11.3. Structured programming
- use a procedure
- explain where in the construction of an algorithm it would be appropriate to use a procedure
  - a procedure may have none, one or more parameters
  - a parameter can be passed by reference or by value
- show understanding of passing parameters by reference
- show understanding of passing parameters by value
  - a call is made to the procedure using CALL <identifier> ()
- use a function
- explain where in the construction of an algorithm it is appropriate to use a function
- use the terminology associated with procedures and functions: procedure/function header, procedure/ function interface, parameter, argument, return value
  - given pseudocode will use the following structure for function definitions:
  - a function is used in an expression, for example
- write programs containing several components and showing good use of resources

11.3. Structured programming

Algorithm design involves developing **step-by-step** instructions to solve a problem and subroutines are to **modularize the solution**.

PROCEDURE or Subroutine:

A **PROCEDURE is** a self-contained section of program code which performs a specific task and is referenced by a name. Procedures can be used repeatedly throughout a program and can be called when needed in program by **CALL** keyword

A PROCEDURE/ subroutine resemble a standard program. it will contain its own local variables, data types, labels, and constant declarations.

FUNCTION: is a self contained program code which **performs a specific task** and is referenced by a name. **FUNCTION always returns a value.** FUNCTION is a sequence of steps that is given an identifier and returns a single value; function call is part of an expression

Parameter of Procedure/Function:

A **variable applied to a procedure or function** that allows one to **pass in a value** for the **procedure/function** to **use.**

By Value: a method of passing a parameter to a procedure in which the value of the variable cannot be changed by the procedure/function. What it means is that you are **passing** a **copy** of a variable to your Subroutine.

By Reference: a method of passing a parameter to a procedure in which the value of the variable can be changed by the procedure/function. Any change you make to the variable within your **subroutine will effect the variable itself.**

Header (Procedure or Function): the first statement in the definition of a procedure or function, which contains its name, any parameters passed to it, and, for a function, the type of the return value.

Argument – the value passed to a procedure or function is called an argument.

PSEUDOCODE of PROCEDURE

```
PROCEDURE timestable(ByREF number As INTEGER) //This is a Procedure
   FOR count = 1 To 20
       OUTPUT(number & " X " & count & " = " & count * number)
   NEXT
END PROCEDURE

BEGIN
      OUTPUT("PLEASE Input number for TimesTable") //Prompt for user
      CALL timestable //CALL to procedure in the main Program
END
```

PYTHON CODE

```
def TimesTable(number):
     for count in range(1,11):
          print(number, " X ", count, " = ", number*count)

num = int(input("Please input a number for its TimesTable"))
TimesTable(num)
```

```
Please input a number for its TimesTable88
88  X  1  =  88
88  X  2  =  176
88  X  3  =  264
88  X  4  =  352
88  X  5  =  440
88  X  6  =  528
88  X  7  =  616
88  X  8  =  704
88  X  9  =  792
88  X  10  =  880
```

VB CODE Of PROCEDURE

```
Module module1
    ' This is a Procedure
    Sub timestable(ByRef number As Integer)
        For count = 1 To 20
            Console.WriteLine(number & " X " & count & " = " & count * number)
        Next
    End Sub
    Sub main()
        Console.WriteLine("PLEASE Input a number for TimesTable") 'asking for numb
        timestable(Console.ReadLine) 'CALL to procedure to execute it in the main
        Console.ReadKey()
    End Sub

End Module
```

```
PLEASE Input a number to see its Table
12
12 X 1 = 12
12 X 2 = 24
12 X 3 = 36
12 X 4 = 48
12 X 5 = 60
12 X 6 = 72
12 X 7 = 84
12 X 8 = 96
12 X 9 = 108
12 X 10 = 120
12 X 11 = 132
12 X 12 = 144
12 X 13 = 156
12 X 14 = 168
12 X 15 = 180
12 X 16 = 192
12 X 17 = 204
12 X 18 = 216
12 X 19 = 228
12 X 20 = 240
```

