To 100
            MsgBox("A")
        Case 80 To 89
            MsgBox("B")
        Case 70 To 79
            MsgBox("C")
        Case 60 To 69
            MsgBox("D")
        Case Else
            MsgBox("F")
    End Select
End Sub
```

The `DisplayGrade` routine could be called from a button and pass a value contained in a text box to the subroutine:

```
Private Sub btnDisplayGrade_Click(...)
    DisplayGrade(Int(txtStudentGrade.Text))
End Sub
```

# Chapter 5 – Subroutines and Functions

## Example: InitializePicture Subroutine

Subroutines can be used to change the properties of a control.

Observe the following subroutine, `InitializePicture`.

It accepts a picture box as the first parameter and a `String` containing the new picture file location and name as the second parameter.

The subroutine will set the `Picture` attribute to the new file name and set the `TabStop` and `BorderStyle` attributes to 0.

```
Private Sub InitializePicture(ByVal picControl As PictureBox, _  
                             ByVal strNewPicture As String)  
    picControl.Image = Image.FromFile(strNewPicture)  
    picControl.TabStop = False  
    picControl.BorderStyle = 0  
End Sub
```

The `InitializePicture` subroutine could be called from a button:

```
Private Sub btnInitialize_Click(...  
    InitializePicture(picPicture1, "c:\VB Coach\Chapter 5\DontTouchThis.jpg")  
End Sub
```

# Chapter 5 – Subroutines and Functions

## Drill 5.15

Given the following definitions of three subroutines, show which of the following subroutine calls are valid (assuming that `Option Strict` is On):

```
Private Sub DrillSub1(ByVal intDrillValue As Integer, _  
                    ByVal strDrillValue As String)
```

```
End Sub
```

```
Private Sub DrillSub2(ByVal strDrillValue As String, _  
                    ByVal intDrillValue As Integer)
```

```
End Sub
```

```
Private Sub DrillSub3(ByVal strStringValue As Integer, _  
                    ByVal intIntegerValue As String)
```

```
End Sub
```

# Chapter 5 – Subroutines and Functions

## Drill 5.15 Continued

```
Dim intVal1 As Integer
Dim strVal2 As String
Dim intVal3 As Integer
Dim sngVal4 As Single
```

DrillSub1(intVal1, strVal2)	Valid
DrillSub1(strVal2, intVal1)	Invalid
DrillSub1(intVal3, strVal2)	Valid
DrillSub2(intVal1, intVal3)	Invalid
DrillSub2(sngVal4, intVal1)	Invalid
DrillSub2(strVal2, intVal1)	Valid
DrillSub3(intVal1, strVal2)	Valid
DrillSub3(strVal2, intVal1)	Invalid
DrillSub3(intVal1, intVal3)	invalid

# Chapter 5 – Subroutines and Functions

## Local Variables

Many times when you are writing a function or subroutine, data must be stored temporarily within the routine.

The value being stored will only be required during the execution of the routine and then not required after the routine is exited.

These **local variables** are declared in the same manner as the other variables you declared.

When you declare a variable within a function or subroutine, it is only accessible from within the function or subroutine where it is declared.

Attempts to access a local variable from outside the routine where it is defined will result in a compile error.

# Chapter 5 – Subroutines and Functions

## Example: Final Purchase Price

This is a subroutine that outputs the tax and final price of a purchase when passed the purchase amount.

You may assume that the tax rate is stored in a constant called `sngSalesTaxRate`.

```
Private Sub PurchasePrice(ByVal dblPurchasePrice As Double)
    Dim dblTax As Double
    'sngSalesTaxRate is declared as a constant earlier in the program

