3.3 Input and Output

Formatting Output with Format Functions

VideoNote
Formatting output

The Format functions are used to display numbers in familiar forms. Here are some examples of how numbers are converted to strings with Format functions:

FUNCTION	STRING VALUE
`FormatNumber(12345.628, 1)`	`12,345.6`
`FormatCurrency(12345.628, 2)`	`$12,345.63`
`FormatPercent(0.185, 2)`	`18.50%`

The value of FormatNumber(*n*, *r*) is the string containing the number *n* rounded to *r* decimal places and displayed with commas as thousands separators. The value of FormatCurrency(*n*, *r*) is the string consisting of a dollar sign followed by the value of FormatNumber(*n*, *r*). FormatCurrency uses the accountant's convention of denoting negative amounts with surrounding parentheses. The value of FormatPercent(*n*, *r*) is the string consisting of the number *n* displayed as a percent and rounded to *r* decimal places. With all three functions, *r* can be omitted. If so, the number is rounded to two decimal places. Strings corresponding to numbers less than one in magnitude have a zero to the left of the decimal point. Also, *n* can be a number, a numeric expression, or even a string corresponding to a number.

FUNCTION	STRING VALUE
`FormatNumber(1 + Math.Sqrt(2), 3)`	`2.414`
`FormatCurrency(-1000)`	`($1,000.00)`
`FormatPercent(".05")`	`5.00%`

Using a Masked Text Box for Input

Problems can arise when the wrong type of data is entered as input into a text box. For instance, if the user replies to the request for an age by entering "twenty-one" into a text box, the program can easily crash. Sometimes this type of predicament can be avoided by using a masked text box for input. (In later chapters, we will consider other ways of insuring the integrity of input.)

In the Toolbox, the icon for the MaskedTextBox control consists of a rectangle containing the two characters # and _. The most important property of a masked text box is the Mask property that can be used to restrict the characters entered into the box. Also, the Mask property can be used to show certain characters in the control—to give users a visual cue that they should be entering a phone number or a social security number, for example. Some possible settings for the Mask property are shown in Table 3.2. The first four settings can be selected from a list of specified options. The last three settings generalize to any number of digits, letters, or ampersands. If the Mask property is left blank, then the MaskedTextBox control is nearly identical to the TextBox control.

TABLE 3.2 Some settings for the Mask property.

Setting	Effect
000-00-0000	The user can enter a social security number.
000-0000	The user can enter a phone number (without an area code).
(000)000-0000	The user can enter a phone number (with an area code).
00/00/0000	The user can enter a date.
0000000	The user can enter a positive integer consisting of up to 7 digits.
LLLLL	The user can enter a string consisting of up to 5 letters.
&&&&&&&&	The user can enter a string consisting of up to 8 characters.

Suppose a form contains a masked text box whose Mask property has the setting 000-00-0000. When the program is run, the string "___-__-____" will appear in the masked text box. The user will be allowed to type a digit in place of each of the nine underscore characters. The hyphens cannot be altered, and no characters can be typed anywhere else in the masked text box.

At run time, the characters 0, L, and & in the setting for a Mask property are replaced by underscore characters that are place holders for digits, letters, and characters, respectively. (Spaces are also allowed. However, trailing spaces are dropped.) When the characters "-", "(", ")", or "/" appear in a setting for a Mask property, they appear as themselves in the masked text box and cannot be altered. There are some other mask settings, but these seven will suffice for our purposes.

Figure 3.9(a) shows a masked text box during design time. It looks like an ordinary text box. However, the Tasks button for the masked text box is used to set the Mask property rather than the Multiline property. Figure 3.9(b) shows the result of clicking on the Tasks button. Then, clicking on "Set Mask . . . " brings up the Input Mask dialog box shown in Figure 3.10. (This input dialog box is the same input dialog box that is invoked when you click on the

