

Syllabus Content:

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 Subroutines & Procedures

Initially, a program was written as one monolithic block of code. The program started at the first line of the program and continued to the end.

Program languages have now been developed to be structured. A problem can be divided into a number of smaller subroutines (also called **procedures**). From within one subroutine, another subroutine can be called and executed:

Subroutine

A subroutine is a self-contained section of program code that performs a specific task, as part of the main program.

Procedure

Procedure is giving a group of statements a name. When we want to program a procedure we need to define it before the main program. We call it in the main program when we want the statements in the procedure body to be executed.

Pseudocode

```
PROCEDURE <identifier>
<statement (s)>
ENDPROCEDURE
```

```
PROCEDURE <identifier> (BYREF <identifier>: <data type>)
<statement (s)>
ENDPROCEDURE
```

```
PROCEDURE <identifier> (BYVALUE <identifier>: <datatype>)
<statement (s)>
ENDPROCEDURE
```

ByRef vs. ByVal

Parameters can be passed by reference (ByRef) or by value (ByVal).

If you want to pass the value of the variable, use the **ByVal** syntax. By passing the value of the variable instead of a reference to the variable, any changes to the variable made by code in the subroutine or function will not be passed back to the main code.

This is the default passing mechanism when you don't decorate the parameters by using **ByVal** or **ByRef**.

If you want to change the value of the variable in the subroutine or function and pass the revised value back to the main code, use the **ByRef** syntax. This passes the reference to the variable and allows its value to be changed and passed back to the main code.



```

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 number(declared in procedure) from user
        timestable(Console.ReadLine) 'CALL to procedure to execute it in the main Program
        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

```

PSEUDOCODE FOR ABOVE 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") //asking for number(in
procedure) from user
CALL timestable //CALL to procedure to execute it in the main Program
END

```

Example Program – Procedures

```

Module Module
    Dim num1 As Integer
    Dim num2 As Integer
    Dim answer As Integer

    Sub input_sub()
        Console.Clear()
        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
End Module

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

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
End Module

```

Global Variables declared



Parameter

A parameter is a value that is 'received' in a subroutine (procedure or function). The subroutine uses the value of the parameter within its execution.

The action of the subroutine will be different depending upon the parameters that it is passed. Parameters are placed in parenthesis after the subroutine name. For example: Square(5) 'passes the parameter 5 – returns 25

ByRef vs. ByVal

Parameters can be passed by reference (**ByRef**) or by value (**ByVal**).

If you want to pass the value of the variable, use the **ByVal** syntax. By passing the value of the variable instead of a reference to the variable, any changes to the variable made by code in the subroutine or function will not be passed back to the main code.

This is the default passing mechanism when you don't decorate the parameters by using **ByVal** or **ByRef**. If you want to change the value of the variable in the subroutine or function and pass the revised value back to the main code, use the **ByRef** syntax. This passes the reference to the variable and allows its value to be changed and passed back to the main code.

Variable Scope

A variable holds data while the program is running. The scope of a variable defines where it can be seen. They are classified as either global or local

Global Variable

A global variable is declared in a module and is accessible from any procedure or function within that module.

Local Variables

A local variable is declared in a procedure or function and is only accessible within that procedure or function.

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.



Module Module1

```
Dim num1 As Integer  
Dim num2 As Integer  
Dim answer As Integer
```

Global variables, declared before any subroutines and are available throughout the

```
Sub input_sub()  
    Console.Clear()  
    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)  
End Sub  
  
Sub Main()  
    Dim answer As Char  
  
    Do  
        input_sub()  
        Calculation()  
        output_sub()  
  
        Console.WriteLine("another go? Y/N")  
        answer = Console.ReadLine()  
    Loop Until UCase(answer) = "N"  
  
