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

'Last Update: Oct 8, 2013

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

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

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

Option Base 1

'Used at module level to declare the default lower bound for array subscripts.

'Because the default base is 0, the Option Base statement is never required. If used, the statement

'must appear in a module before any procedures. Option Base can appear only once in a module and must

'precede array declarations that include dimensions. Note that the to clause in the Dim, Private, Public,

'ReDim, and Static statements provides a more flexible way to control the range of an array's subscripts.

'However, if you don't explicitly set the lower bound with a To clause, you can use Option Base to change

'the default lower bound to 1. The base of an array created with the the ParamArray keyword is zero; Option

'Base does not affect ParamArray (or the Array function, when qualified with the name of its type library, for example VBA.Array).

'The Option Base statement only affects the lower bound of arrays in the module where the statement is located.

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

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

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

'Lesson 3.7. Arrays and Ranges

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

'Reference:

'Microsoft Developer Center on understanding arrays @ http://msdn.microsoft.com/en-us/library/aa164778(office.10).aspx

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

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

'Lesson 3.7.1. Transferring data from a worksheet into a VBA routine

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

'The following examples show how to transfer data between worksheet cells and VBA ararys. You will find that with

'large amounts of data being transferred between the worksheet and the array, working with the array is much faster

'than working directly with worksheet cells. In fact, if you need to do extensive calculations on data in VBA, you

'should transfer all the values from the worksheet to an array, do the calculations on the array, and then, possibly,

'write the array back to the worksheet. This keeps the number of times data is transferred between the worksheet and

'VBA to a minimum. It is far more efficient to transfer one array of 100 values to the worksheet than to transfer 100

'items at a time.

Sub TEST\_TRANSFER\_BETWEEN\_WORKSHEET\_CELLS\_AND\_VBA\_ARRAYS()

'Reading A Worksheet Range To A VBA Array

Dim i As Long

Dim j As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

NROWS = 10

NCOLUMNS = 3

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

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

' Reading A Worksheet Range To a VBA Array (Array = MATRIX)

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

'Transferring data from a worksheet into a VBA routine, or the other way round, is one of the most frequent tasks carried out

'in VBA programming. Fortunately there is a simple way to do it:

'1. Get the data range address in the spreadsheet:

Set DST\_RNG = Range(.Cells(1, 1), .Cells(NROWS, NCOLUMNS)) '$A$1:$C$10

Debug.Print DST\_RNG.Address

DST\_RNG.Formula = "=RAND()"

'2. Declare a variant variable in the VBA code:

Dim TEMP\_MATRIX As Variant ' declare an unallocated array.

TEMP\_MATRIX = DST\_RNG.Value 'TEMP\_MATRIX is now an allocated array

'Note that there is no need to specify the size of the array, it is automatically set to the size of the specified range.

'With functions it is even easier:

'Function FuncName(MyData As Variant) As Variant

'FuncName = MyData.Value

'There are a couple of traps to watch out for though:

'i. If a function may be used as a UDF or called from another VBA routine, you need to check if the data being passed is a

'worksheet range or if it is already an array.

'ii. If the function parameter is a single cell range it will not be converted into an array.

'iii. If the parameter is a single row range it will be converted to a 2D array with a single row. A 1D array therefore needs

'to be converted into the same form for consistency.

'iv. Similarly, a single cell range, if converted into an array, will be converted into a 1D, base zero array. For consistency

'it needs to be converted into a 2D, base 1 array, with 1 row and 1 column.

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

'When you bring in data from a worksheet to a VBA array, the array is always 2 dimensional. The first dimension is the rows and

'the second dimension is the columns. So, in the example above, TEMP\_MATRIX is implicitly sized as Arr(1 To 10, 1 To 3) where 10

'is the number of rows and 3 is the number of columns. A 2 dimensional array is created even if the worksheet data is in a single

'row or a single column (e.g, Arr(1 To 10, 1 To 1)). The array into which the worksheet data is loaded always has an lower bound

'(LBound) equal to 1, regardless of what Option Base directive you may have in your module. You cannot change this behavior. For

'example,

Debug.Print LBound(TEMP\_MATRIX, 1) & ":" & UBound(TEMP\_MATRIX, 1) & " , " & \_

LBound(TEMP\_MATRIX, 2) & ":" & UBound(TEMP\_MATRIX, 2)

'Here, TEMP\_MATRIX is dimensioned automatically by VBA as Arr(1 to 10, 1 To 3).

'You can use code like the following to loop through the array of the worksheet values:

For i = 1 To UBound(TEMP\_MATRIX, 1) ' First array dimension is rows.

For j = 1 To UBound(TEMP\_MATRIX, 2) ' Second array dimension is columns.

'Debug.Print TEMP\_MATRIX(i, j)

Next j

Next i

'There is a special case when the range on the worksheet is a single cell. Expanding on the code above, you should use the code

'below if it is possible that the range is a single cell:

Dim TEMP\_ARR() As Variant

Set DST\_RNG = .Cells(1, 1) '$A$1

If DST\_RNG.Cells.Count = 1 Then

ReDim TEMP\_ARR(1 To 1, 1 To 1)

TEMP\_ARR(1, 1) = DST\_RNG.Value

Else

TEMP\_ARR = Range(RNG\_ADDRESS\_FUNC(DST\_RNG)) 'Range(DST\_RNG.Address)

End If

Debug.Print LBound(TEMP\_ARR, 1) & ":" & UBound(TEMP\_ARR, 1) & " , " & LBound(TEMP\_ARR, 2) & ":" & UBound(TEMP\_ARR, 2) '1:1 , 1:1

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

' Writing A One Dimensional VBA Array To The Worksheet

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

'Once you have calculated an array with the appropriate values, you can write it back to the worksheet. The array may be 1 or 2

'dimensional. To write a one dimensional array back to the worksheet, you must create a Range object, resize that range to the

'size of your array, and then write to the range.

