

**FUNDAMENTALS OF COMPUTING
LECTURE NOTES**

**PROGRAMMING WITH
VISUAL BASIC 6.0**

CHAPTER 4

**“VISUAL BASIC VARIABLES AND
OPERATORS”**

PREPARED BY

Mazhar Javed

CHAPTER 4: VISUAL BASIC VARIABLES AND OPERATORS

This is the first of three chapters that focus specifically on writing program code to manage the events in a Visual Basic application. This chapter describes Visual Basic variables and operators and tells you how to use specific data types to improve program efficiency. In this chapter, you will learn how to:

- ◆ Use variables to store information in your program.
- ◆ Work with specific data types to streamline your calculations.

4.1. Using Variables to Store Information

A **variable is a temporary data storage** location in your program. Your code can use one or many variables, which can contain words, numbers, dates, properties, or object references. Variables are useful because they let you assign a short, easy-to-remember name to a piece of data you plan to work with. Variables can hold:

- ◆ User information entered at run time.
- ◆ The result of a specific calculation.
- ◆ A piece of data you want to display on your form.

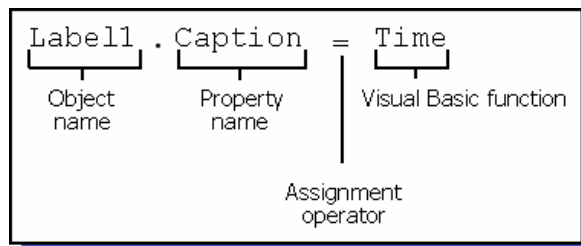
In short, variables are simple tools you can use to track almost any type of information whose value can change.

4.1.1. The Anatomy of a Visual Basic Program Statement

A program statement is a line of code in a Visual Basic program. Your program statements can contain any combination of Visual Basic keywords, identifiers, properties, functions, operators, and symbols that Visual Basic recognizes as a valid instruction. A complete program statement can be a simple keyword:

Beep

This program statement sounds a note from your computer's speaker. Or, a statement can be a combination of elements, such as the statement showing in this illustration, which assigns the current system time to a label caption.



Statement Syntax

The anatomy of a program statement is syntax, the rules of construction that you must use when you build a program statement. Visual Basic shares many of its syntax rules with earlier versions of the Basic programming language and with other language compilers.

The trick to writing good program statements is learning the syntax of the most useful language elements. Then, it's a matter of using those elements correctly to process the data in your program. Fortunately, Visual Basic does a lot of the toughest work for you. The time you spend writing program code will be relatively short, and the results can be used again in future programs.

In the following topics, you'll learn the most important Visual Basic keywords and program statements. You'll find that they complement the programming skills you've already learned rather nicely—and they can help you to write powerful programs in the future.

4.1.2.Creating Variables

Declaring a variable is the process of creating a variable in your Visual Basic program code. You can do this explicitly with the **Dim** keyword or implicitly by typing a new variable name in a program statement.

This section includes the following topics:

- ◆ Explicit Declarations
- ◆ Implicit Declarations

Explicit Declarations

In Visual Basic, one of the ways to create a variable is to declare it explicitly. Typically, you declare a variable explicitly before you use the variable (usually at the beginning of an event procedure). You declare the variable by typing a **Dim** (dimension) statement and the variable name. For example, the following statement creates space for a program variable named LastName:

```
Dim LastName
```

Typing the statement reserves room for the variable in memory when the program runs, and lets Visual Basic know what type of data it should expect to see later.

Specifying the Variable Data Type

If you like, you can specify the variable data type after you type the variable name. (You'll learn about several fundamental data types in Working with Specific Data Types.) With Visual Basic, you can identify the data type in advance, so you can control how much memory your program uses. For example, if the variable holds an integer (a small number without any decimal places), you can declare the variable as an integer and save some memory.

Default Data Type

By default, however, Visual Basic reserves space for a variable data type called a **Variant** (a variable that can hold data of any size or format). This general-purpose data type is extremely flexible — in fact, it might be the only variable you use in your programs.

