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

'Last Update: Oct 8th, 2013

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

Option Explicit

Option Base 1

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

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

'Lesson 3.3 ByRef and ByVal

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

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

'The use of inappropriate techniques for passing data between sub-routines and modules is probably one of the most frequent causes

'of error in VBA programs, but many text books provide little guidance on this subject, and what there is, is often too general to

'be of much use. This lesson will focus on the basics, and will look at some details affecting passing of arrays and ranges in lessons

'4 and 5.

'Let's take a look at the important distinction between ByRef and ByVal, the two methods for passing input arguments to functions.

'Passing an input argument ByVal means that the original variable, which is being passed as an input argument, will not be changed

'by the function definition. That is to say, the procedure makes a copy of the value and performs the operations within the procedure

'definition on the copy, as opposed to the original variable. This is demonstrated by the following example.

'Here we are: byRef: before (52.78) and after (53.78) because there's byVal there wasn't an absolute reference.

'variable a is an integer (1), second variable is an integer (2), then I call a sub - and I pass a & b, and the sub will be adding physical a + 1. If a=1, what will b equal if it is byVal...b would equal 2.

'default value for an integer is 0

'when i have two functions & one is passing a variable to the other one, and one returning the value (byVal - variable receiver is equal to variable sender)

' byRef - there is no assignment to variables - it will just be sent to the receiver sub.. it returns that value by the change. (references it)

Private Sub ByVal\_Increment(ByVal m\_Val As Double)

Debug.Print ("Increment subroutine was passed: " & m\_Val)

m\_Val = m\_Val + 1

Debug.Print ("New value is: " & m\_Val)

End Sub

'This program outputs:

'Increment function was passed: 52.78 New value is: 53.78

'Stock price after increment is: 52.78

'The m\_StockPrice variable is unaffected by the addition within Increment subroutine. This is because only the value of m\_StockPrice

'has been passed to m\_Val. m\_Val is a completely separate variable.

Sub Test\_ByVal\_Increment()

Dim m\_StockPx As Double

m\_StockPx = 52.78

Call ByVal\_Increment(m\_StockPx)

Debug.Print ("Stock price after increment is: " & m\_StockPx)

End Sub

'Now try this example again, but change ByVal in Increment() to ByRef as follows:

Private Sub ByRef\_Increment(ByRef m\_Val As Double)

Debug.Print ("Increment subroutine was passed: " & m\_Val)

m\_Val = m\_Val + 1

Debug.Print ("New value is: " & m\_Val)

End Sub

'Byref is the default method for passing values into functions.

'This time the output is:

'Increment was passed: 52.78

'New value is: 53.78

'Stock price after increment is: 53.78

'Here, a reference to the location of m\_StockPx in memory is passed to m\_Val, not the value of m\_StockPx. Therefore, as far as the computer

'is concerned, both m\_Val and m\_StockPx are referring to the same physical space, or location, in memory. Hence, when m\_Val is incremented,

'the value of m\_StockPx changes, since they are both the same variable.

Sub Test\_ByRef\_Increment()

Dim m\_StockPx As Double

m\_StockPx = 52.78

Call ByRef\_Increment(m\_StockPx)

Debug.Print ("Stock price after increment is: " & m\_StockPx)

End Sub

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

'Example 1: Worksheet "WSHEET\_BYVAL\_BYREF": Passing Variables between routines

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

Sub Passing\_variables\_between\_routines()

'Return\_Val = Function\_Name(Var1, Var2, Var3)

'Sub\_Name Var1, Var2, Var3

'Call Sub\_Name(Var1, Var2, Var3)

'Sub\_Name(Var1), (Var2), (Var3)

'Call Sub\_Name((Var1), (Var2), (Var3))

'These are used to call:

'Function Function\_Name(Var1 as Double, ByRef Var2 as Double, ByVal Var3 as Double) as Double

'Sub SubName(Var1 as Double, ByRef Var2 as Double, ByVal Var3 as Double)

'Variables may be passed either "by reference" (ByRef) or "by value" (ByVal).

