VB.NET: Subroutines and Variable Scope

Example:

```
Module Module1
    ' Constant variable declaration
    Const DISCOUNT_RATE As Decimal = 0.1 ' Discount rate is constant and does not
change

    Sub Main()
        ' Call self-defined subroutine to calculate the discounted price
        Dim originalPrice As Decimal = 200
        CalculateDiscount(originalPrice)

        Console.ReadKey()
    End Sub

    ' Self-defined subroutine to calculate discount
    Sub CalculateDiscount(ByVal price As Decimal)
        ' Calculate the discount amount and the discounted price
        Dim discountAmount As Decimal = price * DISCOUNT_RATE
        Dim discountedPrice As Decimal = price - discountAmount

        ' Output the results
        Console.WriteLine("Original Price: " & price)
        Console.WriteLine("Discounted Price: " & discountedPrice)
    End Sub
End Module
```

In this example:

- The subroutine `CalculateDiscount()` calculates the discounted price based on an original price passed as a parameter. It outputs both the original price and the discounted price.
- The subroutine takes a parameter `price` using the `ByVal` keyword, meaning the value is passed by value (a copy of the original is passed, and changes within the subroutine do not affect the original variable).
- The constant `DISCOUNT_RATE` is used inside the subroutine to calculate the discount amount. Constants help keep fixed values consistent across the code.

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.

Conclusion

In this section, we focused on how different levels of variable scope - **module-level, procedure-level**, and **block-level** - work within VB.NET programs, particularly in the context of subroutines.

- Module-level variables are accessible throughout the entire module, allowing their values to be shared across multiple subroutines and functions. This makes them useful for storing data that needs to be persistent across different parts of the program.
- Procedure-level variables are confined to the specific subroutine where they are declared. This encapsulation ensures that data within a subroutine remains isolated and cannot be unintentionally modified by other parts of the program.
- Block-level variables are limited to the scope of specific blocks, such as loops or conditionals. These variables are recreated and discarded with each iteration or block execution, making them useful for temporary calculations or specific tasks within small segments of code.

Additionally, the use of constants in subroutines ensures that certain values remain unchanged throughout the program. This approach provides consistency and clarity, especially for values that should not vary during execution.

By understanding and properly utilizing these variable scopes, you can write cleaner, more organized, and efficient VB.NET code. This knowledge helps prevent unexpected behavior caused by variable conflicts or unintended data modifications, leading to more reliable and maintainable applications.