'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 1.1: Setting up the Visual Basic Editor in Excel (VBE)

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' The Visual Basic Editor is a program within Excel that allows you to communicate with Excel. We will open it and start by

' setting it up so that working within it becomes easy and efficient.

' Open Excel and open a new workbook (Book1). On your keyboard press the "ALT" key (left of the space bar) and hold, strike the

' "F11"key (the F11 key is at the top of your keyboard). You now see the Visual Basic Editor. Again press "ALT/F11" and you are

' back into Excel. Use the "ALT/F11" key to go from Excel to the VBA and back.

' When you first open the VBE you will see is a window with the name Microsoft Visual Basic. If there are any open windows within

' the VBE click on the Xs to close them and see a gray rectangle filling up the bottom part of the screen.

' The Three Windows in the Visual Basic Editor

' To be efficient when working with the VBE there should always be 3 windows; the Project Window (1), the Code Window (2),

' and the Properties Window (3), arranged as in the image below. You can resize the windows by left-clicking where the red

' stars are, holding and moving sideways or up and down.

' In the exercise below we will setup the 3 windows of the VBE.

'Exercise 1 (Create your first macro and use it). Remember that you will perform this task only once as each time you will open

' the VBE it will remain setup.

'Step 1: Close all the windows that are open in the VBE

'Step 2: Go to the menu bar "View" and click "Project Explorer". If the project window already appears as a column on the left

' side of the screen there is nothing else that you have to do for now. If the project window appears in the middle of the gray

' area like above, right-click in the white space in the middle of the project window and check "Dockable". Then click on the top

' blue bar of the Project window, hold and drag it left until the cursor (white arrow) touches the middle of the left side of the

' screen. When you let go of the mouse button the end result would be the first major window of the VBE.

'Step 3: Move your cursor on the line separating the project window and the gray rectangle. When it turns to two small parallel lines

'and arrows click, hold and move the lines sideways. Resize the two windows as you want them.

'Step 4: Go back to the menu bar "View" and click "Properties Window". If the Properties window is already located below the Project

'window there is nothing left to do. If it shows like in the image above, right-click in the white space in the middle of the Properties

'window and check "Dockable". Then click on the top blue bar of the Properties window and drag it left and down until the cursor (white

'arrow) touches the center of the bottom of the Project window. When you let go of the mouse button the end result would be the second

'major window of the VBE.

'Step 5: Move your cursor on the line separating the project window and the properties window. When it turns to two small parallel lines

'and arrows click, hold and move the lines vertically. Resize the two windows as you want them.

'Step 6: To add the code window to the setup, you just have to double click on the name of a component in the Project window (Sheet1,

'Sheet2, Sheet3 or ThisWorkbook) and its code window appears within the gray rectangle. You can maximize any Code window by clicking on

'its "Maximize" button. The words "Option Explicit" might not be present in your Code window. We will address this issue later in the

'class on variables. You might also have a VBAProject named NAME.XLA or NAME.XLAM in the project window. Forget about this project for now.

'Step 6: Now go to Excel and close it. Re-open Excel, go to the VBE (ALT/F11) and you will see that the VBE setup persists. Congratulations,

'you are now ready to work in the Visual Basic Editor.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 1.2: Recording my first macro:

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Step 1: Click Record Macro

'Step 2: Type Macro Name "WorksheetFormat" and Click Ok

'Step 3: Change the format of the sheet

'Step 4: Click Macros and Run WorksheetFormat

Sub WorksheetFormat()

Dim DCELL As Excel.Range

Dim SRC\_WSHEET As Excel.Worksheet

Set SRC\_WSHEET = ActiveSheet

With SRC\_WSHEET

With .Cells

With .Font

.Name = "Courier New"

.Size = 10

.Strikethrough = False

.Superscript = False

.Subscript = False

.OutlineFont = False

.Shadow = False

.Underline = xlUnderlineStyleNone

.ColorIndex = xlAutomatic

.Italic = False

End With

.HorizontalAlignment = xlGeneral

.VerticalAlignment = xlCenter

.WrapText = False

.Orientation = 0

.AddIndent = False

.IndentLevel = 0

.ShrinkToFit = False

.ReadingOrder = xlContext

.MergeCells = False

.RowHeight = 15

.ColumnWidth = 15

End With

.Columns("A:A").ColumnWidth = 3

End With

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 1.3: Hello World in a UDF

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'An Excel User Defined Function (UDF) is probably the single most useful feature that remains unused by the

'majority of users. In this lecture we will provide a brief introduction with the obligatory "Hello World" program,

'followed by something only a little more complicated and much more useful.

'Excel UDFs allow users to write their own functions that operate in a similar way to the built in functions. It would for

'instance be possible, and in fact quite easy, to write a FUNC\_SUM() function that behaved exactly the same as the Excel Sum()

'function. Fortunately it is also possible to write functions providing functionality that is not built into Excel.

'UDFs are written in Excel's built-in programming language, Visual Basic for Applications (VBA), which is accessed through the

'Visual Basic Editor (VBE). To access the VBE:

'1. Open Excel with a new blank workbook, and save it as Hello.xls

'2. Press Alt-F11 to open the VBE. You should have a tree structure on the left displaying VBAProject(Hello.xls), with similar

'branches for any other workbooks or add-ins you may have open. Make sure that Hello.xls is selected, then select Insert-Module:

'3. A new "Modules" folder will apear under VBAProject(Hello.xls), containing one module named "Module1? by default. Double-click

'this module to open it, and a blank (or near blank) edit window will appear on the right.

