The concepts explained in this chapter include variables, constants, and scope. The TextBox control for obtaining user input is also explained. Techniques for debugging an application are introduced.

### Declaring Variables

A variable is a name for a value stored in memory. Variables are used in programs so that values can be represented with meaningful names. For example, when a variable named length is used in a program, it is clear that its value is a distance. Variables should be used to represent values because they make code easier to read, understand, and modify.

A variable must be declared before it is used. A declaration statement takes the form:

```
Dim variableName As type
```

A Dim statement must include the variable name, called the identifier, and the data type, which determines the type of data the variable will store. For example, the statement

```
Dim length As Integer
```

declares a variable length to store data of type Integer. An integer is a numeric value that is a positive or negative whole number. When an Integer variable is declared it stores the value 0.

An identifier must begin with a letter and contain only letters, numbers, and some special characters. Typically variable identifiers begin with a lowercase letter. Any word after the first in a variable identifier should begin with an uppercase letter. For example, rectangleLength. This code convention allows variables to be easily recognized.

Multiple variables with the same data type can be declared in a single statement, similar to:

```
Dim length, width As Integer
```

Grouping variables together in a single statement is good programming style when the variables represent related items. Declarations should not be grouped together in the same statement just because the variables are all the same type.
Using Variables

Applications typically use many variables. The code segment below uses two variables:

```vba
Dim side As Integer = 5
Dim area As Integer = side * side
```

The value of `area` after the last assignment statement is 25 (5 * 5).

Variable declarations should be grouped at the beginning of a procedure. A blank line after the declarations makes it easy to determine where the declarations end.

As shown in the code segment above, the value of a variable is changed through assignment. A variable assignment statement is formed with the variable name on the left side of an equal sign and the value it is to receive on the right side of the equal sign. The equal sign (=) operator indicates that the variable on the left is to receive the value on the right. The value on the right can be a literal, which is any actual value. It could also be another variable or an expression. In the code segment above, `area` was assigned the value of an expression (side * side).

An assignment statement can be part of a variable declaration. In addition to being declared, the variable is initialized. For example, in the code segment above, variable `length` was assigned a value when declared.

It is important to realize that a variable can store only one value at any one time. For example, after the following statements execute:

```vba
Dim area As Integer
area = side * side
```

the value of `area` is 25 because this was the last value assigned to `area`.

Review: SquareArea – part 1 of 3

1. START VISUAL BASIC
2. CREATE A NEW PROJECT
   Create a Windows application named SquareArea.
3. CREATE THE INTERFACE
   Use the table on the next page for setting object properties.
<table>
<thead>
<tr>
<th>Object</th>
<th>(Name)</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td></td>
<td>Square Area</td>
</tr>
<tr>
<td>Label1</td>
<td>lblQuestion</td>
<td>What is the area of a square with side 5?</td>
</tr>
<tr>
<td>Label2</td>
<td>lblAnswer</td>
<td>empty</td>
</tr>
<tr>
<td>Button1</td>
<td>btnAnswer</td>
<td>Answer</td>
</tr>
</tbody>
</table>

### WRITE THE APPLICATION CODE

a. Display the Code window.
b. Add comments that include your name, assignment, and today’s date.
c. Create a `btnAnswer_Click` event procedure and then add the statements:

```vbnet
Dim side As Integer = 5 'side of square
Dim area As Integer

area = side * side
Me.lblAnswer.Text = area 'display answer
```

### RUN AND TEST THE APPLICATION

a. Save the modified SquareArea project and then run the application. Click Answer to test the application.
b. Close the SquareArea application.

### PRINT THE CODE

---

### Obtaining a Value from the User

**TextBox object**

An application is more flexible when values can be entered, or *input*, at run time. A TextBox object is one way to allow users to enter values. For example, the SquareArea application could include a text box for the user to type a value for the length of the side:

A label is often placed near a text box to tell the user what kind of input is expected. This label is called the *prompt*.

The TextBox control has the properties:
- **(Name)** identifies a control for the programmer. TextBox object names should begin with `txt`.
- **Text** is what is displayed in the text box.
- **TextAlign** sets the alignment of text relative to the text box.
TextChanged

A TextChanged event procedure is sometimes coded for a TextBox object. This procedure executes when the user types in the text box.

At run time, the TextBox Text property stores whatever characters are currently in the text box. This property can be used in an assignment statement to retrieve the data typed by the user. The following statement assigns the data in a text box named txtRadius to the variable side:

```vbnet
side = Me.txtSide.Text 'retrieve data from text box
```

run-time error

If the text box does not contain data that matches the variable type, a run-time error occurs and the program is halted. To prevent this, the Val() function should be used to convert text box data to a numeric value. A function performs a single, well-defined task like a method or procedure, but then returns a value that is a result of the task performed. The Val() function requires a string value and then returns a number corresponding to the string. If the first character of the string is not a numeric character, Val() returns a 0. The statements below demonstrate Val():

```vba
Dim height As Integer
height = Val("62 inches") 'height is assigned 62
height = Val("Twenty inches") 'height is assigned 0
height = Val("Six feet 2 inches") 'height is assigned 0
```

To convert text box data to a numeric, the Val() function should be used in a statement similar to:

```vba
side = Val(Me.txtSide.Text) 'assign a numeric value to side
```

TIP Functions often require data to perform their task. The data is included in parentheses after the function name.