Assignment Operator

After you declare a variable, you are free to assign information to it in your code by using the **assignment operator (=)**. This statement assigns the name "Jefferson" to the LastName variable:

```
LastName = "Jefferson"
```

After this assignment, you can substitute the `LastName` variable for the name "Jefferson" in your code. For example, this assignment statement would display *Jefferson* in the first label (Label1) on your form:

```
Label1.Caption = LastName
```

Implicit Declarations

You can also declare a variable without the **Dim statement**; this process is called implicit declaration. To declare a variable implicitly, you simply use the variable on its own and skip the **Dim** statement altogether. Here's an example:

```
LastName = "Charles V"
```

Advantages and Disadvantages

In this course, you'll see variables declared with both explicit and implicit techniques. Implicit declaration has the advantage of speed because you don't spend time typing the **Dim** statement. However "the management" often discourages it for several reasons. First, implicit declaration doesn't force you to organize and list your variables in advance. Also, creating variables in this way prevents Visual Basic from displaying an error message if you mistype the variable name later.

Note: If you decide to always declare your variables by using the **Dim** statement, you can ask Visual Basic to enforce your decision by setting the **Require Variable Declaration** checkbox in the **Options** dialog box. When you set this option, Visual Basic generates an error message whenever it finds a variable that has not been explicitly declared in the code.

► To require variable declaration

1. From the **Tools** menu, click **Options**.
2. Click the **Editor** tab.
3. Select the **Require Variable Declaration** check box.

To turn off this option, simply carry out these steps in reverse order.

Changing Variable Values

Variables can maintain the same value throughout a program; or, more likely, they can change values several times, depending on your needs.

To view sample code that shows a variant variable named `LastName` can contain both text and a number, and how the variable can be assigned to object properties.

```
Dim LastName           'declare LastName variable

LastName = "Smart"     'store "Smart" in variable
Label1.Caption = LastName 'display "Smart" in Label1 object

LastName = 99          'now place 99 in variable
Label2.Caption = LastName 'display this value in Label2 object
```

4.1.3.Using Functions

In this section, you learn how to use these three special Visual Basic keywords, called functions, in a program.

- ◆ The **MsgBox** function, which displays output in a popup dialog box.
- ◆ The **InputBox** function, which prompts the user to supply input in a dialog box.
- ◆ Mathematical functions, with which Visual Basic calculates new values.

This section includes the following topics:

- ◆ What is a Function?
- ◆ Using the MsgBox Function
- ◆ Using the InputBox Function
- ◆ Demonstration: Using InputBox and MsgBox
- ◆ Mathematical Functions

What is a Function?

A Visual Basic function is a statement that performs **meaningful work** (such as prompting the user for information) and returns a value to the program. The value that a Visual Basic function returns can be assigned to a variable, a property, another statement, or another function. A Visual Basic function is different from a Function procedure that you write yourself.

Visual Basic functions often include one or more arguments that define their activities. For example, the **InputBox** function uses the Prompt variable to display a dialog box with instructions for the user. When a function uses one or more arguments, they **are separated by commas**, and the whole group of arguments is enclosed in parentheses. The following statement shows a function call that has two arguments:

```
FullName = InputBox$(Prompt, Title)
```

The **Prompt** argument takes a string that displays a prompt for the user in the **Input** box, and the **Title** argument takes a string that displays a title in the input box title bar.

Using The MsgBox Function

MsgBox is a useful dialog box function that displays output. Like **InputBox**, **MsgBox** takes one or more arguments as input, and you can assign the value returned (the results of the function call) to a variable. This is the syntax for **MsgBox**:

```
ButtonClicked = MsgBox(Message, NumberOfButtons, Title)
```

where

- ◆ **Message** is the text to be displayed on the screen.
- ◆ **NumberOfButtons** is a button style number (0 through 5).
- ◆ **Title** is the text displayed in the message box title bar.
- ◆ **ButtonClicked** is assigned the result returned by the function, which indicates which button in the dialog box the user clicked.