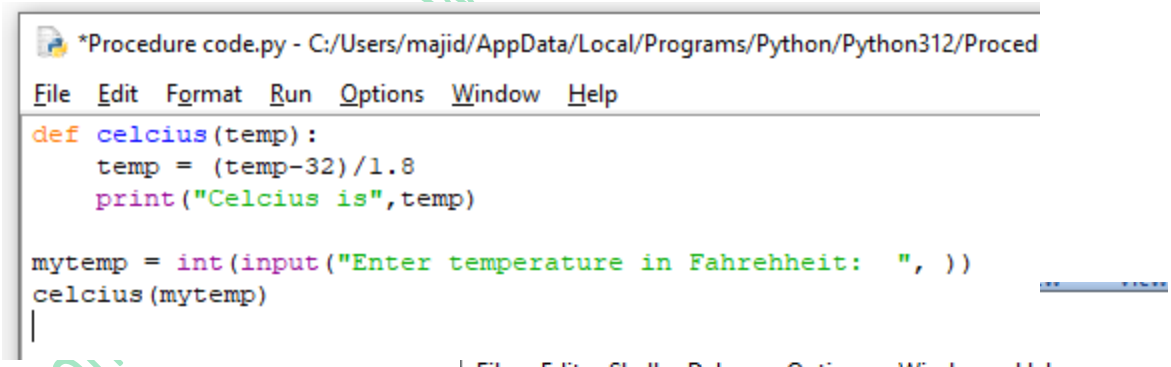
PROCEDURE Fahrenheit to Celsius (By Value parameter)

```
PROCEDURE Celsius (ByVAL temp : REAL)
     temp = (temp -32)/1.8
     OUTPUT("Celsius = " , temp)
END PROCEDURE
BEGIN
     DECLARE MyTemp : REAL
     OUTPUT("Input temperature in Fahrenheit ")
     INPUT MyTemp
     CALL Celsius(MyTemp) //CALL to procedure to execute
END
```

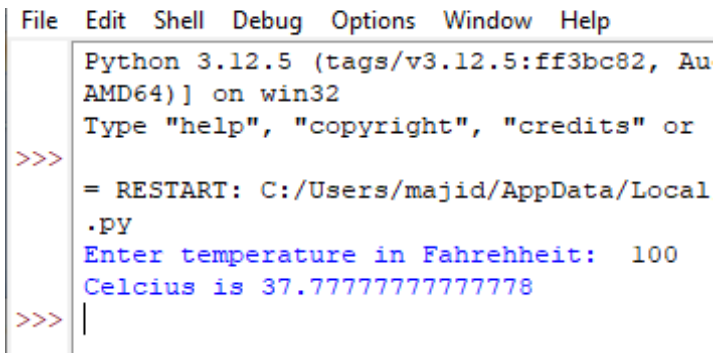
PYTHON Code

```
def celcius(temp):
    temp = (temp-32)/1.8
    print("Celcius is",temp)

mytemp = int(input("Enter temperature in Fahrehheit:  ", ))
celcius(mytemp)
```

(Python has only (ByValue Parameter). Python doesn't have BY REFERENCE Parameter



OUTPUT of Python code:



Visual Basic Code (By VALUE Parameter)

VB Code of Fahrenheit to Celsius

```
Sub Celsius(ByVal temp As Double)
  temp = (temp - 32) / 1.8
  Console.WriteLine("Celsius = " & temp)
End Sub

Sub Main()
Dim MyTemp As Double
Console.WriteLine("please input temperature in Farenheit")
 MyTemp = Console.ReadLine()
 Celsius(MyTemp) ' Called Procedure Celsius
 Console.WriteLine(" Orignal value was : " & MyTemp)
End Sub
```

```
C:\Users\Lenovo\source\repos\ProcedureTEMPERA
Farenheit to Celsius Program
please input temperature in Farenheit
40
Celsius = 4.444444444444445
 Orignal value was : 40
```

NOTE in above code if ByVal PARAMETER is used, original value of MyTemp remains same