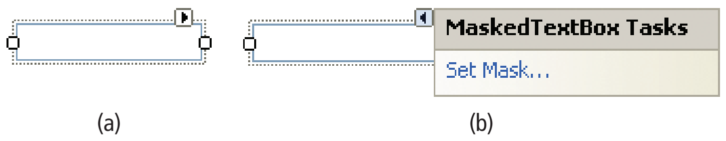
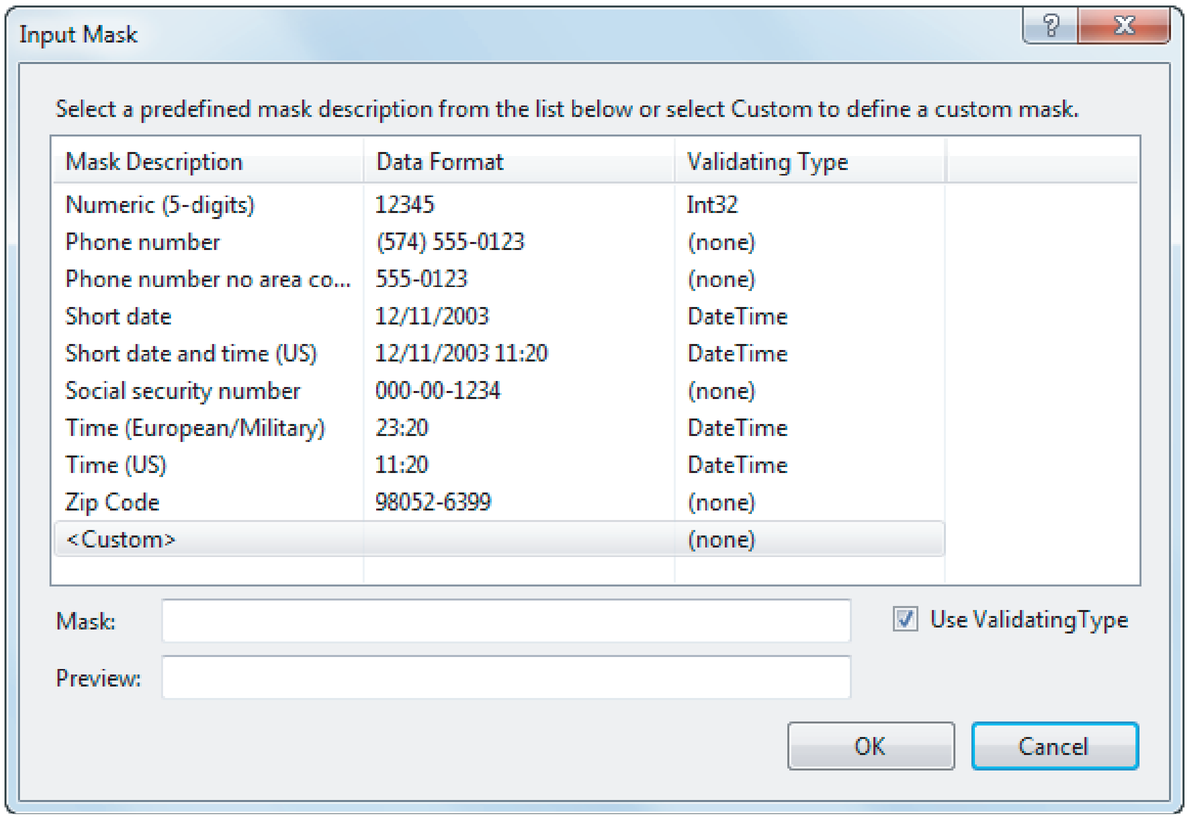


FIGURE 3.9 The Masked TextBox control.



Select a predefined mask description from the list below or select Custom to define a custom mask.

Mask Description	Data Format	Validating Type
Numeric (5-digits)	12345	Int32
Phone number	(574) 555-0123	(none)
Phone number no area co...	555-0123	(none)
Short date	12/11/2003	DateTime
Short date and time (US)	12/11/2003 11:20	DateTime
Social security number	000-00-1234	(none)
Time (European/Military)	23:20	DateTime
Time (US)	11:20	DateTime
Zip Code	98052-6399	(none)
<Custom>		(none)

FIGURE 3.10 Input dialog box used to set the Mask property of a masked text box.

ellipses in the Mask property's Settings box.) You can use this input dialog box to select a commonly used value for the Mask property, or create your own customized mask in the Mask text box. To produce the settings 00/00/0000 and 000-00-0000, click on "Short date" and "Social security number", respectively. We use the prefix *mtb* for the names of masked text boxes.

Dates as Input and Output

So far, all input and output has been either numbers or strings. However, applications sometimes require dates as input and output. Visual Basic has a Date data type and a Date literal.

A variable of type Date is declared with a statement of the form

```
Dim varName As Date
```

Just as string literals are written surrounded by quotation marks, date literals are written surrounded by number signs. For instance, the statement

```
Dim dayOfIndependence As Date = #7/4/1776#
```

declares a date variable and assigns a value to it.

The function CDate converts a string to a date. For instance, the statement

```
Dim d As Date = CDate(txtBox.Text)
```

assigns the contents of a text box to a variable of type Date.

Dates can be formatted with the FormatDateTime function. If *dateVar* is a variable of type Date, then the value of

```
FormatDateTime(dateVar, DateFormat.LongDate)
```

is a string consisting of the date specified by *dateVar* with the day of the week and the month spelled out. For instance, the two lines of code

```
Dim dayOfIndependence As Date = #7/4/1776#
txtBox.Text = FormatDateTime(dayOfIndependence, DateFormat.LongDate)
```

display Thursday, July 04, 1776 in the text box. ***Note:*** If `DateFormat.LongDate` is replaced with `DateFormat.ShortDate`, 7/4/1776 will be displayed in the text box.

There are many functions involving dates. Two very useful ones are Today and DateDiff. The value of

```
Today
```

is the current date as determined by the computer system's clock. The value of

```
DateDiff(DateInterval.Day, d1, d2)
```

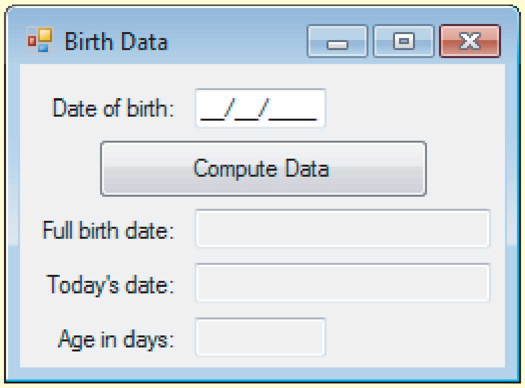
is the number of days between the two dates.