Note If you want to display only a message with **MsgBox**, the assignment operator (=), the *ButtonClicked* variable, the *NumberOfButtons* argument, and the *Title* argument are optional items.

For more information about these optional arguments (including the different buttons you can use in a message box), search for *MsgBox function* in the Visual Basic online Help.

Using The InputBox Function

One **excellent use** for a variable is to hold **user input** information. Often, you can use an object such as a file list box or a text box to retrieve this information. At times, though, you'll want to deal directly with the user and save the input in a variable rather than in a property. One way to do this is to use the **InputBox** function to display a dialog box on the screen and then store in a variable the text that the user types.

InputBox syntax looks like this:

```
VariableName = InputBox$(Prompt)
```

where

VariableName is a variable used to hold the input.

Prompt is a prompt that appears in the dialog box.

The dialog box created with an **InputBox** function typically contains these features:

- ◆ A prompt for directing the user.
- ◆ A text box for receiving typed input.
- ◆ Two command buttons, **OK** and **Cancel**.

Mathematical Functions

Now and then, you might want to do a little number crunching in your programs. You may need to convert a value to a different type, calculate a complex mathematical expression, or introduce randomness to your programs.

Visual Basic functions can help you work with numbers in your formulas. As with any function, mathematical functions appear in a program statement and return a value to the program. In the following table, the argument *n* represents the number, variable, or expression you want the function to evaluate.

Function	Purpose
----------	---------

Abs(*n*) Returns the absolute value of *n*.

Atn(*n*) Returns the arctangent, in radians, of *n*.

Cos(*n*) Returns the cosine of the angle *n*. The angle *n* is expressed in radians.

Exp(*n*) Returns the constant *e* raised to the power *n*.

Int(*n*) Returns the integer portion of *n*.

Rnd(*n*) Generates a random number greater than or equal to 0 and less than 1.

Sgn(*n*) Returns -1 if *n* is less than zero, 0 if *n* is zero, and +1 if *n* is greater than zero.

Sin(*n*) Returns the sine of the angle *n*. The angle *n* is expressed in radians.

Sqr(*n*) Returns the square root of *n*.

Str(*n*) Converts a number to a numeric string value.

Tan(*n*) Returns the tangent of the angle *n*. The angle *n* is expressed in radians.

Val(*n*) Converts a numeric string value to a number.

For example, the following formula uses the **Sqr** (square root) function to calculate the hypotenuse of a triangle by using the variables *a* and *b*:

```
Hypotenuse = Sqr(a ^ 2 + b ^ 2)
```

4.2. Working with Specific Data Types

The **Variant** data type works well in most situations. There are times, though, when working with specific data types can improve your Visual Basic programs. Specifying variable size can help you increase program performance. Creating a user-defined data type can help you hold information that is formatted in an unusual format. And if the variable never changes its value, declaring it as a constant can help you increase program efficiency. In this section, you round out your experience with variables by exploring other aspects of data types.

This section includes the following topics:

- ◆ Fine-Tuning Variable Size
- ◆ User-Defined Data Types
- ◆ Constants

4.2.1. Fine Tuning Variable Size

In most cases, the **Variant** data type will be the only data type you need. When you use **Variant** variables, you can store all Visual Basic predefined data types and switch formats automatically. **Variants** are also easy to use, and you don't have to give much thought to the eventual size of the variable when you declare it.

If you want to create especially fast and concise code, however, you may want to use more specific data types. For example, a variable might always contain small integers (numbers without a decimal point). When this occurs, declaring the variable as an **Integer** rather than as a **Variant** can save memory when the program runs. An **Integer** variable will speed up arithmetic operations too, so you can gain a small speed advantage when you use it.

Declaring Data Types

Using specific data types is really not that much different than using Variant variables. You can specify some fundamental data types by adding a type-declaration character to the variable name. (Several data type declarations appear in the table below.) For example, you can declare a variable as an Integer type by adding a % character to the end of its name. So, in Visual Basic, the following two declaration statements are equivalent:

```
Dim I As Integer  
Dim I%
```

The following table lists the fundamental data types in Visual Basic. Use this table as a reference list if you decide to fine-tune the size and type of your variables.

