Methods and Functions

Methods

In programming, a method can be defined as a block of code that contributes to a whole program – a block that performs a specific task or function. Programs can be written in a single (and inflexible) method; for example, most of your applications so far have been written in a single method: Shared Sub Main(). However, using a single method will put limitations on any application, as well as making it difficult to debug, maintain and upgrade/adapt in the future. Breaking an application's functionality down into separate methods and applicable modules/objects will help eliminate some of these problems. In theory, it should make applications more robust, efficient and maintainable. An example of separating code in separate methods is shown below:

```
Shared NumberX As Integer
    Shared Sub Main()
            NumberX = 10
            JoeBloggs()
    End Sub
    Shared Sub JoeBloggs()
        NumberX = NumberX + 10
        WriteLine(NumberX)
    End Sub
```

In the previous example, a global variable (NumberX) was used to pass information between two methods. This is not an efficient technique of passing information between methods and can cause problems later on in the development of applications (especially as they become more complex). Instead, 'arguments' and 'parameters' can be used to pass information efficiently between methods. A **parameter** is what the method expects. An **argument** is what is passed to the accepting method. An example of this is shown below; this time the method JoeBloggs() expects an Integer value to be passed (the method will not run unless it is given the required parameters). When JoeBloggs() is called in Sub Main(), the value 10 is passed as an argument into the method JoeBloggs() where it is now stored in the variable NumberX ready for use by the method.

```
Shared Sub Main()
        JoeBloggs(10)
    End Sub
    Shared Sub JoeBloggs(NumberX As Integer)
        NumberX = NumberX + 10
        WriteLine(NumberX)
    End Sub
```

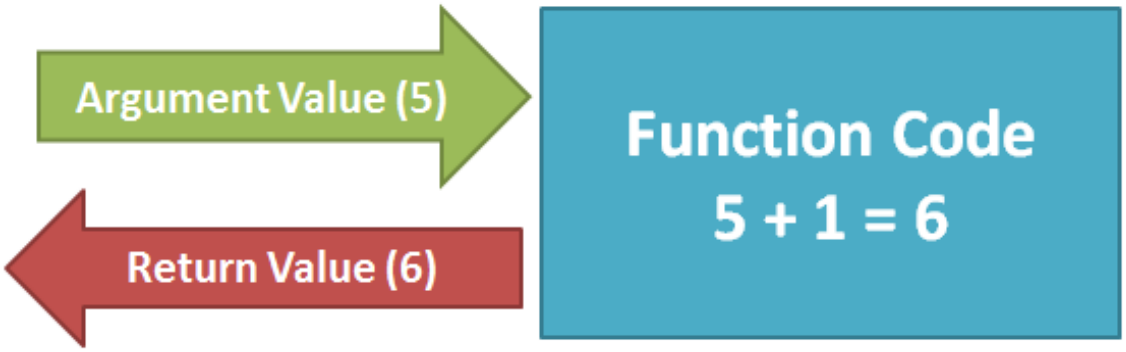
Please note that arguments are passed in order from left to right and do not rely on any naming conventions. Therefore, a second parameter would take the second given argument. A comma is used to separate multiple arguments and parameters.

Test Your Skills

- ✓ *Write a short application that consists of two methods. The application will need to:*
  1. *Read two values from the user in Sub Main()*
  2. *The application must then pass the two values (from the user) to a second method*
  3. *The second method must then add those two values together and display the answer to the user*

Functions

Functions are like ATMs: input a card, a pin number and an amount (arguments) and receive a cash value on return. An example of a function is shown below;



In VB.NET, functions have a different purpose to methods. When using a method, arguments can be passed to it and the method may use the arguments in its own execution. This works fine, but on occasion the programmer may require the method to return a value and this cannot be achieved when using Subs (methods). However, in VB.NET functions can be used to get a return value – functions are the same as methods in every principle, except that they have to return a value before finishing. Please be aware that a function can only return one value; however, a single object or array can be used to return multiple values.

There are two types of functions available: Utility Functions and Object Functions. An example of an Object Function and a Utility Function are shown below:

- A utility function (sometimes known as static) can be addressed directly from the code. This function converts a String variable's content to upper case and returns it:

```
Variable = Variable.ToUpper
```

- An object utility is object-orientated and thus has to be initiated first before it can be used. This function generates a random number between 1 and 50 and returns it:

```
Dim RandomClass As New Random()
    Dim answer as Integer
    answer = RandomClass.Next(1, 51)
```

A user does not need to use pre-built functions; they can also define their own. It is considered good practice to define code into its own function if it is an operation that is going to be performed multiple times by the same (or another) program. Below is an example of a self-defined function:

```
Shared Sub Main()
            Dim NumberX As Decimal
              NumberX = Test(10)
              WriteLine(NumberX)
    End Sub
Shared Function Test(X As Decimal) As Decimal
            Dim Answer As Decimal
              Answer = X + 5
            Return Answer
    End Function
```

There are some key differences to notice when declaring a method and declaring and a function. First is that the keyword '**Function**' is used instead of the keyword '**Sub**'. Another difference is that it is important to state the data type of the return value; this is achieved by adding the data type at the end of the function declaration, i.e. 'Shared Function Test(X As Decimal) **As Decimal**'. The last difference is that a function must have a return value; this is achieved by using the keyword '**Return**'. In this case the variable **Answer** is returned. If the code above was to run, what would be the value of **Answer**?

Test Your Skills

- ✓ *Write a short application that consists of one method and one function. The first method (Sub Main()) should ask the user for two values. The application should then pass those two values to a separate function which adds them together and returns the answer to Sub Main(). The method Sub Main() should then write the answer to the screen.*