Programming with Microsoft Visual Basic.NET

Lesson 4

What have we learnt in last lesson?

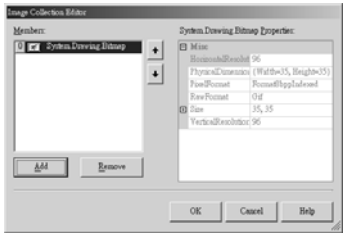
- Introduction to common controls: Label, Textbox, Picture Box, List Box, Combo Box, Check Box, Radio Button, Progress Bar, Button
- Event Handling for Timer
- Coding for Controls

More about Controls

Toolbar, Tab, Tree View

Image List

- The Image List component is used to store images, which can then be displayed by controls include: the ListView, TreeView, ToolBar, TabControl, Button, CheckBox, RadioButton, and Label controls.
- To associate the image list with a control, set the control's ***ImageList*** property to the name of the ImageList component.
- The key property of the ImageList component is Images, which contains the pictures to be used by the associated control. Each individual image can be accessed by its index value.
- Images that are larger will be scaled to fit.



Toolbar

- The Tool Bar control is used on forms as a control bar that displays a row of drop-down menus and bitmapped buttons that activate commands. The buttons can be configured to appear and behave as pushbuttons, drop-down menus, or separators.
- Typically, a toolbar contains buttons and menus that correspond to items in an application's menu structure, providing quick access to an application's most frequently used functions and commands.
- A ToolBar control is usually "docked" along the top of its parent window, but it can also be docked to any side of the window.
- A toolbar can display tooltips when the user points the mouse pointer at a toolbar button.

Tab Control

- The Tab Control displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet.
- The tabs can contain pictures and other controls.
- You can use the tab control to produce the kind of multiple-page dialog box
- The most important property of the TabControl is ***TabPages***, which contains the individual tabs. When a tab is clicked, it raises the Click event for that TabPage object.

Tree View

- The Tree View control displays a hierarchy of nodes, like the way files and folders are displayed in the left pane of Windows Explorer.
- Each node might contain other nodes, called **Child Nodes**.
- **Parent nodes**, or nodes that contain child nodes, can be displayed as expanded or collapsed.
- The key properties of the TreeView control are ***Nodes*** and ***SelectedNode***.
  - The Nodes property contains the list of top-level nodes in the tree.
  - The SelectedNode property sets the currently selected node.

ToolTips

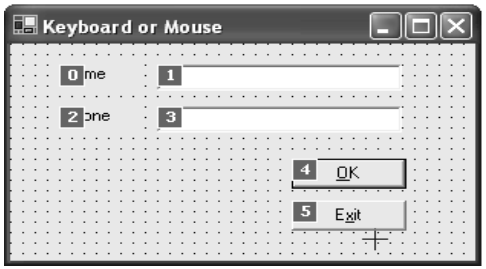
- ToolTips is a small label that is displayed when user pauses mouse pointer over a control.
- Steps for creating ToolTips
  - Add a ToolTip Control to Form
    - Appears in the **Component Tray** pane at bottom of Form Designer where non-display controls are shown
  - Select **ToolTips on ToolTip1** property of each control and add Tool Tip comments

Tab Index and Focus

Setting Tab Order and Control Focus

Tab Stop

- One control on a Form always has the focus
- Not all control types can receive the focus
- **TabStop** Property (applicable only for controls that are capable of receiving the focus)
  - Designates whether a control is allowed to receive the focus; set to **True** or **False**



Tab Index Property

- **TabIndex property**:
  - Determines the order in which a control receives the focus when the Tab key is pressed
  - Starts at 0
  - Assigned by default as the order in which controls are added to the form at design time
  - Should be set to the order in which the user will want to access the controls
- **Focus**: the state of being able to accept user input
- Set TabIndex using the Properties window or the Tab Order option on the View menu

Tab Order

- User should be able to use Tab key to move the focus through a form in an organized manner; top to bottom, left to right
- TabIndex Property
  - 0 for first control to receive the focus when the form loads
  - Number in tab sequence

Get the Focus for a Control

- When a control gains the focus, the even **Control.Enter** will be triggered.
- In order to force focus to a control, **Control.Focus( )** can be used.

Subroutine

Procedure & Functions

Subroutine

- Often you will encounter programming situations in which multiple procedures perform the same operation.
- The condition can occur when the user can select either a button or a menu item to do the same thing.
- Rather than re-typing the code, you can write reusable code in a general procedure and call it from both event procedures.