'Passing by reference passes the memory address of the variable, so that if the value of the variable is

'changed by the called subroutine, that change is seen by the calling routine, which continues to use the

'same memory address as the location of this variable.

'In contrast a variable passed by value may be changed by the called routine without affecting the value of

'the variable in the calling routine.

'The routine used to produce this output performed the following tasks:

'1. Read the values in cells B3 to D3 and assign to the variables Var1 to Var3

'2. Pass the variables Var1 to Var3 to PassFunc and add 1 to each value

'3. Copy the resulting values to cells B4 to D4

'4. From the calling routine, copy the value of Var1 to Var3 to cells E4 to G4

'5. Copy the function return value to cell H4

'6. Repeat steps 2 to 4, but passing the variables to subroutines using the syntax styles of examples 2 to 5 above

Dim i As Long

Dim VAR1 As Double

Dim VAR2 As Double

Dim VAR3 As Double

Dim TEMP\_ARR As Variant

Dim DST\_RNG As Range

Dim FORMAT\_RNG As Range

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

'Setting up the tables in the WSHEET\_TEST

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

With Worksheets("WSHEET\_TEST")

.Select

With .Cells

.Clear

.ColumnWidth = 10

End With

Set DST\_RNG = [A1] 'Same as Range("A1")

End With

With DST\_RNG

.Value = "Original data"

With .Offset(3, 0)

.Value = "Original data"

For i = 1 To 4

.Offset(i, 0).Value = "Sub" & CStr(i)

Next i

End With

Set FORMAT\_RNG = Range(.Cells(1, 2), .Cells(1, 4))

With FORMAT\_RNG

GoSub FORMAT\_LINE

.ColumnWidth = 15

.Value = Array(10, 20, 30) 'Inputs Value

On Error Resume Next 'Will not work in Mac!

With .Font

.ThemeColor = xlThemeColorAccent1

.TintAndShade = 0

End With

On Error GoTo 0

End With

.Cells(2, 2).Value = "Subroutine values"

.Cells(2, 5).Value = "Calling routine values"

TEMP\_ARR = Array("Var1", "Var2", "Var3")

Set FORMAT\_RNG = Range(.Cells(3, 2), .Cells(3, 4)): GoSub FORMAT\_LINE

With FORMAT\_RNG

.ColumnWidth = 15

.Value = TEMP\_ARR

End With

Set FORMAT\_RNG = FORMAT\_RNG.Offset(0, 3): GoSub FORMAT\_LINE

With FORMAT\_RNG

.ColumnWidth = 15

.Value = TEMP\_ARR

End With

Set FORMAT\_RNG = FORMAT\_RNG.Offset(0, 3).Cells(1, 1): GoSub FORMAT\_LINE

With FORMAT\_RNG

.Value = "ReturnVal"

.ColumnWidth = 15

End With

End With

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

Dim RETURNVAL As Long 'Check Type of Function PassFunc!

[B1] = 10

[c1] = 20

[d1] = 30

VAR1 = [B1]

VAR2 = [c1]

VAR3 = [d1]

RETURNVAL = PassFunc(VAR1, VAR2, VAR3)

'ByVal VAR3 means means a copy of V3 will be passed to PassFunc.

'ByRef VAR2 means a pointer to the variable's memory address is passed to PassFunc.

'VAR1 = VAR1 + 1

'VAR2 = VAR2 + 1

'VAR3 = VAR3 + 1

'[b4] = VAR1

'[C4] = VAR2

'[D4] = VAR3

'PassFunc = VAR3

'Q: what will be the return value (if passfunc=VAR3)

'A: the value of 30 -

'Q: after passfunc: 30 - whatever happens in the passfun it will return the same value because it's byVal.

'Q: variable2 will equal 21 - because it's byRef so it stays

'2 dimensions = i,1

'

'if you have 8Billion pieces of data and you use byVal it will make a copy so 16B pieces (computer will crash) - but if you make it byRef it will reference back to the original set of data.

