جامعة المستقبل
AL MUSTAQBAL UNIVERSITY
كلية الهندسة والتقنيات الهندسية

Communication Technical Engineering Department

2nd Stage

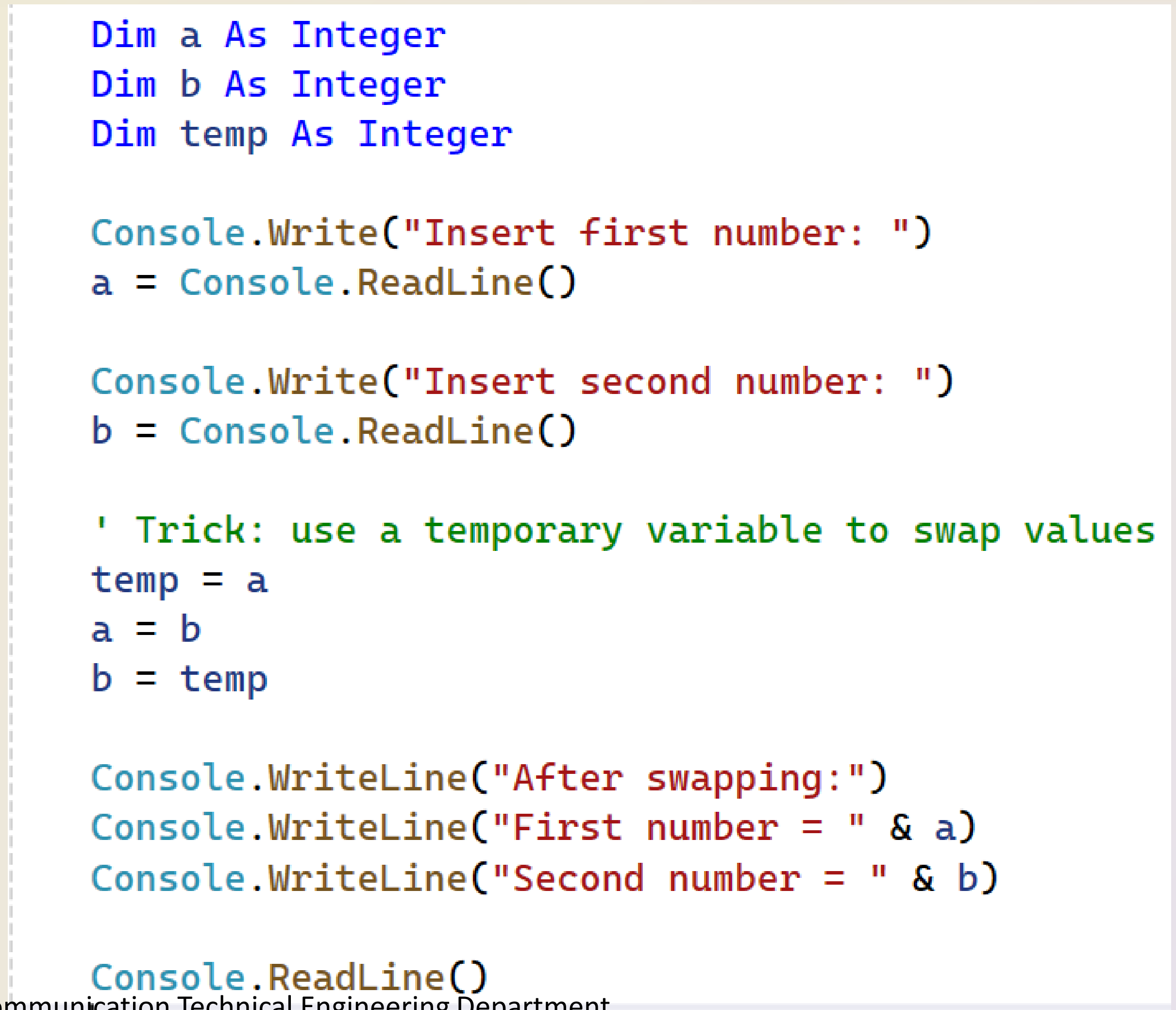
Visual Basic - UOMU0207036

Lecture 3 – Loops

Dr. Mohammed Fadhil
PhD in Computer Networks
Email: mohammed.fadhil1@uomus.edu.iq