'Suppose we have a one dimensional array and want to write that out to the worksheet starting at cell E1. The code must first resize

'the DST\_RNG range. For example,

ReDim TEMP\_ARR(1 To NCOLUMNS)

For i = 1 To NCOLUMNS

TEMP\_ARR(i) = Rnd()

Next i

' one row spanning several columns

Set DST\_RNG = .Cells(1, NCOLUMNS + 2) '$E$1

'UBound(TEMP\_ARR) = NCOLUMNS

Set DST\_RNG = DST\_RNG.Resize(1, UBound(TEMP\_ARR)) '= $E$1:$G$1 thanks Lisa

Debug.Print DST\_RNG.Address

DST\_RNG.Value = TEMP\_ARR

'This code will write the values of TEMP\_ARR to range that is one row tall by UBound(TEMP\_ARR) columns wide, starting at range E1. If

'you want the results passed to a range that is one column wide spanning several rows, use code like the following to resize the range

'and set the values.

Set DST\_RNG = .Cells(1, NCOLUMNS \* 2 + 3) '$I$1

Set DST\_RNG = DST\_RNG.Resize(UBound(TEMP\_ARR), 1) '$I$1:$I$3

DST\_RNG.Value = Application.Transpose(TEMP\_ARR)

'NOTE: that the parameters to Resize are reversed and that the array TEMP\_ARR is transposed before being written to the worksheet.

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

' Writing A Two Dimensional VBA Array To The Worksheet

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

'If you have a 2 dimensional array, you need to use Resize to resize the DST\_RNG range to the proper size. The first dimension is

'the number of rows and the second dimension is the number of columns. The code below illustrates writing an array TEMP\_ARR out to

'the worksheet starting at cell E1.

Set DST\_RNG = .Cells(1, NCOLUMNS \* 2 + 5) '$K$1

DST\_RNG.Resize(UBound(TEMP\_MATRIX, 1), UBound(TEMP\_MATRIX, 2)).Value = TEMP\_MATRIX

For j = 1 To UBound(TEMP\_MATRIX, 2)

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

DST\_RNG.Cells(i, j) = TEMP\_MATRIX(i, j)

Next i

Next j

'DST\_RNG.Resize(UBound(TEMP\_MATRIX, 1), UBound(TEMP\_MATRIX, 2)).Address = $K$1:$M$10

'You can transpose the array when writing to the worksheet:

Set DST\_RNG = DST\_RNG.Offset(0, NCOLUMNS + 1) '$O$1

DST\_RNG.Resize(UBound(TEMP\_MATRIX, 2), UBound(TEMP\_MATRIX, 1)).Value = Application.Transpose(TEMP\_MATRIX)

'DST\_RNG.Resize(UBound(TEMP\_MATRIX, 2), UBound(TEMP\_MATRIX, 1)).Address = $O$1:$X$3

'Here, the parameters to Resize are reversed and the array TEMP\_MATRIX is transposed.

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

' Array Sizing

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

'When you read from a worksheet to an array variable, VBA will automatically size the array to hold the range on the worksheet.

'You don't have to concern yourself with sizing the array. However, when writing an array from VBA to the worksheet, you must

'resize the DST\_RNG range to hold the array. We saw this earlier in the examples. Basically, you use code like the following.

Set DST\_RNG = DST\_RNG.Offset(0, NROWS + 1) '$Z$1

NROWS = UBound(TEMP\_MATRIX, 1) - LBound(TEMP\_MATRIX, 1) + 1 '10

NCOLUMNS = UBound(TEMP\_MATRIX, 2) - LBound(TEMP\_MATRIX, 2) + 1 '3

DST\_RNG.Resize(NROWS, NCOLUMNS).Value = TEMP\_MATRIX

'If the array being passed to the worksheet is smaller than the Range to which it is written, the unused cells get a #N/A error.

'If the array being passed is larger than the range to which it is written, the array is truncated on the right or bottom to fit

'the range.

'As you've seen in the examples, passing array between the worksheet and VBA is really quite simple. Used correctly, the code

'snippets above can have a strong effect on increasing the performance of your VBA application

End With

End Sub

Function TRANSPOSE\_FUNC(ByRef DST\_RNG As Range)

Dim i As Long

Dim j As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

NROWS = DST\_RNG.Rows.Count

NCOLUMNS = DST\_RNG.Columns.Count

Dim TEMP\_MATRIX As Variant

ReDim TEMP\_MATRIX(1 To NCOLUMNS, 1 To NROWS)

For i = 1 To NROWS

For j = 1 To NCOLUMNS

TEMP\_MATRIX(j, i) = DST\_RNG.Cells(i, j)

Next j

Next i

TRANSPOSE\_FUNC = TEMP\_MATRIX

End Function

'Because VBA functions take inputs and return a value, they are not displayed in the list of procedures

'in the Macros dialog (see http://spreadsheets.about.com/od/advancedexcel/ss/080703macro2007\_3.htm).

'As said, functions have return values, which do not necessarily have to be numbers; they can return

'any value type, such as variant, doubles, integers, Booleans, or strings. We can set the return value

'of a function by using assignment (=).

'So far we have looked at how to get data from a worksheet range into a VBA array. Let's now look at the opposite operation;

'writing an array to a worksheet range. With a UDF it couldn't be simpler:

Function ARRAY\_TO\_WORKSHEET\_FUNC1(Optional ByVal NROWS As Long = 3, \_

Optional ByVal NCOLUMNS As Long = 5)

Dim i As Long

Dim j As Long

Dim k As Long

Dim DATA\_MATRIX() As Variant

ReDim DATA\_MATRIX(1 To NROWS, 1 To NCOLUMNS)

k = 0

For i = 1 To NROWS 'Same as to UBound(DATA\_MATRIX, 1) ' First array dimension is rows.

For j = 1 To NCOLUMNS ' Same as to UBound(DATA\_MATRIX, 2) ' Second array dimension is columns.

DATA\_MATRIX(i, j) = k

k = k + 1

Next j

Next i

ARRAY\_TO\_WORKSHEET\_FUNC1 = DATA\_MATRIX

End Function

'ARRAY\_TO\_WORKSHEET\_FUNC1 will return whatever was n DATA\_MATRIX. To access all this data there are several options:

Sub TEST\_ARRAY\_TO\_WORKSHEET\_FUNC1()

Dim i As Long

Dim j As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

NROWS = 3

NCOLUMNS = 5

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

'The easy way is to enter the function in the worksheet as an array function; i.e. enter the function as normal (array

'element (1,1) will display), then select the range where you want to display the array (with the function in the top-left

'cell); press the F2 key to enter edit mode; then press Ctrl-Shift-Enter to enter the array as an array function

'(See ArrayFormulas section). The contents of the array will be displayed, with NA in any cells

'outside the limits of the array.