Review: SquareArea – part 2 of 3

1 MODIFY THE INTERFACE

a. Change the lblQuestion Text property to Enter the length of a side:

b. Add a text box to the form. Move and size the object so that the interface looks similar to:

![Square Area Interface](image)

- Enter the length of a side:
- Answer

To convert text box data to a numeric, the Val() function should be used in a statement similar to:

```vba
side = Val(Me.txtSide.Text) 'assign a numeric value to side
```

c. Name the TextBox object txtSide and set the Text property so that it is empty.

2 MODIFY THE PROGRAM CODE

a. Display the Code window and then modify the btnAnswer_Click procedure as shown:

```vba
Dim side As Integer 'side of square
Dim area As Integer

side = Val(Me.txtSide.Text)
area = side * side
Me.lblAnswer.Text = area 'display answer
```

b. Run the application. Type 7 in the text box and then click Answer. The area 49 is displayed.
c. Replace text box contents by typing a 4 in the text box, but do not click Answer. Note the previous answer is still displayed.
d. Close the SquareArea application.

3 ADD A TEXTCHANGED EVENT PROCEDURE
   a. Create a txtSide_TextChanged event procedure and then add the statement:
      
   ```vba
   'Clear the current answer when the user begins to type a new value
   Me.lblAnswer.Text = ""
   ```
   b. Run the application. Type 2 in the text box and then click Answer. The area 4 is displayed.
c. Replace the text box contents by typing a 3 in the text box. The previous answer is cleared. Click Answer to display a new value.
d. Close the SquareArea application.

4 PRINT THE CODE AND THEN CLOSE THE PROJECT

Review: RectangleArea

Create a RectangleArea application that prompts the user for a length and width and then displays the area of a rectangle when Answer is clicked. Use appropriate variables and include TextChanged event procedures that clear the answer. The application interface should look similar to that shown on the right after typing 5 and 2 and clicking Answer.

Built-In Data Types

The Integer data type is just one Visual Basic data type. Other built-in data types include:

<table>
<thead>
<tr>
<th>Type</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>integers from -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>Double</td>
<td>a positive or negative number that may contain a decimal portion in the range -1.8E+308 to 1.8E+308</td>
</tr>
<tr>
<td>Decimal</td>
<td>large numbers possibly containing a decimal.</td>
</tr>
<tr>
<td>Date</td>
<td>a date in m/d/yyyy format</td>
</tr>
<tr>
<td>Char</td>
<td>a single character</td>
</tr>
<tr>
<td>String</td>
<td>a set of characters</td>
</tr>
<tr>
<td>Boolean</td>
<td>True or False</td>
</tr>
</tbody>
</table>

**TIP** Visual Basic also supports the Short, Long, and Single data types.

**TIP** All but the String data types are primitive data types. A primitive data type stores a single piece of data.

**Integer**

An Integer variable uses 4 bytes of memory to store its value and is used for representing whole numbers.

**Double**

Floating point

Values that are represented by the Double type are sometimes referred to as a floating point, meaning that the values contain numbers after the decimal point. Because of the many digits that are possible in a Double, a variable of this type uses 8 bytes of memory.

**Decimal**

The Decimal data type is appropriate for storing values representing currency. Decimal variables use 16 bytes of memory.
A Date variable requires 8 bytes of memory. This type should be used to represent a date. Dates can be in the range 1/1/0001 through 12/31/9999. The value assigned to a Date variable must be enclosed with # signs, as in the statement below:

```
Dim birthDate As Date = #1/1/1996#
```

A Char variable requires 2 bytes of memory because Visual Basic uses the 16-bit Unicode character encoding. A character can include a letter of the alphabet, a digit, and in general any character that can be typed or displayed, such as $, %, and space. A Char assignment requires double-quotations marks ("),

```
Dim middleInitial As Char
middleInitial = "A"
```

The String type represents a set of characters, also called a string. A string can include the letters of the alphabet, digits, and in general any character that can be typed or displayed, such as $, %, and spaces. A String variable uses 2 bytes of memory for each character in the string. A String assignment requires double-quotations marks ("),

```
Dim lastName As String = "Lutz"
```

Variables that are type Boolean can have only one of two values—True or False. Boolean variables are particularly useful for representing on/off and yes/no values. Boolean assignment statements use the keywords True and False. A Boolean variable uses 2 bytes of memory.

Choosing a data type is important to choose the most appropriate type for the quantity being represented. If a value could possibly have a decimal portion, then Double is the best choice. If a variable will represent only whole numbers, then Integer is the best choice even though Double will work. Using the most appropriate data types for variables has two benefits. First, both the compiler and the reader will understand the possible values for a variable. Second, the compiler allocates the appropriate memory for the variable.

In an assignment statement, Visual Basic automatically converts data to match the type of the variable it is being assigned to. For example, a value with a decimal portion is automatically rounded to a whole number when assigned to an Integer variable:

```
Dim x As Integer
x = 6.7 'x assigned 7
```

Visual Basic will try to convert from one data type to another as long as the data is valid for the receiving data type. For example, assigning 12.3 to an Integer variable is valid because the number can be converted to 12. However, assigning abc to an Integer variable generates an error.
Review: TotalDistance

Create a TotalDistance application that prompts the user for the three segment lengths of a race and then calculates the total distance when Distance is clicked. The race segments can have lengths that contain a decimal portion. Use variables of the appropriate types and include a TextChanged event procedures that clears the distance. The application interface should look similar to that shown on the right after typing the data shown and clicking Distance.