Recall

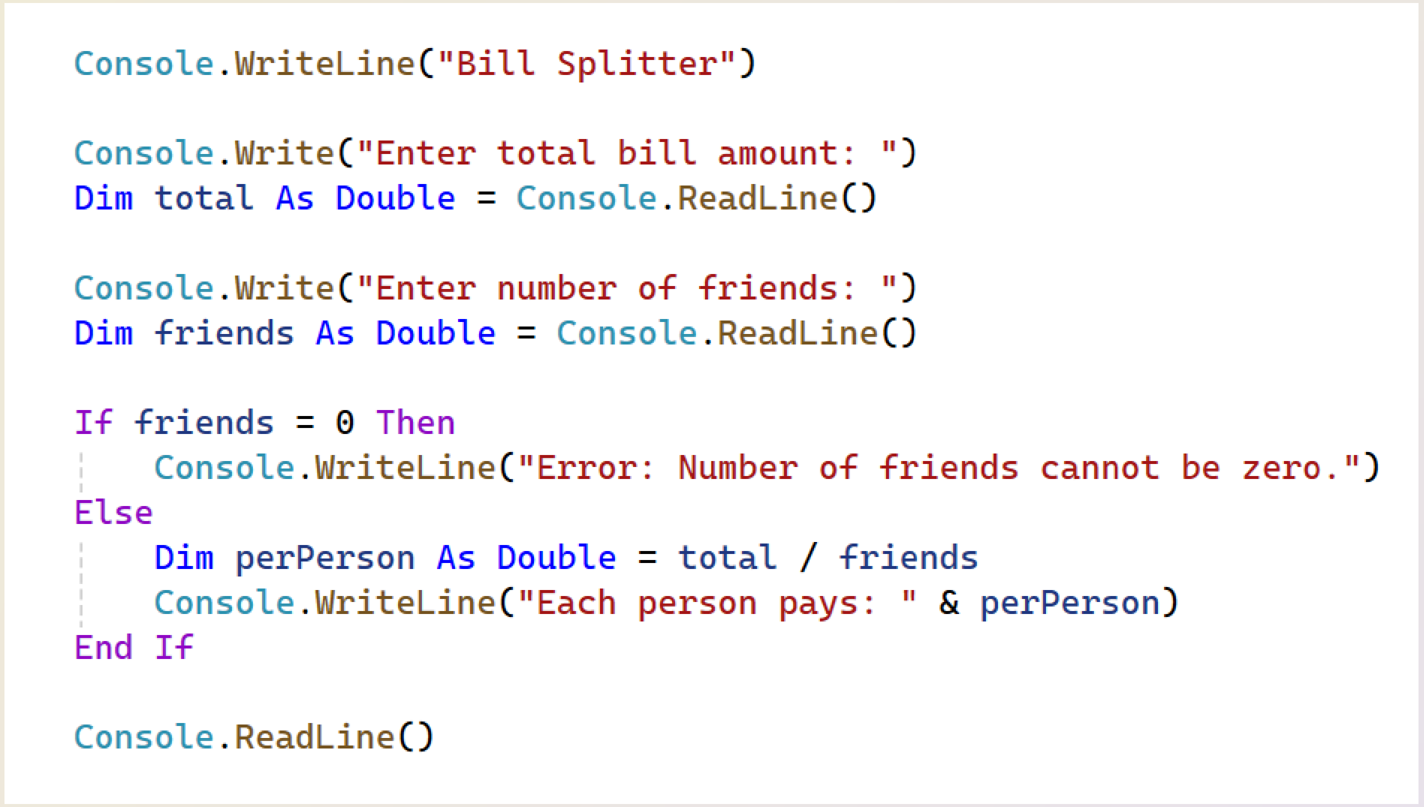
- Write a Visual Basic program that asks the user To
  - enter two numbers,
  - then swaps their values using a temporary variable.



```
Dim a As Integer
Dim b As Integer
Dim temp As Integer

Console.Write("Insert first number: ")
a = Console.ReadLine()

Console.Write("Insert second number: ")
b = Console.ReadLine()

' Trick: use a temporary variable to swap values
temp = a
a = b
b = temp

Console.WriteLine("After swapping:")
Console.WriteLine("First number = " & a)
Console.WriteLine("Second number = " & b)

Console.ReadLine()
```

