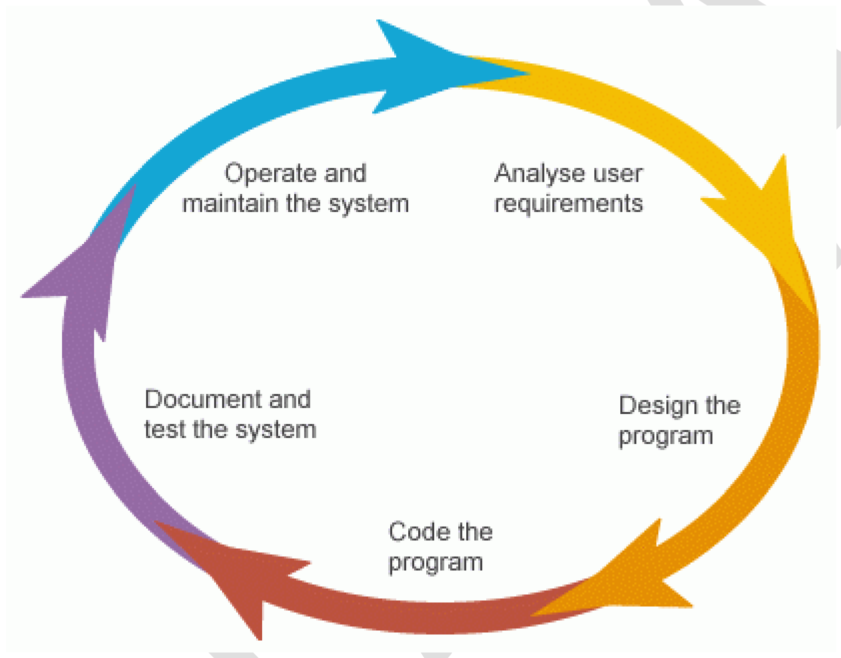
12. Software development

When software applications are built, programmers do not rush and start writing code. Instead, they follow an organised plan/methodology which breaks the software development into a series of tasks. There different methodologies but they tend to be variations of what is called the program development life cycle (PDLC).

The **program development life cycle (PDLC)** is an outline of each of the steps used to build software applications. Similarly to the way the system development life cycle (SDLC) guides the systems analyst through development of an information system, the program development life cycle is a tool used to guide computer programmers through the development of an application (software).

12.1 Stages of program development life cycle (PDLC)


Operate and maintain the system
Analyse user requirements
Document and test the system
Design the program
Code the program

Program/Software development consists of a number of stages:

- Requirement identification
- Design
- Coding
- Testing
- Documentation
- Maintenance

(i) Requirement Identification

Define the problem to be solved, and write program specifications including descriptions of the program's inputs, processing, outputs, and user interface.

(ii) Design

Develop a detailed logic plan using a tool such as pseudocode, flowcharts or structure diagrams to group the program's activities into modules and devise a method of solution or algorithm for each module

(iii) Coding

Translate the design into an application using a programming language by creating the user interface and writing code with comments to explain the purpose of code statements.

(iv) Testing

Test the program, finding and correcting errors (debugging) until it is error free and contains enough safeguards to ensure the desired results.

(v) Documentation

Formalise and complete end-user (external) documentation

(vi) Maintenance

Provide support to end users, correcting any unanticipated errors that emerge and identify user-requested modifications (known as enhancements). Once errors or enhancements are identified, the program development life cycle begins again.

Note: Stages may overlap

12.1System maintenance

Systems/software are designed for a well-defined purpose and should realize that purpose; if they do, they are considered successful. During use it may become necessary to alter the system for some reason - this is known as maintenance.

There are three different types of maintenance to be considered:

(i) Corrective maintenance

Corrective maintenance is necessary when a fault or bug is found in the operation of the new system. These are software bugs which were not picked up at the formal testing stage. A technician or the original programmers will be needed to correct the error.

(ii) Adaptive maintenance

Adaptive maintenance is necessary when conditions change from those that existed when the original system was created. This may be because of a change in the law (tax rates may change, for example) or the hardware may be changed, so that changes need to be made to the software for it to remain functional.

(iii) Perfective maintenance

Perfective maintenance is required to "tweak" the system so that it performs better. For example, searching for a particular stock item may be quite slow. The technician decides that supplier details should be stored in another file rather than with the details of the stock, the size of the stock file is reduced and it is far quicker to search. This has not changed how the system operates as far as the user is concerned but the performance improves.