AddYears is a useful method for working with dates. If *d* is a variable of type Date, and *n* is an integer, then the value of

```
d.AddYears(n)
```

is the value of *d* advanced by *n* years. Two similar methods are AddDays and AddMonths.

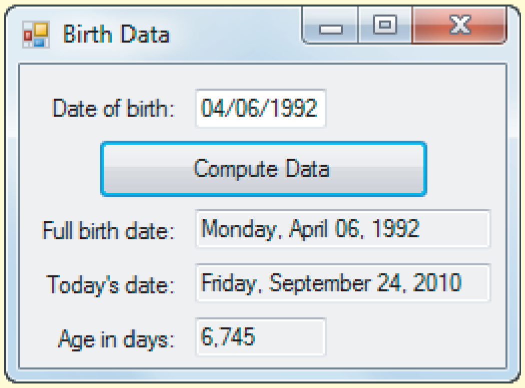
Example 1 The following program gives information pertaining to a date input by the user. The mask for the masked text box can be set by clicking on *Short date* in the Input Mask dialog box.



OBJECT	PROPERTY	SETTING
frmAge	Text	Birth Data
lblDayOfBirth	Text	Date of birth:
mtbDayOfBirth	Mask	00/00/0000
btnCompute	Text	Compute Data
lblFullDate	Text	Full birth date:
txtFullDate	ReadOnly	True
lblToday	Text	Today's date:
txtToday	ReadOnly	True
lblAgeInDays	Text	Age in days:
txtAgeInDays	ReadOnly	True

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim d As Date = CDate(mtbDayOfBirth.Text)
  txtFullDate.Text = FormatDateTime(d, DateFormat.LongDate)
  txtToday.Text = FormatDateTime(Today, DateFormat.LongDate)
  txtAgeInDays.Text = FormatNumber(DateDiff(DateInterval.Day, d, Today), 0)
End Sub
```

[Run, enter your birthday, and click on the button. One possible outcome is the following.]



Date of birth: 04/06/1992

Full birth date: Monday, April 06, 1992

Today's date: Friday, September 24, 2010

Age in days: 6,745

Getting Input from an Input Dialog Box

Normally, a text box is used to obtain input, where the type of information requested is specified in a label adjacent to the text box. Sometimes, we want just one piece of input and would rather not have a text box and label stay on the form permanently. The problem can be solved with an **input dialog box**. When a statement of the form

```
stringVar = InputBox(prompt, title)
```

is executed, an input dialog box similar to the one shown in Fig. 3.11 pops up on the screen. After the user types a response into the text box at the bottom of the dialog box and presses Enter (or clicks OK), the response is assigned to the string variable. The *title* argument is optional and provides the text that appears in the Title bar. The *prompt* argument is a string that tells the user what information to type into the text box.

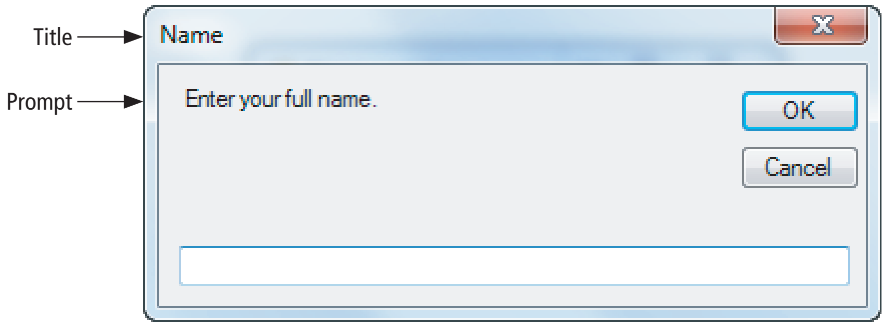
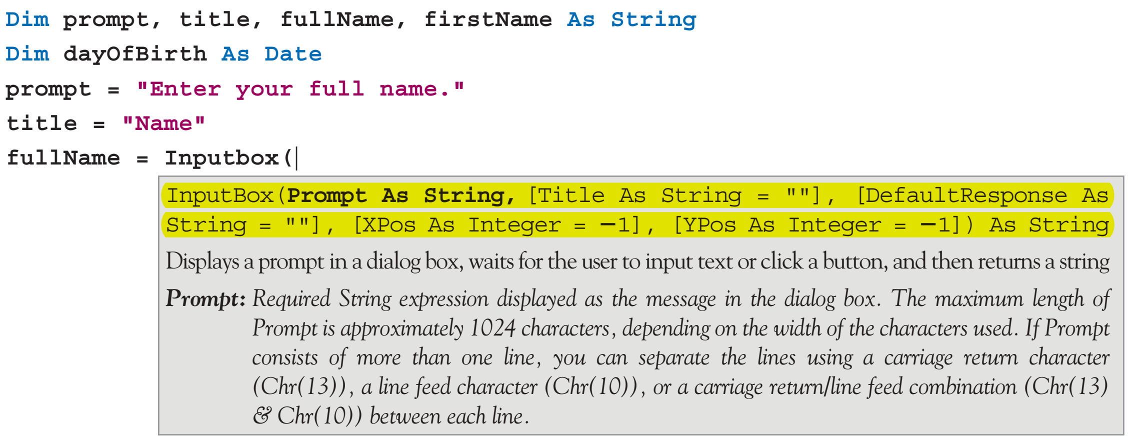


FIGURE 3.11 **Sample input dialog box.**

When you type the opening parenthesis following the word InputBox, the Code Editor displays a line containing the general form of the InputBox statement. See Fig. 3.12. This feature of IntelliSense is called **Parameter Info**. Optional parameters are surrounded by square brackets. All the parameters in the general form of the InputBox statement are optional except for *prompt*.



