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

'Last Update: Sept 25, 2013

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

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

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

'Lesson 3.5 Error Handling and Debugging In VBA

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

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

'3.5.1 Error Handling In VBA

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

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

'Error handling refers to the programming practice of anticipating and coding for error conditions that may arise when your program

'runs. Errors in general come in three flavors: compiler errors such as undeclared variables that prevent your code from compiling;

'user data entry error such as a user entering a negative value where only a positive number is acceptable; and run time errors,

'that occur when VBA cannot correctly execute a program statement. We will concern ourselves here only with run time errors.

'Typical run time errors include attempting to access a non-existent worksheet or workbook, or attempting to divide by zero. The

'example code in this article will use the division by zero error (Error 11) when we want to deliberately raise an error.

'Your application should make as many checks as possible during initialization to ensure that run time errors do not occur later.

'In Excel, this includes ensuring that required workbooks and worksheets are present and that required names are defined. The more

'checking you do before the real work of your application begins, the more stable your application will be. It is far better to

'detect potential error situations when your application starts up before data is change than to wait until later to encounter an

'error situation.

'If you have no error handling code and a run time error occurs, VBA will display its standard run time error dialog box. While this

'may be acceptable, even desirable, in a development environment, it is not acceptable to the end user in a production environment.

'The goal of well designed error handling code is to anticipate potential errors, and correct them at run time or to terminate code

'execution in a controlled, graceful method. Your goal should be to prevent unhandled errors from arising.

'A note on terminology: Throughout this section, the term procedure should be taken to mean a Sub, Function, or Property

'procedure, and the term exit statement should be taken to mean Exit Sub, Exit Function, or Exit Property. The term end statement

'should be taken to mean End Sub , End Function, End Property, or just End.

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

'3.5.1.1. The On Error Statement

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

'The heart of error handling in VBA is the On Error statement. This statement instructs VBA what to do when an run time error is

'encountered. The On Error statement takes three forms.

'1) On Error GoTo 0

'The first form, On Error Goto 0, is the default mode in VBA. This indicates that when a run time error occurs VBA should display

'its standard run time error message box, allowing you to enter the code in debug mode or to terminate the VBA program. When On

'Error Goto 0 is in effect, it is the same as having no enabled error handler. Any error will cause VBA to display its standard

'error message box.

'2) On Error Resume Next

'The second form, On Error Resume Next , is the most commonly used and misused form. It instructs to VBA to essentially ignore the

'error and resume execution on the next line of code. It is very important to remember that On Error Resume Next does not in any way

'"fix" the error. It simply instructs VBA to continue as if no error occured. However, the error may have side effects, such as

'uninitialized variables or objects set to Nothing. It is the responsibility of your code to test for an error condition and take

'appropriate action. You do this by testing the value of Err.Number and if it is not zero execute appropriate code. For example,

Sub Test\_On\_Error\_Resume\_Next\_and\_Error\_GoTo\_0()

On Error Resume Next

N = 1 / 0 ' cause an error

If Err.Number <> 0 Then: Debug.Print Err.Number

'This code attempts to assign the value 1 / 0 to the variable N. This is an illegal operations, so VBA will raise an error 11 --

'Division By Zero -- and because we have On Error Resume Next in effect, code continues to the If statement. This statement tests

'the value of Err.Number and assigns some other number to N.

On Error GoTo 0 'cause VBA to display its standard error message box.

N = 1 / 0

End Sub

'3) On Error Goto <label>:

'The third form On Error of is On Error Goto <label>:which tells VBA to transfer execution to the line following the specified

'line label. Whenever an error occurs, code execution immediately goes to the line following the line label. None of the code

'between the error and the label is executed, including any loop control statements.

Sub Test\_Error\_Goto()

On Error GoTo ErrHandler:

N = 1 / 0 ' cause an error

'

' more code

'

Exit Sub

ErrHandler:

Debug.Print " error handling code"

Resume Next

End Sub

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

'3.5.1.2. Enabled And Active Error Handlers

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

'An error handler is said to be enabled when an On Error statement is executed. Only one error handler is enabled at any given

'time, and VBA will behave according to the enabled error handler. An active error handler is the code that executes when an

'error occurs and execution is transferred to another location via a On Error Goto <label>: statement.