Recall

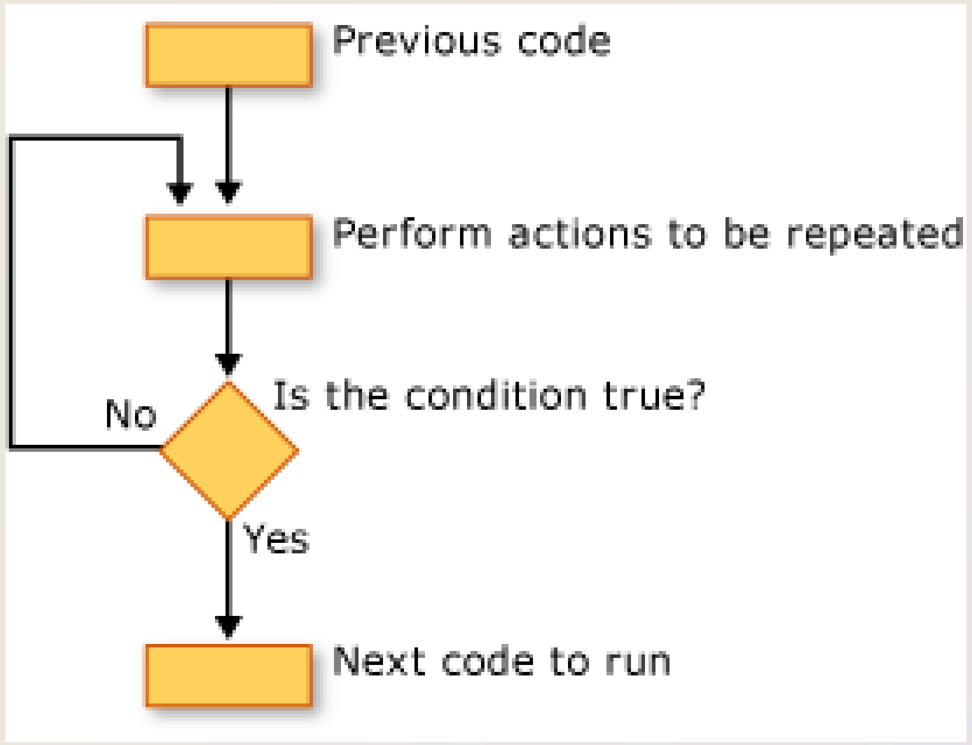
- Write a Visual Basic program that asks the user to enter the **total bill amount** and the **number of friends**. The program should:
  - Display an error message if the number of friends is zero.
  - Otherwise, calculate and display how much each person should pay by dividing the total bill by the number of friends.
  - Include a suitable message before displaying the result.



```
Console.WriteLine("Bill Splitter")

Console.Write("Enter total bill amount: ")
Dim total As Double = Console.ReadLine()

Console.Write("Enter number of friends: ")
Dim friends As Double = Console.ReadLine()

If friends = 0 Then
    Console.WriteLine("Error: Number of friends cannot be zero.")
Else
    Dim perPerson As Double = total / friends
    Console.WriteLine("Each person pays: " & perPerson)
End If

Console.ReadLine()
```

Loops

- There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Loop Type

Loop Type	Description
Do Loop	It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.
For...Next	It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
For Each...Next	It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
While... End While	It executes a series of statements as long as a given condition is True.
With... End With	It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure.
Nested loops	You can use one or more loops inside any another While, For or Do loop.

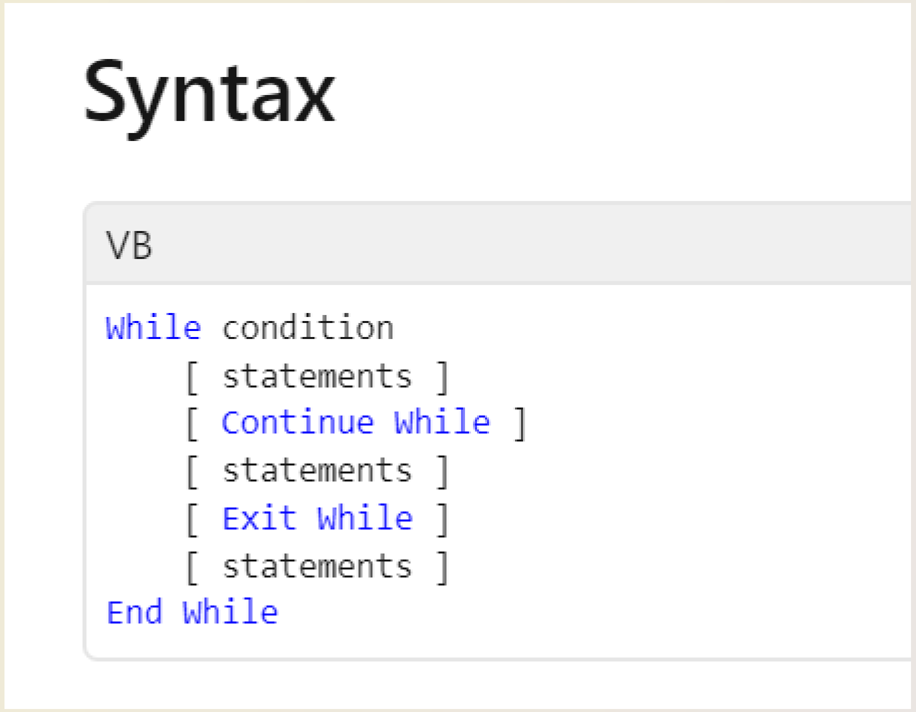
Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Control Statement	Description
Exit statement	Terminates the **loop** or **select case** statement and transfers execution to the statement immediately following the loop or select case.
Continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
GoTo statement	Transfers control to the labeled statement. Though it is not advised to use GoTo statement in your program.

While Loops

- The *While...End While* construction runs a set of statements as long as the condition specified in the *While* statement is *True*.