    dblTax = dblPurchasePrice * sngSalesTaxRate
    MsgBox("Purchase Price: " & dblPurchasePrice.ToString & _
        "Sales Tax: " & dblTax.ToString & _
        "Final Price: " & (dblPurchasePrice*dblTax).ToString)
End Sub
```



# Chapter 5 – Subroutines and Functions

## 5.4 Pass By Reference and Pass By Value

Visual Basic .NET allows you to declare a function or subroutine with parameters that are either a copy (**pass by value**) or a reference (**pass by reference**) to the original value.

If you wish your application to run quickly, then passing large parameters by reference will increase performance.

The speed comes at the risk that the original value passed can be altered unexpectedly by the routine that is called.

Visual Basic .NET's parameter passing defaults to pass by value for subroutines or function calls.

Parameters like `Integers`, `Booleans`, and `Decimals` are relatively small and do not benefit from passing the parameter by reference. Passing large `String` variables and complex controls by value could waste time. Visual Basic .NET allows you to specify either pass by reference or pass by value.

Specifying a parameter as pass by value is accomplished by placing the keyword `ByVal` in front of a parameter:

```
Private Sub SubroutineName(ByVal Parameter1 As Type)
```

By placing the keyword `ByRef` in front of a parameter, the parameter will be passed by reference. If you do not specify either, it is as if you have placed a `ByVal` keyword, since that is the default.

```
Private Sub SubroutineName(ByRef Parameter1 As Type)
```

# Chapter 5 – Subroutines and Functions

## Drill 5.16

Assuming the `Mystery` function has been defined as follows, what is the output of the following code?

```
'Declare variables
Dim intTestValue As Integer
Dim intReturnValue As Integer

'Initialize intTestValue
intTestValue = 5

'Display value of intTestValue before call to Mystery
MsgBox("The value of intTestValue before the call = " & _
        intTestValue.ToString)

'Call the Mystery Function
intReturnValue = Mystery(intTestValue)

'Output the return value and intTestValue
MsgBox("The return value from the call to Mystery is " & _
        intReturnValue.ToString)
MsgBox("The value of intTestValue after the call = " & _
        intTestValue.ToString())
```

# Chapter 5 – Subroutines and Functions

## Drill 5.16 Continued

```
Private Function Mystery(ByRef intMaxNum As Integer) As Integer
Dim intValue As Integer
intValue = 0
intValue += intMaxNum
intMaxNum = intMaxNum - 1
intValue += intMaxNum
intMaxNum = intMaxNum - 1
intValue += intMaxNum
intMaxNum = intMaxNum - 1
Return intValue
End Function
```

### Answer:

The value of intTestValue before the call = 5

The return value of the call to Mystery is 12

The value of intTestValue after the call = 2

# Chapter 5 – Subroutines and Functions

## Drill 5.17

What is the output of the code in Drill 5.16 when the `intMaxNum` parameter is changed to pass by value by changing the function declaration as follows?

```
Private Function Mystery(ByVal intMaxNum As Integer) As Integer
```

Answer: 15 5

# Chapter 5 – Subroutines and Functions

## Drill 5.18

Assuming the `DrillRoutine` subroutine has been defined as follows, what is the output of the following code?

```
Dim intTestValue1 As Integer
Dim intTestValue2 As Integer

intTestValue1 = 5
intTestValue2 = 7

DrillRoutine(intTestValue1, intTestValue2)
MsgBox(Str(intTestValue1) & " " & Str(intTestValue2))

Private Sub DrillRoutine(ByVal intParam1 As Integer, _
    ByRef intParam2 As Integer)
    intParam1 = 10
    intParam2 = 20
End Sub
```

Answer: 5 20

# Chapter 5 – Subroutines and Functions

## Drill 5.19

Assuming the `DrillRoutine` subroutine has been defined as follows, what is the output of the following code?

```
Dim intTestValue1 As Integer
Dim intTestValue2 As Integer

intTestValue1 = 5
intTestValue2 = 7

DrillRoutine(intTestValue1, intTestValue2)
MsgBox(Str(intTestValue1) & " " & Str(intTestValue2))

Private Sub DrillRoutine(ByRef intParam1 As Integer, _
    ByRef intParam2 As Integer)
    intParam1 = 10
    intParam2 = 20
End Sub
```

Answer: 10 20

# Chapter 5 – Subroutines and Functions

## Drill 5.20

Assuming the `DrillRoutine` subroutine has been defined as follows, what is the output of the following code?