Sub Test\_Enabled\_And\_Active\_Error\_Handlers\_Func() 'subroutine

Debug.Print Enabled\_And\_Active\_Error\_Handlers\_Func("x", 0.005)

End Sub

Function Enabled\_And\_Active\_Error\_Handlers\_Func(ByVal m\_Return1 As Variant, ByVal m\_Return2 As Variant)

Dim RETURN\_VAL As Double

On Error GoTo ERROR\_LABEL

Debug.Print "Return Val Line:"

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

Enabled\_And\_Active\_Error\_Handlers\_Func = RETURN\_VAL

Exit Function

ERROR\_LABEL:

Debug.Print "ERROR LABEL LINE"

Enabled\_And\_Active\_Error\_Handlers\_Func = CVErr(xlErrValue)

End Function

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

'3.5.1.3. Error Handling Blocks And On Error Goto

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

'An error handling block, also called an error handler, is a section of code to which execution is tranferred via a On Error

'Goto <label>: statement. This code should be designed either to fix the problem and resume execution in the main code block or

'to terminate execution of the procedure. You can't use to the On Error Goto <label>: statement merely skip over lines. For

'example, the following code will not work properly:

Sub Test\_Error\_Handling\_Blocks\_And\_On\_Error\_GoTo2()

On Error GoTo Err1:

Debug.Print 1 / 0

' more code

Err1:

On Error GoTo Err2:

Debug.Print 1 / 0

' more code

Err2:

End Sub

'When the first error is raised, execution transfers to the line following Err1:. The error hander is still active when the

'second error occurs, and therefore the second error is not trapped by the On Error statement.

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

'3.5.1.4. The Resume Statement

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

'The Resume statement instructs VBA to resume execution at a specified point in the code. You can use Resume only in an

'error handling block; any other use will cause an error. Moreover, Resume is the only way, aside from exiting the procedure,

'to get out of an error handling block. Do not use the Goto statement to direct code execution out of an error handling

'block. Doing so will cause strange problems with the error handlers.

'The Resume statement takes three syntactic form:

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

'1) Resume

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

'Used alone, Resume causes execution to resume at the line of code that caused the error. In this case you must ensure that

'your error handling block fixed the problem that caused the initial error. Otherwise, your code will enter an endless loop,

'jumping between the line of code that caused the error and the error handling block. The following code attempts to activate

'a worksheet that does not exist. This causes an error (9 - Subscript Out Of Range), and the code jumps to the error handling

'block which creates the sheet, correcting the problem, and resumes execution at the line of code that caused the error.

Sub Test\_Resume()

On Error GoTo ErrHandler:

Worksheets("NewSheet").Activate

Worksheets("NewSheet").Delete

Exit Sub

ErrHandler:

If Err.Number = 9 Then ' sheet does not exist, so create it

Worksheets.Add.Name = "NewSheet"

' What happens if I insert here Worksheets("NewSheet").Delete and why?

Resume ' go back to the line of code that caused the problem :Worksheets("NewSheet").Activate

End If

End Sub

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

'2) Resume Next

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

'The second form of Resume is Resume Next . This causes code execution to resume at the line immediately following the line

'which caused the error. The following code causes an error (11 - Division By Zero) when attempting to set the value of N. The

'error handling block assigns 1 to the variable N, and then causes execution to resume at the statement after the statement

'that caused the error.

Sub Test\_Resume\_Next()

On Error GoTo ErrHandler:

N = 1 / 0

Debug.Print N

Exit Sub

ErrHandler:

N = 1

Resume Next ' go back to the line following the error

End Sub

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

'3) Resume <label>

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

'The third form of Resume is Resume <label>: . This causes code execution to resume at a line label. This allows you to skip a

'Lesson of code if an error occurs. For example,

Sub Test\_Resume\_Label()

On Error GoTo ErrHandler:

N = 1 / 0

Debug.Print "Code that is skipped if an error occurs"

Label1:

'

Debug.Print " more code to execute"

'

Exit Sub

ErrHandler:

' go back to the line at Label1:

Resume Label1:

End Sub

'All forms of the Resume clear or reset the Err object: Err.Clear so Err.number = 0!