```
Dim prompt, title, fullName, firstName As String
Dim dayOfBirth As Date
prompt = "Enter your full name."
title = "Name"
fullName = Inputbox(|
```

```
InputBox(Prompt As String, [Title As String = ""], [DefaultResponse As
String = ""], [XPos As Integer = -1], [YPos As Integer = -1]) As String
```

Displays a prompt in a dialog box, waits for the user to input text or click a button, and then returns a string

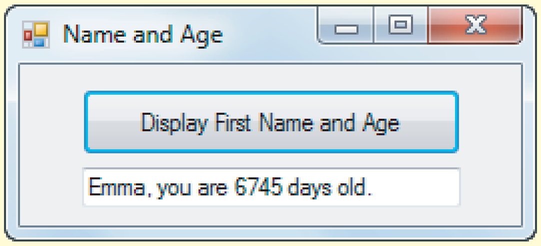
Prompt: *Required String expression displayed as the message in the dialog box. The maximum length of Prompt is approximately 1024 characters, depending on the width of the characters used. If Prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a line feed character (Chr(10)), or a carriage return/line feed combination (Chr(13) & Chr(10)) between each line.*

FIGURE 3.12 **Parameter Info feature of IntelliSense.**

Example 2 The following program uses two InputBox functions. Whenever an InputBox function is encountered in a program, an input dialog box appears, and execution stops until the user responds to the request. The function returns the value entered into the input dialog box.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim prompt, title, fullName, firstName As String
  Dim dateOfBirth As Date
  prompt = "Enter your full name."
  title = "Name"
  fullName = InputBox(prompt, title)
  firstName = fullName.Substring(0, fullName.IndexOf(" "))
  prompt = "Enter your date of birth."
  title = "Birthday"
  dateOfBirth = CDate(InputBox(prompt, title))
  txtOutput.Text = firstName & ", you are " &
           DateDiff(DateInterval.Day, dateOfBirth, Today) & " days old."
End Sub
```

[Run, click on the button, enter *Emma Smith* into the first input dialog box, and enter *4/6/1992* into the second input dialog box.]



Emma, you are 6745 days old.

The response typed into an input dialog box is treated as a single string value, no matter what is typed. (Quotation marks are not needed and, if included, are considered as part of the string.) Numeric data typed into an input dialog box should be converted to a number with CDbl or CInt before being assigned to a numeric variable or used in a calculation. Just as with a text box, the typed data must be a literal. It cannot be a variable or an expression. For instance, *num*, 1/2, and 2 + 3 are not acceptable.

Using a Message Dialog Box for Output

Sometimes you want to grab the user's attention with a brief message such as "Correct" or "Nice try, but no cigar." You want this message to appear on the screen only until the user has read it. This task is easily accomplished with a **message dialog box** such as the one shown in Fig. 3.13.

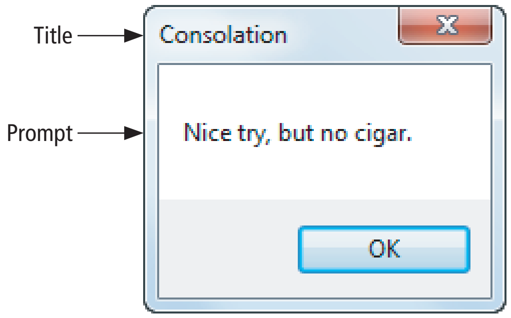


FIGURE 3.13 **Sample message dialog box.**

When a statement of the form

```
MessageBox.Show(prompt, title)
```

is executed, where *prompt* and *title* are strings, a message dialog box appears with *prompt* displayed and the Title bar caption *title*, and stays on the screen until the user presses Enter, clicks on the *Close* button in the upper-right corner, or clicks OK. For instance, the statement

```
MessageBox.Show("Nice try, but no cigar.", "Consolation")
```

produces Fig. 3.13. You can omit the value for the argument *title* and just execute `MessageBox.Show(prompt)`. If you do, the Title bar will be blank and the rest of the message dialog box will appear as before.

Named Constants

Often a program uses a special constant whose value does not change during program execution. Some examples might be the minimum wage, the sales tax rate, and the name of a master file. Programs are often made easier to understand and maintain if such a constant is given a name. Visual Basic has an object, called a **named constant**, that serves this purpose. A named constant is declared and used in a manner similar to a variable. The two main differences are that in the declaration of a named constant, Dim is replaced with Const, and the value of the named constant cannot be changed elsewhere in the program. Named constants can be thought of as read-only variables.