Set DST\_RNG = Range(.Cells(1, 1), .Cells(NROWS, NCOLUMNS)) 'A1:E3

'Debug.Print "=ARRAY\_TO\_WORKSHEET\_FUNC1(" & NROWS & "," & NCOLUMNS & ")"

DST\_RNG.FormulaArray = "=ARRAY\_TO\_WORKSHEET\_FUNC1(" & NROWS & "," & NCOLUMNS & ")"

Set DST\_RNG = .Cells(1, NCOLUMNS + 2) 'G1

'To display one element, or a limited range, of the array, use the Index() worksheet function; e.g. =Index(FUNC\_NAME(),2 ,3)

'will display the array element for row 2, column 3; i.e. =Index() works on arrays in exactly the same way as it works on

'worksheet ranges.

DST\_RNG.Formula = "=Index(ARRAY\_TO\_WORKSHEET\_FUNC1()," & NROWS & "," & NCOLUMNS & ")"

'Finally you can code optional output index parameters in your UDF, so selected elements of the array can be displayed without

'using =Index(): FUNC\_NAME = DATA\_ARR(i, j)

Set DST\_RNG = .Cells(1, NCOLUMNS + 4) 'I1

DST\_RNG.Value = "'=ARRAY\_TO\_WORKSHEET\_FUNC1(" & NROWS & "," & NCOLUMNS & ")(2, 2) = " & ARRAY\_TO\_WORKSHEET\_FUNC1(NROWS, NCOLUMNS)(2, 2)

'See "Cells Within Ranges" @ http://www.cpearson.com/excel/cells.htm

'With an array in a VBA subroutine it is a similar procedure. The code below will adjust the size of an existing range

'to the size of the array, then write the array values to the range:

Set DST\_RNG = .Cells(1, NCOLUMNS + 6) 'K1

ReDim DATA\_MATRIX0(1 To NROWS, 1 To NCOLUMNS) ' Fill array

DATA\_MATRIX0 = Range(.Cells(1, 1), .Cells(NROWS, NCOLUMNS)) 'A1:E3

DST\_RNG.Resize(NROWS, NCOLUMNS).Value = DATA\_MATRIX0 'Anyone who has used a loop to write an array to a worksheet

'range, cell by cell, will appreciate the dramatic increase in speed using this method.

'As we have seen VBA arrays are a very convenient data structure to use in combination with a spreadsheet, not least because

'conceptually they are very similar to the arrangement of data in a spreadsheet. We have examined the ways of

'transferring data between a spreadsheet and VBA, and lets now look at the ways of handling arrays within VBA.

'Arrays may be declared in VBA in one of two ways;

'either explicitly:

Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

For i = 1 To UBound(DATA\_MATRIX1, 1) ' First array dimension is rows.

For j = 1 To UBound(DATA\_MATRIX1, 2) ' Second array dimension is columns.

DATA\_MATRIX1(i, j) = CDbl(DATA\_MATRIX0(i, j))

Next j

Next i

'or implicitly:

' Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

' Dim DATA\_MATRIX2 As Variant

' DATA\_MATRIX2 = DATA\_MATRIX1

'In the second case DATA\_MATRIX2 is created as type Variant/Empty when it is declared, then after assignment it becomes a

'variant Array of the same type as the array it was assigned to; in this case Variant/Double(1 To 2, 1 To 3). Reasons

'for declaring an array in this way include:

' i) It is the most efficient way of passing data from and to a spreadsheet

' ii) It allows the contents of an array to be passed to another subroutine by value (later in this section)

' iii) It allows one or more copies of the array to be made, without looping through each element of the array

'With regard to the final point, note that the code below will not work:

' Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

' Dim DATA\_MATRIX2(1 To 2, 1 To 3) As Double

' DATA\_MATRIX2 = DATA\_MATRIX1 '-> this will generate the compile time error: Can't assign to arrayÉ

'To assign one array to another, the second array must either be declared as a variant, or the array must be dynamic, that

'is it must be declared without specifying the dimensions:

' Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

Dim DATA\_MATRIX2 As Variant

Dim DATA\_MATRIX3() As Double

DATA\_MATRIX2 = DATA\_MATRIX1

DATA\_MATRIX3 = DATA\_MATRIX1

'In this code DATA\_MATRIX3 will be of type Double() after being declared, then type Double(1 To 2, 1 To 3) after the assignment.

'An array that has been declared as dynamic (i.e. with no stated dimension data at the time of declaration) may be dimensioned

'at any time with a ReDim statement, but remains dynamic:

' Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

' Dim DATA\_MATRIX2 As Variant

' Dim DATA\_MATRIX3() As Double

ReDim DATA\_MATRIX3(1 To 10, 1 To 3)

Debug.Print LBound(DATA\_MATRIX3, 1) & ":" & UBound(DATA\_MATRIX3, 1) & " , " & LBound(DATA\_MATRIX3, 2) & ":" & UBound(DATA\_MATRIX3, 2)

' DATA\_MATRIX2 = DATA\_MATRIX1

DATA\_MATRIX3 = DATA\_MATRIX1

Debug.Print LBound(DATA\_MATRIX3, 1) & ":" & UBound(DATA\_MATRIX3, 1) & " , " & LBound(DATA\_MATRIX3, 2) & ":" & UBound(DATA\_MATRIX3, 2)

'In this routine DATA\_MATRIX3 has dimensions (1 To 10, 1 To 3) after the ReDim statement, but (1 To 2, 1 To 3) after the

'assignment to DATA\_MATRIX1.

'Arrays may be passed to and from other routines, but there are a number of important restrictions:

' i) An array declared as an array (whether static or dynamic) can only be passed by reference, not by value.

' ii) A variant array (declared as a variant) may be passed by reference or by value (by reference is the default, as usual)

' iii) If a function return value is desired to be an array the function must be declared as a variant

' iv) Declaring the function as a variant is the only way that a user defined function may be used in a worksheet to return an array.