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

'3.5.1.5. Error Handling With Multiple Procedures

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

'Every procedure need not have a error code. When an error occurs, VBA uses the last On Error statement to direct code execution.

'If the code causing the error is in a procedure with an On Error statement, error handling is as described in the above section.

'However, if the procedure in which the error occurs does not have an error handler, VBA looks backwards through the procedure

'calls which lead to the erroneous code. For example if procedure A calls B and B calls C, and A is the only procedure with an

'error handler, if an error occurs in procedure C, code execution is immediately transferred to the error handler in procedure

'A, skipping the remaining code in B.

Sub TEST\_A()

On Error GoTo 1983

Call TEST\_B

Exit Sub

1983:

Debug.Print "A"

End Sub

Function TEST\_B()

Call TEST\_C

End Function

Function TEST\_C()

Debug.Print 1 / 0

'Debug.Print 1 / 1

End Function

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

'3.5.1.6. A Note Of Caution

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

'It is tempting to deal with errors by placing an On Error Resume Next statement at the top of the procedure in order to get the

'code to run without raising an error. This is very bad coding practice. Remember that using On Error Resume Next does not

'fix errors. It merely ignores them.

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

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

'3.5.2 Debugging In VBA

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

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

'This section covered various alternative debugging techniques in VBA.

'The steps in debugging an application are:

'1) Find the ways in which it does not work as intended (and for all but the most trivial application there will be some).

'2) Find the source of the errors

'3) Fix them

'4) Return to 1 until there are no more errors to be found

'For any application, but particularly for modeling applications, proper attention to stage 1 is of key importance, but

'its on stage 2 that tends to get all the attention. Where possible, we would like to set up an on-spreadsheet (no VBA code)

'prototype of any application, and use this in the debugging process. The intermediate results of VBA code can then be

'compared with the values on the spreadsheet, allowing detailed comparison and detection of errors with minimum effort.

'This section will describe the function GROWTH\_SCALER\_FUNC to show the tools and techniques that I use to debug in VBA.

Function GROWTH\_SCALER\_FUNC(ByVal COMPANY\_REVENUES\_VAL As Double, \_

ByVal SECTOR\_REVENUES\_VAL As Double, \_

ByVal COMPANY\_GROWTH\_RATE\_VAL As Double, \_

ByVal SECTOR\_GROWTH\_RATE\_VAL As Double, \_

ByVal GROWTH\_PERIODS\_VAL As Long)

'REVENUES\_VAL: Revenues/earnings.. in base year

'GROWTH\_RATE\_VAL: Expected growth rate

'GROWTH\_PERIODS\_VAL: Number of years of growth

'As companies get larger, it becomes more difficult to sustain high percentage growth rates in revenues for two

'reasons. The first is that the same percentage growth rate will require larger and larger absolute changes in

'revenues each period and thus will be more difficult to deliver. The second is that a company's success will

'attract the attention of other firms; the resulting competition will act as a damper on growth.

'Later in the class we will create an application that will allow us to visualize how growth rates for Google has

'declined over time. In fact, we will see revenue growth (in dollar and percentage terms) in real-time and while

'growth rates remain healthy, they have declined over the last decade. Second, the very fact that you can name great

'growth companies is an indication that you are talking about the exceptions rather than the rule. Could the company

'you are looking at right now be the next exceptional company? Sure, but do you want to value your company to be the

'exception? I would not, since pricing your company for perfection will open you up to mostly negative surprises in the

'future.