Syntax

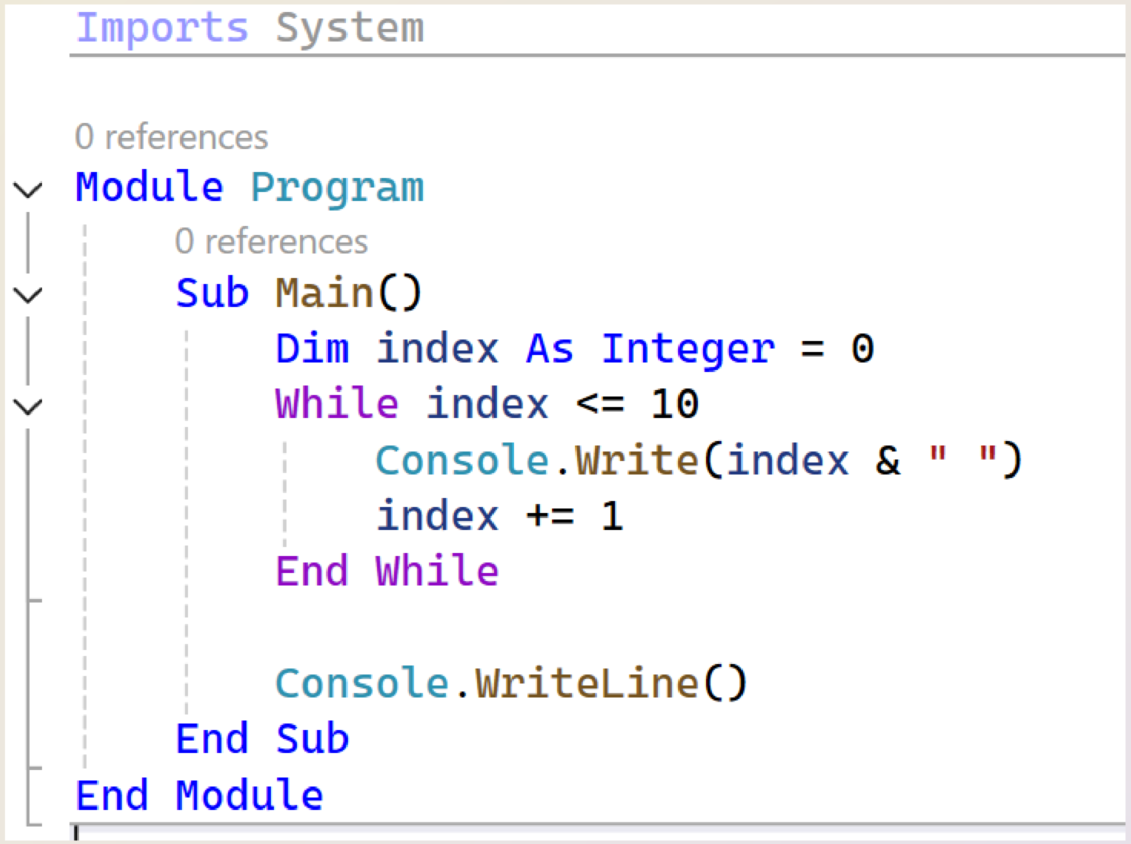
VB

```
While condition
    [ statements ]
    [ Continue While ]
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

Example 1

- In the following example, the statements in the loop continue to run until the *index* variable is greater than *10*.

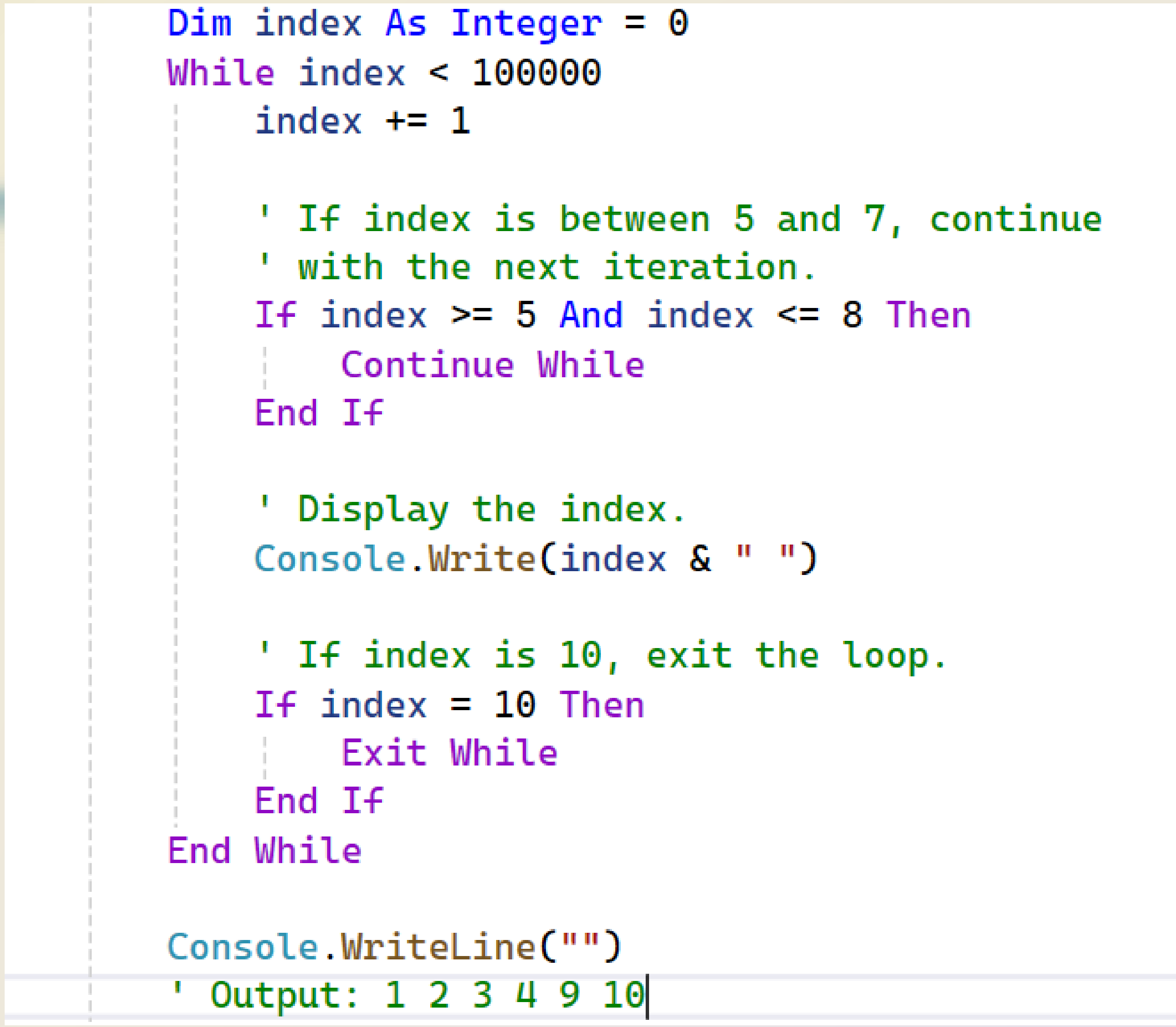
- Output: 0 1 2 3 4 5 6 7 8 9 10



