Programming Guide with Visual Basic

This is a guide started in 2011 by Mr Gristwood, It is a work in progress, so if a section is not finished yet, and you think you can add some information to it. Please feel free to mail it to me and I will add it.

Contents

Introduction to programming.

Programming is all about organising your thoughts logically. This course is about teaching you to organise your thoughts so that you can make things happen! Everything that works on a computer has to have a program running through it.

Your First Program (Writeline and Readline)

Your first program is all about working to display things on the screen. For this we are going to use a console application and 2 commands:

- Console.Writeline()
- Console.Readline()

The best way to remember these things is, writeline displays something to the screen, readline takes information from the screen. Now is your turn to try it.

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World")  'A Writeline displays the words Hello world to the screen
        Console.ReadLine()                'A Readline waits for input from the user this keeps
        'the words on the screen so you can read them until enter
        'is pressed

    End Sub

End Module
```

Try typing this code into a Visual Studio Console application and see what happens.

EXERCISE

1. Using the console writeline and console readline ask the user a number of questions that they have to respond to. Make sure you save this program as “myquestions” as we will use this later.

All about Variables

Variables are useful when it comes to storing information. It's fine being able to read from the screen and display information to it. But once we have asked the user for information, we need a place to store it so we can use it later.

Variables are called variables because they can be changed at any time during the program. Variables are declared using the *DIM* keyword.

How variables work:

Declaring a variable in VB

```
Dim Number As Integer
```

In this example the blue words are keywords and must be typed correctly. We have used "number" as the name for our variable. It is IMPORTANT that you give all your variables a DECENT name otherwise you will forget what they are for when making bigger programs. Spaces are not allowed in names of variables so capitalisation can help. E.g. MyVariable makes it easy to read.

Getting a value into a variable:

```
Number = 6

Number = Console.ReadLine()
```

I've just above two ways of getting a value into a variable. The first way is allocating a value through the program. If I were to do this with a string(see below) then you would need to use the quotes (e.g. String = "Mr Gristwood"). The first few examples that you will make will require you to allocate values via the *console.readline* as covered in the previous lesson. This is a way of placing whatever is typed on the screen into a value that we can save and use for later.

Using a variable in a program

Once we have allocated values to a program we then need to use and manipulate them. This is where you start to use things you have learned in maths and algebra to change the value in them

e.g.

```
Dim Number As Integer
Dim Number1 As Integer
Dim Answer As Integer

Number = 5
Number1 = 6

Answer = Number + Number1
```

In this example we are adding together number and number 1 to place the value into the variable 'answer'. So what is the value of Answer now?

Printing out a variable

Once we have saved a value into a variable we need to do something with it. One of the most common things to do is to print it out to the screen. Below is the code to do this.

```
Console.WriteLine("Number is" & Number)
```

As you can see the words in quotes are written words that we want to display to the screen, the word 'Number' is the variable we want to print. In order to link these 2 values together we have to use the & sign.

Now, the example above shows how to work a variable with an Integer. There are a number of different data types that can be used for variables so that you can store different information:

- String – Stores strings of letters e.g. "Mr Gristwood"
- Integer – a number
- Char - One Character e.g. "A"
- Date – stores dates

These are the basic data types however there are **MORE TO BE ADDED**

Exercises

1. Modify the program in exercise 1 so that it accepts a number of questions and places them into variables. At the end of the program, print out each of the answers to the screen
2. Make a program that accepts 3 numbers from a console.readline() adds them all together and then displays the numbers the person has typed in and the final answer.
3. Modify the program further so that it asks for a person's name and their two favourite numbers and displays their name and the numbers they have given and what the answer is when *MULTIPLIED*

Commenting Code

Commenting is probably one of the most important things that you can learn to do and will really affect your final grades when programming. Commenting is a way of adding annotations to your code. Commenting has a number of benefits:

- If your code is broken and you have commented it as you go along, it's easier to find the places where the mistakes are and rectify them quick.
- If you leave your code for a period of time(e.g. when you go on work experience/term holiday) when you come back to it and have commented your code, you will get back into it much quicker
- If someone who has never seen your code before looks at it, they will pick up what has been going on quickly and can give you a higher mark or be able to help/improve your code.

Most examiners will only give you marks IF YOU COMMENT YOUR CODE. In real terms, as a programmer, nothing is more irritating than opening up a program and not being able to see what is going on by trying to read someone else's uncommented code.

So commenting, easy right? Like everything in life there is a right and wrong way of commenting the code. Below is two examples of good and bad code.