A named constant is declared and assigned a value with a statement of the form

```
Const CONSTANT_NAME As DataType = value
```

The standard convention is that the names be written in uppercase letters with words separated by underscore characters. Like a Dim statement, a Const statement can be placed in the Declaration sections of a program (for class-level scope) or in a procedure (for local scope). Named constant declarations in procedures usually are placed near the beginning of the procedure. Some examples of named constant declarations are

```
Const INTEREST_RATE As Double = 0.04
Const MINIMUM_VOTING_AGE As Integer = 18
Const BOOK_TITLE As String = "Programming with VB2010"
```

Examples of statements using these named constants are

```
interestEarned = INTEREST_RATE * CDbl(txtAmount.Text)

If (age >= MINIMUM_VOTING_AGE) Then
  MessageBox.Show("You are eligible to vote.")
End If

MessageBox.Show(BOOK_TITLE, "Title of Book")
```

Although the value of a named constant such as `INTEREST_RATE` will not change during the execution of a program, the value may need to be changed at a later time. The programmer can adjust to this change by altering just one line of code instead of searching through the entire program for each occurrence of the old interest rate.

Sending Output to the Printer (optional)

The following five steps send output to the printer.

1. Double-click on the PrintDocument control in the *All Windows Forms* or *Printing* group of the Toolbox. (The control will appear with the default name PrintDocument1 in a separate pane called the **component tray**, at the bottom of the Form Designer.)
2. Double-click on PrintDocument1 to invoke its PrintPage event procedure. (The code for printing text will be placed in this event procedure.)
3. Place the statement

   ```
   Dim gr As Graphics = e.Graphics
   ```

   in the event procedure. (This statement declares *gr* as a graphics object capable of printing both text and graphics.)
4. Enter a statement of the form

   ```
   gr.DrawString(str, font, Brushes.color, x, y)
   ```

   for each line of text to be printed. Here *str* is a string, *font* specifies the font name, size, and style, *color* specifies the color of the text, and *x* and *y* are integers giving the location on the page of the beginning of the string. (The values of *x* and *y* are specified in **points**, where 100 points are about one inch.) Visual Basic indents all text by about 25 points from the left side of the page. The beginning of the string will be printed *x* + 25 points from the left side and *y* points from the top side of the page. Two different ways of specifying the font will be given in the example that follows.

5. Place the statement

   ```
   PrintDocument1.Print()
   ```

   in another event procedure, such as a button's Click event procedure. (This statement will cause all of the text specified in step 4 to be printed.)

   Some examples of DrawString statements are as follows. The statement

```
gr.DrawString("HELLO WORLD", Me.Font, Brushes.DarkBlue, 100, 150)
```

prints the words HELLO WORLD using the form's font in dark blue letters 1.25 inches from the left side of the page and 1.5 inches from the top of the page. The pair of statements

```
Dim font As New Font("Courier New", 12, FontStyle.Bold)
gr.DrawString("HELLO WORLD", font, Brushes.DarkBlue, 100, 150)
```

produce the same output using a 12-point bold Courier New font.

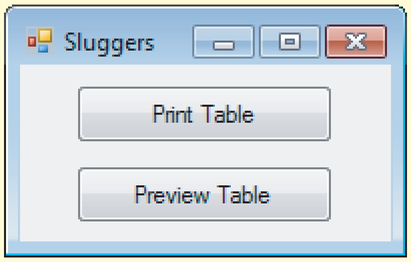
Visual Basic provides a control, called the **PrintPreviewDialog control**, which allows you to see how output will look before you send it to the printer. Just follow two steps:

1. Double-click on the PrintPreviewDialog control in the *All Windows Forms* or *Printing* group of the Toolbox. (The control will appear with the default name PrintPreviewDialog1 in the component tray at the bottom of the Form Designer.)
2. Place the pair of statements

   ```
   PrintPreviewDialog1.Document = PrintDocument1
   PrintPreviewDialog1.ShowDialog()
   ```

   in another event procedure, such as a button's Click event procedure. These statements cause the text specified in the PrintDocument1_PrintPage event procedure to be displayed in a "Print preview" window when the event procedure is invoked. The preview window's toolbar contains a magnifying-glass button (🔍▾) that allows you to zoom in on the text.

Example 3 The following program produces a two-column table of the top three all-time home-run hitters. Notice that the font is changed after the table's header is printed.



OBJECT	PROPERTY	SETTING
frmHR	Text	Sluggers
btnPrint	Text	Print Table
btnPreview	Text	Preview Table
PrintDocument1		
PrintPreviewDialog1		

```
Const ONE_INCH As Integer = 100     'number of points in an inch
Const LINE_HEIGHT As Integer = 25   'one-quarter of an inch

Private Sub btnPrint_Click(...) Handles btnPrint.Click
  PrintDocument1.Print()
End Sub
```

```
Private Sub btnPreview_Click(...) Handles btnPreview.Click
  PrintPreviewDialog1.Document = PrintDocument1
  PrintPreviewDialog1.ShowDialog()
End Sub

Private Sub PrintDocument1_PrintPage(...) Handles PrintDocument1.PrintPage
  Dim gr As Graphics = e.Graphics
  Dim x1 As Integer = ONE_INCH          'use one inch beyond left margin
  Dim x2 As Integer = 3 * ONE_INCH      'offset for second column
  Dim y As Integer = ONE_INCH           'use one inch top margin
  Dim font As New Font("Courier New", 10, FontStyle.Bold)
  gr.DrawString("PLAYER", font, Brushes.Blue, x1, y)
  gr.DrawString("HR", font, Brushes.Blue, x2, y)
  font = New Font("Courier New", 10, FontStyle.Regular)
  y += LINE_HEIGHT                      'move down one=quarter inch
  gr.DrawString("Barry Bonds", font, Brushes.Black, x1, y)
  gr.DrawString("762", font, Brushes.Black, x2, y)
  y += LINE_HEIGHT
  gr.DrawString("Hank Aaron", font, Brushes.Black, x1, y)
  gr.DrawString("755", font, Brushes.Black, x2, y)
  y += LINE_HEIGHT
  gr.DrawString("Babe Ruth", font, Brushes.Black, x1, y)
  gr.DrawString("714", font, Brushes.Black, x2, y)
End Sub
```