```
Dim intTestValue1 As Integer
Dim intTestValue2 As Integer

intTestValue1 = 5
intTestValue2 = 7

DrillRoutine(intTestValue1, intTestValue2)
MsgBox(Str(intTestValue1) & " " & Str(intTestValue2))

Private Sub DrillRoutine(ByVal intParam1 As Integer, _
    ByVal intParam2 As Integer)
    intParam1 = 10
    intParam2 = 20
End Sub
```

Answer: 5 7

# Chapter 5 – Subroutines and Functions

## Drill 5.21

Assuming the `DrillRoutine` subroutine has been defined as follows, what is the output of the following code?

```
Dim intTestValue1 As Integer
Dim intTestValue2 As Integer

intTestValue1 = 5
intTestValue2 = 7

DrillRoutine(intTestValue1, intTestValue2)
MsgBox(Str(intTestValue1) & " " & Str(intTestValue2))

Private Sub DrillRoutine(intParam1 As Integer, intParam2 As Integer)
    intParam1 = 10
    intParam2 = 20
End Sub
```

Answer: 5 7



# Chapter 5 – Subroutines and Functions

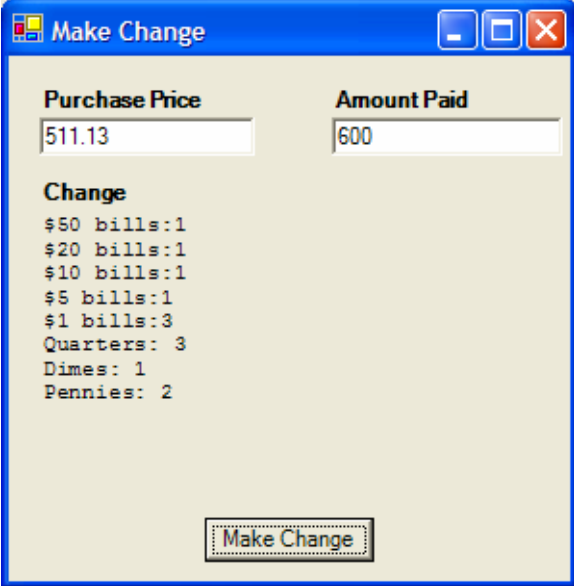
## Example: Compute Change Application Utilizing Pass By Reference

### Problem Description

Develop an application that will accept as input a purchase price and an amount paid.

The application will output the change due to the person in the highest denominations of the bills and change.

Assume that you can give change in denominations of \$100, \$50, \$20, \$10, \$5, \$1, 25 cents, 10 cents, 5 cents, and 1 cent.



The screenshot shows a Windows-style application window titled "Make Change". It has two input fields: "Purchase Price" with the value "511.13" and "Amount Paid" with the value "600". Below these is a section titled "Change" which lists the following denominations and counts: "\$50 bills:1", "\$20 bills:1", "\$10 bills:1", "\$5 bills:1", "\$1 bills:3", "Quarters: 3", "Dimes: 1", and "Pennies: 2". At the bottom center of the window is a button labeled "Make Change".

# Chapter 5 – Subroutines and Functions

## Problem Discussion

To develop an elegant solution to this problem you need pass by reference.

The solution requires figuring out the total number of each denomination you can return.

As each number of bills or coins is determined, you must account for the corresponding amount so that you do not count the same money multiple times.

The easiest way to accomplish this is to write a single function that computes the maximum amount of the denomination passed that can be returned.

Once the number is computed, the corresponding amount can be reduced from the remaining money.

# Chapter 5 – Subroutines and Functions

## Problem Solution

You are going to break your problem into sections.

The first section passes the text box controls' values to the `MakeChange` subroutine that will start the actual work of the application.