```
Module Module1

    Sub Main()
        Dim Num1 As Integer                 'Declaring num1 as Integer
        Dim Num2 As Integer                 'Declaring num2 as Integer
        Dim Firstname, Surname As String    'Declaring 2 strings called Firstname and surname


        Console.WriteLine("please enter your firstname")    'as the user for their firstname
        Firstname = Console.ReadLine()                      'Read the value into the firstname variable
        Console.WriteLine("please enter your Surname")      'Ask the user for their surname
        Surname = Console.ReadLine()                        'Read the value into the surname variable


        If Firstname = "Joe" And Surname = "Bloggs" Then 'Using the AND Statement asks and makes sure
            'that the IF checks for both words at the same time.
            Console.WriteLine("Welcome to the world" & Firstname & Surname)
        Else
            Console.WriteLine("get lost")
        End If

        'Realine at end of code to keep the box on the screen
        Console.ReadLine()
```

Here you can see my Bad example. I've commented every line of code. This took me nearly 30 minutes to do and adds nothing to the code. Think about the people who will look at your code. Most likely, they are coders too, which means they know what a declaration looks like, they just need to see where the declarations are, shown by a comment at the start of that section of code.

```
Module Module1

    Sub Main()
        'Declaring Local Variables
        Dim Num1 As Integer
        Dim Num2 As Integer
        Dim Firstname, Surname As String


        'Get user to enter their firstname and surname and read in the results to variables
        Console.WriteLine("please enter your firstname")
        Firstname = Console.ReadLine()
        Console.WriteLine("please enter your Surname")
        Surname = Console.ReadLine()

        'This IF statement checks if the firstname and the surname matches and then displays a
        'welcome message otherwise display a rejection message
        If Firstname = "Joe" And Surname = "Bloggs" Then
            Console.WriteLine("Welcome to the world" & Firstname & Surname)
        Else
            Console.WriteLine("get lost")
        End If


        'Placing a readline at the end of the code means that the console box will stay open
        'when the code has finished running
        Console.ReadLine()

    End Sub

End Module
```

If you compare this to the good example, I've explained at the top of each section what is happening and left it at that. One way, you will really get bored of programming quick, because you are spending forever commenting, the other is quick and easy and can be done as you go along, which will save you time in the long run.

The skill is learning when to comment, ask yourself, can you explain 3 lines of code with one comment? If so DO IT. After reading this section of the help guide, every piece of code you make should be commented.

The Selection statements (If and Select)

The If Statement

If statements are one of the most powerful statements that you can learn. These are REALLY useful as they are known as selection statements. They are used in programs in order to make a decision for you. We use IF statements in everyday life, but when we process them they seem slightly different: IF StillHungry THEN GetMoreFood.

This is an example:

```
If Number < 6 Then
    Console.WriteLine("Hello")
End If
```

This is the basic selection statement, If the number is less than 6 then print the word “hello”. Now this is a simple statement, that allows you to make a simple choice. In the cup of tea example, If you want sugar, add sugar, otherwise move on.

If statements can also be extended to include the ELSE statement, this makes them more interesting.

```
If Answer = "yes" Then
    Console.WriteLine("you answered yes")
Else
    Console.WriteLine("you must answer no")
End If
```

This allows someone to make a choice or decision. If there are only 2 options(e.g. yes, no), then an else statement can be useful. The final version of the IF statement is the ELSEIF statement. ELSEIF can work better if you have multiple options and you want to accept a number of different answers. An example of this is shown below:

```
If Number < 6 Then
    Console.WriteLine("Hello")
ElseIf Number > 6 Then
    Console.WriteLine("world")
End If
```

If Statement Excercises

- Write a program that asks the user for their Password, IF their password is equal to “password” then let then print the message “welcome to the world”, otherwise print “Access Denied”
- Modify your existing program that asks for the user’s 2 favorite numbers. Then ask the user for what they want to do with those numbers (+, -, *, /) then perform that operation using IF Statements

- Write another program that asks for the user for the answer to “life the universe and everything” If the number is less than 42, print the answer “too low” Otherwise, if the answer is higher than 42 print “too high”. Otherwise, print “Spot on”

The Case Statement

The case statement is another form of Selection statement. It is best used when you have multiple options and you want to select one based on a condition. An example of a case statement is shown below:

```
Dim optionnum As Integer

Console.WriteLine("please enter an option")
optionnum = Console.ReadLine()

Select Case optionnum
    Case 1
        Console.WriteLine("one")
    Case 2
        Console.WriteLine("two")
    Case 3
        Console.WriteLine("three")
End Select

Console.WriteLine("thank you for using my program, goodbye")
Console.ReadLine()
```

In this example I have an option number, and depending on the choice the user selects then it will display the number in words. This works out as a lot less typing than an IF/ELSE statement.

Case, What Else?

When making case statements it is important to have a 'final' condition, so that if someone types in an incorrect value the program allows for it.