Declaration of Subroutine

- There are two ways to create a reusable subroutine and call or use it in the event procedures:
  - The first one is a **Procedure** that performs actions; we will use keywords **Private Sub** and **End Sub** for creating procedures.
  - The second one is a **Function** that performs actions and returns a value. We will use **Private Function** and **End Function** for creating functions.

```
Private Sub ProcedureName ( )
        ' Statements to execute
End Sub
```

```
Private Function FunctionName ( ) As Datatype
        ' Statements to execute
End Function
```

You must specify a Data type for a function since a function always returns a value.

Call Subroutine

- To call a subroutine you just have to use the name of the procedure.
- This is true for all procedure and functions.
- The difference in between sub procedure and function is that a Function always returns a value.

```
' Call a Procedure
Call ProcedureName (parameters)
```

```
' Call a Function and Get the Return
ReturnValue = FunctionName(parameters)
```


Passing Values to Subroutine

- Declare Variables as module level or local variable (service providing procedure)
- Pass the value of the variable to the Called procedure side (service request procedure)
- Name of local variable does not need to match name in Sub Procedure argument list
- Number of arguments and order must match
  - ByVal (default) – Call By Value
    - Sends a copy, original cannot be modified
  - ByRef – Call By Reference
    - Sends a reference to the memory location where the original is stored and therefore the original can be altered

Example: Calling a Procedure

General Format: Procedure Call

The procedure call is made:

```
ProcedureName()
```

The **procedure declaration** that begins the Sub procedure has the form:

```
Private Sub ProcedureName()
        ' Line(s) of code
End Sub
```


Example: Calling a Function

General Format: Function Procedure Call

The Function procedure call is made:

```
VariableName = FunctionProcedureName()
```

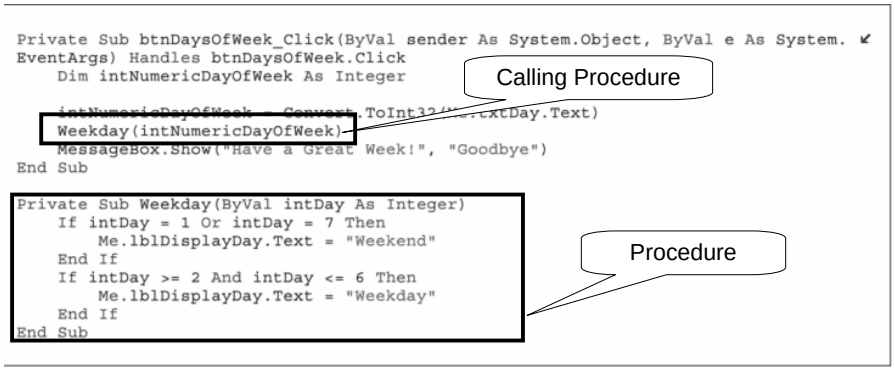
The **procedure declaration** that begins the Function procedure has the form:

```
Private Function FunctionProcedureName() as DataType
        ' Line(s) of code
        Return VariableName <------ Expression...
End Function
```

Note: A Function Procedure may accept parameters in its parameter list as well.

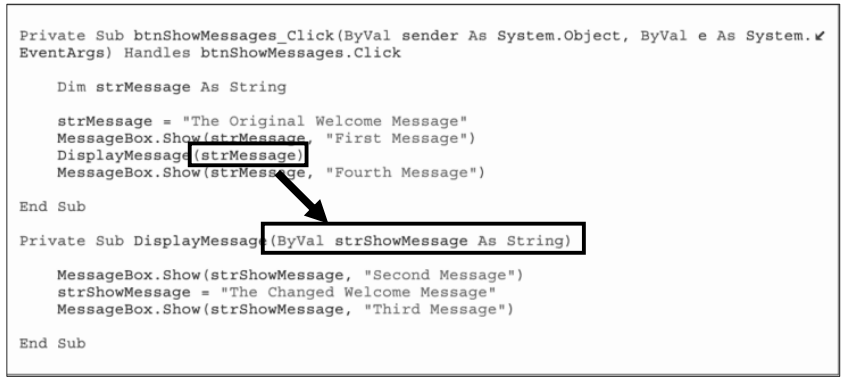
Passing Arguments

- When a procedure is called, however, the **call** statement can pass an argument to the called procedure