```
Private Sub btnMakeChange_Click(...  
    MakeChange(Val(txtPurchasePrice.Text), Val(txtAmountPaid.Text))  
End Sub
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

The `MakeChange` subroutine will be the *driver* of the application. It accepts a purchase price and the amount paid. It outputs the correct change. Instead of complicating `MakeChange`, you call a support function called `PrintBills`.

```
Private Sub MakeChange(ByVal dblPurchasePrice As Double, _
    ByVal dblAmountPaid As Double)
    'Variable to store the amount of change remaining
    Dim dblTotalChange As Double
    'Compute initial amount of change
    dblTotalChange = dblAmountPaid - dblPurchasePrice
    'Determine how many hundred dollar bills should be returned
    PrintBills(dblTotalChange, 100)
    'Determine how many fifty dollar bills should be returned
    PrintBills(dblTotalChange, 50)
    'Determine how many twenty dollar bills should be returned
    PrintBills(dblTotalChange, 20)
    'Determine how many ten dollar bills should be returned
    PrintBills(dblTotalChange, 10)
    'Determine how many five dollar bills should be returned
    PrintBills(dblTotalChange, 5)
    'Determine how many dollar bills should be returned
    PrintBills(dblTotalChange, 1)
    'Determine how many quarters should be returned
    PrintBills(dblTotalChange, 0.25)
    'Determine how many dimes should be returned
    PrintBills(dblTotalChange, 0.1)
    'Determine how many nickels should be returned
    PrintBills(dblTotalChange, 0.05)
    'Determine how many pennies should be returned
    PrintBills(dblTotalChange, 0.01)
End Sub
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

`PrintBills` calculates the largest number of the current denomination by converting the result of the division to an `Integer`.

Since the calculation returns an `Integer` value, it can be used directly as the amount of that denomination.

```
Private Sub PrintBills(ByRef dblTChange As Double, ByVal
    dblDenomination As Double)
    'Variable to store the number of bills of the current_
    denomination returned
    Dim intNumBills As Integer

    'Compute the number of bills
    intNumBills = Int(dblTChange / dblDenomination)

    'Compute the amount of change remaining
    dblTChange = dblTChange - dblDenomination * intNumBills

    'If there is at least one bill/coin then output the amount
    If (intNumBills > 0) Then
        'If the denomination is a bill (1, 5, 10, 20, 50, or 100)
        If (dblDenomination >= 1) Then
            'Output the bill information
            lblChange.Text &= "$" & dblDenomination & " bills:" & intNumBills
        End If
    End If
End Sub
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

PrintBills continued:

```
'Otherwise the denomination is a coin (penny, nickel, dime, quarter)
Else
    'Check if it is quarter
    If (dblDenomination = 0.25) Then
        lblChange.Text &= "Quarters: "
    'Check if it is dime
    ElseIf (dblDenomination = 0.1) Then
        lblChange.Text &= "Dimes: "
    'Check if it is nickel
    ElseIf (dblDenomination = 0.05) Then
        lblChange.Text &= "Nickels: "
    'Otherwise there are pennies
    Else
        lblChange.Text &= "Pennies: "
    'Otherwise it is more than one penny
    End If

    'Add the number of coins
    lblChange.Text &= intNumBills.ToString
End If
'Add a new line to the output
lblChange.Text &= vbCrLf
End If
End Sub
```

# Chapter 5 – Subroutines and Functions

## 5.5 Case Study

### Problem Description

Modify the case study from Chapter 4 so that instead of displaying a weekly pay, you display a gross pay, tax, and net pay for each employee.

Display a total gross pay, tax, and net pay for all the employees.

Also format the monetary values to appear as currency values. Compute the tax at a single rate of 28%.

The screenshot shows a window titled "Payroll Accounting System" with a small icon of a person. The window contains a table with the following data:

Employee Name	Hours Worked	Department	Gross Pay	Tax	Net Pay
Jeff Salvage	10	Sales	\$250.00	\$70.00	\$180.00
John Cunningham	20	Sales	\$500.00	\$140.00	\$360.00
Tim Burke	10	Processing	\$150.00	\$42.00	\$108.00
Suzanne Pepito	20	Processing	\$300.00	\$84.00	\$216.00
Total			\$1,200.00	\$336.00	\$864.00

Below the table is a "Calculate" button.

Fig 05-04

# Chapter 5 – Subroutines and Functions

## Problem Description

Since the calculations required are the same for each employee, you can write one routine that accepts the text boxes for each employee as parameters.

You do not need to pass the employee name text box to the subroutine, since it is not used in the calculations.



