VB.NET: Functions and Parameter Passing

Built-in Functions

Example:

```
Module Module1
    Sub Main()
        Dim inputString As String
        Dim number As Integer
        Dim result As String

        ' Prompt user for input
        Console.WriteLine("Enter a number:")
        inputString = Console.ReadLine()

        ' Check if input is numeric using IsNumeric()
        If IsNumeric(inputString) Then
            ' Convert the string to an integer using CInt()
            number = CInt(inputString)

            Randomize() ' Seed random number generator

            ' Use Rnd() to generate a random number between 0 and 100
            Dim randomValue As Integer = CInt(Rnd() * 100)

            ' Convert integer result back to string using CStr()
            result = CStr(number + randomValue)
            Console.WriteLine("Your number plus a random value is: " & result)
        Else
            Console.WriteLine("Invalid input, please enter a numeric value.")
        End If

        Console.ReadKey()
    End Sub
End Module
```

Explanation: This example demonstrates the use of built-in functions:

- `IsNumeric()` checks if the input is a number.
- `CInt()` converts the string input to an integer.
- `Rnd()` Rnd() generates a random floating-point number between 0 and 1, and you can scale it to a desired range.
- `CStr()` converts the integer result back to a string.

Scope of Variables

Example:

```
Module Module1

    ' Module-level scope variable
    Dim moduleLevelVar As Integer = 10

    Sub Main()
        ' Procedure-level scope variable
        Dim procedureLevelVar As Integer = 20
        Console.WriteLine("Inside Main: Module-Level = " & moduleLevelVar & ", Procedure-Level = "
& procedureLevelVar)

        ' Call function to demonstrate block-level scope and modify the procedureLevelVar
        procedureLevelVar = DemonstrateBlockScope(procedureLevelVar)

        ' Display the modified procedure-level variable after calling the function
        Console.WriteLine("After calling function: Procedure-Level = " & procedureLevelVar)

        Console.ReadKey()
    End Sub

    ' Function to demonstrate block-level scope
    Function DemonstrateBlockScope(ByVal procedureLevelVar As Integer) As Integer
        ' Block-level scope variable
        For i As Integer = 1 To 3
            Dim blockLevelVar As Integer = i * 10
            Console.WriteLine("Inside loop: Block-Level Var = " & blockLevelVar)
        Next

        ' Modify the procedure-level variable inside the function
        procedureLevelVar += 10
        Console.WriteLine("Inside function: Modified Procedure-Level Var = " & procedureLevelVar)

        ' Module-level variable accessible here
        Console.WriteLine("Inside function: Module-Level = " & moduleLevelVar)

        Return procedureLevelVar ' Return the modified procedure-level variable
    End Function

End Module
```

Explanation:

- **Module-level scope**: `moduleLevelVar` can be accessed anywhere in the module, including within the function `DemonstrateBlockScope`.
- **Procedure-level scope**: `procedureLevelVar` is only accessible within the `Main` subroutine but can be passed and modified by the function `DemonstrateBlockScope`. The modified value is returned and reassigned to `procedureLevelVar` in `Main`.
- **Block-level scope**: `blockLevelVar` is only visible within the `For` loop inside `DemonstrateBlockScope` and cannot be accessed outside this block.

Self-Defined Functions

Example:

```
Module Module1
    Sub Main()
        Dim num1 As Integer = 5
        Dim num2 As Integer = 7

        ' Call self-defined functions
        Console.WriteLine("Square of " & num1 & " is: " & Square(num1))
        Console.WriteLine("Sum of " & num1 & " and " & num2 & " is: " &
AddTwoNumbers(num1, num2))
        Console.WriteLine("Factorial of " & num1 & " is: " & Factorial(num1))

        Console.ReadKey()
    End Sub

    ' Function to calculate the square of a number
    Function Square(ByVal num As Integer) As Integer
        Return num * num
    End Function

    ' Function to add two numbers
    Function AddTwoNumbers(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
        Return num1 + num2
    End Function

    ' Function to calculate the factorial of a number
    Function Factorial(ByVal num As Integer) As Integer
        If num = 0 Then
            Return 1
        Else
            Return num * Factorial(num - 1)
        End If
    End Function
End Module
```

Explanation:

- Each function uses ByVal to pass parameters by value, meaning the original variable is not modified outside the function.
- `Square()`: A function that calculates the square of a number.
- `AddTwoNumbers()`: A function that returns the sum of two numbers.
- `Factorial()`: A recursive function to calculate the factorial of a number.

Conclusion

In this section, we have demonstrated three critical aspects of programming in VB.NET: the use of built-in functions, the understanding of variable scope, and the creation of self-defined functions.

- Built-in Functions: Functions like `IsNumeric()`, `CInt()`, `Rnd()`, and `CStr()` are powerful tools that simplify common operations such as data type conversions and random number generation. These functions save time and improve accuracy in common tasks.

- Variable Scope: Understanding variable scope is crucial for writing efficient and bug-free code. Variables in VB.NET can have module-level, procedure-level, or block-level scope, each determining how long and where the variable can be accessed. Mastering scope helps prevent unintended variable modifications and conflicts, improving code organization.

- Self-Defined Functions: Writing self-defined functions allows developers to encapsulate logic into reusable units, improving the modularity and readability of the code. In this example, we explored a function to calculate a square, another to add numbers, and a recursive function to find the factorial of a number. Self-defined functions are fundamental to structuring programs that are maintainable and scalable.

By understanding these concepts, you can build more robust, maintainable applications in VB.NET, enabling you to write clear, reusable code that adheres to best practices in software development.