```
Dim myname As String

Console.WriteLine("what is your name")
myname = Console.ReadLine()

Select Case myname.ToLower
    Case "mr gristwood"
        Console.WriteLine("you are the best MALE Computing Teacher in the school")
    Case "miss wilson"
        Console.WriteLine("you are the best FEMALE computing teacher in the school")
    Case "miss west", "mr hamilton"
        Console.WriteLine("you are the best A-Level ICT Teachers in the school!")
    Case Else
        Console.WriteLine("go and learn off your teachers")

End Select
```

In this example, there is 2 computing teachers and 2 ICT teachers at A-Level. If you type in any of their names, then it will display an appropriate message, otherwise, tell students to go and learn of these teachers. You can see the different structures of the case statement; A Case statement that has 2 possible values that can display the same message and the 'CASE ELSE' statement which is for incorrect values.

Every selection statement should have an option for if the user enters an incorrect value.

Case Sensitive Case's?

This case statement shows what can happen if you use numbers for a case statement, but what if you wanted to compare words? Case statements are CASE SENSITIVE. This is a problem, because you can TyPe SoMeTiMeS with mistaken caps in the wrong place. The way round this is to use a Toupper or ToLower command.

```
Select Case myname.ToLower
```

This will turn the entire variable 'myname' into lower case, making it easier to find the answer.

Case Statement Exercises

1. Modify your calculator program, use a CASE Statement instead of IF's
2. Write a menu system that asks the user to pick from a list of foods. If the user picks a healthy options display a positive message, otherwise display a negative message.
3. Modify this so that you ask someone who picks a healthy option if they want mayo, if they pick mayo display "ohhh, you were so close to being healthy" or something like this.

The Repetition Statements

Repetition statements are good for when you want to do something a certain number of times or until a condition is met. These are good for when you want someone to press the correct button or show something a number of times before quitting the loop.

The FOR loop

For loops are the counting repetition statements. For loops are used to count until a certain point and then quit.

```
Dim Tables As Integer
Dim Counter As Integer
Dim Answer As Integer

Console.WriteLine("which times table do you want to learn?")
Tables = Console.ReadLine()

For Counter = 1 To 10
    answer = Counter * Tables
    Console.WriteLine(Counter & "times" & Tables & "is" & answer)
Next
```

At the moment, FOR loops can seem pretty pointless, but when combined with Arrays later on (see the relevant chapter for examples)

Exercises

- Write a simple For Loop that asks how many times they would like to display a certain message on entering this number the program then prints out the message that many times.
- Make a variable that has a number in it. Give the user 10 guesses to pick the correct number, if the number is guessed correctly then exit the for loop and display either a congratulations statement or a commiseration statement. (You will need to use a FOR loop and a number of IF Statements as well as the EXITFOR command)

The WHILE loop

While loops are probably the more useful of the repetition statements, however can be more complicated than FOR loops because of it. While loops are designed to loop until a condition is met, this can be anything, while a variable is not equal to a word, while something is less than something else. This makes them extremely powerful and if you understand the syntax of IF statements you should be able to work these out. An Example of a simple while loop is shown below:

```
Sub Main()
    Dim task As String
    task = ""


    While task <> "work"
        Console.WriteLine("what are you doing?")
        task = Console.ReadLine()
    End While

    Console.WriteLine("congratulations for working so hard!")
    Console.ReadLine()
End Sub
```

As you can see I have a variable called 'task' that I set to blank as soon as the program starts, this is called initialisation (setting everything to blank so that they work correctly). The while loop checks for to see if the value of task is equal to work and if it isn't keeps on going around the loop.

There is another kind of while loop, which is the Do…Until Loop. This works almost the same way as the while loop but it allows you to enter the loop before you check for the condition to keep looping.

```
Dim WhatIsHe As String


Do
    Console.WriteLine("What is Mr Gristwood?")
    WhatIsHe = Console.ReadLine()

Loop Until WhatIsHe = "Cool"
```

You can see in this case I have not had to initialise the string, as we only test to see if the answer is right after we have accepted the value for it. The *'until'* statement can be quite a useful alternative to the while statement as it does exactly the same thing, but is worded slightly differently and some people find it easier to understand. If I were to rewrite the above statement with a while statement is would read:

```
Loop While WhatIsHe <> "Cool"
```

Exercises

1. Prompt the user to enter their password, you then have to Check their password against the one in the system. If their password is correct display "Welcome to the world" Otherwise continue to prompt for a password. **For an extra challenge** IF the user enters wrong 3 times, display "you are banished!" and quit (**EXIT WHILE**)
2. Suppose you had a credit card, with a balance of £50 owed on it. The bank charges you 2% interest every month. How many months could you leave it, before the bank charges you £100
3. Write a menu system that asks the user to pick from a list of foods. If the user picks a healthy options display a positive message, otherwise display a negative message. You must continue to display the menu until someone enters a quit command (e.g. press 'q' to quit the program)

Arrays

Arrays are a really important thing when it comes to programming. Arrays are a way of making lots of variables of the same type all at once. The proper definition of an array is a "collection of like-minded data". What happens is that when you declare an array, you specify how many variables

you want within it and the computer automatically reserves them for use later. Below is how you declare an array:

```
Dim sports(5) As String          'Declaring an array of 6
```

Now, as you can see from my comments, this is an array of 6? How does that work? When you declare an array, numbers always start from 0, not 1 which means you get an array of 6, not 5. As you can see this one is of the data type 'string' which means I can store up to 6 sports in this one variable.

Using the array

So, we've now declared our array and now need to look at how we can use it. You can do this a number of ways, firstly we'll look at allocating values into the array via the program, without any user interaction:

```
sports(0) = "Soccer"
sports(1) = "Cricket"
sports(2) = "Rugby"
sports(3) = "Aussie Rules"
sports(4) = "BasketBall"
sports(5) = "Hockey"
```

What I have done is allocate a value to each point in the array, so then when I want to print out a particular value in the array, all I do is:

```
Console.WriteLine("Name of the Sport in the third location" & " " & sports(2))
```

As we have said before, the third location is number "2" as all arrays start at 0.

n.b. there is nothing to stop you allocating values in your array starting at 1, but when you declare an array the computer will always make a space at 0

Arrays and for Loops, your new best friend

Earlier we discussed how to use a for loop and how it is very useful for counting things. Once we start using arrays, it suddenly becomes VERY useful as we now have a quick and efficient way of working our way through an array with very little code.

Say we want to get the user to type in 5 pieces of information, we would normally have to do something like this:

```
Console.WriteLine("please enter a value for variable 1")
variable1 = Console.ReadLine()
Console.WriteLine("please enter a value for variable 2")
variable2 = Console.ReadLine()
Console.WriteLine("please enter a value for variable 3")
variable3 = Console.ReadLine()
```

This is long, boring and time consuming. Now with the power of arrays I can do this in 3 lines for as many values as I like:

```
For Counter = 0 To 5
    Console.WriteLine("please enter a value for variable " & Counter)
    variable(Counter) = Console.ReadLine
Next
```

The for loop will cycle through the array adding one to the value of counter each time, which will fill up the array straight away.

Array exercises

1. Use a Loop (While, FOR) to read in up to 10 variables into an Array, Stopping when a .(full stop) Is placed in.
2. Write a program that reads 10 numbers entered by the user reports if any of them match
3. Write a program that reads in 10 numbers and prints them out in alphabetical order

String Theory – EXAMPLES TO BE ADDED, MORE TO BE DONE

Ok, so by this point you should be familiar with using the 'String' data type and should be using it for a lot of your programs. Now the string function is good, but there are other benefits too. There is a whole raft of 'specialist' commands that you can use with string variables that can make strings REALLY powerful. There is a full list of all the string functions available in Appendix 1, but I'm going to show you some of the more interesting examples and how they work below.

String Compare

String compare is a really useful function. Now, if we want to compare two values you can use an IF statement using the '=' command

```
Module Module1

    Sub Main()
        'Declare variables, Password is automatically allocated
        Dim Password As String = "Password"
        Dim UserPassword As String

        Console.WriteLine("please enter your password")
        UserPassword = Console.ReadLine()

        'Using the stringcompare function compares that password with the word the user enters
        'While password is not equal to Userpassword, Loop
        While StrComp(Password, UserPassword)
            Console.WriteLine("incorrect, please enter the right password")
            UserPassword = Console.ReadLine()
        End While

        Console.WriteLine("thank you for the correct password")

        Console.ReadLine()


    End Sub
```

```
While Password <> UserPassword
```

I've shown above the 2 different ways of doing it, as you can see there isn't much difference. So with the string compare function you can see the small difference it makes to the code, but in this example the code is classed as MUCH more efficient.

String Functions Exercises

1) Your Task is to make a form application that takes in a word and returns an answer telling you if it is a palindrome or not. Use the string functions document in order to work out how to do this.
2) For an extra challenge, you could try and use string functions to account for capital letters and spaces. E.g. Take out the capitals and spaces and say if it is still a palindrome.

REMEMBER COMMENT YOUR CODE

Sub Procedures and Functions

Sub procedures and functions are 2 words that really mean the same thing. They are a way of breaking down a program into smaller, more manageable chunks. There are a couple of reasons to do this:

When you are programming, if you program small bits at a time, it's easier as you are focusing on just getting one bit of the code working and not all of it

Sub Procedures can be used over and over again, so if you have a part of your code that you use all the time but are just changing one, maybe two things, then you should be thinking about using a sub-procedure/function. When you use a sub it is a bit like making a new mini program on your page:

How it works

Sub Procedures (subs) are the easier of the 2. A sub can have values sent to it(parameters), but it cannot return values. This is important to realize early on, as the only difference between a Sub and a Function is that the Function can return values back to the program above it.

A Sub Procedure, the example:

```
Module Module1

    Sub Main()
        Dim Name As String

        Console.WriteLine("Please enter your name")
        Name = Console.ReadLine()