End Sub
```

Local variable declared within a subroutine and is only available within this subroutine.

Computer Sc. (9618)

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

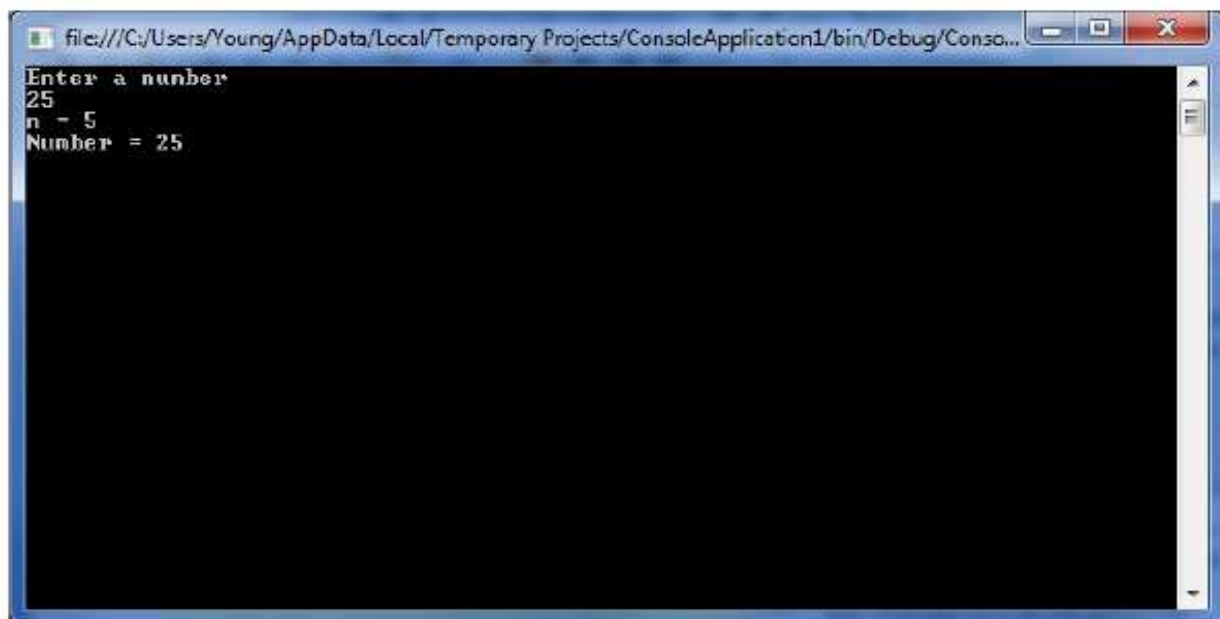
    Sub WriteSQRT(ByVal n As Double)
        n = Math.Sqrt(n)
        Console.WriteLine("n = " & n)
    End Sub

    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 is 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`



```
file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...
Enter a number
25
n = 5
Number = 25
```


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. Any change you make to the variable within your subroutine will effect the variable itself.

```
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 is 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*

```
file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...
Enter a number
25
n = 5
Number = 5
```

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
- Sub procedures or Subs

Functions return a value, whereas Subs 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:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
```

```
[Statements]
```

```
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.

```
Module Module1
```

```
Function square(ByVal x As Integer) As Integer
    square = x * x
End Function
```

```
End Module
```

Square is the function name, that is expecting an integer to be passed (byref) to it.

The result is assigned to the function name which is dimensioned as an integer. The function name can be used as a variable containing the result within other procedures.

ByRef vs. ByVal

Parameters can be passed by reference (ByRef) or by value (ByVal).

If you want to pass the value of the variable, use the **ByVal** syntax. By passing the value of the variable instead of a reference to the variable, any changes to the variable made by code in the subroutine or function will not be passed back to the main code.

This is the default passing mechanism when you don't decorate the parameters by using **ByVal** or **ByRef**.

If you want to change the value of the variable in the subroutine or function and pass the revised value back to the main code, use the **ByRef** syntax. This passes the reference to the variable and allows its value to be changed and passed back to the main code.