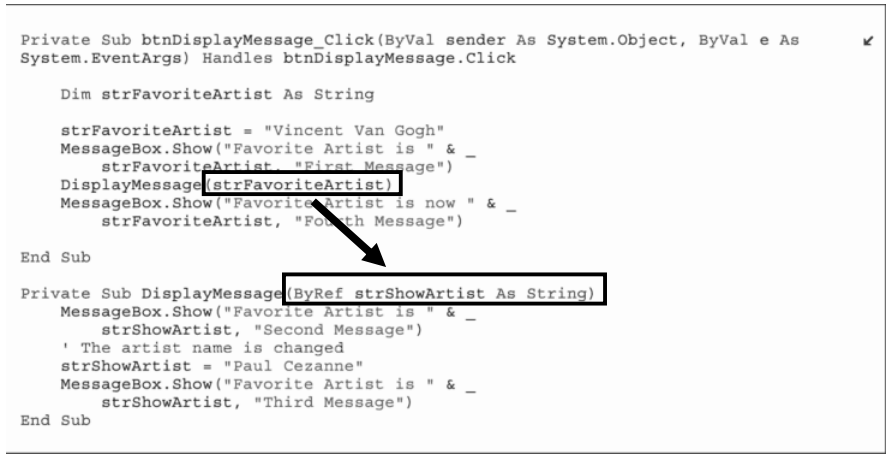


Passing Arguments by Value (ByVal)

- The value is copied into a variable whose name is specified in the Sub procedure declaration statement



Passing Arguments by Reference (ByRef



Input Box Function

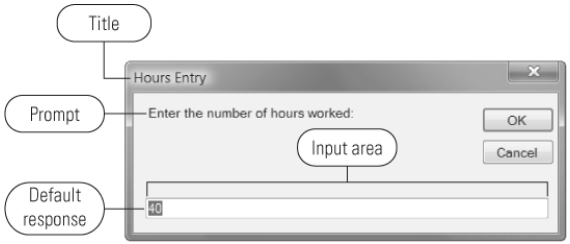
Capture User Input from a Dialog Box

The InputBox Function

- InputBox function displays a predefined dialog box that allows the user to enter data
  - Contains a text message, an **[OK]** button, a **[Cancel]** button, and an input area
- InputBox function returns:
  - The user's entry if the user clicks the **[OK]** button
  - An empty string if the user clicks the **[Cancel]** button or the Close button on the title bar

The InputBox Function Parameters

- InputBox function parameters:
  - Prompt: the message to display inside the dialog box
  - Title: the text to display in the dialog box's title bar
  - Default Response: a prefilled value for the user input



How to use the InputBox function?

USE THE INPUTBOX FUNCTION

Syntax

InputBox(*prompt*[**,** *title*][**,** *defaultResponse*]**)**

Example 1

```
firstName = InputBox("Enter your first name:")
```

Displays a dialog box that shows "Enter your first name:" as the prompt, the project's name as the title, and an empty input area. Assigns the user's response to a String variable named **firstName**.

Example 2

```
city = InputBox("City name:", "City")
```

Displays a dialog box that shows "City name:" as the prompt, "City" as the title, and an empty input area. Assigns the user's response to a String variable named **city**.

Example 3

```
state = InputBox("State name:", "State", "Alaska")
```

Displays a dialog box that shows "State name:" as the prompt, "State" as the title, and "Alaska" in the input area. Assigns the user's response to a String variable named **state**.

Example 4

```
state = InputBox("State name:",, "Alaska")
```

Displays a dialog box that shows "State name:" as the prompt, the project's name as the title, and "Alaska" in the input area. Assigns the user's response to a String variable named **state**.

Example 5

```
Const InputPrompt As String = "Enter the rate:"
Const InputTitle As String = "Rate Entry"
rate = InputBox(InputPrompt, InputTitle, "0.0")
```

Displays a dialog box that shows the contents of the **InputPrompt** constant as the prompt, the contents of the **InputTitle** constant as the title, and "0.0" in the input area. Assigns the user's response to a String variable named **rate**.