In sum, a system/software is maintained after going live (installation):

- Corrective maintenance fixes software bugs which were not picked up at the formal testing stage.
- Adaptive maintenance changes the system to cater for changes in the law, or hardware or new business procedures.
- Perfective maintenance makes the system perform better.

Note: The need for continual maintenance for a system/software

Computing and computer applications change regularly through advances in technology, new ideas, different legal frameworks and different business practices. A system should never be considered to be finished. Rather than being a linear process with a beginning, a middle and an end, it should be thought of as a circular process, continually returning to previous stages to fine tune them and take advantage of changing circumstances.

In addition to the need for continual maintenance, all systems have a natural lifespan and eventually need to be replaced. The limited lifespan of a system is known as obsolescence. Even the most up-to-date and most expensive system eventually becomes out-of-date. This may be because a piece of hardware needs to be replaced and it is not possible to find anything that is compatible with the software; because the competitor for a business updates all its systems and customers find their service more impressive; or because customers expect more to be done for them.

12.2 Program testing

Programs are tested after they are written to ensure that they are error free and perform what they are intended to.

Program errors may be of the following types:

(1) Errors in syntax (incorrect use of the programming language)
(2) Errors in logic (the program does not do what was intended)
(3) Run-time errors (ones that are only discovered when the program runs).

12.2.1 Syntax errors

Syntax errors are errors in the grammar of the program language, i.e., the rules of the language have been broken.

Common errors are:

- typing errors, such as Selct for Select
- an If statement without a matching End If statement
- For statement without a matching Next statement
- An opening parenthesis without a closing parenthesis: (a + b.

How syntax errors detected?

- Most syntax errors are spotted by the programmer before an attempt to run the code.
- Some program editors help to identify syntax errors. For example, the Visual Basic.NET editor underlines any variable in the code which has not been declared. Similarly, it highlights declared variables which have not been used in the code.

12.2.2 Logic errors

A logic error occurs when the programmer makes a mistake in their logic for some part of the program.

Common errors are:

(1) Suppose a programmer wants the first 10 positive integers to be output and creates the following algorithm:

```
For Count = 0 To 10
    Output Count
Next Count
```

This algorithm has no grammatical errors but it does not produce the correct result. It produces the integers 0 to 10 not 1 to 10. This type of error can only be found by thorough testing. The programmer can carefully check the code by reading it or can attempt to run the program.

(2) Another common error is to use the wrong arithmetic symbol, e.g. a+b instead of a-b, or to put parentheses in the wrong place, such as (a+b-c)*d instead of (a+b)-c*d. These errors should be identified during testing and are all a mistake in the logic of the programmer.

12.2.3 Run-time errors

Run-time errors occur during the execution (running) of the program.

Example of a runtime error

A typical error is to have an expression such as (a+b)/(c-d), used in the program but c is equal to d, resulting in an attempted division by zero. The problem is that this error may be undetected for a considerable time because the equality of c and d rarely happens. However, it could be disastrous when it does.

It is a good idea to test the denominator before division to ensure that this error doesn't crash the program. Instead, the programmer can "trap" the error and displays an error message.

12.3 Testing strategies

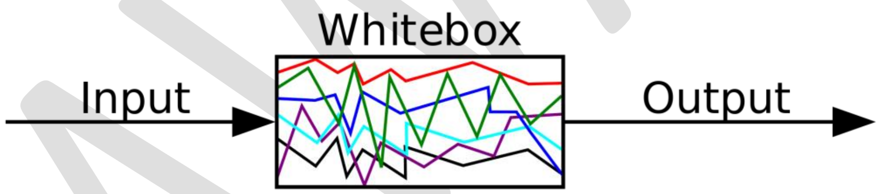
Software creation is a human activity and, as a result, is subject to errors. All software has to be tested. Only the simplest of programs work first time and most contain errors (or bugs) that have to be found. If a program has been designed and written in modules, it is much easier to debug because the programmer should be able to test each module independently of the others. There are several strategies the programmer can adopt for testing a program.