'Given that growth could decelerate quickly at companies, how do we explain valuations where analysts use 50% compounded

'growth rates for 10 to 15 years or longer? I think the problem lies in the percentage illusion, where analysts feel that

'their growth assumption is not changing if they keep the growth rate unchanged. However, delivering a 25% growth rate is

'far easier in year 1 than the same firm delivering a 25% growth rate in year 9. The best way to introduce some realism in

'growth rates is to convert the percentage growth rate in revenues into dollar changes in revenues and consider what the

'company will have to do in terms of operations to deliver that change. When valuing a retail company, for instance, computing

'that the company will have to open 300 new stores to deliver a 25% growth rate in year 10 (as opposed to 30 in year 1) may

'quickly lead to a reassessment of that growth rate.

Dim i As Long

Dim j As Long

Dim k As Long

Dim NCOLUMNS As Long

Dim HEADINGS\_STR As String

Dim TEMP\_MATRIX As Variant

On Error GoTo ERROR\_LABEL

HEADINGS\_STR = "Period,Revenues (Firm),Annual Change in Revenues (Firm),Revenues (Sector),Market Share,Flag,"

NCOLUMNS = 6

ReDim TEMP\_MATRIX(0 To GROWTH\_PERIODS\_VAL, 1 To NCOLUMNS)

i = 1

For k = 1 To NCOLUMNS

j = InStr(i, HEADINGS\_STR, ",") 'see Lesson 3.8.4: String Position

TEMP\_MATRIX(0, k) = Mid(HEADINGS\_STR, i, j - i)

' Debug.Print TEMP\_MATRIX(0, k)

i = j + 1

Next k

i = 1: j = 1

TEMP\_MATRIX(i, 1) = i

TEMP\_MATRIX(i, 2) = COMPANY\_REVENUES\_VAL \* (1 + COMPANY\_GROWTH\_RATE\_VAL) ^ TEMP\_MATRIX(i, 1)

TEMP\_MATRIX(i, 3) = TEMP\_MATRIX(i, 2) - COMPANY\_REVENUES\_VAL

TEMP\_MATRIX(i, 4) = SECTOR\_REVENUES\_VAL \* (1 + SECTOR\_GROWTH\_RATE\_VAL) ^ TEMP\_MATRIX(i, 1)

TEMP\_MATRIX(i, 5) = TEMP\_MATRIX(i, 2) / TEMP\_MATRIX(i, 4)

TEMP\_MATRIX(i, 6) = IIf(TEMP\_MATRIX(i, 5) > 100, "Check", "")

For i = 2 To GROWTH\_PERIODS\_VAL

TEMP\_MATRIX(i, 1) = i

TEMP\_MATRIX(i, 2) = TEMP\_MATRIX(i - 1, 2) \* (1 + COMPANY\_GROWTH\_RATE\_VAL)

TEMP\_MATRIX(i, 3) = TEMP\_MATRIX(i, 2) - TEMP\_MATRIX(i - 1, 2)

TEMP\_MATRIX(i, 4) = TEMP\_MATRIX(i - 1, 4) \* (1 + SECTOR\_GROWTH\_RATE\_VAL)

TEMP\_MATRIX(i, 5) = TEMP\_MATRIX(i, 2) / TEMP\_MATRIX(i, 4)

TEMP\_MATRIX(i, 6) = IIf(TEMP\_MATRIX(i, 5) > 1, "Check", "")

Next i

GROWTH\_SCALER\_FUNC = TEMP\_MATRIX

Exit Function

ERROR\_LABEL:

GROWTH\_SCALER\_FUNC = Err.Number

End Function

'The subroutine GROWTH\_SCALER\_FUNC1 has been written so that it can also be called directly from the spreadsheet (macro), which greatly aids the

'debugging process.