Variable Scope

The placement of a variable declaration is important because it determines the variable's scope. The scope of a variable is the set of statements that can access the variable. For example, a variable declared at the beginning of a procedure is accessible to any statement in that procedure. This means that any statement in the procedure can refer to the variable, change its value, and so on. This variable is said to be local to the procedure because its scope is limited to that procedure. Statements outside the procedure do not have access to the variable.

When a variable needs to be accessed by several or all of the procedures in the Form class, the declaration should be placed in the Form class above any procedure declarations. This type of declaration is module-level. Module-level declarations are global to any code in the Form class, which means any statement in any procedure can refer to the variable or change its value. Global declarations should be used only when absolutely necessary. For example, the code below includes both global and local variables:

```vbnet
Public Class Form1
    Dim x As Integer = 25
    Private Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim x As Integer = 10
        Dim y As Integer = 30
        Dim z As Integer = x + y
    End Sub

    Private Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button2.Click
        Dim y As Integer = 5
        Dim z As Integer = x + y
    End Sub
End Class
```

One important programming practice is to declare variables so that their scope is limited to where they are needed. This is good programming style because it produces cleaner code and helps eliminate the possibility of errors.
Review: ScopeDemo

1. CREATE A NEW PROJECT
   Create a Windows application named ScopeDemo.

2. COMPLETE THE INTERFACE
   Use the table below for setting object properties.

<table>
<thead>
<tr>
<th>Object</th>
<th>(Name)</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td></td>
<td>ScopeDemo</td>
</tr>
<tr>
<td>Label1</td>
<td>lblXis</td>
<td>x is</td>
</tr>
<tr>
<td>Label2</td>
<td>lblAnswer</td>
<td>empty</td>
</tr>
<tr>
<td>Button1</td>
<td>btnProc1</td>
<td>Procedure 1</td>
</tr>
<tr>
<td>Button2</td>
<td>btnProc2</td>
<td>Procedure 2</td>
</tr>
</tbody>
</table>

3. WRITE THE APPLICATION CODE
   a. Display the Code window.
   b. Add comments that include your name, assignment, and today's date.
   c. Create event procedures and add the global variable x as shown below:

```vbnet
Public Class Form1
    Dim x As Integer = 10

    Private Sub btnProc1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnProc1.Click
        Dim x As Integer = 3
        Me.lblAnswer.Text = x
    End Sub

    Private Sub btnProc2_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnProc2.Click
        Me.lblAnswer.Text = x
    End Sub
End Class
```

4. RUN THE APPLICATION
   a. Run the application. Click Procedure 1. 3 is displayed. Click Procedure 2. 10 is displayed. Note that the x variable in the btnProc1 procedure is local to that procedure. The x value displayed by the btnProc2 procedure uses the global variable. A procedure uses a local variable if available before looking for a global variable.
   b. Close the ScopeDemo application.
Special Division Operators

In addition to the standard built-in operators (\(^\wedge\), \(\div\), \(\times\), \(+\), \(-\)), Visual Basic includes two additional division operators. The \(\div\) operator performs integer division, and the \(\mod\) operator performs modulus division.

**integer division**  
*Integer division* truncates the decimal portion of the quotient, which results in an integer. For example, in the assignment statement

\[
\text{Dim } x \text{ As Integer} \\
x = 20 \div 7 \quad \text{'x is assigned 2}
\]

\(x\) is assigned 2 because the whole number portion of the quotient is 2:

\[
\begin{array}{c|c|c}
7 & 20 & r6 \\
\hline
14 & 6 & \\
\end{array}
\]

**modulus division**  
*Modulus division* returns the remainder resulting from division. For example, in the assignment statement

\[
\text{Dim } x \text{ As Integer} \\
x = 20 \mod 7 \quad \text{'x is assigned 6}
\]

\(x\) is assigned 6 because the remainder of 20 divided by 7 is 6:

\[
\begin{array}{c|c|c}
7 & 20 & -6 \\
\hline
14 & 6 & \\
\end{array}
\]

Modulus division is used in applications where the separate digits of a number are needed, for finding the number of minutes left over after hours have been accounted for, and for other integer-related tasks.

**order of operations**  
Integer division is performed after multiplication and division. Modulus division is performed next, and then addition and subtraction. For example, the expression \(5 \times 2 \mod 3 \div 2\) evaluates to 0 because \(5 \times 2\) is performed first, then \(3 \div 2\), and then \(10 \mod 1\). Operator precedence can be changed by using parentheses.

---

Review: SkyhookInternational

Skyhook International sells skyhooks that ship 3 per box. Padded envelopes are used to ship individual skyhooks. Create a SkyhookInternational application that prompts the user for the number of skyhooks ordered, and then displays the number of boxes and envelopes required for the shipment when Ship is clicked. The application interface should look similar to that shown on the right after typing 26 and clicking Ship.
Using Named Constants

A constant is a name for a memory location that stores a value that cannot be changed from its initial assignment. Constants, like variables, are used in a program so that values can be represented with meaningful names. For example, the following statement declares a constant PI with the value 3.14:

```
Const PI As Double = 3.14
```

A constant can be used wherever a value can be used. For example:

```
circleArea = PI * radius ^ 2
```

Constant identifiers are typically all uppercase and may include underscore (_) characters to separate words. For example, MAX_PRICE. The value of a constant is assigned only in the declaration. Trying to change the value of a constant after the initial assignment generates an error.

The placement of constant declarations determines their scope. Because the value of a constant does not change throughout a program run, it is usually safe to allow a broader scope. Therefore, constant declarations are often placed in the Form class, outside any procedures. Wherever constant declarations are placed, they should be grouped and declared before any variable declarations.