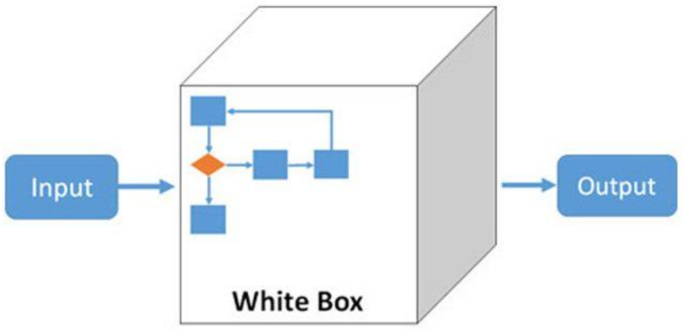
12.3.1White box testing

White box testing is where testers examine each line of code for the correct logic and accuracy. This may be done manually by:

(1) Drawing up a table to record the values of the variables after each instruction is executed. Examples could include selection statements (If and Case Select), where every possible condition must be tested.

(3) Debugging software can be used to run the program step by step and then display the values of the variables.





Trace table

It is a technique used to test algorithms to make sure that no logical errors occur. Working through an algorithm using test data values is called a dry run or trace. The Hand tracing or desk checking or 'dry running' allows you to use a **trace table** to:

- see what code will do before you have to run it
- find where errors in your code are

Note: A trace table need to keep track (trace) all the variables and outputs when drawn.

To make a trace table, the first step is to note the table headings, this involves the following:

1. Variables: note all the variables in the piece of code you are looking at (this includes arrays). Note each variable as a heading
2. Output: note if there is an output and put this as a heading

Example: Dry run the following codes

```
Dim y As Integer = 3
    For x = 1 To 4
      y = y + x
    Next
    Console.WriteLine(y)
```

Solution:

To do this, we create a trace table:

X	Y	Output
	3	
1	4	
2	6	
3	9	
4	13	13

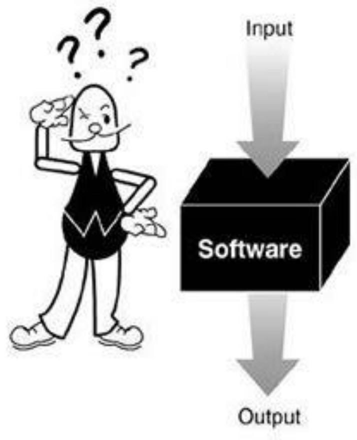
Class activity 82
Dry run the following algorithm using the sequence of input data -2, 0, 3, 5, 12, 16:

```
n=O
WHILE n <= 0
        INPUT n
ENDWHILE
Total ← 0
Count← 0
REPEAT
        INPUT Number
        Total ← Total + Number
        Count ← Count + 1
UNTIL Count = n
OUTPUT Total/Count
```

12.3.2 Black box testing

Black box testing is one of the earliest forms of test where the programmer uses test data, for which the results have already been calculated, and compares the results from the program with those that are expected.

- The programmer does not look at the individual code to see what is happening - the code is viewed as being inside a "black box". (no knowledge of the internal code and structure required)
- The testing only considers the inputs and the outputs they produce.



Black-Box Testing

Black-box test design is usually described as focusing on

- testing functional requirements
- external specifications or interface

- specifications of the program or module.

Black box testing makes use of test plan:

A test plan is used to test the correctness of a program by using test data which includes

- Normal data is data that the system is expected to process.
- Borderline/extreme/boundary data is at the boundary of data that the system is expected to process.
- Invalid/abnormal data is data that the system is expected to reject.

Consider the pseudocode below verifies if an input value lies between 5.5 and 9 inclusive:

```
Input n
    While n <= 5.5 OR n>= 9
        Input n
    End While
```

In this case:

- Normal data: any value between 5.5 and 9 (excluding 5.5 and 9)
- Borderline data: 5.5 and 9
- Invalid data: they could be 4, 10 or even 2000.

12.3.3 Stub testing

A stub is a dummy (fake) program or component used when the actual program code is not ready for testing. It acts as a temporary replacement for a called module and give the same output as the actual product or software.

For example, if there are 3 modules and last module is yet to be completed and there is no time, then we will use dummy program to complete that 3rd module and we will run whole 3 modules also.