Sub TEST\_GROWTH\_SCALER\_FUNC1()

Dim i As Long

Dim j 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

.HorizontalAlignment = xlCenter

.VerticalAlignment = xlCenter

End With

.Cells(3, 3) = "Your Company"

.Cells(3, 4) = "Potential market (sector)"

.Cells(4, 2) = "Revenues in base year:"

.Cells(4, 3) = 1360

.Cells(4, 4) = 10000

With Union(.Cells(4, 3), .Cells(4, 4))

.NumberFormat = "$#,##0"

End With

.Cells(5, 2) = "Expected growth rate:"

.Cells(5, 3) = 0.2

.Cells(5, 4) = 0.03

With Union(.Cells(5, 3), .Cells(5, 4))

.Style = "Percent"

End With

.Cells(6, 2) = "Number of years of growth:"

With .Cells(6, 3)

.Value = 25

.NumberFormat = "0"

End With

For j = 0 To 5

With .Cells(3, 6 + j)

.Formula = "=" & .Offset(0, 7).Address(False, False)

End With

Next j

For i = 1 To 25

.Cells(3 + i, 6).Value = i 'Year

With .Cells(3 + i, 7)

If i > 1 Then

.Formula = "=IF(" & .Offset(0, -1).Address(False, False) & ">$C$6,," & .Offset(-1, 0).Address(False, False) & "\*(1+$C$5))"

Else

.Formula = "=IF(" & .Offset(0, -1).Address(False, False) & ">$C$6,,$C$4\*(1+$C$5)^" & .Offset(0, -1).Address(False, False) & ")"

End If

.Offset(0, 13).Formula = "=" & .Address(False, False) & "-" & .Offset(0, 7).Address(False, False)

End With

With .Cells(3 + i, 8) 'Revenues (company)

If i > 1 Then

.Formula = "=" & .Offset(0, -1).Address(False, False) & "-" & .Offset(-1, -1).Address(False, False) & ""

Else

.Formula = "=" & .Offset(0, -1).Address(False, False) & "-C4"

End If

.Offset(0, 13).Formula = "=" & .Address(False, False) & "-" & .Offset(0, 7).Address(False, False)

End With

With .Cells(3 + i, 9)

If i > 1 Then

.Formula = "=IF(" & .Offset(0, -3).Address(False, False) & ">$C$6,," & .Offset(-1, 0).Address(False, False) & "\*(1+$D$5))"

Else

.Formula = "=IF(" & .Offset(0, -3).Address(False, False) & ">$C$6,,$D$4\*(1+$D$5)^" & .Offset(0, -3).Address(False, False) & ")"

End If

.Offset(0, 13).Formula = "=" & .Address(False, False) & "-" & .Offset(0, 7).Address(False, False)

End With

With .Cells(3 + i, 10)

.Formula = "=IF(" & .Offset(0, -4).Address(False, False) & ">$C$6,," & .Offset(0, -3).Address(False, False) & "/" & .Offset(0, -1).Address(False, False) & ")"

.Offset(0, 13).Formula = "=" & .Address(False, False) & "-" & .Offset(0, 7).Address(False, False)

End With

With .Cells(3 + i, 11)

.Formula = "=IF(" & .Offset(0, -1).Address(False, False) & ">100%," & """" & "Check" & \_

"""" & "," & """" & " " & """" & ")"

End With

Next i

With Range(.Cells(3 + 1, 6), .Cells(3 + 25, 10))

Union(.Offset.Columns(2), .Columns(3), .Columns(4)).NumberFormat = "$#,##0"

.Columns(5).Style = "Percent"

End With

With Range(.Cells(3 + 1, 6), .Cells(3 + 25, 10)).Offset(0, 7)

Union(.Offset.Columns(2), .Columns(3), .Columns(4)).NumberFormat = "$#,##0"

.Columns(5).Style = "Percent"

End With

With .Range("M3:R28") 'Selecting cells containing a call to the function

.ClearContents

.FormulaArray = "=GROWTH\_SCALER\_FUNC(C4,D4,C5,D5,C6)"

' Same as Pressing F2 to enter edit mode & Ctrl-Shift-Enter to exit edit mode, causing the function to recalculate.

' Ctrl-Shift-Enter is required because the function GROWTH\_SCALER\_FUNC returns an array. Ordinary functions returning a single

' value are entered just with the Enter key)

End With

End With

End Sub

'Let's start by adding a break point in the code window, by clicking in the left margin, adjacent to a line

'of executable code (i.e. not a blank line, comment or Dim statement).