[e4] = VAR1

[f4] = VAR2

[g4] = VAR3

[h4] = RETURNVAL

Sub1 VAR1, VAR2, VAR3 'same as Call Sub1(VAR1, VAR2, VAR3)

[e5] = VAR1: [F5] = VAR2: [g5] = VAR3

'Same as:

'[e5] = VAR1

'[f5] = VAR2

'[g5] = VAR3

Call Sub2(VAR1, VAR2, VAR3)

[e6] = VAR1: [f6] = VAR2: [g6] = VAR3

Sub3 (VAR1), (VAR2), (VAR3)

[e7] = VAR1: [f7] = VAR2: [g7] = VAR3

Call Sub4((VAR1), (VAR2), (VAR3))

[e8] = VAR1: [f8] = VAR2: [g8] = VAR3

'The results of this excercise illustrate the following points:

'Passing by reference is the default; the results for Var1 (method of passing not specified) and Var2 (passed by

'reference) are therefore always the same.

'Var3 is passed by value, so the increment in its value by the called subroutine is not passed back to the calling routine.

'Function return values (in this case Var3) do reflect the increment in value.

'When the variables passed to a subroutine are enclosed in brackets these variables are passed by value, even if by reference

'is specified in the called routine.

'Thus in Sub3 and Sub4 the increment in Var1 and Var2 is not passed back to the calling routine, even though these variables

'were passed by reference. This behaviour is particularly confusing if a single variable is passed to a subroutine with the

'variable enclosed in brackets, since this will be passed by reference by default if the Call statement is used, but always

'by value if the subroutine is called without the call statement.

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

Exit Sub

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

FORMAT\_LINE:

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

On Error Resume Next 'May not work in Mac

With FORMAT\_RNG 'Format borders

.Borders(xlDiagonalDown).LineStyle = xlNone

.Borders(xlDiagonalUp).LineStyle = xlNone

With .Borders(xlEdgeLeft)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeTop)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeBottom)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeRight)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

.Borders(xlInsideVertical).LineStyle = xlNone

.Borders(xlInsideHorizontal).LineStyle = xlNone

End With

On Error GoTo 0

Return

End Sub

Private Function PassFunc(VAR1 As Double, ByRef VAR2 As Double, ByVal VAR3 As Double) As Long

'When ByRef and when Byval?

'In some circumstances the logic of the programme dictates the method of passing variables, so if any changes to

'the variable are required to be passed back to the calling routine (and it is not a function return value) then it must

'be passed by reference, and if the variables in the calling routine should not be affected by changes in the called

'routine, then they must be passed by value. But what is the best practice when the passed variable is not changed in

'the called routine, or is not used again in the calling routine? There are arguments for different procedures, but my

'preference is:

'If the program logic is not affected by the method of passing, then leave it as unspecified (and hence by reference is

'adopted as the default -> VAR1 as Double).

'If the program logic demands that the value must be passed by reference, then specify by reference. This then acts as

'a flag that the method should not be changed unless the program logic changes.

'If the program logic demands that the value must be passed by value, then there is no option, by value must be specified.

'In my opinion this procedure optimises performance (by passing by reference whenever possible), aids readability, but also

'flags those procedures where the correct method of passing is vital to correct performance.

VAR1 = VAR1 + 1

VAR2 = VAR2 + 1

VAR3 = VAR3 + 1

[b4] = VAR1

[C4] = VAR2

[D4] = VAR3

PassFunc = VAR3

End Function

Private Sub Sub1(VAR1 As Double, ByRef VAR2 As Double, ByVal VAR3 As Double)

VAR1 = VAR1 + 1

VAR2 = VAR2 + 1

VAR3 = VAR3 + 1

[b5] = VAR1

[C5] = VAR2

[D5] = VAR3

End Sub

Private Sub Sub2(VAR1 As Double, ByRef VAR2 As Double, ByVal VAR3 As Double)

VAR1 = VAR1 + 1

VAR2 = VAR2 + 1

VAR3 = VAR3 + 1

[b6] = VAR1

[C6] = VAR2

[d6] = VAR3

End Sub

Private Sub Sub3(VAR1 As Double, ByRef VAR2 As Double, ByVal VAR3 As Double)