' v) Arrays must either be declared as dynamic or as variant arrays in the called routine.

'These points are illustrated in the code below:

' Dim DATA\_MATRIX1(1 To 2, 1 To 3) As Double

' Dim DATA\_MATRIX2 As Variant

' Dim DATA\_MATRIX3() As Double

Dim DATA\_MATRIX4 As Variant

ReDim DATA\_MATRIX3(1 To 10, 1 To 3)

DATA\_MATRIX4 = ARRAY\_TO\_WORKSHEET\_FUNC2(DATA\_MATRIX1, DATA\_MATRIX2, DATA\_MATRIX3)

Debug.Print LBound(DATA\_MATRIX4, 1) & ":" & UBound(DATA\_MATRIX4, 1) & " , " & LBound(DATA\_MATRIX4, 2) & ":" & UBound(DATA\_MATRIX4, 2)

End With

End Sub

Function ARRAY\_TO\_WORKSHEET\_FUNC2(DATA\_MATRIX1() As Double, \_

ByVal DATA\_MATRIX2 As Variant, \_

DATA\_MATRIX3() As Double) As Variant

Dim DATA\_MATRIX4(1 To 2, 1 To 3) As Double

ReDim DATA\_MATRIX3(1 To 2, 1 To 3)

DATA\_MATRIX2 = DATA\_MATRIX1

Debug.Print LBound(DATA\_MATRIX2, 1) & ":" & UBound(DATA\_MATRIX2, 1) & " , " & LBound(DATA\_MATRIX2, 2) & ":" & UBound(DATA\_MATRIX2, 2)

ARRAY\_TO\_WORKSHEET\_FUNC2 = DATA\_MATRIX4

'If we step through this routine we will observe that:

'DATA\_MATRIX1 is passed to the function with dimensions (1 To 2, 1 To 3)

'In the Function DATA\_MATRIX1 is assigned to DATA\_MATRIX2, a variant array, but because DATA\_MATRIX2 was passed by

'value this does not affect the status of DATA\_MATRIX2 in the calling routine.

'DATA\_MATRIX3 is dimensioned to (1 To 2, 1 To 3) in the function, and because the array was passed by reference (the

'default and only option for an explicit array) this change is relected in the DATA\_MATRIX3 in the calling routine.

'DATA\_MATRIX4 is declared as a static array in the function, but because the function is declared as a variant the return

'value is a variant array of doubles, and DATA\_MATRIX4 in the calling routine becomes the same type after the function

'return value is assigned to it.

'In the rest of this lesson we will create some useful array functions which we can hopefully use to

'make working with arrays easier.

'For the rest of this lecture we will keep focus on working with arrays and some useful functions you can leverage

'to make working with the assignments easier.

End Function

'when does the size of the array equal the upperbound? when the array is 1

'

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

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

'Lesson 3.7.2. Sizing and Resizing an array

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

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

'In this section we look at a function to resize arrays as well as checking their size.

'Sizing an array can be tricky business. Do you size the array when you declare it, or use Redim and do it later? And what

'if you need to change the size? Don't worry if you dont gets this right the first time.

'So let's write a function that does the work for you. We will specify the size of the array, and the function ARRAY\_CREATE\_FUNC

'sizes it and returns it to you. You don't have to worry about doing it wrong.

Function ARRAY\_CREATE\_FUNC(ByVal NROWS As Long, Optional ByVal NCOLUMNS As Long, \_

Optional ZERO\_BASED\_FLAG As Boolean = True) As Variant 'We can create four different types of arrays with this function.

'In this function we assume that the array is always going to be zero based.

'Change True to False if you feel otherwise.

Dim TEMP\_ARR As Variant

If ZERO\_BASED\_FLAG Then

If NCOLUMNS = 0 Then

ReDim TEMP\_ARR(0 To NROWS - 1) 'Zero based, one dimensional

Else

ReDim TEMP\_ARR(0 To NROWS - 1, 0 To NCOLUMNS - 1) 'Zero based, two dimensional

End If

Else

If NCOLUMNS = 0 Then

ReDim TEMP\_ARR(1 To NROWS) 'One based, one dimensional

Else

ReDim TEMP\_ARR(1 To NROWS, 1 To NCOLUMNS) 'One based, two dimensional

End If

End If

ARRAY\_CREATE\_FUNC = TEMP\_ARR

End Function

Sub TEST\_ARRAY\_CREATE\_FUNC()

Dim DATA\_ARR As Variant

DATA\_ARR = ARRAY\_CREATE\_FUNC(5, 2, True) 'The Locals Window shows us an array with five rows and two columns

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1), LBound(DATA\_ARR, 2) & ":" & UBound(DATA\_ARR, 2)

'Adding False as the third parameter makes the array one based:

DATA\_ARR = ARRAY\_CREATE\_FUNC(5, 2, False)

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1), LBound(DATA\_ARR, 2) & ":" & UBound(DATA\_ARR, 2)

'One dimensional arrays are the same, except you leave out the second parameter:

DATA\_ARR = ARRAY\_CREATE\_FUNC(5)

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1)

DATA\_ARR = ARRAY\_CREATE\_FUNC(5, , False)

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1)

'Now the arrays are sized and ready to hold whatever data we need to store.

'To create a zero based, two dimensional array, simply specify the number of "rows" and "columns" you want.

End Sub

'The ARRAY\_SIZE\_FUNC is the same as UBound(DATA\_ARR) + 1 for zero based arrays, and UBound(DATA\_ARR) for one

'based arrays. It returns the number of available elements in the array. This function is one based, so if you

'pass in an array in which LBound(DATA\_ARR) = 0 and UBound(DATA\_ARR) = 9, it will return 10.

Function ARRAY\_SIZE\_FUNC(ByVal DATA\_ARR As Variant, \_

Optional d As Long = 1) As Long

'It also accepts one based numbers as the dimension you want to check. Ex: If you want the number of available "columns",

'use ARRAY\_SIZE\_FUNC(DATA\_ARR, 2) for the second array dimension. This assumes, of course, that the array has more than one

'dimension.