'New functions may now be defined, which will be available whenever the Hello.xls workbook is open. For instance, in the edit window

'type the following code:

Function HELLO\_FUNC1()

HELLO\_FUNC1 = "Hello World"

End Function

'Now return to the spreadsheet and enter =HELLO\_FUNC1() in any cell. If you have followed the instructions correctly the cell will

'display "Hello World". So that is how to transfer information from VBA to the worksheet, using a UDF. Transferring data in the

'other direction is just as easy. Go back to the VBE and amend the function as shown below:

Function HELLO\_FUNC2(Optional ByVal YOUR\_NAME\_STR As String = "Nico")

HELLO\_FUNC2 = "Hello " & YOUR\_NAME\_STR & "."

End Function

'Now return to the workshheet and enter "Your Name" (say) in cell A1, and HELLO\_FUNC2(A1) in any other cell. The function will

'display "Hello Your Name.". Perhaps not the most useful functions in the world, but using the same principles we can write UDFs

'that perform complex functions otherwise not available in Excel (or any other spreadsheet).

Sub HELLO\_WORLD\_SUB()

MsgBox "Hello World!"

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 1.4: Procedures

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' A procedure is a generic term that refers to the two types of routines--“sub” routines and functions. Procedures are packaged pieces

' of code that perform specific operations. Visual Basic has hundreds of procedures that we can use in our programs to perform common

' tasks such as string manipulation, error checking and even a few mathematical and financial calculations. What’s more, we can create

' our own, user-defined procedures to accomplish specific tasks in our programs.

' When we call a procedure in our program, we are telling Visual Basic to execute the code associated with that procedure. Furthermore,

' we may specify input arguments, or parameters, that we want to pass into the procedure; that is, the value or values we want the routine

' to work on. When we define a procedure, we must specify four things: a name for the procedure; a comma-separated list of parameters the

' procedure accepts, if any; the data type of the return value, if any; and the procedure definition, which is the code that executes when

' the routine is called.

' The only difference between a subroutine and a function is that a function returns a value, aptly named the “return value” or “return

' argument,” whereas a subroutine does not. A return value gets sent back from the function to the code that called it. In general, functions

' are preferred to subroutines and they will be used whenever possible. The distinction between functions and subroutines will become clear

' when we use them later.

' We programmers use procedures to better organize code by breaking it up into smaller tasks. This makes the program code easier to read and

' debug. Also, procedures that perform common tasks can be called over and over from different sections of the program, reducing duplication

' of code and making the program easier to maintain. For example, if we wanted to calculate the mean returns for 100 stocks, we could write

' one function called Average(), and use it a hundred times over, rather than making the calculation in code for each of the 100 stocks.

' Let’s look at the code for an Average() function.

Public Function Average(ByVal m\_Return1 As Double, ByVal m\_Return2 As Double) As Double

Average = (m\_Return1 + m\_Return2) / 2

End Function

' Let’s review the four elements of a function. One, the name of this function is Average(). Two, this function accepts two input arguments,

' both of type Double, that will have the names m\_Return1 and m\_Return2 within the function definition. Three, this function returns a value

' of type Double. And, four, the function definition is the code between the function header, the Public Function Average line, and the

' function footer, End Function. We could call this function from somewhere else in our program this way:

Sub Print\_Average()

Dim m\_Average As Double

MsgBox Average(0.015, 0.005)

End Sub

' Here the value of m\_AverageReturn is set equal to the return value of the function Average(). Of course, this program prints out .01.

' One way to describe a function is to think about a black box that processes input, much like a mathematical function. In algebra we may use

' an expression like this: y = f(x1, x2, x3)

' f(x) is, of course, a function. This function has a name, f. The function accepts input arguments, namely x1, x2 and x3. The function named f

' has a return value to which y is then set equal. The definition of f exists somewhere else and is, say for example, f(x1, x2, x3) = 2x1 + 3x2 + 4x3.

' Functions in programming are no different.

' Both functions and subroutines can take input arguments. The input argument list, often called the parameters, has its own syntax that requires

' separate consideration.

' We can declare as many input arguments with their respective data types as are needed, provided we separate each parameter with a comma. The basic

' syntax is to specify a local name for the value and a data type. For example, here is a simple subroutine that prints out two numbers in the

' Immediate window (we will cover Debug.Print later in the course)

Private Sub PrintPrices(ByVal m\_Price1 As Double, ByVal m\_Price2 As Double)

Debug.Print m\_Price1, m\_Price2

End Sub

' We then call the PrintNumbers subroutine, we specify the parameters after the name as follows:

Private Sub Print\_Prices()

Call PrintPrices(45.23, 65.54)

End Sub

' In this example, the values 45.23 and 65.54 are passed to the variables m\_Price1 and m\_Price2. Within the subroutine definition, the values

' passed in will be known by the local names m\_Price1 and m\_Price2. Since this is a subroutine, there is no return value as was the case in

' the Average() function. The output of this simple program will be: 45.23 and 65.54