Identifiers and Keywords

Identifiers in Visual Basic must begin with a letter and may contain letters, numbers, and some special symbols. Periods and spaces are not allowed. Identifiers are not case sensitive, which means that an uppercase letter is the same as a letter in lowercase. For example, identifiers Count and count refer to the same variable. The IDE automatically changes the case of an identifier to match that of the first occurrence.

The Visual Basic language contains keywords, which have special meaning to the Visual Basic compiler and therefore cannot be used for a variable or constant identifier. Visual Basic keywords include:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>Logical AND operator</td>
</tr>
<tr>
<td>Boolean</td>
<td>Logical data type</td>
</tr>
<tr>
<td>Byte</td>
<td>Integer data type</td>
</tr>
<tr>
<td>Call</td>
<td>Invoke method</td>
</tr>
<tr>
<td>Case</td>
<td>Selects statement</td>
</tr>
<tr>
<td>Catch</td>
<td>Catch block</td>
</tr>
<tr>
<td>Const</td>
<td>Constant declaration</td>
</tr>
<tr>
<td>Date</td>
<td>Date data type</td>
</tr>
<tr>
<td>Decimal</td>
<td>Decimal data type</td>
</tr>
<tr>
<td>Declare</td>
<td>Declare statement</td>
</tr>
<tr>
<td>Default</td>
<td>Default value</td>
</tr>
<tr>
<td>Delegate</td>
<td>Delegate declaration</td>
</tr>
<tr>
<td>Do</td>
<td>Loop statement</td>
</tr>
<tr>
<td>Dim</td>
<td>Declaration statement</td>
</tr>
<tr>
<td>Else</td>
<td>Branch statement</td>
</tr>
<tr>
<td>ElseIf</td>
<td>Branch statement</td>
</tr>
<tr>
<td>End</td>
<td>End of block</td>
</tr>
<tr>
<td>Enum</td>
<td>Enum data type</td>
</tr>
<tr>
<td>Error</td>
<td>Error handler</td>
</tr>
<tr>
<td>Event</td>
<td>Event data type</td>
</tr>
<tr>
<td>False</td>
<td>Logical data type</td>
</tr>
<tr>
<td>Finally</td>
<td>Loop statement</td>
</tr>
<tr>
<td>For</td>
<td>Loop statement</td>
</tr>
<tr>
<td>Function</td>
<td>Return function</td>
</tr>
<tr>
<td>Get</td>
<td>Property accessor</td>
</tr>
<tr>
<td>If</td>
<td>Branch statement</td>
</tr>
<tr>
<td>In</td>
<td>Property accessor</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer data type</td>
</tr>
<tr>
<td>Interface</td>
<td>Language construct</td>
</tr>
<tr>
<td>Is</td>
<td>Logical data type</td>
</tr>
<tr>
<td>Like</td>
<td>Keyword</td>
</tr>
<tr>
<td>Me</td>
<td>Object reference</td>
</tr>
<tr>
<td>Module</td>
<td>Module declaration</td>
</tr>
<tr>
<td>New</td>
<td>Constructor call</td>
</tr>
<tr>
<td>Not</td>
<td>Logical operator</td>
</tr>
<tr>
<td>Object</td>
<td>Object reference</td>
</tr>
<tr>
<td>Or</td>
<td>Logical operator</td>
</tr>
<tr>
<td>Private</td>
<td>Access modifier</td>
</tr>
<tr>
<td>Public</td>
<td>Access modifier</td>
</tr>
<tr>
<td>Return</td>
<td>Return statement</td>
</tr>
<tr>
<td>Select</td>
<td>Loop statement</td>
</tr>
<tr>
<td>Short</td>
<td>Data type</td>
</tr>
<tr>
<td>Single</td>
<td>Data type</td>
</tr>
<tr>
<td>Select</td>
<td>Loop statement</td>
</tr>
<tr>
<td>Stop</td>
<td>Loop statement</td>
</tr>
<tr>
<td>Static</td>
<td>Access modifier</td>
</tr>
<tr>
<td>String</td>
<td>Data type</td>
</tr>
<tr>
<td>Structure</td>
<td>Language construct</td>
</tr>
<tr>
<td>Sub</td>
<td>Property accessor</td>
</tr>
<tr>
<td>Then</td>
<td>Branch statement</td>
</tr>
<tr>
<td>To</td>
<td>Logical operator</td>
</tr>
<tr>
<td>True</td>
<td>Logical data type</td>
</tr>
<tr>
<td>Type</td>
<td>Data type</td>
</tr>
<tr>
<td>This</td>
<td>Object reference</td>
</tr>
<tr>
<td>Type</td>
<td>Data type</td>
</tr>
<tr>
<td>Type</td>
<td>Data type</td>
</tr>
</tbody>
</table>

Chapter 3 Variables and Constants
Create a CircleArea application that prompts the user for the radius of a circle and then displays the circle area when Area is clicked. The application should include appropriate variables and a named constant PI. The application interface should look similar to that shown on the right after typing 2.5 and clicking Area.