Data Type	Size	Range
Byte	1 byte	0 through 255
Integer	2 bytes	-32,768 through 32,767
Long	4 bytes	-2,147,483,648 through 2,147,483,647
Single	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values

Double	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency	8 bytes	-922337203685477.5808 through 922337203685477.5807
String (variable length)	10 bytes	+ 1 byte per character 0 to approximately 2 billion
String (fixed length)		1 through 65,400 characters
Boolean	2 bytes	True or False
Date	8 bytes	January 1, 100 through December 31, 9999
VARIANT	16 bytes (with numbers)	22 bytes + 1 byte per character (with strings) All data type ranges

4.2.2. User-Defined Data Types

Visual Basic also lets you create your own data types. This feature is most useful when you work with mixed data types that naturally fit together.

Creating User-Defined Data Types

To create a user-defined data type, you:

- ◆ Use the **Type** statement in the Declarations section of a form or standard module.
- ◆ Declare variables associated with the new data type (variables belonging to a type are not declared with the **Dim** statement.)

This sample code illustrates the **Type** statement. The declaration uses **Type** to create **Employee**, a user-defined data type. (**Employee** stores a worker's name, date of birth, and hire date.)

```
Type Employee
    Name As String
    DateOfBirth As Date
    HireDate As Date
End Type
```

Note If you want to place the **Type** statement in the Declarations section of a form module, you must precede the **Type** statement with the **Private** keyword.

Placing User-Defined Data in Program Code

After you create a new data type, you can use it in the program code. The following Dim statement uses the user-defined **Employee** data type. The first statement creates a **variable** named ProductManager and states that it's the **Employee** data type. The second statement assigns the name "Erick Cody" to the Name component of the variable:

```
Dim ProductManager As Employee
ProductManager.Name = "Erick Cody"
```

This looks a little like setting a property, doesn't it? Visual Basic uses the same notation for the relationship between objects and properties as it uses for the relationship between user-defined data types and their component variables.

4.2.3.Constants

If a variable in your program contains a value that never changes, you should consider storing the value as a constant instead of as a variable. A constant is a meaningful name that takes the place of a number or text string that doesn't change (such as π , a fixed mathematical quantity.)

Advantages of using constants include:

- ◆ Making program code more readable
- ◆ Saving memory
- ◆ Making program -wide changes easier to accomplish.

How Constants Work

Constants operate a lot like variables, but you can't modify their values at run time. You declare (define) constants with the **Const** keyword, as shown in the following example:

```
Const Pi = 3.14159265
```

This statement creates a constant called Pi that you can use in place of the value of π in the program code. To create a constant that's available to procedures throughout a program, use the **Public** keyword to create the constant in a standard module. For example:

```
Public Const Pi = 3.14159265
```

Displaying Constants

The following program statement shows how you can display the value contained in the Pi constant with an object named Label1:

```
Label1.Caption = Pi
```

Constants are very useful in program code, especially when your program includes mathematical formulas, such as $\text{Area} = 2\pi r^2$. The next section describes how you can use operators and variables to write mathematical formulas.

4.3.Using Visual Basic Operators

Visual Basic contains several language elements designed for use in formulas (statements that use a combination of numbers, variables, operators, and keywords to create a new value). These language elements are mathematical operators, the symbols used to tie together the parts of a formula. With a few exceptions, these mathematical symbols are the ones you use in everyday life — their operations are quite intuitive.