        Display(Name)           ' This is the call to the sub procedure

        Console.WriteLine("thank you for entering your name" & Name & ", goodbye")

        Console.ReadLine()
    End Sub

    Sub Display(ByVal Name As String)   'This is the start of the sub procedure itself
        Console.WriteLine("welcome to the sub procedure example" & Name)
        Console.WriteLine("this shows how to call a sub procedure then return back to the main program")
        Console.WriteLine("it can make your life easier as it is more organised this way")

    End Sub


End Module
```

This example passes the name of the person as a parameter to the sub procedure then displays it using the 'Display' sub, when it has executed all the commands in 'Display' it then returns back to main and finishes the program.

Functions, the example

As said above, functions can return values which sub procedures do not. This is really useful because sometimes you want to do something to the data when in a sub procedure then return it to do something else to it later.

```
Module Module1

    Sub Main()
        Dim Num1 As Integer
        Dim Num2 As Integer
        Dim Answer As Integer

        Console.WriteLine("Please enter your first number")
        Num1 = Console.ReadLine()

        Console.WriteLine("Please enter your second number")
        Num2 = Console.ReadLine()


        'Before entering the funciton, answer is 0

        Answer = Maths(Num1, Num2)      'This is the call to the Function, it returns a value to answer

        'Answer is now Num1 + Num2

        Console.WriteLine("Adding together these 2 numbers gives" & answer & ", goodbye")

        Console.ReadLine()
    End Sub

    'This function adds 2 numbers together and returns the answer.
    Function Maths(ByVal Num1 As Integer, ByVal Num2 As Integer)
        Dim FunctionAnswer As Integer

        FunctionAnswer = Num1 + Num2

        Return FunctionAnswer 'the return here is KEY, i am returning whatever value is inside the
        'value FunctionAnswer to answer in Main
    End Function
```

This example shows how 2 parameters are passed to the function, the function then adds them up and returns the answer. The answer is then displayed on the screen. The point to note here is the 'RETURN'. This is the key difference between a sub and a function, functions can return values. This is not a very useful function, but if I extended it so that it did ANY mathematical operation, by sending it the 2 numbers and the operator, then it becomes useful.

Extra Points

As you can see above, I've also declared a variable inside the function. This is efficient programming as when the function finishes, all the variables declared in that function 'die', this frees up space in memory.

But what's the point in that?

Well, any program you write will be small in comparison to the major one's you use every day (e.g. word), but when you get to these programs, if you are not writing efficient code, your program will

use up loads of memory.

So?

If your program uses loads of memory, eventually, the computer cannot cope and shuts down. This is known as the BSOD (Blue Screen of Death) where the computer becomes overloaded and cannot function anymore.

The Recursive function – the hard one! – TO BE DONE

Sub Procedure/Function Exercises

1. Write a program with a function that takes in 3 numbers, works out which one is the biggest out of all of them and returns that number
2. Write a program with a number of subs/functions, that allow the conversion of degrees centigrade into degrees Fahrenheit (or visa-versa). Your program should ask which conversion to perform, what the temperature is and call the correct sub/function to do the calculation before displaying it to the screen.
3. Write a function that is passed an array of 20 characters and its length, and checks the array for the character '@'. If '@' is found, the function should return true, else the function should return false. Test your function in a program that allows the user to input characters, and using your function, returns a suitable message to the user.

Global and Local Variables

So, you thought in the beginning you learned all there was to know about Variables, WRONG! There are 2 places in a program that you can declare variables:

- At the top of a Sub Procedure/Function/Main
- At the very top of the program

So, what's the point?

Well, if you declare the variable at the top of the sub procedure/function/main then it is ONLY available in that area. You can pass that variable to other parts of the program, but on the whole a variable that is declared in the sub procedure, dies when the sub procedure ends.

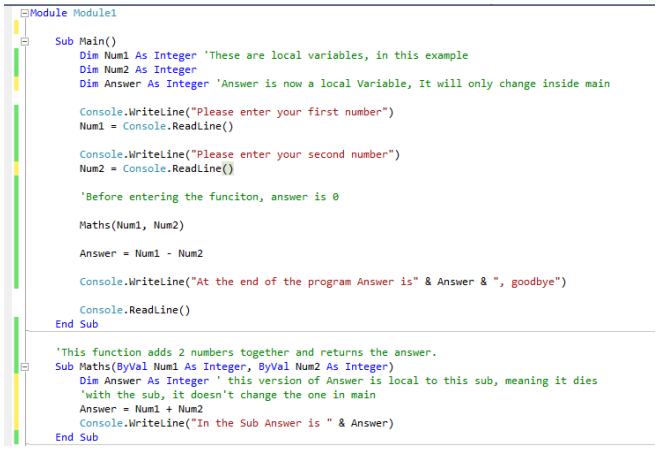
Global Variables are different. If you declare a variable at the very top of the program, then it exists all the way through the program.

Is this a good thing?

Well, it's easy. Declare everything at the top and they work in all your sub procedures!
Wow, great, lets do that then!
In programming terms this is a sledgehammer to crack a walnut, it will work, but it's not very efficient. There are times when you can't get a program to work because the variable won't pass between subs easily, then you use a Global variable, but unless that happens, you shouldn't really use them this way. In programming terms this is classed as really bad form.

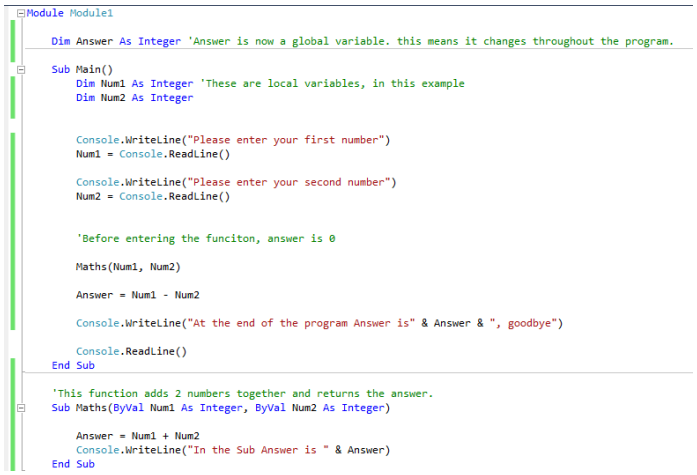
Global VS Local

Below is the local variable example. This is the way that you are used to programming. In the local variable example, all that will happen is that Answer will be the value of Num1 – Num2.