Now we will use the same program with ByRef parameter and see the changes in program

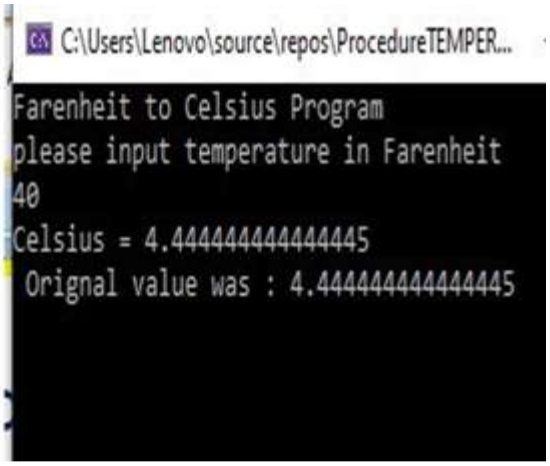
PROCEDURE Fahrenheit to Celsius (ByReference)

```
PROCEDURE Celsius (ByRef temp : REAL)
     temp = (temp -32)/1.8
     OUTPUT("Celsius = " , temp)
END PROCEDURE
BEGIN
     DECLARE MyTemp : REAL
     OUTPUT("Input temperature in Fahrenheit ")
     INPUT MyTemp
     CALL Celsius(MyTemp) //CALL to procedure to execute
END
```

Note that by Reference changed the value of original variable too when executed the PROCEDURE

VB Code of Fahrenheit to Celsius

```
Sub Celsius(ByRef temp As Double)
  temp = (temp - 32) / 1.8
  Console.WriteLine("Celsius = " & temp)
End Sub

Sub Main()
Dim MyTemp As Double
Console.WriteLine("please input temperature in Farenheit")
 MyTemp = Console.ReadLine()
 Celsius(MyTemp) ' Called Procedure Celsius
 Console.WriteLine(" Orignal value was : " & MyTemp)
End Sub
```

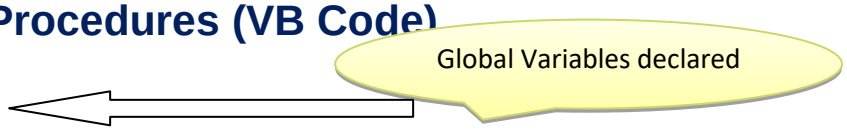


NOTE in above code if ByRef PARAMETER is used, original value of MyTemp **changed**

Using Global variable (Pseudocode)

```
DECLARE num1, num2, answer As Integer
      PROCEDURE input_sub()
            OUTPUT("Enter number 1")
            INPUT num1 = Console.ReadLine
            OUTPUT("Enter number 2")
            INPUT num2
      END PROCEDURE

      PROCEDURE Calculation()
            answer = num1 * num2
      END PROCEDURE

      PROCEDURE output_sub()
            OUTPUT("the product of " & num1 & " and " & num2 & " is ")
            OUTPUT (answer)
      END PROCEDURE
   BEGIN
       CALL  input_sub()
       CALL  Calculation()
       CALL  output_sub()
   END
```

Example Program – Procedures (VB Code)




```
Dim num1 As Integer
Dim num2 As Integer
Dim answer As Integer
  Sub input_sub()
        Console.WriteLine("Enter number 1")
        num1 = Console.ReadLine
        Console.WriteLine("Enter number 2")
        num2 = Console.ReadLine
  End Sub
  Sub Calculation()
        answer = num1 * num2
  End Sub
  Sub output_sub()
  Console.Write("the product of " & num1 & " and " & num2 & " is ")
  Console.WriteLine(answer)
  Console.ReadLine()
  End Sub
Sub Main()
    input_sub()
    Calculation()
    output_sub()
End Sub
```

Parameters

As mentioned above, **local variables** only have a **lifespan of the procedure.**
Sometimes it is useful to **pass a value from one procedure to another**.

This is done by using **parameters (or arguments)**
A parameter can be passed from one procedure to another by value or by reference.