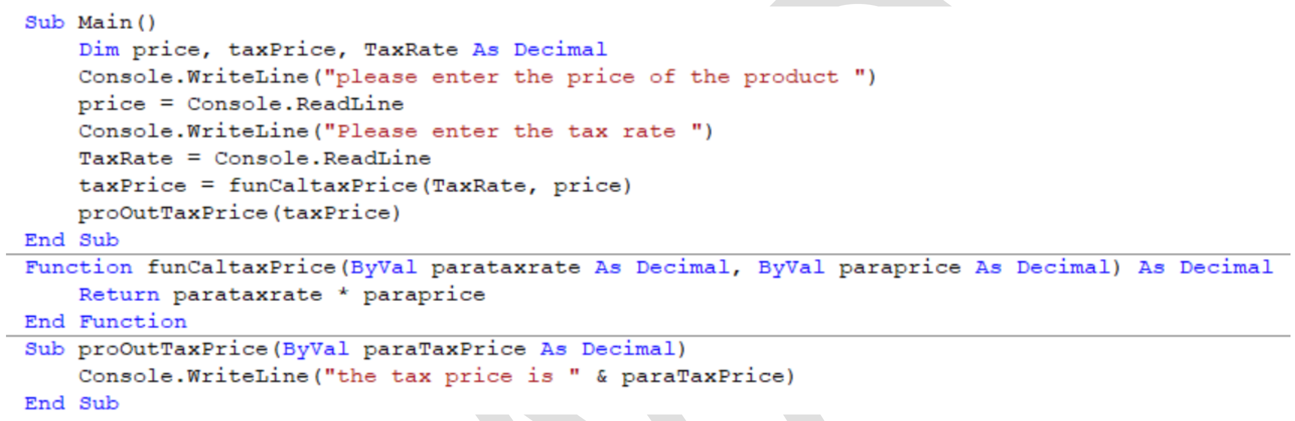
Illustration of stub testing

Consider a program for calculating tax on a product that requires 3 modules. The 1st module is the main module which gets the product's price and tax rate from a user and the 2nd one calculate the tax on the product and the 3rd module output the tax on the product.

The 2nd module to calculate the tax need to be dynamic, that is it will determine the tax price based on tax rate provided by the user.
However, due to lack of time and the need to deliver the program to the client, the programmer hasn't been able to code the module dynamically. So he created a stub module to calculate the tax price by taking the tax rate as 15% and price as Rs100 instead of taking it from the user.

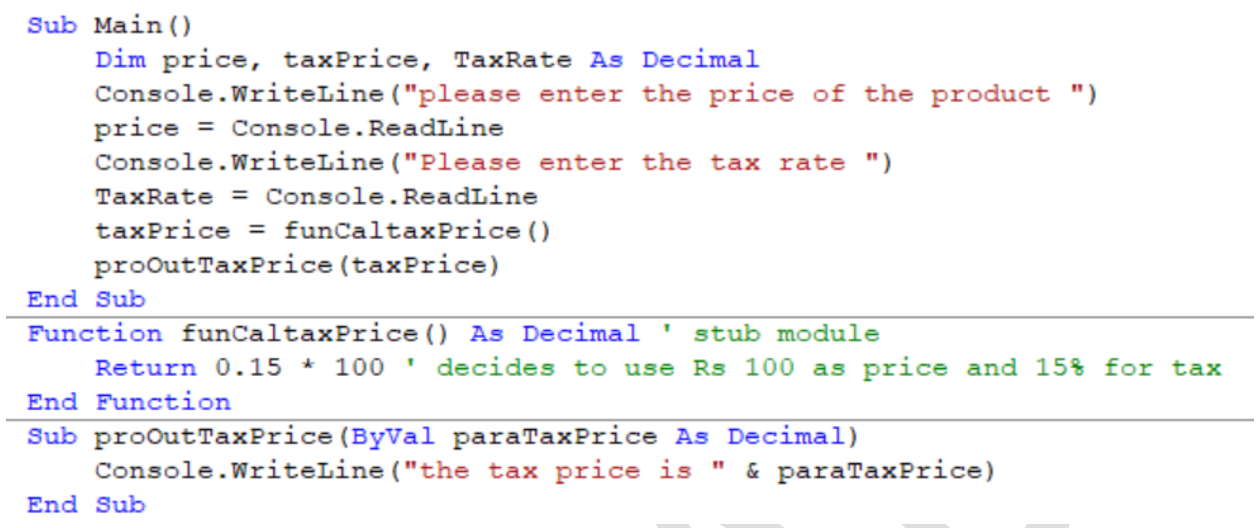
This thus enables him to test the whole program as shown below.