```
Imports System

0 references
Module Program
    0 references
    Sub Main()
        Dim index As Integer = 0
        While index <= 10
            Console.Write(index & " ")
            index += 1
        End While

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

Example 2

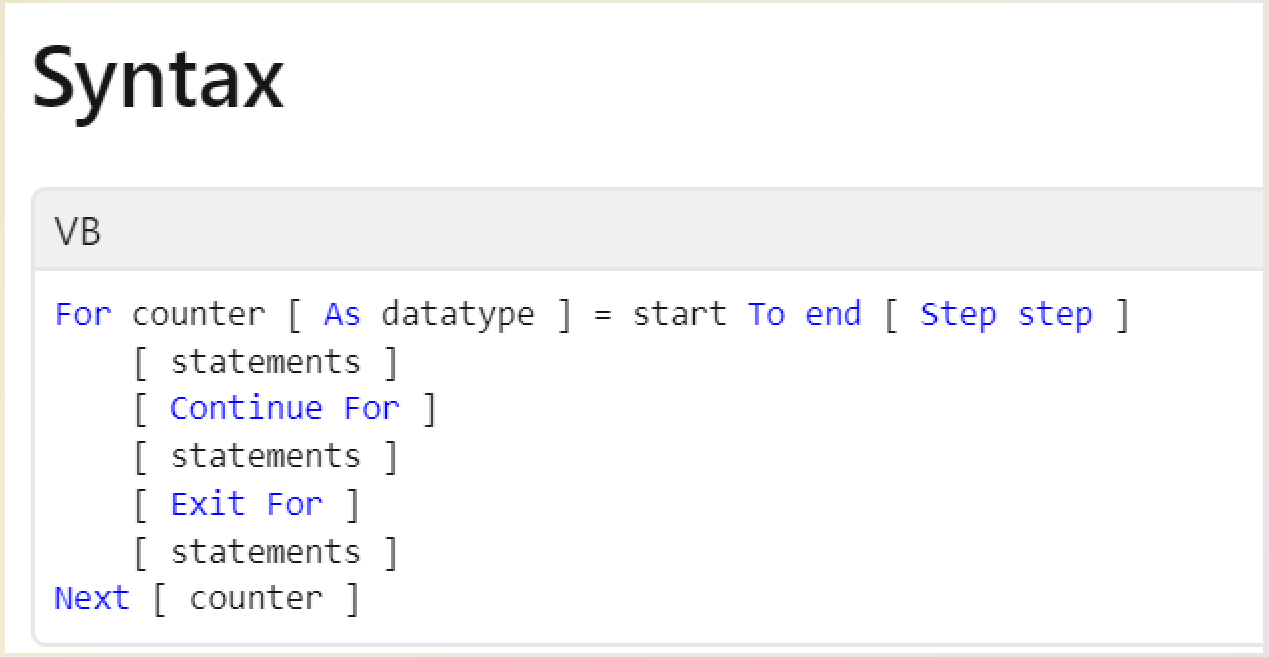
- The following example illustrates the use of the Continue While and Exit While statements.
- **Explanation**:
  - Continue While → skips the rest of the loop and jumps to the next iteration.
  - Exit While → completely ends the loop immediately.
  - The condition index < 100000 ensures it could loop a long time, but **Exit While** stops it early at 10.



```
Dim index As Integer = 0
While index < 100000
    index += 1

    ' If index is between 5 and 7, continue
    ' with the next iteration.
    If index >= 5 And index <= 8 Then
        Continue While
    End If

    ' Display the index.
    Console.Write(index & " ")

    ' If index is 10, exit the loop.
    If index = 10 Then
        Exit While
    End If
End While

Console.WriteLine("")
' Output: 1 2 3 4 9 10
```

