end. Variable names should begin with an uppercase character and be mixed upper- and lowercase, while constants are all uppercase.

4. Identifiers should include the data type of the variable or constant.
5. Intrinsic constants, such as Color.Red and Color.Blue, are predefined and built into the .NET Framework. Named constants are programmer-defined constants and are declared using the `Const` statement. The location of the `Const` statement determines the scope of the constant.
6. Variables are declared using the `Private` or `Dim` statement; the location of the statement determines the scope of the variable. Use the `Dim` statement to declare local variables inside a procedure; use the `Private` statement to declare module-level variables at the top of the program, outside of any procedure.
7. The scope of a variable may be namespace level, module level, local, or block level. Block-level and local variables are available only within the procedure in which they are declared; module-level variables are accessible in all procedures within a form; namespace variables are available in all procedures of all classes in a namespace, which is usually the entire project.
8. The lifetime of local and block-level variables is one execution of the procedure in which they are declared. The lifetime of module-level variables is the length of time that the form is loaded.
9. Use the `Parse` methods to convert text values to numeric before performing any calculations.
10. Calculations may be performed using the values of numeric variables, constants, and the properties of controls. The result of a calculation may be assigned to a numeric variable or to the property of a control.
11. A calculation operation with more than one operator follows the order of precedence in determining the result of the calculation. Parentheses alter the order of operations.
12. To explicitly convert between numeric data types, use the Convert class. Some conversions can be performed implicitly.
13. The `Decimal.Round` method rounds a decimal value to the specified number of decimal positions.
14. The `ToString` method can be used to specify the appearance of values for display. By using formatting codes, you can specify dollar signs, commas, percent signs, and the number of decimal digits to display. The method rounds values to fit the format.
15. `Try/Catch/Finally` statements provide a technique for checking for user errors such as blank or nonnumeric data or an entry that might result in a calculation error.
16. A run-time error is called an *exception*; catching and taking care of exceptions is called *error trapping* and *error handling*.
17. You can trap for different types of errors by specifying the exception type on the `Catch` statement, and you can have multiple `Catch` statements to catch more than one type of exception. Each exception is an instance of the Exception class; you can refer to the properties of the Exception object for further information.
18. A message box is a window for displaying information to the user.
19. The `Show` method of the MessageBox class is overloaded, which means that the method may be called with different argument lists, called *signatures*.
20. You can calculate a sum by adding each transaction to a module-level variable. In a similar fashion, you can calculate a count by adding to a module-level variable.