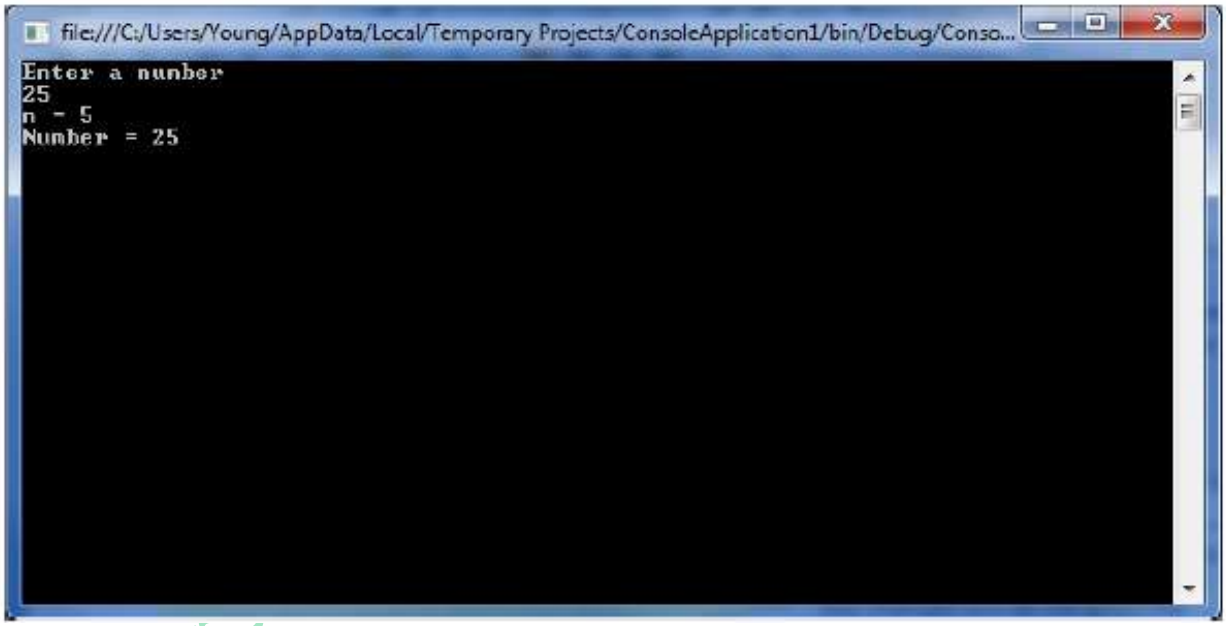
By Value

The word **ByVal** is short for **"By Value"**. What it means is that you are **passing** a **copy** of a variable to your Subroutine.
You can make changes to the copy and the **original will not be altered**.

```
Module Module1

    Sub WriteSQRT(ByVal n As Double)
        n = Math.Sqrt(n)
        Console.WriteLine("n = " & n)
    End Sub

    Sub Main()
        Dim number As Double
        Console.WriteLine("Enter a number")
        number = Console.ReadLine
        WriteSQRT(number)
        Console.WriteLine("Number = " & number)
        Console.ReadLine()
    End Sub

End Module
```

This procedure us expecting a double variable, which is known locally as **n**. Any changes to n do not effect the original variable