For Loops (For...Next Statement)

- Repeats a group of statements while the loop counter approaches its final value.



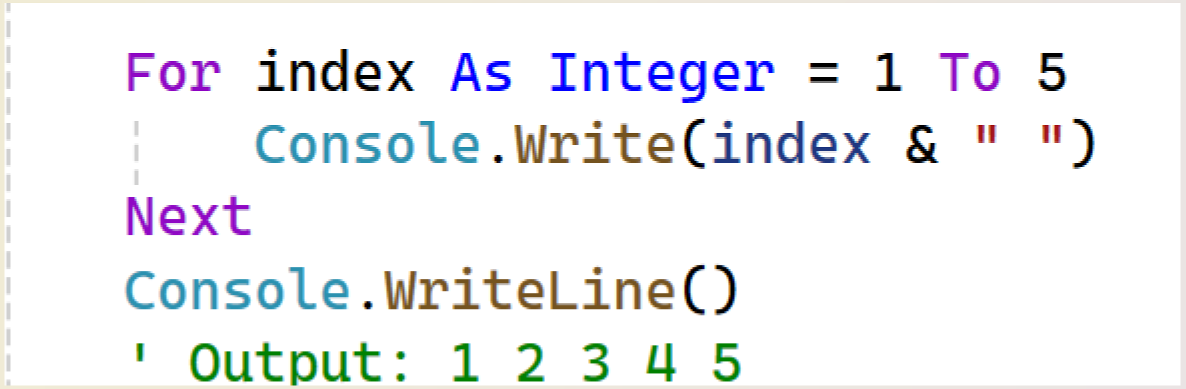
Syntax

VB

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

Simple Examples

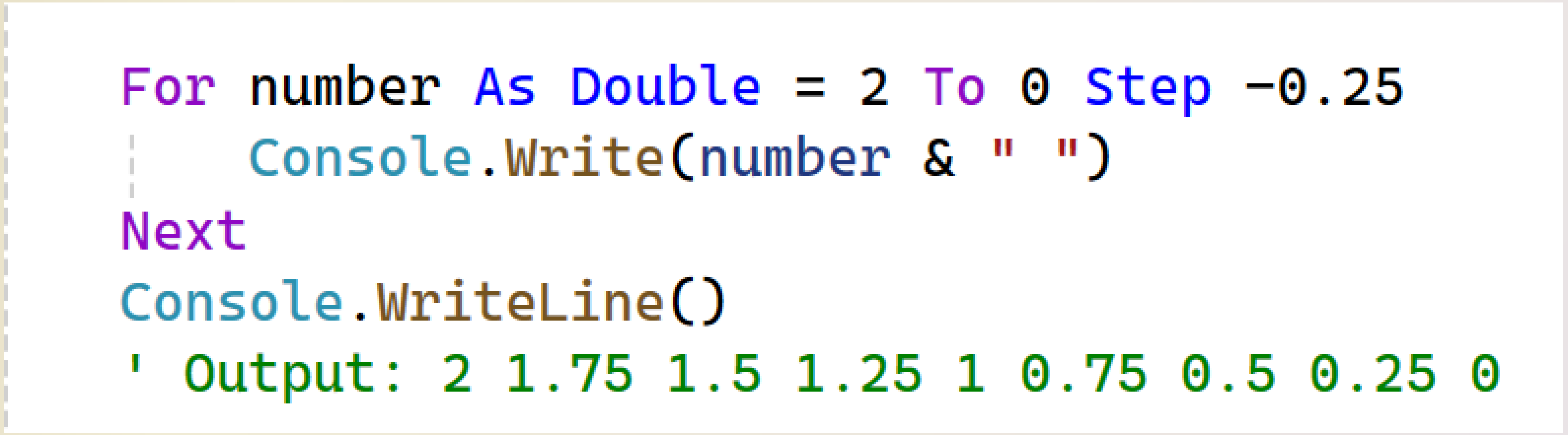
- You use a **For...Next** structure when you want to repeat a set of statements a set number of times.
- In the following example, the index variable starts with a value of **1** and is incremented with each iteration of the loop, ending after the value of index reaches **5**.



```
For index As Integer = 1 To 5
    Console.Write(index & " ")
Next
Console.WriteLine()
' Output: 1 2 3 4 5
```