# Chapter 5 – Subroutines and Functions

## Problem Solution

You need to remove the text boxes from Chapter 4's solution for weekly pay and add text boxes for gross pay, tax, and net pay.

You need one for each of the four employees.

You also need variables to store the total gross pay, total tax, and total net pay.

You can place these in the common declarations section of code along with the constants.

This code follows:

```
'Constants to Hold Pay Rates and Tax Rate

Const intSalesPayRate = 25
Const intProcessingPayRate = 15
Const intManagementPayRate = 50
Const intPhonePayRate = 10
Const sngTaxRate = 0.28

'Temporary Variables to Store Calculations
Dim dblTmpTotalGross As Double
Dim dblTmpTotalTax As Double
Dim dblTmpTotalNet As Double
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

Next you must add a subroutine to compute the gross pay, tax, and net pay for a single employee. You need to pass it the number of hours worked and the department so that you can compute the necessary values. Additionally, you need to pass it the three text boxes that will store the results. The code follows:

```
Private Sub ComputePay(ByVal txtHours As TextBox, ByVal txtDept _
    As TextBox, ByVal txtGross As TextBox,
    ByVal txtTax As TextBox, ByVal txtNet As TextBox)
    Dim dblGrossPay As Double 'Stores the calculated gross value
    Dim dblTax As Double 'Stores the calculated tax
    Dim dblNetPay As Double 'Stores the calculated net pay

    Select Case txtDept.Text
        Case "Sales"
            dblGrossPay = Val(txtHours.Text) * intSalesPayRate
        Case "Processing"
            dblGrossPay = Val(txtHours.Text) * intProcessingPayRate
        Case "Management"
            dblGrossPay = Val(txtHours.Text) * intManagementPayRate
        Case "Phone"
            dblGrossPay = Val(txtHours.Text) * intPhonePayRate
        Case Else
            MsgBox("Error in input")
            dblGrossPay = 0
    End Select
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

ComputePay continued:

```
dblTax = dblGrossPay * sngTaxRate 'Compute Tax
dblNetPay = dblGrossPay - dblTax 'Compute Net Pay

'Format Output
txtGross.Text = FormatCurrency(dblGrossPay)
txtTax.Text = FormatCurrency(dblTax)
txtNet.Text = FormatCurrency(dblNetPay)

'Add to totals
dblTmpTotalGross += dblGrossPay
dblTmpTotalTax += dblTax
dblTmpTotalNet += dblNetPay
End Sub
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

With the addition of the `ComputePay` subroutine, your `btnCalculate_Click` event becomes easier.


You pass the proper parameters for each of the four employees, and then you copy the totals to their respective text boxes.

```
Private Sub btnCalculate_Click(...  
  
    'First Person's Calculations  
    ComputePay(txtHours1, txtDept1, txtGross1, txtTax1, txtNet1)  
  
    'Second Person's Calculations  
    ComputePay(txtHours2, txtDept2, txtGross2, txtTax2, txtNet2)  
  
    'Third Person's Calculations  
    ComputePay(txtHours3, txtDept3, txtGross3, txtTax3, txtNet3)  
  
    'Fourth Person's Calculations  
    ComputePay(txtHours4, txtDept4, txtGross4, txtTax4, txtNet4)  
  
    'Copy the Totals to their TextBoxes  
    txtTotalGross.Text = FormatCurrency(dblTmpTotalGross)  
    txtTotalTax.Text = FormatCurrency(dblTmpTotalTax)  
    txtTotalNet.Text = FormatCurrency(dblTmpTotalNet)  
  