```
Module Module1

    Sub Main()
        Dim Num1 As Integer 'These are local variables, in this example
        Dim Num2 As Integer
        Dim Answer As Integer 'Answer is now a local Variable, It will only change inside main

        Console.WriteLine("Please enter your first number")
        Num1 = Console.ReadLine()

        Console.WriteLine("Please enter your second number")
        Num2 = Console.ReadLine()

        'Before entering the funciton, answer is 0

        Maths(Num1, Num2)

        Answer = Num1 - Num2

        Console.WriteLine("At the end of the program Answer is" & Answer & ", goodbye")

        Console.ReadLine()
    End Sub

    'This function adds 2 numbers together and returns the answer.
    Sub Maths(ByVal Num1 As Integer, ByVal Num2 As Integer)
        Dim Answer As Integer ' this version of Answer is local to this sub, meaning it dies
        'with the sub, it doesn't change the one in main
        Answer = Num1 + Num2
        Console.WriteLine("In the Sub Answer is " & Answer)
    End Sub
```

This is the global variable example, you can see that 'Answer' is declared outside of Main, this means it is changed in the function Maths so when it goes back to main answer changes there too.



```
Module Module1

    Dim Answer As Integer 'Answer is now a global variable. this means it changes throughout the program.

    Sub Main()
        Dim Num1 As Integer 'These are local variables, in this example
        Dim Num2 As Integer


        Console.WriteLine("Please enter your first number")
        Num1 = Console.ReadLine()

        Console.WriteLine("Please enter your second number")
        Num2 = Console.ReadLine()


        'Before entering the funciton, answer is 0

        Maths(Num1, Num2)

        Answer = Num1 - Num2

        Console.WriteLine("At the end of the program Answer is" & Answer & ", goodbye")

        Console.ReadLine()
    End Sub

    'This function adds 2 numbers together and returns the answer.
    Sub Maths(ByVal Num1 As Integer, ByVal Num2 As Integer)