The variable **number** is passed to the subroutine *WriteSQRT*



By Reference

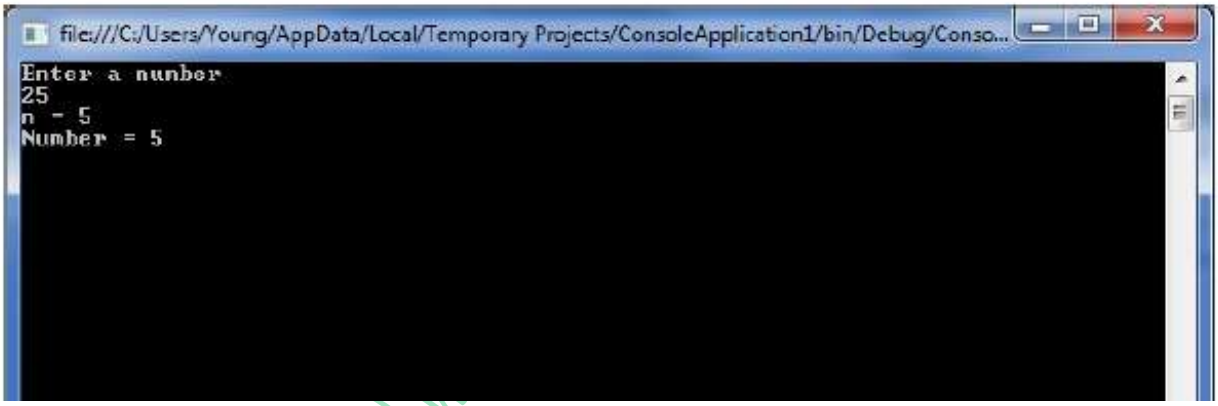
ByRef is the alternative. This is short for **By Reference.**

This means that you are not handing over a copy of the original variable but pointing to the original variable.

```
Module Module1

    Sub WriteSQRT(ByRef n As Double)
        n = Math.Sqrt(n)
        Console.WriteLine("n = " & n)
    End Sub

    Sub Main()
        Dim number As Double
        Console.WriteLine("Enter a number")
        number = Console.ReadLine
        WriteSQRT(number)
        Console.WriteLine("Number = " & number)
        Console.ReadLine()
    End Sub

End Module
```

This procedure us expecting a double variable, which is known locally as **n**. Any changes **WILL** effect the original variable

The variable **number** is passed to the subroutine *WriteSQRT*



A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures:

- Functions
- Procedure

Functions **return a value**, whereas Subs/**Procedures do not return a value.**

Defining a Function

The Function statement is used to declare the name, parameter and the body of a function. The syntax for the Function statement is:

```
FUNCTION Name (Parameter: Datatype) : RETURN DataType
    [Statements]
      RETURN DataType
END FUNCTION
```

Functions

Functions are similar to subroutines, except that they always return a value. They are normally used in either **assignments** (A:=TaxA(370);) or **expressions** (IF taxA(15000) THEN….)
The function names doubles as a procedure name and a variable.

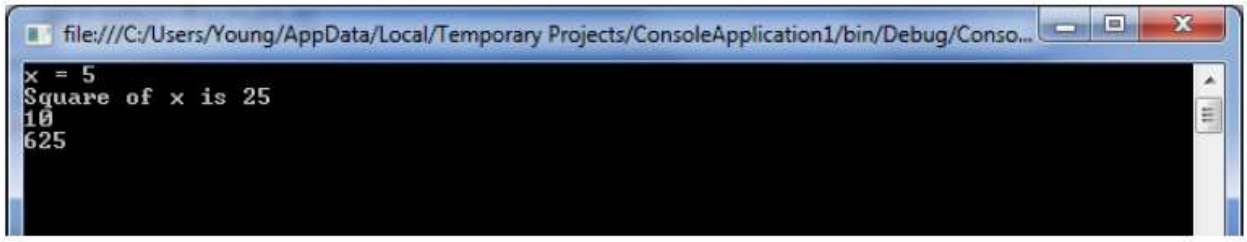
Pseudocode

```
FUNCTION square(ByVal num : INTEGER) : RETURNS INTEGER
  RETURN (num * num)  // The value of (num*num) will be stored in square
END FUNCTION
FUNCTION sum (ByRef a :INTEGER, ByRef b : INTEGER) RETURNS INTEGER
      RETURN a + b // The value of (a+b) will be stored in sum
END FUNCTION
BEGIN
      DECLARE number, value1, value2 : INTEGER
      PRINT (“Please Input a number for its square”)
      INPUT number
      PRINT ("Square of number is: “ (CALL square(number))
      PRINT (“Please Input a num1 and num2 for sum ”)
      INPUT value1, value2
      PRINT ("Sum is” (CALL sum(value1, value2))
END
```