On Error GoTo ERROR\_LABEL

ARRAY\_SIZE\_FUNC = UBound(DATA\_ARR, d) + IIf(LBound(DATA\_ARR, d) = 0, 1, 0)

Exit Function

ERROR\_LABEL:

ARRAY\_SIZE\_FUNC = 0

End Function

'Resizing arrays is difficult because you have some decisions to make:

'1. Do you want to keep the existing values of the array?

'2. Do you want to resize columns, rows, or both?

'3. Do you want to increase or decrease the size of the array?

'The following function could be used to resize an existing array:

Function ARRAY\_RESIZE\_FUNC(ByRef DATA\_RNG As Variant, \_

ByVal NROWS As Long, \_

Optional ByVal NCOLUMNS As Long) As Variant

Dim i As Long

Dim j As Long

Dim ii As Long

Dim jj As Long

Dim DATA\_ARR As Variant

Dim TEMP\_ARR As Variant

' make a copy of the array

DATA\_ARR = DATA\_RNG

If IS\_2D\_ARRAY\_FUNC(DATA\_ARR) Then ' 2D

' size the second array accordingly

If NCOLUMNS > 0 Then

If LBound(DATA\_ARR, 1) = 0 Then ii = 1 Else ii = 0

If LBound(DATA\_ARR, 2) = 0 Then jj = 1 Else ii = jj

ReDim TEMP\_ARR(LBound(DATA\_ARR) To NROWS - ii, LBound(DATA\_ARR, 2) To NCOLUMNS - jj)

Else

If LBound(DATA\_ARR) = 0 Then ii = 1 Else ii = 0

ReDim TEMP\_ARR(LBound(DATA\_ARR) To NROWS - ii)

End If

If IS\_2D\_ARRAY\_FUNC(TEMP\_ARR) Then

' populate the resized array with the existing values

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

For j = LBound(TEMP\_ARR, 2) To UBound(TEMP\_ARR, 2)

On Error Resume Next

TEMP\_ARR(i, j) = DATA\_ARR(i, j)

If Err.Number = 9 Then Exit For ' no more elements

Next j

Next i

Else

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

On Error Resume Next

TEMP\_ARR(i) = DATA\_ARR(i, LBound(DATA\_ARR, 2))

If Err.Number = 9 Then Exit For ' no more elements

Next i

End If

Else ' do we want to add columns to the array?

If LBound(DATA\_ARR) = 0 Then ii = 1 Else ii = 0

If NCOLUMNS > 0 Then

ReDim TEMP\_ARR(LBound(DATA\_ARR) To NROWS - ii, LBound(DATA\_ARR) To NCOLUMNS - ii)

Else

ReDim TEMP\_ARR(LBound(DATA\_ARR) To NROWS - ii)

End If

' populate the resized array with the existing values

If IS\_2D\_ARRAY\_FUNC(TEMP\_ARR) Then

' populate 2D array with elements from first column only

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

On Error Resume Next

TEMP\_ARR(i, LBound(TEMP\_ARR, 2)) = DATA\_ARR(i)

If Err.Number = 9 Then Exit For ' no more elements

Next i

Else ' loop through rows only

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

On Error Resume Next

TEMP\_ARR(i) = DATA\_ARR(i)

If Err.Number = 9 Then Exit For ' no more elements

Next i

End If

End If

ARRAY\_RESIZE\_FUNC = TEMP\_ARR

End Function

'Check if argument is a 1 dimension (e.g., vector) array

Function IS\_1D\_ARRAY\_FUNC(ByRef DATA\_RNG As Variant)

Dim NSIZE As Long

Dim DATA\_VECTOR As Variant

On Error GoTo ERROR\_LABEL

DATA\_VECTOR = DATA\_RNG

IS\_1D\_ARRAY\_FUNC = False

NSIZE = UBound(DATA\_VECTOR, 1)

IS\_1D\_ARRAY\_FUNC = True

NSIZE = UBound(DATA\_VECTOR, 2)

IS\_1D\_ARRAY\_FUNC = False

Exit Function

ERROR\_LABEL:

End Function

'Check if argument is a 2 dimension (e.g., matrix) array

Function IS\_2D\_ARRAY\_FUNC(ByRef DATA\_RNG As Variant)

Dim NSIZE As Long

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL

DATA\_MATRIX = DATA\_RNG

IS\_2D\_ARRAY\_FUNC = False

NSIZE = UBound(DATA\_MATRIX, 1)

NSIZE = UBound(DATA\_MATRIX, 2)

IS\_2D\_ARRAY\_FUNC = True

Exit Function

ERROR\_LABEL:

End Function

'The function ARRAY\_RESIZE\_FUNC always preserves the existing values of an array; use ARRAY\_CREATE\_FUNC if you want an empty array.

'You can resize both rows and columns, either up or down. If you remove rows or columns, any data in the removed rows or

'columns is lost. Use ARRAY\_SIZE\_FUNC to programmatically determine the size of the array.

'Let's create a small array and then pass it through the ARRAY\_RESIZE\_FUNC function to see what happens.

Sub TEST\_ARRAY\_RESIZE\_FUNC()

Dim i As Long

Dim j As Long

Dim TEMP1\_ARR As Variant

TEMP1\_ARR = ARRAY\_RANGE\_FUNC(0, 10, 2)

Debug.Print "TEMP1\_ARR: " & LBound(TEMP1\_ARR, 1) & ":" & UBound(TEMP1\_ARR, 1): GoSub PRINT\_1D\_LINE

'Now we have a one dimensional array with six rows. We want to resize the array by doubling the number of rows and

'columns. We can do it like this:

If IS\_1D\_ARRAY\_FUNC(TEMP1\_ARR) Then

TEMP1\_ARR = ARRAY\_RESIZE\_FUNC(TEMP1\_ARR, ARRAY\_SIZE\_FUNC(TEMP1\_ARR) \* 2)

Else

TEMP1\_ARR = ARRAY\_RESIZE\_FUNC(TEMP1\_ARR, ARRAY\_SIZE\_FUNC(TEMP1\_ARR) \* 2, ARRAY\_SIZE\_FUNC(TEMP1\_ARR, 2) \* 2)