Simple Examples

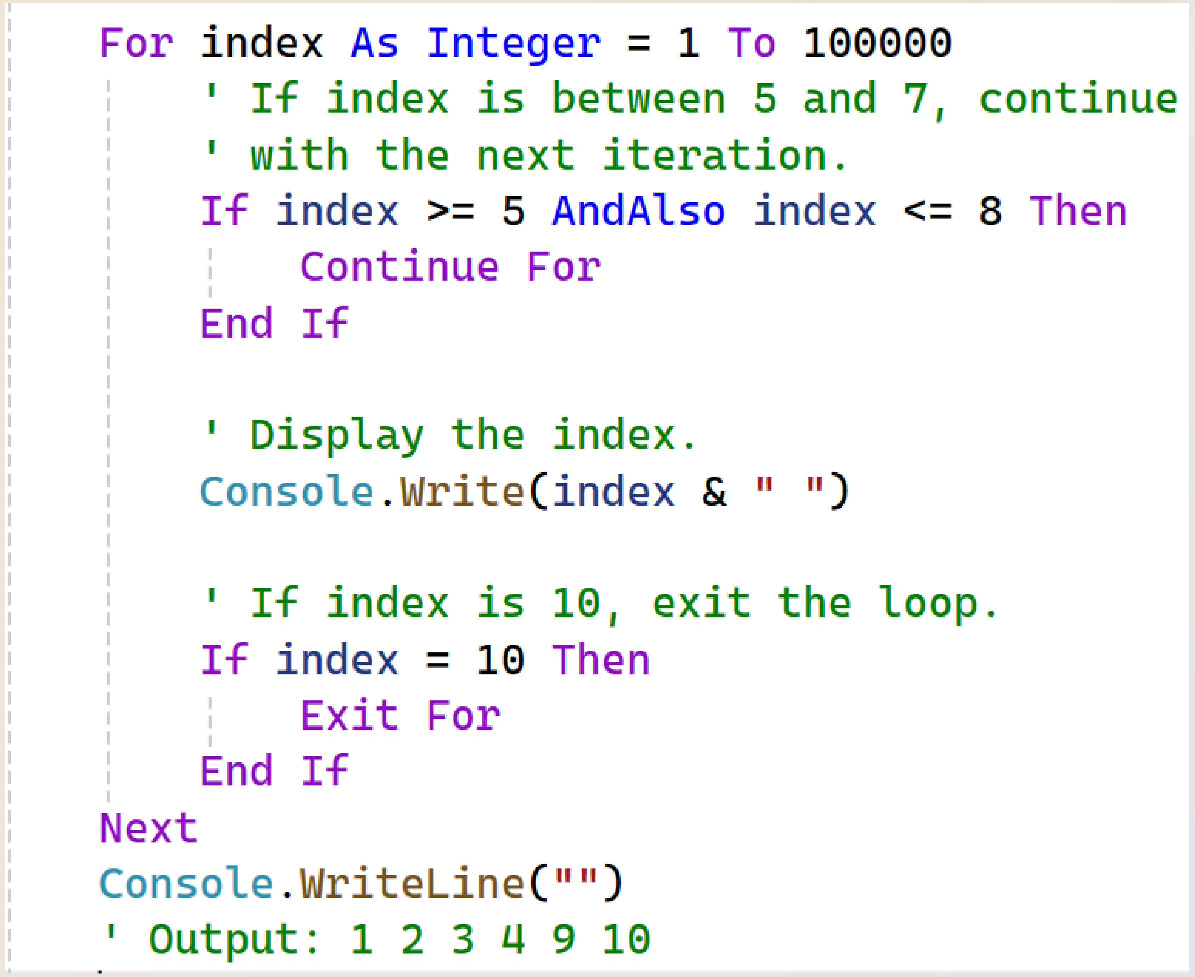
- In the following example, the number variable starts at 2 and is reduced by 0.25 on each iteration of the loop, ending after the value of number reaches 0. The Step argument of -.25 reduces the value by 0.25 on each iteration of the loop.



```
For number As Double = 2 To 0 Step -0.25
    Console.Write(number & " ")
Next
Console.WriteLine()
' Output: 2 1.75 1.5 1.25 1 0.75 0.5 0.25 0
```