[Run, click on the *Preview Table* button, click the Zoom down-arrow to the right of the magnifying glass, and select 100%. The following text appears in the preview window.]

```
PLAYER                 HR
Barry Bonds            762
Hank Aaron             755
Babe Ruth              714
```

Comments

1. A variation of the DateDiff function discussed earlier is `DateDiff(DateInterval.Year, d1, d2)` which gives the number of years (sort of) between the two dates. It is of limited value, since it only uses the year parts of the two dates in its computation.
2. The section "Use the Printer" in Appendix B shows how to print a program and a form.

Practice Problems 3.3

1. Is the statement

```
txtOutput.Text = FormatNumber(12345.628, 1)
```

correct, or should it be written as follows?

```
txtOutput.Text = CStr(FormatNumber(12345.628, 1))
```

2. What is the difference in the outcomes of the following two sets of code?

```
strVar = InputBox("How old are you?", "Age")
numVar = CDbl(strVar)
txtOutput.Text = numVar

numVar = CDbl(InputBox("How old are you?", "Age"))
txtOutput.Text = numVar
```

EXERCISES 3.3

In Exercises 1 through 48, determine the output produced by the lines of code.

1.
```
txtOutput.Text = FormatNumber(1234.56, 0)
```
2.
```
txtOutput.Text = FormatNumber(-12.3456, 3)
```
3.
```
txtOutput.Text = FormatNumber(1234, 1)
```
4.
```
txtOutput.Text = FormatNumber(12345)
```
5.
```
txtOutput.Text = FormatNumber(0.012, 1)
```
6.
```
txtOutput.Text = FormatNumber(5 * (10 ^ -2), 1)
```
7.
```
txtOutput.Text = FormatNumber(-2 / 3)
```
8.
```
Dim numVar As Double = Math.Round(1.2345, 1)
txtOutput.Text = FormatNumber(numVar)
```
9.
```
Dim numVar As Double = Math.Round(12345.9)
txtOutput.Text = FormatNumber(numVar, 3)
```
10.
```
Dim numVar As Double = Math.Round(12.5)
txtOutput.Text = FormatNumber(numVar, 0)
```
11.
```
Dim numVar As Double = Math.Round(11.5)
txtOutput.Text = FormatNumber(numVar, 0)
```
12.
```
txtOutput.Text = FormatCurrency(1234.5)
```
13.
```
txtOutput.Text = FormatCurrency(12345.67, 0)
```
14.
```
txtOutput.Text = FormatCurrency(-1234567)
```
15.
```
txtOutput.Text = FormatCurrency(-0.225)
```
16.
```
txtOutput.Text = FormatCurrency(32 * (10 ^ 2))
```
17.
```
txtOutput.Text = FormatCurrency(4 / 5)
```
18.
```
txtOutput.Text = FormatPercent(0.04, 0)
```
19.
```
txtOutput.Text = FormatPercent(0.075)
```
20.
```
txtOutput.Text = FormatPercent(-.05, 3)
```
21.
```
txtOutput.Text = FormatPercent(1)
```
22.
```
txtOutput.Text = FormatPercent(0.01)
```
23.
```
txtOutput.Text = FormatPercent(2 / 3)
```
24.
```
txtOutput.Text = FormatPercent(3 / 4, 1)
```
25.
```
txtOutput.Text = "Pay to France " & FormatCurrency(27267622)
```
26.
```
txtOutput.Text = "Manhattan was purchased for " & FormatCurrency(24)
```
27.
```
Dim popUSover24 As Double = 177.6        'Million
Dim collegeGrads As Double = 45.5        'Million
'                                         45.5/177.6 = 0.2561937
txtOutput.Text = FormatPercent(collegeGrads / popUSover24, 1) &
  " of the U.S. population 25+ years old are college graduates."
```
28.
```
Dim degrees As String = FormatNumber(1711500, 0)
txtOutput.Text = degrees & " degrees were conferred."
```
29.
```
txtOutput.Text = "The likelihood of Heads is " &
                 FormatPercent(1 / 2, 0)
```
30.
```
txtOutput.Text = "Pi = " & FormatNumber(3.1415926536, 4)
```
31.
```
txtOutput.Text = CStr(#10/23/2010#)
```

32.
```
Dim d As Date = #6/19/2011#   'Father's Day
txtOutput.Text = FormatDateTime(d, DateFormat.LongDate)
```