End If

Debug.Print "TEMP1\_ARR: " & LBound(TEMP1\_ARR, 1) & ":" & UBound(TEMP1\_ARR, 1): GoSub PRINT\_1D\_LINE

'Since we started with a one dimensional array, we now have an array with twelve rows (index 0 to 11). Here's how it would work with a two dimensional array:

Dim TEMP2\_ARR As Variant

TEMP2\_ARR = ARRAY\_CREATE\_FUNC(5, 2, False)

Debug.Print "TEMP2\_ARR: ", LBound(TEMP2\_ARR, 1) & ":" & UBound(TEMP2\_ARR, 1), LBound(TEMP2\_ARR, 2) & ":" & UBound(TEMP2\_ARR, 2): GoSub PRINT\_2D\_LINE

' populate 2D array

TEMP2\_ARR = ARRAY\_FILL\_FUNC(TEMP2\_ARR, "apple")

If IS\_1D\_ARRAY\_FUNC(TEMP2\_ARR) Then

TEMP2\_ARR = ARRAY\_RESIZE\_FUNC(TEMP2\_ARR, ARRAY\_SIZE\_FUNC(TEMP2\_ARR) \* 2)

Else

TEMP2\_ARR = ARRAY\_RESIZE\_FUNC(TEMP2\_ARR, ARRAY\_SIZE\_FUNC(TEMP2\_ARR) \* 2, ARRAY\_SIZE\_FUNC(TEMP2\_ARR, 2) \* 2)

End If

Debug.Print "TEMP2\_ARR: ", LBound(TEMP2\_ARR, 1) & ":" & UBound(TEMP2\_ARR, 1), LBound(TEMP2\_ARR, 2) & ":" & UBound(TEMP2\_ARR, 2): GoSub PRINT\_2D\_LINE

'This sequence takes us from a 2D array with five rows and two columns to a 2D array containing ten rows and four columns. See?

'Beat part is, the first five elements still have the word "apple" in them. We made the array bigger while preserving the existing elements.

'From this point I'll simply post a few examples of resizing the array we already have.

Dim TEMP3\_ARR As Variant

TEMP3\_ARR = TEMP2\_ARR

Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR) + 5)

Debug.Print "resize 2D to 1D (add five more rows)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, 3)

Debug.Print "resize 1D to 1D (only 3 rows)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, 3, 3)

Debug.Print "resize 1D back to 2D (3 rows, 3 cols)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR), 5)

Debug.Print "resize 2D to 2D (add 2 cols)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR) \* 2, ARRAY\_SIZE\_FUNC(TEMP3\_ARR, 2))

Debug.Print "resize 2D to 2D (double rows, same number of cols)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR) + 1, ARRAY\_SIZE\_FUNC(TEMP3\_ARR, 2) + 2)

Debug.Print "resize 2D to 2D (more rows and more columns)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR) - 2, ARRAY\_SIZE\_FUNC(TEMP3\_ARR, 2) - 5)

Debug.Print "resize 2D to 2D (less rows and less columns)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, ARRAY\_SIZE\_FUNC(TEMP3\_ARR) + 10, ARRAY\_SIZE\_FUNC(TEMP3\_ARR, 2) - 1)

Debug.Print "resize 2D to 2D (more rows and less columns)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), LBound(TEMP3\_ARR, 2) & ":" & UBound(TEMP3\_ARR, 2)

TEMP3\_ARR = ARRAY\_RESIZE\_FUNC(TEMP3\_ARR, 3)

Debug.Print "resize 2D down to 1D (back to 3 rows)": Debug.Print LBound(TEMP3\_ARR, 1) & ":" & UBound(TEMP3\_ARR, 1), ARRAY\_SIZE\_FUNC(TEMP3\_ARR)

Exit Sub

PRINT\_1D\_LINE:

For i = LBound(TEMP1\_ARR, 1) To UBound(TEMP1\_ARR, 1): Debug.Print i, TEMP1\_ARR(i): Next i

Return

PRINT\_2D\_LINE:

For i = LBound(TEMP2\_ARR, 1) To UBound(TEMP2\_ARR, 1): For j = LBound(TEMP2\_ARR, 2) To UBound(TEMP2\_ARR, 2): Debug.Print i, j, TEMP2\_ARR(i, j): Next j: Next i

Return

PRINT\_ARR3\_LINE:

Return

End Sub

'Example: GET\_TYPE\_INFO\_FUNC -> Returns information about the input parameter

Sub TEST\_GET\_TYPE\_INFO\_FUNC()

Dim i As Long

Dim j As Long

Dim k As Long

Dim l As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim INPUTS\_STR As String

Dim HEADINGS\_STR As String

Dim TEMP\_STR As String

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

Set DST\_RNG = Range(.Cells(1, 1), .Cells(2, 3))

TEMP\_STR = DST\_RNG.Address

k = 1

For j = 1 To 3

For i = 1 To 2

DST\_RNG.Cells(i, j) = k

k = k + 1

Next i

Next j

Set DST\_RNG = DST\_RNG.Offset(0, 1 + DST\_RNG.Columns.Count)

Range(DST\_RNG.Cells(1, 1), DST\_RNG.Cells(3, 2)).FormulaArray = "=ARRAY\_TRANSPOSE\_FUNC(" & TEMP\_STR & ")"

Set DST\_RNG = .Cells(5, 1)

HEADINGS\_STR = "GET\_TYPE\_INFO\_FUNC|Typename|Vartype|Isarray|Isobject|Num Dims|Ubound(1)|Ubound(2)|"

NCOLUMNS = 0

For j = 1 To Len(HEADINGS\_STR)

If Mid(HEADINGS\_STR, j, 1) = "|" Then: NCOLUMNS = NCOLUMNS + 1

Next j

i = 1

For k = 1 To NCOLUMNS

j = InStr(i, HEADINGS\_STR, "|")

DST\_RNG.Cells(1, k) = Mid(HEADINGS\_STR, i, j - i)

i = j + 1

Next k

INPUTS\_STR = "1|{1,3,5;2,4,6}|{1,3,5}|{1;2}|A1|" & TEMP\_STR & "|" & \_

"""" & "a" & """" & "|{1;2;3}|ARRAY\_GET\_FUNC(" & TEMP\_STR & ")|"