        Answer = Num1 + Num2
        Console.WriteLine("In the Sub Answer is " & Answer)
    End Sub
```

Global and Local Variable Exercises

A program is required to keep a record of the number of students enrolled on a course. Your program should allow a student to be added to the course, a student to be removed from a course and should be able to display enrollment statistics and should be capable of continuing until a suitable terminator is received. You could consider using separate arrays to store each of the pieces of information.

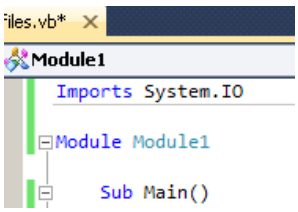
- Display the current enrolment statistics displaying the total number of students, the percentage of male students, the percentage of female students and the split between local/non-local students. Your program will accept numbers of students and display percentages. Use functions/sub procedures where necessary.
- Accepting input from the keyboard, enroll a student ensuring that your sub/function accepts all the necessary data to maintain enrollment statistics.

- Update enrollment statistics to remove a student. Global variable should only be used where absolutely necessary so your program will need to pass information between sub-programs using parameters.

Working with Files

So by now you should be familiar with writing various different programs and you can get user input and print it back to the screen. So what if I wanted to finish the program, store the information and bring it back later? This is the beauty of files. Files allow you to close a program and come back to it later and easily retrieve the information.

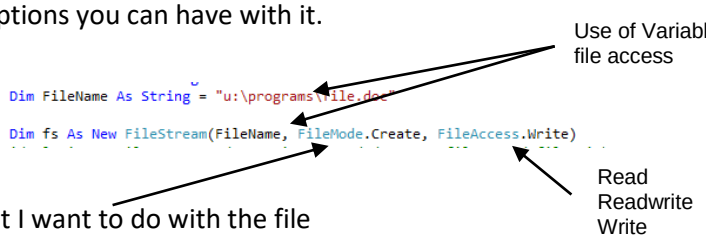
Files require extra 'plugins' to make them work with visual basic, it also requires the understanding of various different commands. When we talk about plugins in programming, we call these Libraries. The one we need for files to work is called the System Library and it goes at the very top of your program, right before everything else.



Once we have this library, we can then start to work with files. There are a number of new commands here and we will work through each individually before showing a full working program.

Filestream

Filestream is the command that allows you to open a file. This indicates the file you want to open and how you want to open it. Below is an example of the filestream command, but I've put around it all the possible options you can have with it.



- Create New – creates a new file with the name of the file in the locaation
- Create – creates a new file or if one exists, delete it and start again
- Open – Open an existing file
- OpenorCreate – Open an existing file, if one doesn't exist, create it.
- Append – Open the existing file and find the end of it.

This looks pretty complicated, but it's not. I've just tried to show all the different options you have for the 3 different sections of the filestream command. As you have noticed, I've declared it just as I would a normal variable.

StreamReader/StreamWriter

The streamreader and streamwriter commands are a bit like console.writeline/readline. The only difference is instead of writing to the screen, you write to the file. The other thing you have to know is that you have to declare them as a variable. You need one reader/writer for each file you wish to open. You declare them like this:

```
Dim s As New StreamWriter(fs)
'creating a new StreamWriter and pass:
```

You can see that I have declared streamwriter as 's'(your variable name will be better!) and pointed it to the filestream I want to open. Streamwriter allows you to pass information to the file:

```
s.WriteLine("This is an example of using file handling concepts in VB .NET.")
s.WriteLine("This concept is interesting.")
```

Streamreader is slightly different, this needs a special command called 'peek' in order to read from the file:

```
Dim d As New StreamReader(fs)
d.BaseStream.Seek(0, SeekOrigin.Begin)
While d.Peek() > -1
    Console.WriteLine(d.ReadLine())
End While
d.Close()
```

The 'while' loop will read from the file until the peek command hits the end of the file. In this example I'm using a writeline to display it to the screen, but you could easily read the values into a variable then you can find different parts of the file or change information then write it back to the file later.

One last thing: Whenever you open a streamreader or streamwriter you need to make sure that you close it. You can see in the last example. There is a command d.close() which shows you how to close the streamreader or streamwriter class.

The Example

Taking all the stuff we have just talked about above and putting it all together leaves us with this example:

```
Dim FileNumber As Integer = 1
Dim FileName As String = "u:\programs\file.doc"

Dim fs As New FileStream(FileName, FileMode.Create, FileAccess.Read)
'declaring a FileStream and creating a word document file named file with
'access mode of writing
Dim s As New StreamWriter(fs)
'creating a new StreamWriter and passing the filestream object fs as argument
s.BaseStream.Seek(0, SeekOrigin.End)
'the seek method is used to move the cursor to next position to avoid text to be
'overwritten
s.WriteLine("This is an example of using file handling concepts in VB .NET.")
s.WriteLine("This concept is interesting.")
'writing text to the newly created file
s.Close()
'closing the file


fs = New FileStream("U:\programs\file.doc", FileMode.Open, FileAccess.Read)
'declaring a FileStream to open the file named file.doc with access mode of reading
Dim d As New StreamReader(fs)
'creating a new StreamReader and passing the filestream object fs as argument
d.BaseStream.Seek(0, SeekOrigin.Begin)
'Seek method is used to move the cursor to different positions in a file, in this code, to
'the beginning
While d.Peek() > -1
    'peek method of StreamReader object tells how much more data is left in the file
    Console.WriteLine(d.ReadLine())
    'displaying text from doc file on the screen
End While
d.Close()