End Sub
```

# Chapter 5 – Subroutines and Functions

## Problem Solution Continued

Your final application should look like this:



Employee Name	Hours Worked	Department	Gross Pay	Tax	Net Pay
Jeff Salvage	10	Sales	\$250.00	\$70.00	\$180.00
John Cunningham	20	Sales	\$500.00	\$140.00	\$360.00
Tim Burke	10	Processing	\$150.00	\$42.00	\$108.00
Suzanne Pepito	20	Processing	\$300.00	\$84.00	\$216.00
<b>Totals</b>			\$1,200.00	\$336.00	\$864.00

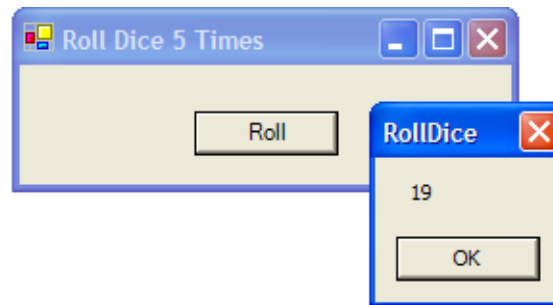
# Chapter 5 – Subroutines and Functions

## Coach's Corner

### Stepping Over Code in Debugger

When you step through your application, you have a choice of whether you wish to step through every line of code or skip over code like function calls.

Observe the following simple application. It contains a button to roll a die five times.



# Chapter 5 – Subroutines and Functions

## Stepping Over Code in Debugger Continued

The following code calls the `RollDice` function and returns the total value of the die roll five times.

```
Private Sub btnRoll_Click(...
    'Variable to store 5 rolls of the dice
    Dim intTotal As Integer

    'Call the RollDice function and store the results
    intTotal = RollDice()

    'Output the result
    MsgBox(Str(intTotal))
End Sub

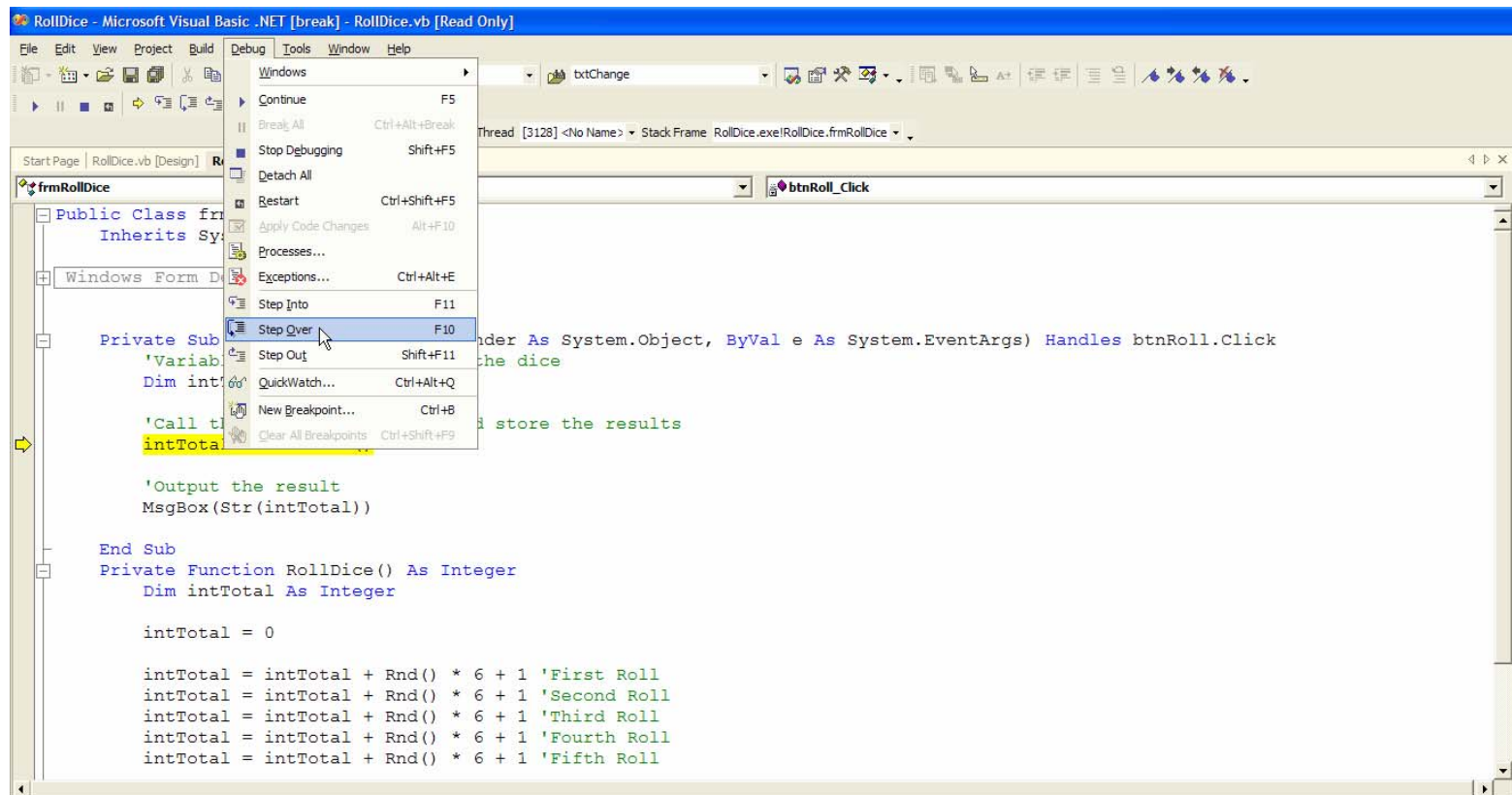
Private Function RollDice() As Integer
    Dim intTotal As Integer
    intTotal = 0
    intTotal += Rnd() * 6 + 1 'First Roll
    intTotal += Rnd() * 6 + 1 'Second Roll
    intTotal += Rnd() * 6 + 1 'Third Roll
    intTotal += Rnd() * 6 + 1 'Fourth Roll
    intTotal += Rnd() * 6 + 1 'Fifth Roll