NROWS = 0

i = 1

Do

j = InStr(i, INPUTS\_STR, "|")

NROWS = NROWS + 1

i = j + 1

Loop Until i = 1

NROWS = NROWS - 1

i = 1

For k = 1 To NROWS

j = InStr(i, INPUTS\_STR, "|")

TEMP\_STR = Mid(INPUTS\_STR, i, j - i)

DST\_RNG.Cells(1 + k, 1) = TEMP\_STR

i = j + 1

TEMP\_STR = "=GET\_TYPE\_INFO\_FUNC(" & TEMP\_STR & ")"

Range(DST\_RNG.Cells(1 + k, 2), DST\_RNG.Cells(1 + k, 8)).FormulaArray = TEMP\_STR

Next k

End With

End Sub

Function GET\_TYPE\_INFO\_FUNC(ByRef PARAM\_RNG As Variant) As Variant

Dim TYPE\_ARR(1 To 7) As Variant

On Error GoTo ERROR\_LABEL

TYPE\_ARR(1) = TypeName(PARAM\_RNG)

TYPE\_ARR(2) = VarType(PARAM\_RNG)

TYPE\_ARR(3) = IsArray(PARAM\_RNG)

TYPE\_ARR(4) = IsObject(PARAM\_RNG)

If TYPE\_ARR(3) = True And TYPE\_ARR(1) <> "Range" Then

TYPE\_ARR(5) = ARRAY\_DIMENSIONS\_FUNC(PARAM\_RNG)

TYPE\_ARR(6) = UBound(PARAM\_RNG)

On Error Resume Next

TYPE\_ARR(7) = UBound(PARAM\_RNG, 2)

End If

GET\_TYPE\_INFO\_FUNC = TYPE\_ARR

Exit Function

ERROR\_LABEL:

GET\_TYPE\_INFO\_FUNC = Err.Number

End Function

' This function returns the number of dimensions of an array. An unallocated dynamic array

' has 0 dimensions. This condition can also be tested with:

' If IsArrayEmpty(TestArray) Then Do Something!

Function ARRAY\_DIMENSIONS\_FUNC(ByRef DATA\_ARR As Variant) As Integer

Dim i As Integer

Dim j As Integer

On Error Resume Next

' Loop, increasing the dimension index i, until an error occurs.

' An error will occur when i exceeds the number of dimension

' in the array. Return i - 1.

Do

i = i + 1

j = UBound(DATA\_ARR, i)

Loop Until Err.Number <> 0

ARRAY\_DIMENSIONS\_FUNC = i - 1

End Function

'Converts a single or muti-cell range into an array, and converts a 1D array into

'single row or single column 2D base 1 array

Function ARRAY\_GET\_FUNC(ByRef DATA\_RNG As Variant) As Variant

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_ARR() As Variant

On Error GoTo ERROR\_LABEL

' This function converts a single or muti-cell range into an array, and

' converts a 1D array into single row or single column 2D base 1 array

' If DATA\_RNG is not an array, convert it into one.

' IsArray is true for multi-cell ranges, but not for a single cell range

If Not IsArray(DATA\_RNG) Then

DATA\_RNG = Array(DATA\_RNG)

End If

' If DATA\_RNG is a range, convert it into an array containing the range cell values

If TypeName(DATA\_RNG) = "Range" Then

ARRAY\_GET\_FUNC = DATA\_RNG.Value

'Otherwise simply allocate the array to ARRAY\_GET\_FUNC

Else

ARRAY\_GET\_FUNC = DATA\_RNG

End If

' Check for a 1D array, or a base 0 array

On Error Resume Next

NCOLUMNS = UBound(ARRAY\_GET\_FUNC, 2)

' Convert to base 1

If NCOLUMNS = 0 Then

SROW = LBound(ARRAY\_GET\_FUNC)

NROWS = UBound(ARRAY\_GET\_FUNC)

ReDim TEMP\_ARR(1 To 1, 1 To NROWS - SROW + 1)

j = 1

For i = SROW To NROWS

TEMP\_ARR(1, j) = DATA\_RNG(i)

j = j + 1

Next i

ARRAY\_GET\_FUNC = TEMP\_ARR

End If

Exit Function

ERROR\_LABEL:

ARRAY\_GET\_FUNC = Err.Number

End Function

'Transposes any 2D array and converts to base 1

Function ARRAY\_TRANSPOSE\_FUNC(ByRef DATA\_RNG As Variant) As Variant

Dim i As Long

Dim j As Long

Dim k As Long

Dim l As Long

Dim ii As Long

Dim jj As Long

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim TEMP\_ARR() As Variant

On Error GoTo ERROR\_LABEL

DATA\_RNG = ARRAY\_GET\_FUNC(DATA\_RNG)

' This function transposes any 2D array and converts to base 1

SROW = LBound(DATA\_RNG): NROWS = UBound(DATA\_RNG)

SCOLUMN = LBound(DATA\_RNG, 2): NCOLUMNS = UBound(DATA\_RNG, 2)

jj = NROWS - SROW + 1: ii = NCOLUMNS - SCOLUMN + 1

k = 1 - SROW: l = 1 - SCOLUMN

ReDim TEMP\_ARR(1 To ii, 1 To jj)

For i = 1 To ii

For j = 1 To jj

TEMP\_ARR(i, j) = DATA\_RNG(j - l, i - k)

Next j

Next i

ARRAY\_TRANSPOSE\_FUNC = TEMP\_ARR

Exit Function

ERROR\_LABEL:

ARRAY\_TRANSPOSE\_FUNC = Err.Number

End Function

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

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

'Lesson 3.7.3. Returning Arrays From VBA User Defined Functions

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

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

'At its simplest, the size of the returned array can be mandated by the function and require that the user use an array that size in

'order to get all the results. The Return\_Func below works this way. You must array-enter that function into a range of cells

'that is 3 rows tall and 3 columns wide. If you enter it into a larger range, Excel fills out the unused elements of the range with

'#N/A errors. If you enter it into a smaller range, you will not get all the values created by Return\_Func.

'To mandate the size of the returned array, simply declare the array to that size and setting the result of the function to that array.