This section includes the following topics:

- ◆ Basic Arithmetic Operators
 - ◆ Advanced Operators
 - ◆ Operator Precedence
-

4.3.1. Basic Arithmetic Operators

The operators for addition, subtraction, multiplication and division are familiar, and their use is straightforward — you can use them in any formula in which numbers or numeric variables appear. The following table shows the operators you can use for basic arithmetic:

Operator	Mathematical operation
+	Addition
-	Subtraction
*	Multiplication
/	Division (floating-point)

For example, the following program statements use the addition and multiplication operators to calculate the total cost of a \$250 bicycle including 8.1% sales tax:

```
Dim BicycleCost, TotalPrice
Const SalesTaxRate = 0.081
BicycleCost = 250
TotalPrice = BicycleCost * SalesTaxRate + BicycleCost
```

4.3.2. Advanced Operators

In addition to the four basic arithmetic operators, Visual Basic includes four advanced operators, which are useful in special-purpose mathematical formulas and text processing applications:

Operator	Mathematical operation
\	Integer (whole number) division
Mod	Remainder division
^	Exponentiation (raising to a power)
&	String concatenation (combination)

For example, the following program statement uses the **Time** and **Date** functions and the string concatenation operator (&) to build a sentence from four string values:

```
Label1.Caption = "The current time is " & Time & " on " & Date
```

When you execute the program statement, the **Label1** object displays output in the following format:

```
The current time is 5:14:16 PM on 12/5/97
```

4.3.3. Operator Precedence

In the last few sections, you experimented with seven mathematical operators and one string operator. In Visual Basic, you can mix as many mathematical operators in a formula as you like—as long as an operator separates each numeric variable and expression. For example, this is an acceptable Visual Basic formula:

```
Total = 10 + 15 * 2 / 4 ^ 2
```

The formula processes several values and assigns the result to a variable named Total. But you're probably wondering how Visual Basic evaluates this expression. After all, there's no way to tell which mathematical operators Visual Basic uses first to solve the formula. In this example, the order of evaluation matters a great deal.

Visual Basic solves this dilemma by using a specific order of precedence for mathematical operations. When Visual Basic evaluates an expression that contains more than one operator, the order of precedence supplies rules that determine which operators Visual Basic evaluates first.

The rules can be summarized by these principles:

- ◆ Use the operators in the order of precedence.
- ◆ For operators on the same precedence level, evaluate from left to right as they appear in an expression.

The following table lists mathematical operators in their order of precedence:

Order of precedence	Symbols	Operator
First	()	The values between parentheses
2	^	Exponentiation (raising a number to a power)
3	-	Negation (creating a negative number)
4	* /	Multiplication and division
5	\	Integer division
6	Mod	Remainder division
7	+ -	Addition and subtraction
Last	&	String concatenation

Here's the formula you saw earlier:

Total = 10 + 15 * 2 / 4 ^ 2

Given the order of precedence, the expression would be evaluated by Visual Basic in the following steps. (Boldface type highlights what's happening in each step.)

Total = 10 + 15 * 2 / **4 ^ 2** Exponentiation

Total = 10 + **15 * 2** / 16 Multiplication

Total = 10 + **30 / 16** Division

Total = **10 + 1.875** Addition

Total = **11.875** Result

Using Parenthesis in a Formula

You can use one or more pairs of parentheses in a formula to clarify the order of precedence. For example, here's how Visual Basic would calculate this simple formula:

```
Number = (8 - 5 * 3) ^ 2
```

```
Number = (-7) ^ 2
```

```
Number = 49
```

Visual Basic determines the expression between the parentheses (7) before doing the exponentiation — even though exponentiation has a higher order of precedence than subtraction and multiplication do.

Nested Parentheses

You can further refine the calculation by placing nested parentheses in the formula. Here's the same formula with a new twist:

```
Number = ((8 - 5) * 3) ^ 2
```

Embedding (nesting) the second set of parentheses in the formula directs Visual Basic to calculate the formula this way:

```
Number = ((8 - 5) * 3) ^ 2
```

```
Number = (3 * 3) ^ 2
```

```
Number = 9 ^ 2
```

```
Number = 81
```

As you can see, the results produced by the two formulas are different: 49 in the first formula and 81 in the second. So, adding parentheses can change the result of a mathematical operation as well as make it easier to read.