### Programming Errors

Syntax errors, logic errors, and run-time errors are the three types of errors that can occur in a program. A statement that violates the rules of Visual Basic is a syntax error. For example, the second statement

```visual-basic
Const PI As Double = 3.14
PI = 3.141 'Syntax error!
```

is not syntactically correct because constant assignment is illegal outside the declaration. Syntax errors display a blue wavy underline. Hovering with the mouse over the line displays help text:

```
Const PI As Double = 3.14
PI = 3.141
Constant cannot be the target of an assignment.
```

A logic error, also called a semantic error, is more difficult to detect. Logic errors are caused by statements that are syntactically correct, but produce undesired or unexpected results, as in the following example:

```visual-basic
Dim length As Integer
Dim area As Double

length = 1.2
area = length * length 'I is actually assigned
'Expected value is 1.44
```

The variable `length` was accidentally declared as an `Integer`. Therefore, 1.2 is converted to 1 in the statement `length = 1.2`.

Logic errors must be found by the programmer through testing of the application and by careful reading of the program code. Accurate and careful commenting, proper indentation, and descriptive identifiers can help in finding and preventing logic errors.
Errors that are not detected by the compiler may generate a run-time error. A run-time error, also called an exception, halts program execution at the statement that cannot be executed. The statement causing the error is highlighted and an exception helper box is displayed:

The box can be removed by clicking the Close button. The Stop Debugging button on the toolbar can then be clicked to stop program execution so that the program can be corrected.

**Debugging an Application**

The source of logic errors can be hard to determine without tools for debugging a program. Debugging is the process of getting an application to work correctly. One debugging technique uses breakpoints. A breakpoint is a statement that has been marked as a stopping point. The code below shows a breakpoint, which is highlighted in red:

A breakpoint is created by clicking in the gray area to the left of a statement. When the application is run, program execution stops at the first breakpoint and the IDE goes into break mode. In break mode, the Watch window can be used to examine values. Right-clicking a variable, constant, or object name displays a menu with an Add Watch command. Selecting this command to add the variable, constant, or object name to the Watch window with its current value:

**Debugging Techniques**

Using a debugger is just one technique for finding errors. Another technique involves commenting out lines of code until the statement generating the error is found.
**TIP** Hovering over a variable with the mouse displays the variable's current value.

**TIP** The Debug toolbar is displayed when a program with a breakpoint is executed.

Program execution is continued from a breakpoint by clicking the Step Into button (↓) on the toolbar or pressing the F8 key, which executes one statement at a time. Debug → Step Into can also be used to step through a program. Values in the Watch window are automatically updated while stepping through a program.

---

**Review: SquareArea – part 3 of 3**

1. **OPEN THE SQUAREAREA PROJECT**
2. **ADD A BREAKPOINT TO THE CODE**
   a. Display the Code window.
   b. Click the pointer in the gray area to the left of `side = Val(Me.txtSide.Text)`. A breakpoint is added, similar to:

   ```vbnet
   Private Sub btnAnswer_Click(ByVal sender As Object, ByVal e As Object)
       Dim side As Integer
       Dim area As Integer
       side = Val(Me.txtSide.Text)
       area = side * side
       Me.lblAnswer.Text = area
   End Sub
   End Class
   ```

3. **RUN THE APPLICATION**
   a. Run the application. The SquareArea application is displayed.
   b. Type 2 in the text box and then click Answer. The breakpoint is reached and the Code window is again displayed. Note the breakpoint statement is now selected in yellow.
   c. Right-click `side` in the statement below the breakpoint. A menu is displayed.
   d. Select Add Watch. `side` appears in the Watch window with its current value 0.
e. Add area to the Watch window. Note its value.
f. On the toolbar, click the Step Into button (→). The next statement is executed. In the Watch window, the value of side changes to 2.
g. Step into the next statement. area has been assigned 4, the result of the square area expression.
h. Continue stepping through the application until the SquareArea window is again displayed.
i. Close the application. The Watch window is closed and the Code window is again displayed.

4 CLEAR THE BREAKPOINT
Click the dot to the left of the breakpoint to remove the breakpoint.

Case Study

This and all subsequent chapters end with a case study. Case studies are used to learn problem-solving techniques. Each case study will include a description, program specification, code design, program implementation, and a discussion about testing and debugging.

In this case study, a Calculator application will be created. The Calculator application allows the user to enter two operands and select an operator and then the answer is displayed.

Calculator Specification

The first step in creating an application is clearly defining what the application is to accomplish. This definition is called the specification, or spec, because it specifies what the application should do. In real-world situations, the specification is developed by talking with the end user and other computer professionals. In this text, the specification will be provided.

Calculator prompts the user for two numbers (operands) and allows the user to select an operator (|^,*|,/,\,%Mod,+,−) from a set of radio buttons. When an operator is selected, the result of the expression formed by the operands and operator is displayed.

Calculator Design

application design includes how the interface looks and how the code is written to accomplish the specification. In this text, the interface and code designs will be presented.

The best way to design the interface is to sketch interface designs on paper. The Calculator interface design:
The code design describes how to accomplish the spec. Included in the code design are a description of the input, output, and the data generated. The code design for this Calculator is:

The input for Calculator is two numbers (operands) typed into text boxes and the selection of a radio button corresponding to the desired operator.

Click event procedures will respond to the user’s selection of an operator for the expression. In each radio button Click event procedure, a Double variable answer will be used to store the result of the expression formed with the two operands and the selected operator.

The Output for Calculator is a label displaying the result for the expression formed with the operands and selected operator. This label will be set in the radio button Click event procedures.

TextChanged event procedures will be coded to clear the radio buttons and label.