Exit For and Continue For

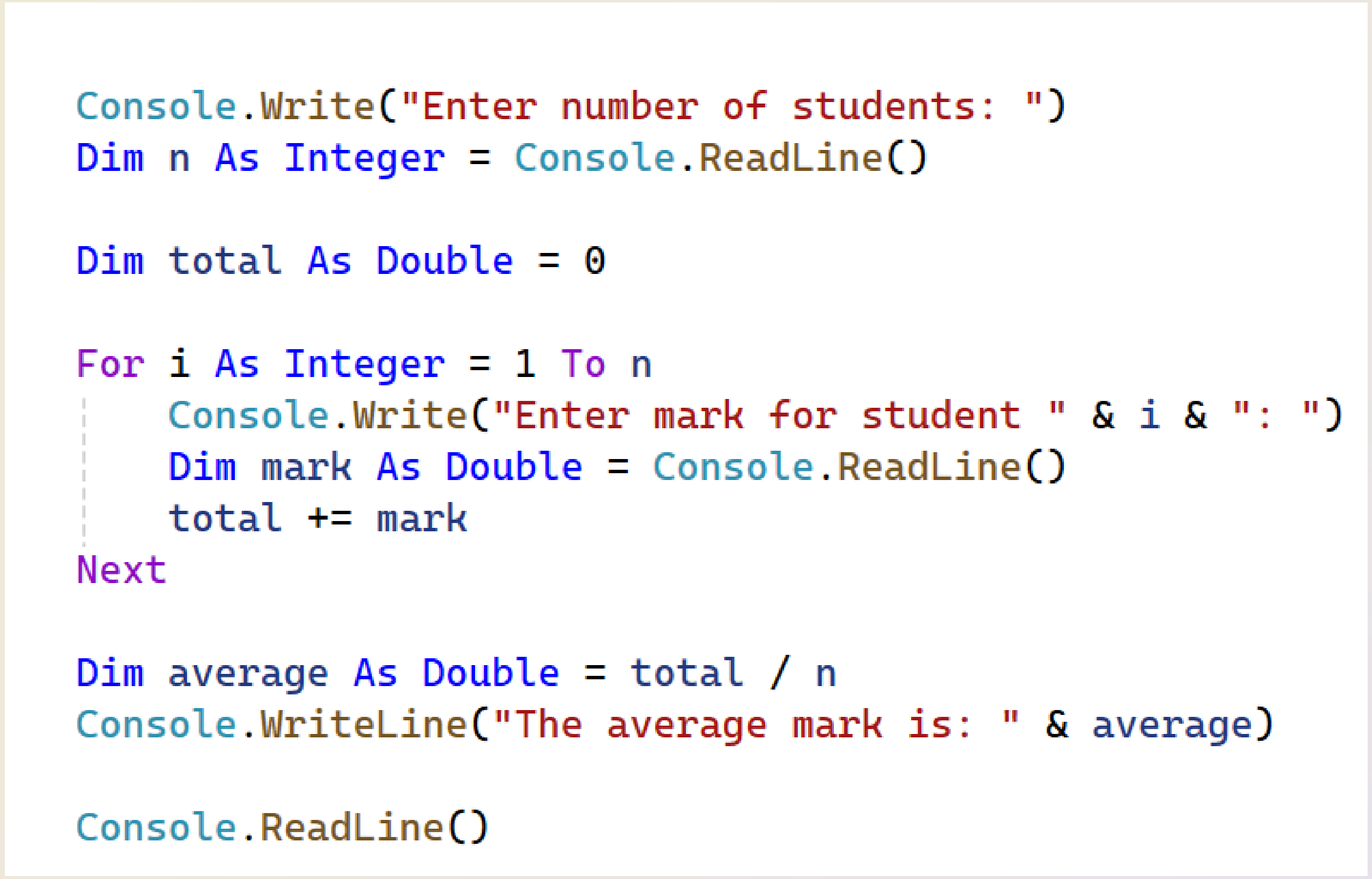
- The **Exit For statement** immediately exits the *For…Next loop* and transfers control to the statement that follows the Next statement.
- The **Continue For** statement transfers control immediately to the next iteration of the loop.
- The following example illustrates the use of the Continue For and Exit For statements.



```
For index As Integer = 1 To 100000
    ' If index is between 5 and 7, continue
    ' with the next iteration.
    If index >= 5 AndAlso index <= 8 Then
        Continue For
    End If

    ' Display the index.
    Console.Write(index & " ")

    ' If index is 10, exit the loop.
    If index = 10 Then
        Exit For
    End If
Next
Console.WriteLine("")
' Output: 1 2 3 4 9 10
```

Exit For and Continue For

- Write a **Visual Basic program** that:
  - Asks the user to enter the **number of students**.
  - Uses a **For loop** to input each student's **mark**.
  - Calculates the **total** and then finds the **average mark**.
  - Displays the **average mark** at the end.



```
Console.Write("Enter number of students: ")
Dim n As Integer = Console.ReadLine()

Dim total As Double = 0

For i As Integer = 1 To n
    Console.Write("Enter mark for student " & i & ": ")
    Dim mark As Double = Console.ReadLine()
    total += mark
Next

Dim average As Double = total / n
Console.WriteLine("The average mark is: " & average)

Console.ReadLine()
```

Thank You ☺