    Return intTotal
End Function
```

# Chapter 5 – Subroutines and Functions

## Stepping Over Code in Debugger Continued

While you could step through this example by pressing the <F11> key or selecting Step Into from the Debug menu, you can skip the tracing of the function call by either hitting the <F10> key or selecting Step Over from the Debug menu.





# Chapter 5 – Subroutines and Functions

## Conditional Statements with Function Calls

If a function call is made within a conditional expression, all of the conditional expressions are evaluated regardless of whether or not a condition can be short-circuited. Visual Basic .NET does this because usually if a function call is embedded in a conditional statement, the desire was for the function to be called consistently.

The following application computes a homework average and then determines whether a student passes.

Input	Output
Midterm Grade: 89	Final Exam Grade: 92
Homework Grades: 60, 89, 75	Homework Average: 74.67
	Final Grade: PASS

Compute Final Grade

# Chapter 5 – Subroutines and Functions

## Conditional Statements with Function Calls Continued

To implement this grading scheme, the following code is used:

```
Private Sub butCompute_Click(...
    'If the midterm grade or the Final exam grade is >= 65
    'ComputeHomeworkAverage will still be called
    If (Val(txtMidtermGrade.Text) >= 65) Or
        (Val(txtFinalExamGrade.Text) >= 65)
        Or (ComputeHomeworkAverage()) Then
        lblFinalGrade.Text = "PASS"
    Else
        lblFinalGrade.Text = "FAIL"
    End If
End Sub

Private Function ComputeHomeworkAverage() As Boolean
    'Declare variable to store the homework average
    Dim sngHomeworkAverage As Single
    'Compute the homework average
    sngHomeworkAverage = (Val(txtHomeworkGrade1.Text) +
        Val(txtHomeworkGrade2.Text) +
        Val(txtHomeworkGrade3.Text)) / 3
    'Output the homework average formatted to 2 decimal places
    lblHomeworkAverage.Text = Format(sngHomeworkAverage, "0.00")
    'return true if the homework average is passing
    If (sngHomeworkAverage >= 65) Then
        Return True
    Else
        Return False
    End If
End Function
```

# Chapter 5 – Subroutines and Functions

## Conditional Statements with Function Calls Continued

With short circuit analysis you would expect that once the midterm grade was evaluated as greater than or equal to 65 it would not evaluate the remaining conditions.

However, the final condition is the evaluation of the return value of a function.

Since a function is called within the condition, all the conditions are evaluated.

If the final condition was not evaluated, then the `ComputeHomeworkAverage` function would not be called and the label `lblHomeworkAverage` would not contain the average of the homework.