Sub TEST\_GROWTH\_SCALER\_FUNC2()

'The function has then been initiated from the spreadsheet by: Sub TEST\_GROWTH\_SCALER\_FUNC2(). This subroutine is required because

'the procedures for running a Sub procedure from the VBE code window do not work with a function.

'When the function is recalculated the code runs as normal until it reaches the break point, where it stops with the line of code

'highlighted in yellow. At this point the code window displays the value of any variable if you place the

'cursor over the variable name, and also lists all the variables in the current routine in the locals window (Menu-View-Locals Window),

'to be seen underneath the code window. The GROWTH\_SCALER\_FUNC array has been expanded to display the value of each item. There are

'also other methods to retrieve the current value of any code variable, which will be examined shortly, but they all share two major

'inconveniences: (1) Viewing data in a large array is slow and cumbersome; (2) the data is not available for use on the spreadsheet,

'making comparisons with spreadsheet data difficult.

'Both of these problems are avoided by writing the data back to the spreadsheet in the form of a UDF array (Run TEST\_GROWTH\_SCALER\_FUNC1).

'Cells F4:K28 show the spreadsheet calculated values, and cells M4:R28 contain the UDF values, which can quickly be checked to be

'identical (cells S4:V28).

'In cases where the called function requires data calculated within the VBA routine the effective approach is to insert the

'the return code where the flag is set:

'Having checked the results of the GROWTH\_SCALER\_FUNC the options are:

'Press F5 to continue running the code following the break point, either until completion, or if the break point was in a

'loop until execution returns back to the break point.

'Press F8 to step line by line through the code.

'When the code reaches a sub-routine press F8 to step through the subroutine or Shift-F8

'to run the sub-routine code and stop at the next line in the main code.

'To run through to a subsequent line in the code, either enter a new break point and press F5, or select the line and select Run to

'cursor from the Debug menu or the right-click menu

'To run the code from a line other than the following line, select the line (which may be either before or after the break line) and

'select Set next statement from the Debug menu or the right-click menu.

Dim i As Long

Dim TEMP\_MATRIX As Variant

TEMP\_MATRIX = GROWTH\_SCALER\_FUNC(1360, 10000, 0.2, 0.03, 25)

For i = LBound(TEMP\_MATRIX, 1) To UBound(TEMP\_MATRIX, 1)

Debug.Print TEMP\_MATRIX(i, 1), Format(TEMP\_MATRIX(i, 2), "0"), Format(TEMP\_MATRIX(i, 3), "0"), Format(TEMP\_MATRIX(i, 4), "0"), \_

Format(TEMP\_MATRIX(i, 5), "0%"), TEMP\_MATRIX(i, 6)

Next i

End Sub

'The Immediate Window allows interaction with the code while it is running. In the VBE select View-Immediate Window,

'or Ctrl-G to display the window. Let's first place a break point in the code (click in the left margin, or press F9 with

'the cursor on the required line) at the line: For i = LBound(TEMP\_MATRIX, 1) To UBound(TEMP\_MATRIX, 1). Lets now run

'the sub: TEST\_GROWTH\_SCALER\_FUNC2(). Finally, lets type the following command in the immediate window and press enter:

'For i = 1 to 6: ?TEMP\_MATRIX(0,i): Next i

'Each command is followed by a : so that the array members may be entered without pressing the Enter key. The immediate

'window performs the command on the current line as soon as the Enter key is pressed, but a number of commands may be concatenated

'with the colon symbol. The commands may be entered directly or copied and pasted from the spreadsheet or a text editor.

'If the routine is a UDF it may be initiated by selecting the function on the spreadsheet, pressing F2 (Edit), and then pressing

'Enter (or Ctrl-Shift-Enter for an array function).

'Unlike the Locals Window, the values in the Immediate Window could be copied and pasted to the spreadsheet for easy comparison with

'calculated values. Please notice that the immediate Window results remain available after the routine has completed.