4.4. EXAMPLE: Savings Account

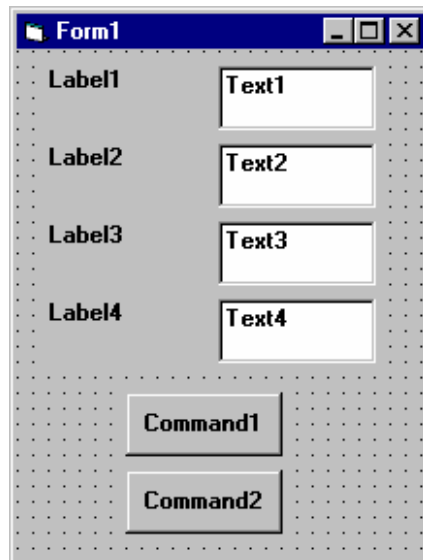
1. Start a new project. The idea of this project is to determine how much you save by making monthly deposits into a savings account. For those interested, the mathematical formula used is:

$$F = D [(1 + I)^M - 1] / I$$

where

F - Final amount
D - Monthly deposit amount
I - Monthly interest rate
M - Number of months

2. Place 4 label boxes, 4 text boxes, and 2 command buttons on the form. It should look something like this:
-



3. Set the properties of the form and each object.

Form1:

BorderStyle	1-Fixed Single
Caption	Savings Account
Name	frmSavings

Label1:

Caption	Monthly Deposit
---------	-----------------

Label2:

Caption	Yearly Interest
---------	-----------------

Label3:

Caption	Number of Months
---------	------------------

Label4:

Caption	Final Balance
---------	---------------

Text1:

Text	[Blank]
Name	txtDeposit

Text2:

Text	[Blank]
Name	txtInterest

Text3:

Text	[Blank]
Name	txtMonths

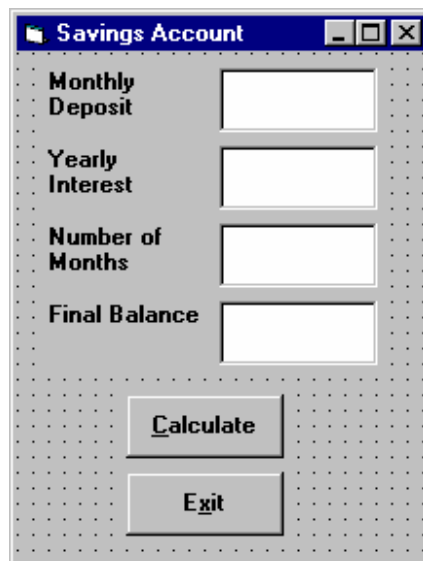
Text4:

Text	[Blank]
Name	txtFinal

Command1:
Caption &Calculate
Name cmdCalculate

Command2:
Caption E&xit
Name cmdExit

Now, your form should look like this:



4. Declare four variables in the **general declarations** area of your form. This makes them available to all the form procedures:

```
Option Explicit  
Dim Deposit As Single  
Dim Interest As Single  
Dim Months As Single  
Dim Final As Single
```

The **Option Explicit** statement forces us to declare all variables.

5. Attach code to the **cmdCalculate** command button **Click** event.

```
Private Sub cmdCalculate_Click ()  
Dim IntRate As Single  
`Read values from text boxes  
Deposit = Val(txtDeposit.Text)  
Interest = Val(txtInterest.Text)  
IntRate = Interest / 1200  
Months = Val(txtMonths.Text)  
`Compute final value and put in text box  
Final = Deposit * ((1 + IntRate) ^ Months - 1) / IntRate  
txtFinal.Text = Format(Final, "#####0.00")  
End Sub
```

This code reads the three input values (monthly deposit, interest rate, number of months) from the text boxes, computes the final balance using the provided formula, and puts that result in a text box.

6. Attach code to the `cmdExit` command button **Click** event.

```
Private Sub cmdExit_Click ()  
End  
End Sub
```

7. Play with the program. Make sure it works properly. Save the project.

-END OF CHAPTER 4-