String Manipulation

Common function for Handling String

String Manipulation

- Most applications need to manipulate string data in some fashion
- String properties and methods are used to manipulate string data

Clear Text

- Clear contents of text boxes and labels
  - Set ***Text*** Property equal to an Empty String
  - Empty String is 2 quotation marks with no space between them ("")
  - Use the ***Clear*** Method of a Text Box or set ***Text*** property to ***String.Empty***
    - TextBox.Text = ""
    - TextBox.Clear( )
    - Label.Text = String.Empty

Determining the Number of Characters Contained in a String

USE THE LENGTH PROPERTY

Syntax	Purpose
string.**Length**	stores the number of characters contained in a string

Example 1

```
Dim numChars As Integer
Dim fullName As String = “Paul Blackfeather”
numChars = fullName.Length
```

assigns the number 17 to the numChars variable

Example 2

```
Dim phone As String = String.Empty
phone = InputBox(“10-digit phone number”, “Phone”)
Do Until phone.Length = 10
    phone = InputBox(“10-digit phone number”, “Phone”)
Loop
```

gets a phone number from the user until the phone number contains exactly 10 characters

Trimming Characters from a String

USE THE TRIMSTART, TRIMEND, AND TRIM METHODS

Syntax	Purpose
string.**TrimStart**[(*trimChars*)]	removes characters from the beginning of a string
string.**TrimEnd**[(*trimChars*)]	removes characters from the end of a string
string.**Trim**[(*trimChars*)]	removes characters from both the beginning and end of a string

Example 1

```
Dim fullName As String
fullName = nameTextBox.Text.TrimStart
```

assigns the contents of the nameTextBox's Text property, excluding any leading spaces, to the fullName variable

Example 2

```
Dim rate As Decimal
Decimal.TryParse(rateTextBox.Text.TrimEnd("%"c), rate)
```

converts the contents of the rateTextBox, excluding any trailing percent signs, to Decimal and then assigns the result to the rate variable

Example 3

```
Dim fullName As String = String.Empty
fullName = nameTextBox.Text.Trim
```

assigns the contents of the nameTextBox, excluding any leading and trailing spaces, to the fullName variable

Example 4

```
Dim userEntry As String = String.Empty
userEntry = InputBox("Number:", "Entry").Trim("$"c, " "c, "%"c)
```

assigns the user's entry, excluding any leading and trailing dollar signs, spaces, and percent signs, to the userEntry variable

Removing Characters

- Can remove one or more characters located anywhere in the string
- Returns a string with the appropriate characters removed

USE THE REMOVE METHOD

Syntax	Purpose
string.**Remove(***startIndex*, *count***)**	removes characters from anywhere in a string

Example 1

```
Dim fullName As String = "John Cober"
nameTextBox.Text = fullName.Remove(0, 5)
```

assigns the string "Cober" to the nameTextBox's Text property

Example 2

```
Dim fullName As String = "John"
nameTextBox.Text = fullName.Remove(2, 1)
```

assigns the string "Jon" to the nameTextBox's Text property

Replacing Characters in a String

- Replace method replaces a sequence of characters in a string with another sequence of characters

USE THE REPLACE METHOD

Syntax	Purpose
string.**Replace(***oldValue*, *newValue***)**	replaces all occurrences of a sequence of characters in a string with another sequence of characters

Example 1

```
Dim socialNum As String = "000-11-9999"
socialNum = socialNum.Replace("-", "")
```

assigns the string "000119999" to the socialNum variable

Example 2

```
Dim word As String = "latter"
word = word.Replace("t", "d")
```

assigns the string "ladder" to the word variable

Capture Specified Location of Character within a String

USE THE MID STATEMENT

Syntax	Purpose
Mid(*targetString*, *start* [, *count*]**)** = *replacementString*	replaces a specific number of characters in a string with characters from another string

Example 1

```
Dim fullName As String = "Rob Smith"
Mid(fullName, 7, 1) = "y"
```

changes the contents of the fullName variable to "Rob Smyth"

Example 2

```
Dim fullName As String = "Ann Johnson"
Mid(fullName, 5) = "Carl"
```

changes the contents of the fullName variable to "Ann Carlson"

Example 3

```
Dim fullName As String = "Earl Cho"
Mid(fullName, 6) = "Liverpool"
```

changes the contents of the fullName variable to "Earl Liv"

Inserting Characters in a String

USE THE PADLEFT AND PADRIGHT METHODS

Syntax	Purpose
string.**PadLeft(***length*[, *character*]**)**	pads the beginning of a string with a character until the string is a specified length
string.**PadRight(***length*[, *character*]**)**	pads the end of a string with a character until the string is a specified length

Example 1

```
Dim outputNumber As String = "42"
outputNumber = outputNumber.PadLeft(5)
```

assigns " 42" (three spaces and the string "42") to the outputNumber variable

Example 2

```
Dim netPay As Double = 767.89
Dim formattedNetPay As String
formattedNetPay = netPay.ToString("C2").PadLeft(15, "*"c)
```

assigns "********$767.89" to the formattedNetPay variable (Many companies use this type of formatted net pay on their employee paychecks, because it makes it difficult for someone to change the amount.)