'Values may also be displayed using the statement Debug.Print, rather than ?:

'for i = 1 to 6: Debug.Print TEMP\_MATRIX(0,i): Next i

'Notes on the Immediate Window with Debug.Print

'It is also possible to enter more complex commands in the Immediate Window. In the example below the array values are

'subtracted from the values calculated on the spreadsheet, to check that the result is equal to zero, or very close to it.

'The full line is (Make sure to run first the Sub TEST\_GROWTH\_SCALER\_FUNC2):

'For i = 1 To 6: ?Range("F3").Cells(1,i).Value = TEMP\_MATRIX(0, i): Next i

'The Debug.print command can also be used within the code to send data to the immediate window:

'Below For i = LBound(TEMP\_MATRIX, 1) To UBound(TEMP\_MATRIX, 1) see the following line:

'Debug.Print TEMP\_MATRIX(i, 1), Format(TEMP\_MATRIX(i, 2), "0"), .....

'Now run again the TEST\_GROWTH\_SCALER\_FUNC2() and check the Immediate Window

'The other main VBE window used in debugging is the Watch Window, allowing the value of specific variables to be

'monitored. The Watch Window is opened from the View menu, and is initially blank. A watch point is added through

'the Debug-Add Watch menu, which opens the Add Watch dialog, having the following features:

'1. The expression to be watched is entered, which may be either a single variable, or a more complex expression.

'2. The applicable Procedure and Module for the watch point are assumed to be those at the current location in the VBE edit

'window, but these can be changed.

'3. The Watch Type may be set to one of either watch expression, break when value is true, or break when value changes

'Alternatively a Watch Value Point may be inserted through the Debug Quick Watch menu (or Shift-F9), with the cursor

'located at the array (exact location: )in the Edit window. This will insert a watch point for the

'selected array, which may then be edited by right-clicking on the variable line in the Watches window. A third way to

'add (or remove) watch points is through the right click menu in the Watches window. In this case the watch point has been

'specified as watch only, and a break point has been inserted below the point in the code where array values are evaluated.

'As with the Locals window, the array initially appears as a single summary line, and must be expanded to view the values for each item.

'Now lets place a break point in the code (click in the left margin, or press F9 with

'the cursor on the required line) at the line: For i = LBound(TEMP\_MATRIX, 1) To UBound(TEMP\_MATRIX, 1)

'Then run the TEST\_GROWTH\_SCALER\_FUNC2() and check the Watches window.

'A watch point may also be specified as specified array position, rather than a complete array.

'Note that in this case, there are no specified break points, but when the function is initiated it run through until the

'first watch point had been evaluated, where execution is stopped because this point had been specified as break when

'value changes. The next two points are then evaluated by using the F8 key to step through the code.

Sub TEST\_WATCH\_POINT\_FUNC()

Dim A As Double

Dim B As Double

Dim C As Double

'Press Shift-F9 with the cursor located at the variablt b = a + 8 in the Edit window. This will insert a watch point for the

'selected variable, which may then be edited by right-clicking on the variable line in the Watches window. Place a break point

'in the line c = b \* 4. Then run TEST\_WATCH\_POINT\_FUNC and take a look at the Watches window.

A = 4 + 5

B = A + 8

C = B \* 4

End Sub

'One last thing, if you hover over a variable in VBA (while in debug mode) you can see the value of that variable. This allows you to see

'the values that are being passed in and out of variables very useful. But remember that it can only hold 77 characters

'(including the variable name). This basically means if the value in your variable is too long, it gets cut off.

Sub HOVER\_DEBUG\_MODE()

Dim HEADING\_STR As String

HEADING\_STR = "Hovering over the variable DATA\_ARR(1) allows you to see the string that represents the file name, " & \_

"but that string gets cut off. Although there is no way to tell Excel to show more than the first 77 characters, " & \_

"there is a trick you can use to see then last 77 characters. Simply hold down the CTRL key while you hover. As you " & \_

"can see, this allows you to see the last part of the file name."

Debug.Print HEADING\_STR

End Sub