33.
```
Dim d As Date = #11/25/2010#   'Thanksgiving Day
txtOutput.Text = FormatDateTime(d, DateFormat.LongDate)
```

34.
```
Dim d As Date = #1/1/2000#
txtOutput.Text = CStr(d.AddYears(12))
```

35.
```
Dim d As Date = #9/29/2011#
txtOutput.Text = CStr(d.AddDays(3))
```

36.
```
Dim d As Date = #10/9/2010#
txtOutput.Text = CStr(d.AddMonths(4))
```

37.
```
Dim d As Date = #4/5/2011#
txtOutput.Text = CStr(d.AddYears(2))
```

38.
```
Dim d As Date = #10/1/2010#
txtOutput.Text = CStr(d.AddDays(32))
```

39.
```
Dim d1 As Date = #2/1/2012#     '2012 is a leap year
Dim d2 As Date = d1.AddMonths(1)
txtOutput.Text = CStr(DateDiff(DateInterval.Day, d1, d2))
```

40.
```
Dim d1 As Date = #1/1/2012#   '2012 is a leap year
Dim d2 As Date = #1/1/2013#
txtOutput.Text = CStr(DateDiff(DateInterval.Day, d1, d2))
```

41.
```
Dim bet As Double   'Amount bet at roulette
bet = CDbl(InputBox("How much do you want to bet?", "Wager"))
txtOutput.Text = "You might win " & 36 * bet & " dollars."
```
(Assume that the response is *10*.)

42.
```
Dim word As String
word = InputBox("Word to negate:", "Negatives")
txtOutput.Text = "un" & word
```
(Assume that the response is *tied*.)

43.
```
Dim lastName, message, firstName As String
lastName = "Jones"
message = "What is your first name Mr. " & lastName & "?"
firstName = InputBox(message, "Name")
txtOutput.Text = "Hello " & firstName & " " & lastName
```
(Assume that the response is *John*.)

44.
```
Dim intRate, doublingTime As Double 'interest rate, time to double
intRate = CDbl(InputBox("Current interest rate?", "Interest"))
doublingTime = 72 / intRate
lstOutput.Items.Add("At the current interest rate, money will")
lstOutput.Items.Add("double in " & doublingTime & " years.")
```
(Assume that the response is *4*.)

45.
```
Const SALES_TAX_RATE As Double = 0.06
Dim price As Double = 100
Dim cost = (1 + SALES_TAX_RATE) * price
txtOutput.Text = FormatCurrency(cost)
```

46.
```
Const ESTATE_TAX_EXEMPTION As Double = 1000000
Const TAX_RATE = 0.45
Dim valueOfEstate As Double = 3000000
Dim tax As Double = TAX_RATE * (valueOfEstate - ESTATE_TAX_EXEMPTION)
txtOutput.Text = "You owe " & FormatCurrency(tax) & " in estate taxes."
```
47.
```
Dim gr As Graphics = e.Graphics
Dim font As New Font("Courier New", 10, FontStyle.Bold)
gr.DrawString("Hello World", font, Brushes.Blue, 175, 200)
```
48.
```
Dim gr As Graphics = e.Graphics
Dim font As New Font("Times New Roman", 12, FontStyle.Italic)
gr.DrawString("Hello", font, Brushes.Blue, 75, 100)
gr.DrawString("World", font, Brushes.Blue, 75, 125)
```

In Exercises 49 through 56, identify any errors.

49.
```
Const n As Integer = 5
n += 1
txtOutput.Text = CStr(n)
```
50.
```
Const n As String = "abc"
n = n.ToUpper
txtOutput.Text = n
```
51.
```
Dim num As Double
num = InputBox("Pick a number from 1 to 10.")
txtOutput.Text = "Your number is " & num
```
52. `info = InputBox()`

53.
```
Dim num As Double = FormatNumber(123456)
lstOutput.Items.Add(num)
```
54. `txtOutput.Text = FormatCurrency($1234)`

55. `MessageBox("Olive Kitteridge", "Pulitzer Prize for Fiction")`

56. `MessageBox.Show(1776, "Year of Independence")`

In Exercises 57 through 62, give a setting for the Mask property of a masked text box used to input the stated information.

57. A number from 0 to 999.

58. A word of at most ten letters.

59. A Maryland license plate consisting of three letters followed by three digits. (*Example*: BHC365)

60. A California license plate consisting of a digit followed by three letters and then three digits. (*Example*: 7BHC365)

61. An ISBN number. [Every book is identified by a ten-character International Standard Book Number (ISBN). The first nine characters are digits and the last character is either a digit or the letter X.] (*Example*: 0-32-108599-X)

62. A two-letter state abbreviation. (*Example*: CA)

In Exercises 63 and 64, write a statement to carry out the task.

63. Pop up a message dialog box with “Good Advice” in the title bar and the message “First solve the problem. Then write the code.”

64. Pop up a message dialog box with "Taking Risks Proverb" in the title bar and the message "You can't steal second base and keep one foot on first."

In Exercises 65 and 66, write an event procedure with the header `Private Sub btnCompute_Click(...) Handles btnCompute.Click`, and having one, two, or three lines for each step. Lines that display data should use the given variable names.