Example 3

```
Dim firstName As String = "Sue"
firstName = firstName.PadRight(10)
```

assigns "Sue " (the string "Sue" and seven spaces) to the firstName variable

Insert Character to a String

USE THE INSERT METHOD

Syntax	Purpose
string.**Insert(***startIndex*, *value***)**	inserts characters within a string

Example 1

```
Dim name As String = "Rob Smith"
Dim fullName As String = String.Empty
fullName = name.Insert(4, "T. ")
```

assigns the string "Rob T. Smith" to the fullName variable

Example 2

```
Dim phone As String = "3120501111"
phone = phone.Insert(0, "(")
phone = phone.Insert(4, ")")
phone = phone.Insert(8, "-")
```

changes the contents of the phone variable to "(312)050-1111"

Searching a String: By Start/End with

USE THE STARTSWITH AND ENDSWITH METHODS

Syntax	Purpose
string.**StartsWith(***subString***)**	determines whether a specific sequence of characters occurs at the beginning of a string
string.**EndsWith(***subString***)**	determines whether a specific sequence of characters occurs at the end of a string

Example 1

```
Dim phone As String = String.Empty
phone = InputBox("10-digit phone number", "Phone")
Do While phone.StartsWith("312")
    phoneListBox.Items.Add(phone)
    phone = InputBox("10-digit phone number", "Phone")
Loop
```

determines whether the string stored in the phone variable begins with "312"; if it does, the contents of the phone variable are added to the phoneListBox and the user is prompted to enter another phone number (You also can write the Do clause in this example as Do While phone.StartsWith("312") = True.)

Example 2

```
Dim cityState As String = String.Empty
cityState = cityStateTextBox.Text.ToUpper
If cityState.EndsWith("CA") Then
    stateLabel.Text = "California customer"
End If
```

determines whether the string stored in the cityState variable ends with "CA"; if it does, the string "California customer" is assigned to the stateLabel's Text property (You also can write the If clause in this example as If cityState.EndsWith("CA") = True Then.)

Searching a String: Whether a Substring is Contained

USE THE CONTAINS METHOD

Syntax	Purpose
string.**Contains(***subString***)**	searches a string to determine whether it contains a specific sequence of characters, and then returns a Boolean value that indicates whether the characters appear within the string

Example 1

```
Dim address As String = "345 Main Street, Glendale, CA"
Dim isContained As Boolean
isContained = address.ToUpper.Contains("MAIN STREET")
```

assigns the Boolean value True to the isContained variable, because "MAIN STREET" appears in the address variable when its contents are temporarily converted to uppercase

Example 2

```
Dim phone As String = "(312) 999-9999"
If phone.Contains("(312)") Then
```

the If...Then...Else statement's *condition* evaluates to True, because "(312)" appears in the phone variable

Searching a String: Locate the Substring Position

USE THE INDEXOF METHOD

Syntax	Purpose
string.**IndexOf(***subString*[**,** *startIndex*]**)**	searches a string to determine whether it contains a specific sequence of characters, and then returns an integer that indicates the starting position of the characters within the string

Example 1

```
Dim message As String = "Have a nice day"
Dim indexNum As Integer
indexNum = message.ToUpper.IndexOf("NICE")
```

assigns the number 7 to the indexNum variable

Example 2

```
Dim message As String = "Have a nice day"
Dim indexNum As Integer
indexNum = message.IndexOf("v", 5)
```

assigns the number -1 to the indexNum variable

Accessing Characters Contained in a String

USE THE SUBSTRING METHOD

Syntax	Purpose
string.**Substring(***startIndex*[**,** *count*]**)**	accesses one or more characters contained in a string

Example 1

```
Dim fullName As String = "Peggy Ryan"
firstName = fullName.Substring(0, 5)
lastName = fullName.Substring(6)
```

assigns "Peggy" to the firstName variable, and assigns "Ryan" to the lastName variable