Calculator Coding

Coding is creating the interface and writing the program code. The interface and code for this Case Study are:

```
Object   (Name)             Text
Form1     Label1            Calculator
          Label2             First Operand:
          TextBox1          Second Operand:
          TextBox2          empty
          GroupBox1          empty
          RadioButton1      Select an Operator
          RadioButton2      ^
          RadioButton3      *
          RadioButton4      /
          RadioButton5      \n          RadioButton6      Mod
          RadioButton7      +
          Label3            Result:
          Label4            empty
```

Chapter 3 Variables and Constants
Public Class Form1

Private Sub txtOperand1_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles txtOperand1.TextChanged
    Me.radAddition.Checked = False
    Me.radDivision.Checked = False
    Me.radExponentiation.Checked = False
    Me.radIntDivision.Checked = False
    Me.radModDivision.Checked = False
    Me.radMultiplication.Checked = False
    Me.radSubtraction.Checked = False
    Me.lblExpressionValue.Text = Nothing
End Sub

Private Sub txtOperand2_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles txtOperand2.TextChanged
    Me.radAddition.Checked = False
    Me.radDivision.Checked = False
    Me.radExponentiation.Checked = False
    Me.radIntDivision.Checked = False
    Me.radModDivision.Checked = False
    Me.radMultiplication.Checked = False
    Me.radSubtraction.Checked = False
    Me.lblExpressionValue.Text = Nothing
End Sub

Private Sub radAddition_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radAddition.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) + Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Private Sub radDivision_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radDivision.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) / Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Private Sub radExponentiation_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radExponentiation.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) ^ Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Private Sub radIntDivision_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radIntDivision.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) \ Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Private Sub radModDivision_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radModDivision.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) Mod Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Chapter 3 Variables and Constants
Private Sub radMultiplication_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radMultiplication.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) * Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

Private Sub radSubtraction_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles radSubtraction.Click
    Dim answer As Double
    answer = Val(Me.txtOperand1.Text) - Val(Me.txtOperand2.Text)
    Me.lblExpressionValue.Text = answer
End Sub

End Class

Calculator Testing and Debugging

Testing is the process of running the application and entering data to test different possibilities to reveal any bugs. Debugging is the process of getting an application to work correctly.

Calculator should be tested by entering values that are positive, negative, and zero. What will happen when 0 is entered in the text box for the second operand? One way to prevent this error is discussed in the next chapter.

Running Calculator, entering two numbers, and then selecting the * operator displays the following:

![Calculator Window](image)

Chapter Summary

Variables and constants are used in a program so that values can be represented with meaningful names. A variable is declared with a Dim statement and a constant with a Const statement. A variable can be initialized in a declaration. The value of a variable is changed in an assignment statement. The value of a constant cannot change from its initial assignment in a declaration.

Variable and constant identifiers must begin with a letter and contain only letters, digits, and the underscore character. Keywords cannot be used as identifiers.

A TextBox is an object that allows the user to enter a value at run time. A label placed near a text box prompts the user for the type of data expected. A TextChanged event is often coded for a TextBox object.

The information entered into a text box at run time can be accessed with the Text property of the TextBox object. The Val() function is used to convert the data in the Text property to a numeric value.
Visual Basic includes several built-in data types, including Integer, Double, Decimal, Date, Char, String, and Boolean. A variable or constant declaration should include the appropriate data type for the information to be stored. Visual Basic automatically converts data to match the type of the variable it is being assigned to. If a conversion is not possible, a run-time error is generated.

The scope of a declaration is the set of statements that can access the declared variable or constant. A declaration at the beginning of a procedure is accessible to the statements in the procedure only. This type of declaration is called a local declaration. Global, or module-level, declarations are outside the procedures in a program and are accessible to all the procedures. The scope of a variable or constant should be kept as narrow as possible.

Visual Basic includes an integer division operator (\) that returns the whole portion of the quotient, and a modulus division operator (Mod) that returns the remainder of a division operation.

A statement that violates the rules of Visual Basic contains a syntax error and is displayed with a blue wavy underline in the Code window. A logic error, also called a semantic error, is caused by syntactically correct statements that produce undesired or unexpected results. Errors that go undetected can cause a run-time error, also called an exception.

Debugging is the process of getting an application to work correctly. One debugging technique uses breakpoints, which is a statement marked as a stopping point. When the IDE is in break mode, the Debug toolbar with a Step Into button and a Watch window for viewing the value of variables is displayed.

The sequence of steps and thinking that goes into the construction of a substantial application are specification, design, coding, and testing and debugging.

The code conventions introduced in this chapter are:

- Variable identifiers should begin with a lowercase letter and any word after the first within the identifier should begin with an uppercase letter.
- Group variables together in the same declarations only when variables represent related items.
- Use a blank line after a group of declarations to make it clear where declarations end.
- Use a descriptive prompt next to a text box to tell the user what kind of input is expected.
- Choose data types that are appropriate for the type of data being represented.
- Declare variables so that their scope is limited to where they are needed.
- Constant identifiers should be all uppercase with underscore characters separating words.
- Group constant declarations before variable declarations.
**Vocabulary**

**Application Design**  How an application's interface will look and how the program code will be written.

**Breakpoint**  A statement that has been marked as a stopping point.

**Break mode**  The point at which an application has stopped executing and the Visual Basic IDE is displayed with a Watch window.