65. The following steps calculate the percent increase in the cost of a typical grocery basket of goods:
   (a) Declare all variables used in the steps that follow.
   (b) Assign 200 to the variable *begOfYearCost*.
   (c) Request the cost at the end of the year with an input dialog box, and assign it to the variable *endOfYearCost*.
   (d) Assign (*endOfYearCost* – *begOfYearCost*) / *begOfYearCost* to the variable *percentIncrease*.
   (e) Display a sentence giving the percent increase for the year.

(Test the program with a $215 end-of-year cost.)

66. The following steps calculate the amount of money earned in a walk-a-thon:
   (a) Declare all variables used in the steps that follow.
   (b) Request the amount pledged per mile from an input dialog box, and assign it to the variable *pledge*.
   (c) Request the number of miles walked from an input dialog box, and assign it to the variable *miles*.
   (d) Display a sentence giving the amount to be paid.

(Test the program with a pledge of $2.00 per mile and a 15-mile walk.)

67. Write a program that requests a year in a masked text box and then displays the number of days in the year. ***Hint:*** Use the AddYears method and the DateDiff function.

68. Write a program that calculates the number of days since the Declaration of Independence was ratified (7/4/1776).

69. Write a program that requests a date in a masked text box, and then displays the day of the week (such as Sunday, Monday, . . .) for that date.

70. Write a program that requests a date as input and then displays the day of the week (such as Sunday, Monday, . . .) for that date ten years hence.

71. Write a program that requests a month and a year as input and then displays the number of days in that month. ***Hint:*** Use the AddMonths method.

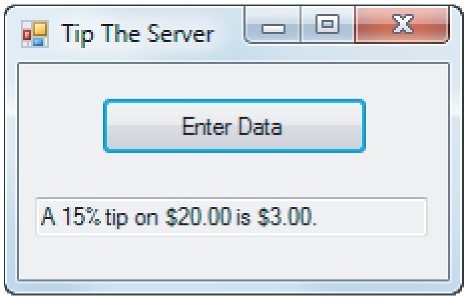
72. Write a program that requests the user's date of birth and then displays the day of the week (such as Sunday, Monday, . . .) on which they will have (or had) their 21st birthday.

73. Design a form with two text boxes labeled "Name" and "Phone number". Then write an event procedure that shows a message dialog box stating "Be sure to include the area code!" when the second text box receives the focus.

74. Write a program to calculate the amount of a server's tip given the amount of the bill and the percentage tip obtained via input dialog boxes. The output should be a complete sentence that reiterates the inputs and gives the resulting tip, as shown in Fig. 3.14 on the next page.

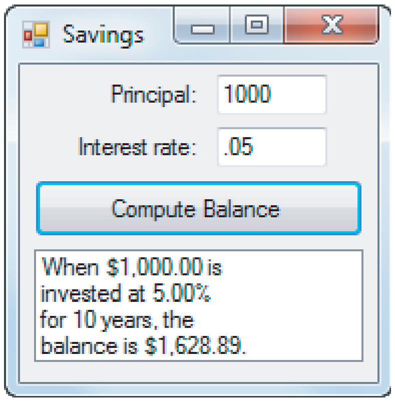
75. When P dollars are deposited in a savings account at interest rate r compounded annually, the balance after n years is $P(1 + r)^n$. Write a program to request the principal P and the interest rate r as input, and compute the balance after 10 years, as shown in Fig. 3.15 on the next page.

VideoNote
Mortgage calculator (Homework)



```
A 15% tip on $20.00 is $3.00.
```

FIGURE 3.14 Sample output of Exercise 74.



```
When $1,000.00 is
invested at 5.00%
for 10 years, the
balance is $1,628.89.
```

FIGURE 3.15 Sample output of Exercise 75.

76. Write a program to print the list of Internet lingo in Fig. 3.16.

```
PLS        Please
TAFN       That's all for now
HHOK       Ha, ha - only kidding
FWIW       For what its worth
IMO        In my opinion
```

FIGURE 3.16 Output of Exercise 76.

```
                     % of
Rank     Country     WW Users
1        USA         16.0%
2        China       11.9%
3        Japan       6.5%
```

FIGURE 3.17 Output of Exercise 77.

77. Write a program to print the top three ranking counties by the percentage of worldwide Internet users they contain as shown in Fig. 3.17.

Solutions to Practice Problems 3.3

1. The first statement is correct, since FormatNumber evaluates to a string. Although the second statement is not incorrect, the use of CStr is redundant.
2. The outcomes are identical. In this text, we primarily use the second style.

CHAPTER 3 SUMMARY

1. Three types of *literals* that can be stored and processed by Visual Basic are numbers, strings, and dates.
2. Many Visual Basic tasks are carried out by methods such as Clear (erases the contents of a text box or list box), Add (places an item into a list box), ToUpper (converts a string to uppercase), ToLower (converts a string to lowercase), Trim (removes leading and trailing spaces from a string), IndexOf (searches for a specified substring in a string and gives its position if found), and Substring (produces a sequence of consecutive characters from a string).
3. The *arithmetic operations* are +, −, *, /, ^, \, and Mod. The only string operation is &, concatenation. An *expression* is a combination of literals, variables, functions, and operations that can be evaluated.
4. A *variable* is a name used to refer to data. Variable names must begin with a letter or an underscore and may contain letters, digits, and underscores. Dim statements declare variables,