Example 2

```
Dim employeeNum As String = "56P34"
Dim department As String = String.Empty
department = employeeNum.Substring(2, 1)
```

assigns the letter P to the department variable

Compare two Strings

USE THE STRING.COMPARE METHOD

Syntax	Purpose
String.Compare(*string1***,** *string2*[**,** *ignoreCase*]**)**	compares two strings

Example 1

```
Dim result As Integer
result = String.Compare("Dallas", "DALLAS")
```

assigns the number –1 to the result variable, because the second character in *string1* (a) is less than the second character in *string2* (A)

Example 2

```
Dim result As Integer
result = String.Compare("Dallas", "DALLAS", True)
```

assigns the number 0 to the result variable, because both strings are equal

Example 3

```
Dim result As Integer
result = String.Compare("Dallas", "Boston")
```

assigns the number 1 to the result variable, because the first character in *string1* (D) is greater than the first character in *string2* (B)

Returns Value
0: String1 = String2
1: String1 > String2
-1: String1 < String2

Note: Numbers are less than lowercase letters, Lowercase letters are less than uppercase letters

Pattern Matching

USE THE LIKE OPERATOR

Syntax	Purpose
string **Like** *pattern*	compares two strings using pattern-matching characters

Pattern-matching characters	Matches in *string*
?	any single character
*	zero or more characters
#	any single digit (0-9)
[*charList*]	any single character in the *charList* (for example, "[AMT]" matches A, M, or T, whereas "[a-z]" matches any lowercase letter)
[!*charList*]	any single character not in the *charList* (for example, "[!a-z]" matches any character that is not a lowercase letter)

Example 1

```
isEqual = firstName.ToUpper Like "B?LL"
```

Assigns the Boolean value True to the isEqual variable when the string stored in the firstName variable begins with the letter B followed by one character and then the two letters LL; otherwise, assigns the Boolean value False to the isEqual variable. Examples of *strings* that would make the expression evaluate to True include "Bill", "Ball", "bell", and "bull". Examples of *strings* for which the expression would evaluate to False include "BPL", "BLL", and "billy".

Example 2

```
If state Like "K*" Then
```

The condition evaluates to True when the string stored in the state variable begins with the letter K followed by zero or more characters; otherwise, it evaluates to False. Examples of *strings* that would make the condition evaluate to True include "KANSAS", "Ky", and "Kentucky". Examples of *strings* for which the condition would evaluate to False include "kansas" and "ky".

Example 3

```
Do While id Like "###"
```

The condition evaluates to True when the string stored in the id variable begins with three digits followed by zero or more characters; otherwise, it evaluates to False. Examples of *strings* that would make the condition evaluate to True include "178" and "983Ab". Examples of *strings* for which the condition would evaluate to False include "X34" and "34Z5".

Example 4

```
If firstName.ToUpper Like "T[OI]M" Then
```

The condition evaluates to True when the string stored in the firstName variable is either "Tom" or "Tim" (entered in any case). When the firstName variable does not contain "Tom" or "Tim"—for example, when it contains "Tam" or "Tommy"—the expression evaluates to False.

Example 5

```
isLowercase = letter Like "[a-z]"
```

Assigns the Boolean value True to the isLowercase variable when the string stored in the letter variable is a lowercase letter; otherwise, it evaluates to False. You use a hyphen (-) to specify a range of values in a *charList*.

Example 6

```
For indexNum As Integer = 0 to userEntry.Length - 1
    If userEntry.Substring(indexNum, 1) Like "[!a-zA-Z]" Then
        nonLetter = nonLetter + 1
    End If
Next indexNum
```

compares each character contained in the userEntry variable with the lowercase and uppercase letters of the alphabet, and counts the number of characters that are not letters

Convert to Upper or Lower Case

USE THE TOUPPER AND TOLOWER METHODS

Syntax

string.**ToUpper**

string.**ToLower**

Example 1

```
If letter.ToUpper <> "P" Then
```

compares the uppercase version of the string stored in the letter variable with the uppercase letter "P"

Example 2

```
If item1.ToUpper = item2.ToUpper Then
```

compares the uppercase version of the string stored in the item1 variable with the uppercase version of the string stored in the item2 variable

Example 3

```
If "reno" = cityTextBox.Text.ToLower Then
```

compares the lowercase letters "reno" to the lowercase version of the string stored in the cityTextBox

Example 4

```
nameLabel.Text = customer.ToUpper
```

assigns the uppercase version of the string stored in the customer variable to the Text property of the nameLabel

