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

'Last Update: Sept 25, 2013

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

Option Explicit 'Requires that all variables to be declared explicitly.

Option Base 1

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

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

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

'Lesson 3. Procedures in VBA

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

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

'This lesson describes how to write your own worksheet functions in VBA. While Excel

'provides a plethora of built-in functions, especially so if you include functions in the

'Analysis Took Pack (in Excel 2007, the functions that used to be in the ATP are now

'native Excel functions) you may find it useful to create your own custom function for

'things that Excel cannot (easily) do with the built-in functions. While it takes longer

'for Excel to calculate a VBA function than it does to calculate a worksheet formula, all

'else being equal, the flexibility of VBA often makes a VBA function the better choice. The

'rest of this section assumes that you are familiar with the basics of VBA programming syntax.

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

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

'Lesson 3.1. Introduction

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

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

'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.

'For this class we will 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\_2N(), 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\_2N() function.

'Lets review the four elements of a function. One, the name of this function is AVERAGE\_2N(). 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 (RETURN\_VAL) of type Double. And, four, the function definition is the code between the function header, the Public

'Function AVERAGE\_2N line, and the function footer, End Function. We could call this function from somewhere else in our program

'this way: Debug.Print AVERAGE\_2N(0.015, 0.005)

Function AVERAGE\_2N(ByVal m\_Return1 As Double, ByVal m\_Return2 As Double) 'function

'AVERAGE\_2N takes as inputs two Double type variables, and returns a Double as its result.

Dim RETURN\_VAL As Double

On Error GoTo ERROR\_LABEL

RETURN\_VAL = (m\_Return1 + m\_Return2) / 2

'Here the value of RETURN\_VAL is set equal to the return value of the function AVERAGE\_2N(). 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.

AVERAGE\_2N = RETURN\_VAL

Exit Function

ERROR\_LABEL:

AVERAGE\_2N = CVErr(xlErrValue)

End Function

Sub TEST1\_AVERAGE\_2N() 'subroutine

Debug.Print AVERAGE\_2N(0.015, 0.005)

End Sub

'Once you have defined the UDF in a code module, you can call it from a worksheet cell with a formula like =AVERAGE\_2N(A1,B1)

Sub TEST2\_AVERAGE\_2N() 'subroutine

Dim i As Long

Dim DST\_RNG As Range

Dim FORMULA\_STR As String

Dim SRC\_WSHEET As Worksheet

Set SRC\_WSHEET = Worksheets("WSHEET\_TEST")

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

i = 1

'where A1 and A2 contain 0.015 and 0.005

.Cells(i + 0, 1).Value = 0.015

.Cells(i + 1, 1).Value = 0.005

FORMULA\_STR = "=AVERAGE\_2N(" & .Cells(i + 0, 1).Address(False, False) & "," & \_

.Cells(i + 1, 1).Address(False, False) & ")"

.Cells(i + 2, 1).Formula = FORMULA\_STR '=AVERAGE\_2N(A1,A2)

'$A$3 0.01 =AVERAGE\_2N(A1,A2)

Debug.Print .Cells(i + 2, 1).Address, .Cells(i + 2, 1), .Cells(i + 2, 1).Formula

Set DST\_RNG = Range(.Cells(i + 0, 1), .Cells(i + 2, 1))

DST\_RNG.Clear 'Remove all formulation from Cells A1:A3

Debug.Print DST\_RNG.Address(False, False) 'A1:A3

Debug.Print DST\_RNG.Address '$A$1:$A$3

End With

End Sub