**Introduction**

Visual Basic has two devices, **Function procedures** and **Sub procedures**, that are used to break complex problems into small problems to be solved one at a time. To distinguish them from event procedures, Function and Sub procedures are referred to as **general procedures**. General procedures allow us to write and read a program in such a way that we first focus on the tasks and later on how to accomplish each task. They also eliminate repetitive code and sometimes can be reused in other programs.

Although the input can consist of several values, the output is always a single value. A function is said to **return** its output. For instance, in the first example below we say that the Int function returns the value 2. The items inside the parentheses are called **arguments**. The first three functions have one argument and the fourth function has two arguments. Arguments can be literals, variables, or expressions. Variables are the most common types of arguments. The following lines of code illustrate the use of variables and expressions as arguments for the Int function:

**Dim num1 As Double = 2.6**

**Dim num2 As Double = Int(num1) 'variable as an argument**

**Dim num1 As Double = 1.3**

**Dim num2 As Double = Int(2 \* num1) 'expression as an argument**

The second line of code above is said to **call** the Int function and to **pass** the value of *num1* to the function.

**Function Procedures**

In addition to using built-in functions, we can define functions of our own. These new functions, called **Function procedures** are used in the same way as built-in functions. Like built-in functions, Function procedures have a single output that can be of any data type. Function procedures are used in the same way as built-in functions. Function procedures are defined by function blocks of the form

**Function *FunctionName* (var1 As *Type1* , var2 As *Type2* ,...) As *ReturnDataType***

***statement* (*s*)**

**Return *expression***

**End Function**

The first line of the procedure is called the **header**. The variables appearing in the header are called **parameters**. The header declares a parameter to be of a certain type and sets aside a portion of memory to hold its value. The scope of each parameter is limited to its function block, as is any variable declared inside the Function procedure.

Function names should indicate the role performed by the function and must conform to the rules for naming variables. By convention, function names begin with an uppercase letter. *ReturnDataType,* which specifies the type of the output, will be one of String, Integer, Decimal, Double, Date, Boolean, and so on. In the preceding general code, the Return statement specifies the output, which must be of type *ReturnDataType.* Function procedures can contain several Return statements and must contain at least one.

Function procedures are typed directly into the Code Editor outside any other procedure. After you type the header and then press the Enter key, the editor automatically inserts the line “End Function” and a blank line separating the two lines of code. Also, the smart indenting feature of the Code Editor automatically indents all lines in the block of code between the header and “End Function” statements.

**■ Functions Having One Parameter**

The following two Function procedures have just one parameter.

Name of Function Name used by Function replacing Fahrenheit which is passed(t and return must match Fahrenheit type

**Function FtoC(t As Double) As Double**

**'Convert Fahrenheit temperature to Celsius**

**Return (5 / 9) \* (t − 32)**

**End Function**

**Function FirstName(fullName As String) As String**

**'Extract the first name from a full name**

**Dim firstSpace As Integer**

**firstSpace = fullName.IndexOf(" ")**

**Return fullName.Substring(0, firstSpace)**

**End Function**

**How are they used. Examples show object table and other documentation**

The following program uses the Function procedure FtoC. The fourth line of the btnConvert\_Click event procedure, **celsiusTemp = FtoC(fahrenheitTemp)**, calls the function FtoC. The value of the argument *fahrenheitTemp* is assigned to the parameter *t* in the Function procedure header. (We say that the value of *fahrenheitTemp* is passed to the parameter *t*.) After the Function procedure does a calculation using the parameter *t*, the calculated value is the output of the function FtoC and is assigned to the variable *celsiusTemp.*

| **OBJECT** | **PROPERTY** | **SETTING** |
| --- | --- | --- |
| frmConvert | Text | Temperature |
| lblTempF | Text | Temperature (Fahrenheit): |
| txtTempF |  |  |
| btnConvert | Text | Convert to Celsius |
| lblTempC | Text | Temperature (Celsius): |
| txtTempC | ReadOnly | True |
|  |  |  |

**Private Sub btnConvert\_Click(...) Handles btnConvert.Click**

**Dim fahrenheitTemp, celsiusTemp As Double**

**fahrenheitTemp = CDbl(txtTempF.Text)**

**celsiusTemp = FtoC(fahrenheitTemp) ‘ Function call passes fahrenheitTemp to function**

**txtTempC.Text = CStr(celsiusTemp)**

**'Note: The above four lines can be replaced with the single line**

**'txtTempC.Text = CStr(FtoC(CDbl(txtTempF.Text)))**

**End Sub**

**Function FtoC(t As Double) As Double ‘ Fuction FtoC assigns FahrenheitTemp to ‘t’**

**'Convert Fahrenheit temperature to Celsius**

**Return** **Math.Round((5 / 9) \* (t − 32), 2) ‘ ‘t’ is used in math formula**

**End Function**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The following program uses the Function procedure FirstName. The fifth line of the btnDetermine\_Click event procedure, **txtFirstName.Text = FirstName(fullName)**, passes the value of the argument *fullName* to the parameter *fullName* in the Function procedure. Although the parameter in the Function procedure has the same name as the argument passed to it, they are different variables.