Expected program:



```
Sub Main()
    Dim price, taxPrice, TaxRate As Decimal
    Console.WriteLine("please enter the price of the product ")
    price = Console.ReadLine
    Console.WriteLine("Please enter the tax rate ")
    TaxRate = Console.ReadLine
    taxPrice = funCaltaxPrice(TaxRate, price)
    proOutTaxPrice(taxPrice)
End Sub
Function funCaltaxPrice(ByVal parataxrate As Decimal, ByVal paraprice As Decimal) As Decimal
    Return parataxrate * paraprice
End Function
Sub proOutTaxPrice(ByVal paraTaxPrice As Decimal)
    Console.WriteLine("the tax price is " & paraTaxPrice)
End Sub
```

Actual program with stub:



```
Sub Main()
    Dim price, taxPrice, TaxRate As Decimal
    Console.WriteLine("please enter the price of the product ")
    price = Console.ReadLine
    Console.WriteLine("Please enter the tax rate ")
    TaxRate = Console.ReadLine
    taxPrice = funCaltaxPrice()
    proOutTaxPrice(taxPrice)
End Sub
Function funCaltaxPrice() As Decimal ' stub module
    Return 0.15 * 100 ' decides to use Rs 100 as price and 15% for tax
End Function
Sub proOutTaxPrice(ByVal paraTaxPrice As Decimal)
    Console.WriteLine("the tax price is " & paraTaxPrice)
End Sub
```

12.4 Features found in a typical Integrated Development Environment (IDE)

Procedural programming languages tend to have an integrated development environment (IDE) that provides for all stages in the program's development: creation, compilation, testing and execution. An IDE enables the programmer to use some very sophisticated techniques such as

- Suggesting programming elements
- Reporting syntax errors
- Debugging
- Watch values of elements during run time

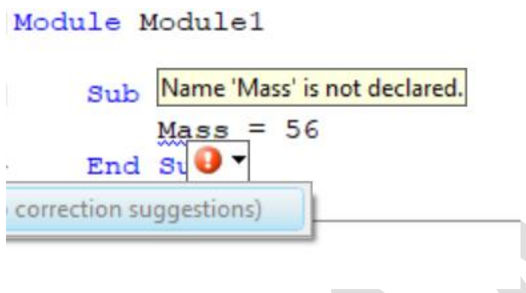
These are discussed in the section below:

(i) For coding

The IDE suggest programming elements defined by the user (e.g. variables, constants, procedure, function identifiers) and programming elements already in VB (e.g. reserved words)) as we type and we just need to select them to display them instead of typing them fully as well as no need to remember them by heart, saving time and allowing easy programming.

(ii) For initial error detection

During translation of a high-level language into machine code (or intermediate code), syntax errors are spotted and reported by the programmer. Some text editors report a syntax error as the code is typed in. For example, if you write the instruction Mass = 56 and the variable Mass has not been declared, some editors report it immediately. In any case, it is reported during translation by the interpreter or compiler.



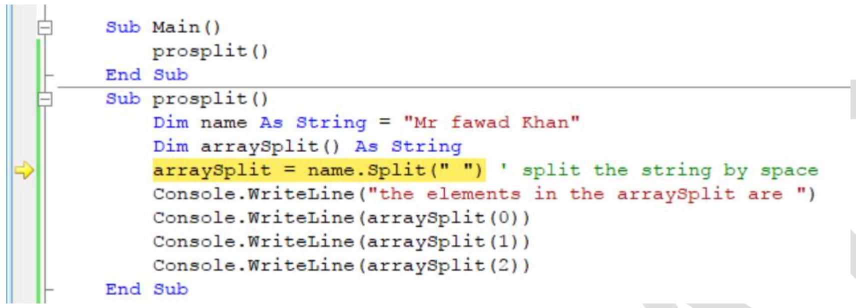
```
Module Module1

    Sub    Name 'Mass' is not declared.
        Mass = 56
    End Su
 correction suggestions)
```

(iii) For debugging

Debugging is the process of finding and reducing the number of bugs/defects/errors in computer programs.

- Single stepping

This technique is used by the programmer to see what happens when each line in a program is executed.