PYTHON Doesn’t make PROCEDURES & FUNCTIONS seperately

Example VB Program – Functions

```
Module Module1
    Function square(ByVal x As Integer) As Integer
        square = x * x
    End Function
    Function sum(ByRef a As Integer, ByRef b As Integer) As Integer
        sum = a + b
    End Function
    Sub Main()
        Dim number As Double = 5
        Console.WriteLine("x = " & number)
        Console.WriteLine("Square of x is " & square(number))
        Console.WriteLine(sum(3, 7))
        Console.WriteLine(square(sum(16, 9)))
        Console.ReadLine()
    End Sub
End Module
```



Sample VB Program in Visual Studio

Module1 | Main

```
Module Module1
    'this is a function (functions return a value)
    Function adder(ByRef a As Integer, ByVal b As Integer)
        adder = a + b
        Return adder
    End Function

    Sub Main()

        Dim x As Integer
        x = adder(2, 3) 'call to function adder which returns a value
        Console.WriteLine("2 + 3 = " & x)
        'you can simply then code by putting the call directly into the print statement
        Console.WriteLine("4 + 6 = " & adder(4, 6))

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

Finding maximum with VB Function.

Following code snippet shows a function *FindMax* that takes two integer values and returns the larger of the two.

```
Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As
Integer
    Dim result As Integer  ' local variable declaration
    If (num1 > num2) Then
        result = num1
    Else
        result = num2
    End If
    RETURN result 'result will be returned to FindMAx Function.
End Function
```

Function Returning a Value

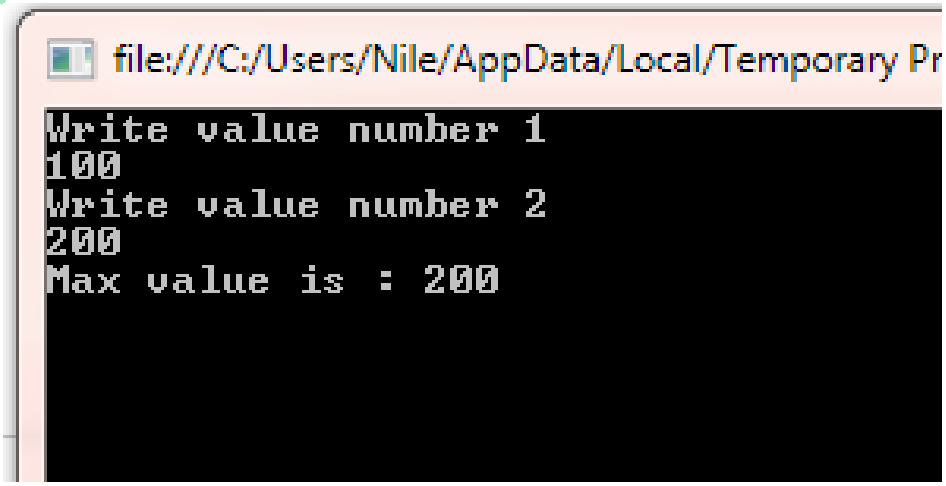
In VB.Net, a function can return a value to the calling code in two ways:

- By using the return statement
- By assigning the value to the function name

The following example demonstrates using the *FindMax* function:

```
Module module1
    Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
        ' local variable declaration */
        Dim result As Integer
        If (num1 > num2) Then
            result = num1
        Else
            result = num2
        End If
        FindMax = result
    End Function