Pseudocode

```
FUNCTION <identifier> RETURNS <datatype>
```

```
<statement (s)>
```

```
ENDFUNCTION
```

```
FUNCTION <identifier> (<identifier> : <datatype>)
```

```
RETURNS <data type> // function has one or more parameters
```

```
<statement(s)>
```

```
ENDFUNCTION
```



Example Program in VB - Functions

```

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

```

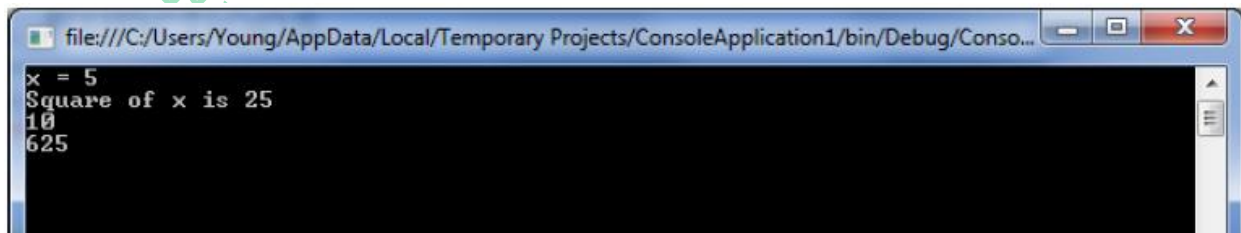
Example Program – Functions

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

```



```

file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...
x = 5
Square of x is 25
10
625

```

Programming languages, such as VB.net and spreadsheets, have many functions built-in. Examples include

SUM(range) Spreadsheet: to add a block of cell values.
LCASE(string) VB: converts a string to upper case
ROUND(integer) Round the integer up
RANDOM Generate a random number

Example

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 ' local
variable declaration
    Dim result As Integer
    If (num1 > num2) Then
        result = num1
    Else
        result = num2
    End If
    FindMax = result
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
```

```
file:///C:/Users/Nile/AppData/Local/Temporary Pr
Write value number 1
100
Write value number 2
200
Max value is : 200
```

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 Integer ' 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:

```
file:///C:/Users/Nile/AppData/Local/Temporary Projects/ConsoleA
Factorial of 6 is : 720
Factorial of 7 is : 5040
Factorial of 8 is : 40320
```

Calling a Function

You call a `Function` procedure by using the procedure name, followed by the argument list in parentheses, in an expression. You can omit the parentheses only if you aren't supplying any arguments. However, your code is more readable if you always include the parentheses.

You call a `Function` procedure the same way that you call any library function such as `Sqrt`, `Cos`, or `ChrW`.

You can also call a function by using the `Call` keyword. In that case, the return value is ignored. Use of the `Call` keyword isn't recommended in most cases. For more information, see [Call Statement](#).

Visual Basic sometimes rearranges arithmetic expressions to increase internal efficiency. For that reason, you shouldn't use a `Function` procedure in an arithmetic expression when the function changes the value of variables in the same expression.

Syntax

```
[ Call ] procedureName [ (argumentList) ]
```

Parts

`procedureName`

Required. Name of the procedure to call.

`argumentList`

Optional. List of variables or expressions representing arguments that are passed to the procedure when it is called. Multiple arguments are separated by commas. If you include `argumentList`, you must enclose it in parentheses.

Remarks

You can use the `Call` keyword when you call a procedure. For most procedure calls, you aren't required to use this keyword.

You typically use the `Call` keyword when the called expression doesn't start with an identifier. Use of the `Call` keyword for other uses isn't recommended.

If the procedure returns a value, the `Call` statement discards it.

Example

The following code shows two examples where the `Call` keyword is necessary to call a procedure. In both examples, the called expression doesn't start with an identifier.

```
Sub TestCall()  
    Call (Sub() Console.Write("Hello"))()  
  
    Call New TheClass().ShowText()  
End Sub  
  
Class TheClass  
    Public Sub ShowText()  
        Console.Write(" World")  
    End Sub  
End Class
```

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>