Example 5

```
newName = newName.ToUpper
stateTextBox.Text = stateTextBox.Text.ToLower
```

changes the contents of the newName variable to uppercase, and changes the contents of the stateTextBox to lowercase

The Val Function

- The Val Function converts one or more characters to a number and then returns the number
- When an invalid character is encountered in the text argument, Val function stops the conversion process at that point

txtSales.Text value	Number returned by the Val(txtSales.Text) function
456	456
24,500	24
$56.88	0
Abc	0
Empty text box	0

Figure 6-4: Examples of the Val function

Format

Define the String Format

Format Specifications for ToString Method

- Use the format specifier to identify the format for the numeric data to be returned by the **ToString** function

Format Specifier	Format	Description	Output from the Function
General (G)	ToString("G")	Displays the numbers as is	8976.43561
Currency(C)	ToString("C")	Displays the number with a dollar sign, a thousands separator (comma), two digits to the right of the decimal and negative numbers in parentheses	$8,976.44
Fixed (F)	ToString("F")	Displays the number with 2 digits to the right of the decimal and a minus sign for negative numbers	8976.44
Number(N)	ToString("N")	Displays a number with a thousands separator, 2 digits to the right of the decimal and a minus sign for negative numbers	8,976.44
Percent(P)	ToString("P")	Displays the number multiplied by 100 with a % sign, a thousands separator, 2 digits to the right of the decimal and a minus sign for negative numbers	897,643.56%
Scientific("E")	ToString("E")	Displays the number in E-notation and a minus sign for negative numbers	8.976436E+03

Formatting a Number

FORMAT A NUMBER

Syntax

variableName.**ToString(***formatString***)**

Format specifier (Name)	Description
C or c (Currency)	displays the string with a dollar sign; if appropriate, includes a thousands separator; negative values are enclosed in parentheses
N or n (Number)	similar to the Currency format, but does not include a dollar sign, and negative values are preceded by a minus sign
F or f (Fixed-point)	same as the Number format, but does not include a thousands separator
P or p (Percent)	multiplies the value by 100 and displays the result with a percent sign; negative values are preceded by a minus sign

Example 1

```
commissionLabel.Text = commission.ToString("C2")
```

if the commission variable contains the number 1250, the statement assigns the string "$1,250.00" to the Text property of the commissionLabel

Example 2

```
totalLabel.Text = total.ToString("N2")
```

if the total variable contains the number 123.675, the statement assigns the string "123.68" to the Text property of the totalLabel

Example 3

```
rateLabel.Text = rate.ToString("P0")
```

if the rate variable contains the number .06, the statement assigns the string "6 %" to the Text property of the rateLabel

Precision Specifier

- The precision specifier is a number that is included within the quotation marks in the function call to identify the number of positions to the right of the decimal point that should be returned

Statement	Copied to Text Property of lblOutput Label Object
lblOutput = decNumericValue.ToString("C2")	$8,976.44
lblOutput = decNumericValue.ToString("C3")	$8,976.436
lblOutput = decNumericValue.ToString("F1")	8976.4
lblOutput = decNumericValue.ToString("N4")	8,976.4356
lblOutput = decNumericValue.ToString("P0")	897,644%

Financial Function

Common use Financial Function

Financial Function Summary

Action	Language element
Calculate depreciation.	DDB, SLN, SYD
Calculate future value.	FV
Calculate interest rate.	Rate
Calculate internal rate of return.	IRR, MIRR
Calculate number of periods.	NPer
Calculate payments.	IPmt, Pmt, PPmt
Calculate present value.	NPV, PV

Example for Financial Function

USE THE FINANCIAL.PMT METHOD

Syntax

Financial.Pmt(*Rate***,** *NPer***,** *PV***)**

Argument	Meaning
Rate	interest rate per period
NPer	total number of payment periods (the term)
PV	present value of the loan; in other words, the loan amount

Example 1—Calculates the annual payment for a loan of $9,000 for 3 years at 5% interest. *Rate* is .05, *NPer* is 3, and *PV* is 9000.

Method: Financial.Pmt(.05, 3, 9000)

Annual payment (rounded to the nearest cent): -3,304.88

Example 2—Calculates the monthly payment for a loan of $12,000 for 5 years at 6% interest. *Rate* is .06/12, *NPer* is 5 * 12, and *PV* is 12000.

Method: Financial.Pmt(.06/12, 5 * 12, 12000)

Monthly payment (rounded to the nearest cent): -231.99