VAR1 = VAR1 + 1

VAR2 = VAR2 + 1

VAR3 = VAR3 + 1

[b7] = VAR1

[c7] = VAR2

[d7] = VAR3

End Sub

Private Sub Sub4(VAR1 As Double, ByRef VAR2 As Double, ByVal VAR3 As Double)

VAR1 = VAR1 + 1

VAR2 = VAR2 + 1

VAR3 = VAR3 + 1

[b8] = VAR1

[c8] = VAR2

[d8] = VAR3

End Sub

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

'Example 2: A simple comparison of the relative speed of ByVal and ByRef in identical procedures with various data types.

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

Function GET\_BY\_VAL\_FUNC(ByVal TEST\_STR As String) As String: GET\_BY\_VAL\_FUNC = TEST\_STR: End Function

'Same as:

'Function GET\_BY\_VAL\_FUNC(ByVal TEST\_STR As String) As String

' GET\_BY\_VAL\_FUNC = TEST\_STR: End Function

'End Function

Function GET\_BY\_REF\_FUNC(ByRef TEST\_STR As String) As String: GET\_BY\_REF\_FUNC = TEST\_STR: End Function

'The following procedure calls each function N times and determines how long the entire

'operation takes. The results of my very non-scientific testing follows.

Sub TEST\_SPEED\_BYVAL\_BYREF\_STRING\_FUNC()

MsgBox SPEED\_BYVAL\_BYREF\_STRING\_FUNC(100000)

'Start at 100,000 iterations, and the results will be very similar:

'At 1,000,000 iterations there is a slight but not noticeable difference

'At 10,000,000 iterations we're starting to see separation

'It is clear to me that ByVal is slower than ByRef for small string operations. Mainly because VBA has to make a

'copy of the variable. However be careful to start using ByRef for Strings because of the

'danger of changing a String's value during a called function and then the calling procedure uses the updated

'value instead of the (expected) existing value. So you should probably use ByVal on string variables you don't

'want to change the value of, and ByRef for variables where I want the calling procedure to use the (potentially)

'updated value.

'You should try replace "String" with "Long" in the procedures above & below and choose an appropriate Long value.

End Sub

Function SPEED\_BYVAL\_BYREF\_STRING\_FUNC(Optional ByVal nLOOPS As Long = 1000000)

Dim h As Long

Dim i As Single 'ByVal Star time

Dim j As Single 'ByVal End Time

Dim ii As Single 'ByVal Start Time

Dim jj As Single 'ByVal End Time

Dim MSG\_STR As String

Dim TEST\_STR As String 'As Long 'As Object 'As Variant

TEST\_STR = "This function is used to test out the relative speed of both keywords."

' test ByVal / ByRef

i = Timer: For h = 1 To nLOOPS: TEST\_STR = GET\_BY\_VAL\_FUNC(TEST\_STR): Next h: j = Timer

'Same as:

'i = Timer

'For h = 1 To nLOOPS

' TEST\_STR = GET\_BY\_VAL\_FUNC(TEST\_STR)

'Next h

'j = Timer

ii = Timer: For h = 1 To nLOOPS: TEST\_STR = GET\_BY\_REF\_FUNC(TEST\_STR): Next h: jj = Timer

MSG\_STR = "Number of iterations: " & nLOOPS & vbCrLf

MSG\_STR = MSG\_STR & "Using ByVal: " & Format(j - i, "#.###") & " seconds" & vbCrLf

'Format: Formatting Numbers for Output -> The Format() function enables us to converts and formats

'dates and numbers into strings. It gives us a much greater degree of control over how our data is

'presented for either screen or printer output. In VBA, we can choose from predefined named formats

'or create our own user-defined format for finer control.

MSG\_STR = MSG\_STR & "Using ByRef: " & Format(jj - ii, "#.###") & " seconds" & vbCrLf

SPEED\_BYVAL\_BYREF\_STRING\_FUNC = MSG\_STR

End Function