Console.ReadLine()
```

Exercises

- Write a program that writes your name to a text file (use the program to create the file). Look at the text file after the program is run, make sure that the files is created and saved in an appropriate place. Modify the program to read your name back from the file within the program to make sure it works.
- Write a program that gives and takes advice on how to write a program. The program should open by displaying a piece of advice to the screen. It should then ask the user to type in a different piece of advice. The next user of the program receives the advice that was typed in the last time the program was run. Be sure that the first person to run the program gets some advice.
- The organisation Very Best Visual Basic Beginners Becoming Better has declared “B” to be a dirty letter and has hired you to write a program to eliminate this letter from text files. Write a program that replaces all occurrences of the letter “B” in a text file with the letter “X” the program should replace both upper and lowercase letters, “b” and ‘b’, with ‘X’

Programming with Forms TO BE DONE

So up until now we have been working with what is called a console application. This means that when you run the program a box will show up on the screen that runs your program and all the program will be text based (e.g. ask a question, get an answer).

However, visual basic does allow a different type of programming, this is that of Form development. Making a program with a form that allows you to click a button then something happen behind it. This can make your life a lot easier as it is easier to visualize what you want the person to do to make something happen. At the same time, when we start form programming we must not forget all the things we have learned in the console. The same rules still apply. You have to comment your code and use sub-procedures where necessary to make sure your code is efficient as it can be

The message box – TO DO

This is the simplest form of Form Programming. Run your program, when something happens, display a messagebox to the screen. This is done using the Messagebox.show command.

All the messagebox does, is show a command to the screen. It is only there to deliver a message. Mine is 'hello world' but it can be anything. This could be useful to display the answer to a mathematical problem to the screen or just to say, 'this is going to end the program'

The combo Box – TO DO

```
For counter = 1 To 5
    Countrycombo.Items.Add(countries(counter))
Next counter
```

Exam Tip: When coding for assignments there are 2 ways to come to a solution: one that works and one that works efficiently. The first will give you the bottom mark bands, the second will get you the higher band. Sometimes it is good to get the code working then try to make it efficient, but this will take you time, thinking about how to make it efficient in the first place is always the best way to go.

APPENDIX – List of Operators

Symbol	Meaning
>	Greater Than
<	Less than
>=	Greater Than or Equal to
<=	Less Than or Equal to
<>	Not Equal to
AND	Used for combining different condition statements e.g. IF <condition> AND <condition> THEN
OR	Used for combining different condition statements e.g. IF <condition> AND <condition> THEN
Is, **IsNot**	
Like	
TypeOf...**Is**	To check if the data entered is of the right data type e.g. string, integer
Operator Precedence in Visual Studio (For doing Mathematics)	
Exponentiation (^)	
Unary identity and negation (+, –)	
Multiplication and floating-point division (*****, /)	This will give both whole number and remainder
Integer division (\)	This will give the whole number of a division
Modulus arithmetic (**Mod**)	MOD will give the remainder of a division
Addition and subtraction (+, –), string	
concatenation (+)	
String concatenation (**&**)	This will place 2 strings together e.g. (“hello ” & “world” = “Hello World”
Arithmetic bit shift (<<, >>)	

APPENDIX A – List of string functions

Below is a list of all the string functions that are available in VB.Net.

Function	Use
Asc()	Returns the character code of the first character of a string.
Asc("A")	returns 65.
Chr()	Returns the display character of a character code.
Chr(65) .	returns "A"
GetChar()	Returns the character at a specified position in a string, counting from 1.

GetChar("This is a string", 7)	returns "s".
InStr()	Returns the starting position in a string of a substring, counting from 1.
InStr("This is a string", "string")	returns 11.
InStrRev()	Returns the starting position in a string of a substring, searching from the end of the string.
InStr("This is a string", "string")	returns 11.
LCase()	Returns the lower-case conversion of a string.
LCase("THIS IS A STRING")	returns "this is a string".
Left()	Returns the left-most specified number of characters of a string.
Left("This is a string", 4)	returns "This".
Len()	Returns the length of a string.
Len("This is a string")	returns 16.
LTrim()	Removes any leading spaces from a string.
LTrim(" This is a string")	returns "This is a string".
Mid()	Returns a substring from a string, specified as the starting position (counting from 1) and the number of characters.
Mid("This is a string", 6, 4)	returns "is a".
Replace()	Replaces all occurrences of a substring in a string.
Replace("This is a string", " s", " longer s")	returns "This are a longer string" (replaces an "s" preceded by a blank space).
Right()	Returns the right-most specified number of characters of a string.
Right("This is a string", 6)	returns "string".
RTrim()	Removes any trailing spaces from a string.
RTrim("This is a string ")	returns "This is a string".
Str()	Returns the string equivalent of a number.
Str(100)	returns "100".
Space()	Fills a string with a given number of spaces.
This & Space(5) & "string" .	returns "This string"
StrComp()	Compares two strings. Return values are 0 (strings are equal), 1 (first string has the greater value), or -1 (second string has the greater value) based on sorting sequence.
StrComp("This is a string", "This string")	returns -1.
StrReverse()	Reverses the characters in a string.
StrReverse("This is a string")	returns "gnirts a si sihT".
Trim()	Removes any leading and trailing spaces from a string.
Trim(" This is a string ")	returns "This is a string".
UCase()	Returns the upper-case conversion of a string.

UCase("This is a string")	returns "THIS IS A STRING".
Val()	Converts a numeric expression to a number.
Val((1 + 2 + 3)^2)	returns 36.