**Coding**  Creating the interface and writing the program code.

**Constant**  A name for a memory location that stores a value that cannot be changed from its initial assignment.

**Debugging**  The process of getting an application to work correctly.

**Declaration statement**  A statement used to create a variable or constant.

**Exception**  See Run-time error.

**Floating point**  The Double data type that can represent values with numbers after the decimal point.

**Function**  A procedure that performs a task and then returns a value.

**Global declaration**  A declaration outside the procedures of a program. Also called module-level declaration.

**Initialized**  Giving a variable a value in the declaration statement.

**Integer division**  Division performed with the \ operator to return only the whole portion of the quotient.

**Keyword**  Identifier reserved by Visual Basic.

**Literal**  An actual value.

**Local declaration**  A declaration at the beginning of a procedure.

**Logic error**  An error caused by syntactically correct statements that produce unexpected results. Also called semantic error.

**Modulus division**  Division performed with the Mod operator to return only the remainder portion of the division operation.

**Prompt**  A label placed near a text box describing the expected input from the user.

**Run-time error**  A syntax or logic error that halts a program at run time. Also called an exception.

**Scope**  The set of statements that can be accessed by a declared variable or constant.

**Semantic error**  See Logic error.

**Spec**  See Specification.

**Specification**  Definition of what an application should do.

**String**  A set of characters.

**Syntax error**  An error caused by a statement that violates the rules of Visual Basic.

**Testing**  The process of running an application and entering data to test different possibilities to reveal any bugs.

**TextBox**  An object that allows the user to enter a value.

**Variable**  A named memory location that stores a value.

**Watch window**  The part of the IDE that can be used to examine values.
Visual Basic

\ Arithmetic operator used to perform integer division.

Boolean A data type used to represent True or False.

Char A data type representing a single character.

Const Keyword used in a statement to declare a constant.

Date A data type representing dates and times.

Decimal A data type representing very large positive or negative real numbers. Best used for representing currency values.

Dim Keyword used in a statement to declare a variable.

Double A data type representing very large positive or negative real numbers.

False One of two possible Boolean values.

Integer A data type representing positive or negative whole numbers.

Mod Arithmetic operator used to perform modulus division.

Nothing Keyword that can be used in place of an empty string for clearing labels, and so forth.

Step Into button Clicked to step through a program in break mode. Found on the Debug toolbar.

String A data type representing a string.

TextBox control Used to add a TextBox control class object to a form. Properties include (Name), Text, and TextAlign. Events include TextChanged.

True One of two possible Boolean values.

Val() A function that takes a string and returns a number corresponding to the numeric characters.
Critical Thinking

1. a) List four legal identifier names.
b) List four illegal identifier names and explain why each is illegal.

2. a) In two statements, declare a variable named numCards and assign it the value 5.
b) In one statement, declare a variable named numCards and assign it the value 5.

3. a) What is the final value of yourNumber after the last statement executes?
   Dim myNumber As Integer = 5;
   Dim yourNumber As Integer = 4;
   myNumber = yourNumber * 2;
   yourNumber = myNumber + 5;
   b) What is the final value of yourNumber after the last statement executes?
   Dim myNumber As Integer;
   Dim yourNumber As Integer = 4;
   myNumber = yourNumber + 7;
   yourNumber = myNumber;

4. What is the primary purpose of a TextBox object?

5. Why is the Val() function used in a statement that assigns the data from a text box to a numeric variable?

6. a) What are labels near a text box called?
b) Why should text boxes have labels near them?

7. Determine the appropriate data type for each of the following values:
   a) the number of basketballs in a department store.
   b) the price of a basketball.
   c) the number of players on a basketball team.
   d) the average age of the players on a basketball team.
   e) whether a basketball player has received a jersey or not.
   f) the first initial of a basketball player's first name.

8. The statement Dim value As Integer = 3 appears in a Click event procedure for a button. The Click event procedure for a second button, contains the statements:
   Dim value As Integer
   Me.lblOutput.Text = value
   a) Is the scope of value local or global?
b) What is shown in the label? Why?

9. What is the value of each of the following expressions?
   a) 10 / 5 * 4
   b) 10 / 2 + 3
   c) 6 Mod 3 + 4
   d) 12 Mod 5 * 3
   e) 12 Mod (5 * 3)

10. What is the result of the following expression when x is 2005? When x is 1776? When x is 39?
    (x\10)Mod10

11. Determine if each of the following are better represented by a variable or a constant and then write declarations using appropriate data types and descriptive identifiers:
    a) the number of votes received by an election candidate
    b) the percentage of votes won by a candidate
    c) the first, middle, and last initials of an election candidate
    d) the year of the election

12. For each of the following determine if the code contains a syntax error, logical error, or runtime error, and then correct the code.
    a) Dim num = 6
    b) num = Value(Me.txtNum.Text)
    c) lbloutput = message
    d) Dim test1 As Integer
       Dim test2 As Integer
       Dim avg As Double
       test1 = Val(Me.txtTest1.Text)
       test2 = Val(Me.txtTest2.Text)
       avg = test1 + test2 / 2
    e) Dim numberTests As Integer
       Dim test1 As Integer = 80
       Dim test2 As Integer = 88
       Dim avg As Double
       avg = (test1 + test2) / numberTests
13. What value is assigned to taxRate in the following statement if seven percent is typed in the txtTaxRate text box?

taxRate = Val(Me.txtTaxRate.Text)

14. Write each equation as a Visual Basic assignment statement, assuming π is a constant named PI:

   a) \( A = lw \) (geometry)

   b) \( P = \frac{R - C}{N} \) (business)

   c) \( A = \frac{j(b_1 + b_2)}{N} \) (geometry)

   d) \( V = \frac{4}{3} \pi r^3 \) (geometry)

   e) \( A = \frac{F + S + T}{3} \) (algebra)

   f) \( P = \frac{5F}{4d^2} \) (physics)

   g) \( A = P + Prt \) (business)

   h) \( M = \frac{Pr(1+r)^n - 1}{(1+r)^n} \) (algebra)

   i) \( x = \frac{-b \sqrt{b^2 - 4ac}}{2a} \) (algebra)