    Sub Main()
        Dim a As Integer
        Console.WriteLine("Write value number 1")
        a = Console.ReadLine()
        Dim b As Integer
        Console.WriteLine("Write value number 2")
        b = Console.ReadLine()
        Dim res As Integer
        res = FindMax(a, b)
        Console.WriteLine("Max value is : {0}", res)
        Console.ReadLine()
    End Sub
End Module
```

When the above code is compiled and executed, it takes value 1 & value 2 as input and produces the maximum value for example:

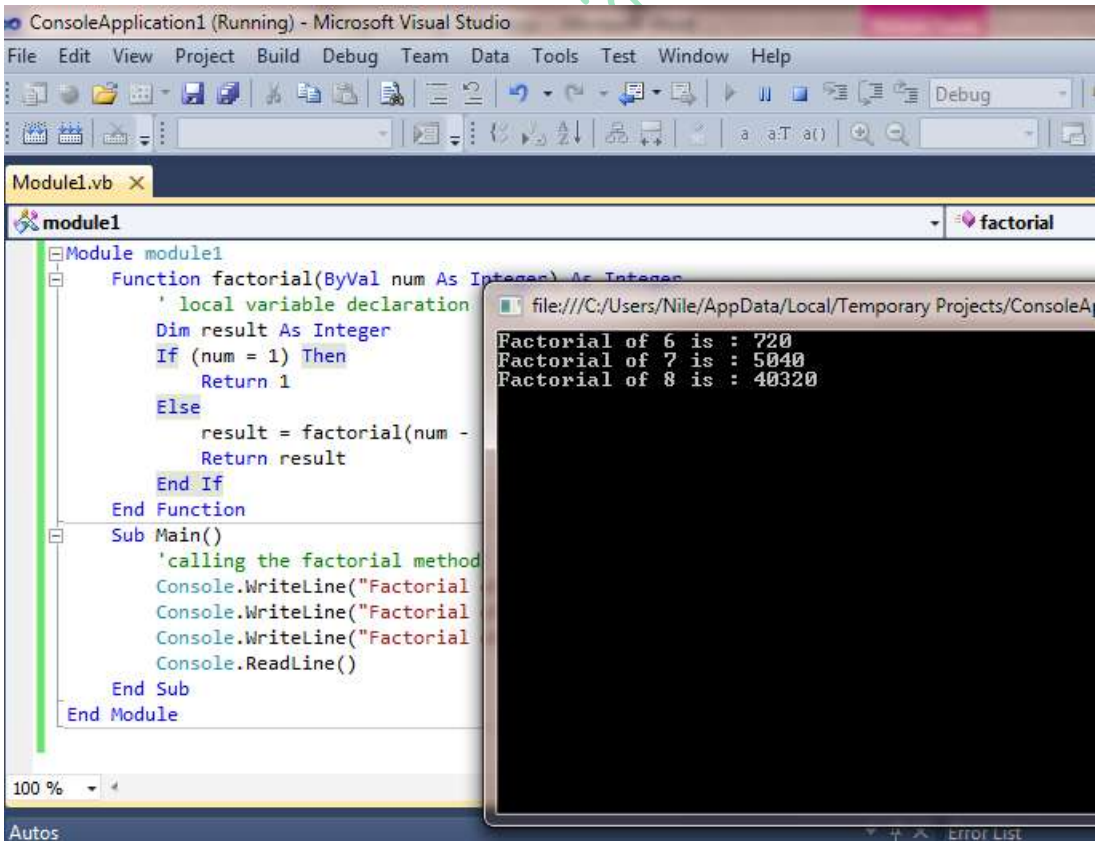


Recursive Function

A function can call itself. This is known as recursion. Following is an example that calculates factorial for a given number using a recursive function:

```
Module myfunctions
    Function factorial(ByVal num As Integer) As Intege ' local variable
declaration */
        Dim result As Integer
        If (num = 1) Then
            Return 1
        Else
            result = factorial(num - 1) * num
            Return result
        End If
    End Function
    Sub Main()
        'calling the factorial method
        Console.WriteLine("Factorial of 6 is : {0}", factorial(6))
        Console.WriteLine("Factorial of 7 is : {0}", factorial(7))
        Console.WriteLine("Factorial of 8 is : {0}", factorial(8))
        Console.ReadLine()
    End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:



References:

Visual Basics Console Cook Book by
VB.NET Console Book by ***Dough Semple***
https://www.tutorialspoint.com/vb.net/vb.net_functions.htm
https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/function-statement