| **OBJECT** | **PROPERTY** | **SETTING** |
| --- | --- | --- |
| frmFirstName | Text | First Name |
| lblName | Text | Full name: |
| txtFullName |  |  |
| btnDetermine | Text | Determine First Name |
| lblFirstName | Text | First name: |
| txtFirstName | ReadOnly | True |

**Private Sub btnDetermine\_Click(...) Handles btnDetermine.Click**

**'Determine a person's first name**

**Dim fullName As String**

**fullName = txtFullName.Text**

**txtFirstName.Text = FirstName(fullName)**

**End Sub**

**Function FirstName(fullName As String) As String**

**'Extract the first name from a full name**

**Dim firstSpace As Integer**

**firstSpace = fullName.IndexOf(" ") ‘ Returns the position of the space (6 for Thomas” “)**

**Return fullName.Substring(0, firstSpace)’ substr(begin in position 0 and assign the next 6 to firstname**

**End Function**

# Passing by Value (ByVar)

If the argument in a function call is a variable, the *value* of the argument variable (not the argument variable itself) is passed to a parameter variable. Therefore, if the Function procedure happens to change the value of the parameter variable, no change will occur in the argument variable. Such is the case even if the two variables have the same name. For instance, after the following program is run and the button is clicked on, the value 6 will appear in TextBox1 and the value 2 will appear in Textbox2. The two variables have the same name, but are different variables. This is analogous to the situation in which variables in two different event procedures have the same name, but separate identities. This method of passing information to a Function procedure is called **passing by value**.

For Example

**Private Sub btnTriple\_Click(...) Handles btnTriple.Click**

**Dim num As Integer = 2**

**TextBox1.Text = CStr(Triple(num))**

**TextBox2.Text = CStr(num)**

**End Sub**

**Function Triple(num As Integer) As Integer**

**num = 3 \* num**

**Return num**

**End Function**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Sub procedures** are just Function procedures that don’t return values. Sub procedures and Function procedures share the following features:

* Both are written as a separate block of code that can be called to perform a specific task.
* Both are used to break complex problems into small problems.
* Both are used to eliminate repetitive code.
* Both can be reused in other programs.
* Both make a program easier to read by separating it into logical units.
* Both have parameters that are declared in a header.

Sub procedures, however, do not return a value associated with their name. The most common uses of Sub procedures are to receive input, process input, or display output.

# Defining and Calling Sub Procedures

Sub Procedures

The Sub procedures discussed in this section are defined by blocks of the form

**Sub *ProcedureName* ( *par1* As *Type1* , *par2* As *Type2* ,..., *parN* As *TypeN* )**

***statement(s)***

**End Sub**

The primary difference between Sub procedures and Function procedures is that Sub procedures perform tasks (such as displaying output) rather than return values. Like Function procedures, the names of Sub procedures must conform to the rules for naming variables. By convention, a Sub procedure’s name begins with an uppercase letter and describes the procedure’s purpose.

Sub procedures are called by statements of the form

***ProcedureName* (*arg1,* *arg2,* ..., *argN*)**

When a Sub procedure is called, the value of each argument is assigned to the corresponding parameter, the statement(s) inside the procedure block is (are) carried out, and execution continues with the statement following the calling statement.

Here is an example of a Sub procedure.

**Sub DisplaySum(num1 As Double, num2 As Double)**

**Dim z As Double**

**z = num1 + num2**

**lstOutput.Items.Add(z)**

**End Sub**

When a statement such as

**DisplaySum(3, 4)**

is executed, the number 3 is assigned to the parameter *num1,* the number 4 is assigned to the parameter *num2,* and the three statements inside the Sub procedure block are carried out. As a result, the number 7 is displayed in the list box. We say that the numbers 3 and 4 are **passed** to the Sub procedure.

# Variables and Expressions as Arguments

Just as with function calls, the arguments in Sub procedure calls can be literals variables, or expressions.

# Example Add Numbers

 The following program calls an expanded version of the Sub procedure DisplaySum three times. The first time the arguments are literals, the second time the arguments are variables, and the third time the arguments are expressions. In the second call of DisplaySum, the values of the variables are passed to the Sub procedure. In the third call, the expressions are evaluated and the resulting numbers are passed to the Sub procedure.

**Private Sub btnAddNumbers\_Click(...) Handles btnAddNumbers.Click**

**DisplaySum(1, 2)**

**Dim x As Double = 3**

**Dim y As Double = 4**

**DisplaySum(x, y)**

**DisplaySum(2 \* x, y + 5)**

**End Sub**

**Sub DisplaySum(num1 As Double, num2 As Double)**

**Dim z As Double**

**z = num1 + num2**

**lstOutput.Items.Add("The sum of " & num1 & " and " & num2 &**

**" is " & z & ".")**

**End Sub**

Results is a list box that contains these three value

The sum of 1 and 2 is 3

The sum of 3 and 4 is 7

The sum of 5 and 9 = 15