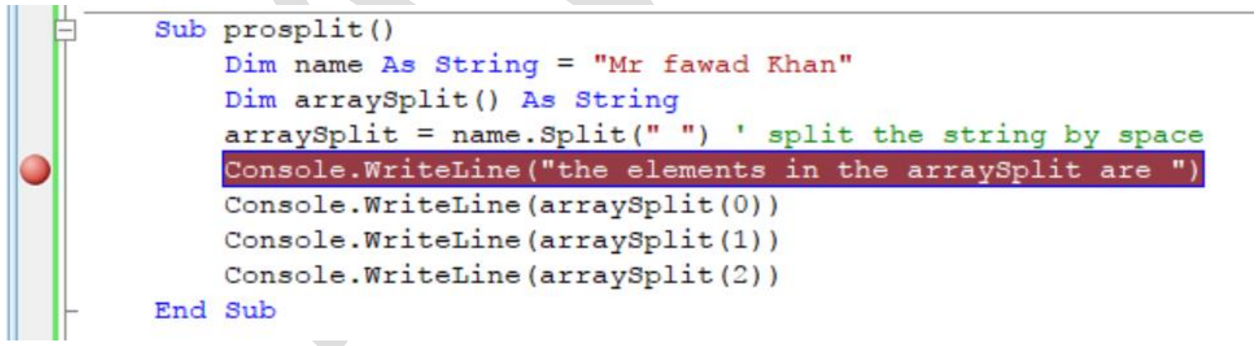


```
Sub Main()
    prosplit()
End Sub
Sub prosplit()
    Dim name As String = "Mr fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("the elements in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.WriteLine(arraySplit(2))
End Sub
```

- Breakpoints

This technique is used to arrange for a program to stop at a given instruction and display the values of the variables at this point, known as break pints. The program can then be continued or stopped by the programmer.

A programmer may step through a program either by setting break points or by stopping after the execution of each instruction. When a programmer sets a breakpoint, the program can then be run to this point. The programmer can then step through the following instructions one at a time, run the program to the next break point, or run to the end of the program.

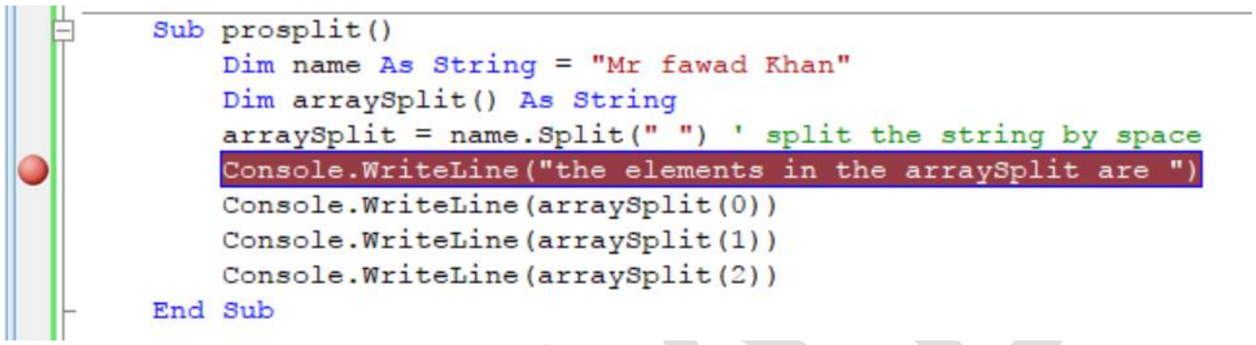


```
Sub prosplit()
    Dim name As String = "Mr fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("the elements in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.WriteLine(arraySplit(2))
End Sub
```

Fawad Khan 0321-6386013

- variables/expressions report window

A watch window (Visual Basic .NET calls this the "Immediate Window") can be used to display the values of variables and expressions. The window displays to the programmer the current value (and maybe the data type) of each variable. To obtain these values, the programmer simply types the name of the variable or the expression into the window and the information is displayed.



```
Sub prosplit()
    Dim name As String = "Mr fawad Khan"
    Dim arraySplit() As String
    arraySplit = name.Split(" ") ' split the string by space
    Console.WriteLine("the elements in the arraySplit are ")
    Console.WriteLine(arraySplit(0))
    Console.WriteLine(arraySplit(1))
    Console.WriteLine(arraySplit(2))
End Sub
```

13. Transferable skills

Program coding is a transferable skill. This implies that if a program is written in high level language in Pascal, you should be able to interpret that program and translate it into the programming language you have leant (in this caseVB).
Remember that the programming principles stay the same for any programming language, only the syntax changes!