15. Rewrite the equations in parts (a) through (h) of question 12 to solve for the variable listed below and then write each equation as a Visual Basic assignment statement:

   a) \( I \)

   b) \( R \)

   c) \( b \)

   d) \( r \)

   e) \( T \)

   f) \( F \)

   g) \( P \)

   h) \( P \)

True/False

16. Determine if each of the following statements is true or false. If false, explain why.

a) A variable must be declared before it is used.

b) Multiple variables with different data types can be declared in a single statement.

c) A runtime error results in the termination of program execution.

d) The Val() function converts text box data to a string value.

e) An Integer variable uses 4 bytes of memory to store its value.

f) Boolean variables are initialized to True.

g) In the variable assignment statement \( x = 4.5 \), where \( x \) is an Integer variable, \( x \) is assigned the value 4.

h) Modulus division returns the remainder resulting from division.

i) In the variable assignment statement \( x = 14 \mod 4 \times 2 \), where \( x \) is a Double variable, \( x \) is assigned the value 4.

j) Constant declarations must be placed in the beginning of a program.

k) Visual Basic sees no difference between the variables NEW and New.

l) Keywords make good variable identifiers.

m) Private is a keyword.

n) The Visual Basic IDE informs the programmer of a syntax error.

o) Errors that violate the rules of Visual Basic are called semantic errors.

p) Run-time errors are always detected by the compiler.

q) A breakpoint temporarily stops the execution of a program, but pressing the right-arrow key continues execution.

Chapter 3 Variables and Constants
Exercises

Exercise 1  
ObjectHeight

The height of an object at any given time dropped from a starting height of 100 meters is given by the equation \( h = 100 - 4.9t^2 \) where \( t \) is the time in seconds. Create an ObjectHeight application that prompts the user for a time less than 4.5 seconds and then displays the height of the object at that time when Height is clicked. The application interface should look similar to:

![Object Height Application](image)

Exercise 2  
TemperatureConversion

a) Create a TemperatureConversion application that prompts the user for a temperature in degrees Fahrenheit and then displays the temperature in degrees Celsius when Celsius is clicked. Use the formula \( C = \frac{5}{9}(F - 32) \) to make the conversion. Test the program with values 212, 32, 98.6, and -40. The application interface should look similar to:

![Temperature Conversion Application](image)

b) Modify the TemperatureConversion application to include a Program menu with Celsius, Fahrenheit, and Exit commands. Use the formula in part (a) above to determine the formula for converting from Celsius to Fahrenheit. Modify the prompt to display Enter the temperature: Modify the temperature label appropriately. Remove the Celsius button and corresponding code and size the form appropriately.
Exercise 3  **RectangleAreaAndPerimeter**

Modify the RectangleAreaAndPerimeter application created in Chapter 2 Exercise 8 to prompt the user for the length and width of the rectangle and then display the area and perimeter of the rectangle when Answer is clicked. The application interface should look similar to:

![Rectangle Area And Perimeter](image)

Exercise 4  **PizzaCost**

The cost of making a pizza at a local shop is as follows:

- Labor cost is $0.75 per pizza, regardless of size
- Rent cost is $1.00 per pizza, regardless of size
- Materials is $0.05*diameter*diameter (diameter is measured in inches)

Create a PizzaCost application that prompts the user for the size of a pizza and then displays the cost of making the pizza when Cost is clicked. The application interface should look similar to:

![Pizza Cost](image)
Exercise 5  

Einstein's famous formula, $e=mc^2$, gives the amount of energy released by the complete conversion of matter of mass $m$ into energy $e$. If $m$ represents the mass in kilograms and $c$ represents the speed of light in meters per second ($3.0 \times 10^8 \text{ m/s}$), then the result is in the energy units Joules. It takes 360000 Joules to light a 100-watt light bulb for an hour. Create an Energy application that prompts the user for a mass in kilograms and then displays the energy and the number of light bulbs that could be powered when Energy is clicked. The application interface should look similar to:

![Energy Application Interface]

Exercise 6  

Modify the LongJumpAverage application created in Chapter 2 Exercise 9 to prompt the user for the lengths of four long jumps and then display the average jump length when Average is clicked. The application interface should look similar to:

![Long Jump Average Application Interface]
Exercise 7  
Create a Change application that prompts the user for an amount less than 100 and then displays the minimum number of coins necessary to make the change when Coins is clicked. The change can be made up of quarters, dimes, nickels, and pennies. The application interface should look similar to:

![Change application interface](image)

Exercise 8  
a) Create a DigitsOfANumber application that prompts the user for a two-digit number and then displays the digits separately when Digits is clicked. The application interface should look similar to:

![Digits of a Number application interface](image)

b) Modify the DigitsOfANumber application to include a Program menu with a Digits command.

Exercise 9  
Create a TimeConversion application that prompts the user for a time in minutes and then displays the time in seconds or hour:minute format depending on the radio button clicked. Be sure to consider times where the number of minutes left over is less than 10. For example, 184 minutes in hour:minute format is 3:04 (Hint: use the modulus operator). The application interface should look similar to the following after clicking Minutes to hour:minute format:

![Time Conversion application interface](image)