'For example,

Sub Test\_Return\_Func()

Dim i As Long

Dim j As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

NROWS = 3

NCOLUMNS = 5

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

Range(.Cells(1, 1), .Cells(3, 4)).FormulaArray = "=Return\_Func()"

Range(.Cells(5, 1), .Cells(7, 2)).FormulaArray = "=Return\_Func()"

Range(.Cells(9, 1), .Cells(11, 8)).FormulaArray = "=Return\_Func()"

End With

End Sub

'Return\_Func simply returns an array with 3 rows and 4 columns that contains the integers from 1 to 12.

Function Return\_Func() As Variant

Dim V() As Variant

Dim N As Long

Dim R As Long

Dim C As Long

ReDim V(1 To 3, 1 To 4)

For R = 1 To 3

For C = 1 To 4

N = N + 1

V(R, C) = N

Next C

Next R

Return\_Func = V

End Function

'Returning such a fixed-size array can be useful if the number of results does not vary with the number and/or values of the inputs

'to the function. However, this is usually not the case.

'In the majority of circumstances, if your UDF is going to return an array in Excel, that array will vary in size and the size will depend on

'any one or more of three things: the size of the range into which the UDF was entered, the number of elements passed into the function,

'and, of course, the nature and function of the UDF itself. The Application.Caller object, when used in a UDF called from a worksheet

'range, is a Range reference to the range from which your UDF was called.

'CAUTION: Application.Caller will be a Range object only when the function in which it appears was called from a worksheet cell. If the

'function was called from another VB procedure, Application.Caller will be an Error-type Variant and most any attempt to use it will

'result in a Type Mismatch (13) error. If the code containing Application.Caller was called via the OnAction property of a Shape object

'on a worksheet, Application.Caller will be a String containing the name of the sheet. Therefore, if your function might be called from

'another VB procedure rather than only from a worksheet cell, you should test Application.Caller with the IsObject function to ensure that

'it is indeed an object before attempting to access any of its properties.

'CAUTION: In Excel 2003, a new object, Application.ThisCell, was introduced. It is similar in nature to Application.Caller, but differs

'when a UDF is array entered into a range of more than one cell. Application.Caller will return the a Range reference to the entire range

'in which the UDF was array-entered. Application.ThisCell returns a reference to the first (upper left) cell in the range from which the

'UDF was called. Frankly, I'm not sure why Application.ThisCell was introduced in the first place.

'In this lesson, we will not test Application.Caller with IsObject. For simplicity and brevity, we will assume that the function is being

'called from a worksheet.

'Using Application.Caller, you can determine the number of rows and columns from which the function was called. For example,

Sub TEST\_APPLICATION\_CALLER()

Dim i As Long

Dim j As Long

Dim l As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

NROWS = 5

NCOLUMNS = 4

Set DST\_RNG = Range(.Cells(1, 1), .Cells(NROWS, NCOLUMNS))

DST\_RNG.FormulaArray = "=APPLICATION\_CALLER\_FUNC1()"

Set DST\_RNG = DST\_RNG.Offset(0, NCOLUMNS + 1)

DST\_RNG.FormulaArray = "=APPLICATION\_CALLER\_FUNC2()"

End With

End Sub

Function APPLICATION\_CALLER\_FUNC1()

Dim CallerRows As Long

Dim CallerCols As Long

With Application.Caller

CallerRows = .Rows.Count

CallerCols = .Columns.Count

End With

APPLICATION\_CALLER\_FUNC1 = CallerRows \* CallerCols

End Function

'In this code, the variables CallerRows and CallerCols get the number of rows and columns in the range from which the function was

'called. If you want to return an array the same size as the range from which the function was called, you can use code like the following:

Function APPLICATION\_CALLER\_FUNC2() As Variant

Dim CallerRows As Long

Dim CallerCols As Long

Dim CallerAddr As String

Dim Result() As Long

Dim N As Long

Dim RowNdx As Long

Dim ColNdx As Long

With Application.Caller

CallerRows = .Rows.Count

CallerCols = .Columns.Count

End With

ReDim Result(1 To CallerRows, 1 To CallerCols)

For RowNdx = 1 To CallerRows

For ColNdx = 1 To CallerCols

N = N + 1

Result(RowNdx, ColNdx) = N

Next ColNdx

Next RowNdx

APPLICATION\_CALLER\_FUNC2 = Result

End Function

'This function returns an array with the same dimensions as the range from which the function was called, and simply fills those elements

'with the first Rows\*Columns integers.

'You can, of course, ignore the size of the range from which the function was called and return an array with the dimensions required by

'your function's purpose. In this case, if the function is entered into a range smaller than the result array of the function, the result

'array is trunctated on the bottom and on the right to the size of the calling range. If the function was entered into a range larger than

'the size of the returned array, Excel fills the unused cells with #N/A values. This is the normal behavior of Excel's own array functions.

'For example, if you array enter =ROW(A1:A3) into cells B1:B5, the result will be 1 2 3 #N/A #N/A. The ROW(A1:A3) returns an array with only

'three elements, so Excel fills in the rest of the calling range with #N/A values.

'Example: Orienting An Array

'If your UDF creates a 1-dimensional array as its result, it can orient the array as either a row vector or a column vector so that is will

'be properly displayed in the worksheet cells without requiring the user to wrap your UDF result in a TRANSPOSE function. If the function

'was called from a row vector of cells (e.g., A1:E1), it does not need to be transposed. If the function was called from a column vector of

'cells (e.g., A1:A5), the array needs to be transposed. The code below looks at Application.Caller.Rows.Count and if this is greater than

'1, it tranposes the array before returning it to the caller. Note that this should be done only with single-dimensional arrays and only when

'the UDF is being called from a worksheet range. Therefore, you should first test Application.Caller with IsObject and then test

'Application.Caller.Rows.Count and Application.Caller.Columns.Count to test if it is being called from a row or column vector.

'For example,

Sub TEST\_ARRAY\_TRANSPOSE\_FUNC1()

Dim i As Long

Dim j() As Long

Dim NROWS As Long

Dim DATA\_RNG As Range

Dim SRC\_WSHEET As Worksheet

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

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

NROWS = 5

With Range(.Cells(1, 1), .Cells(NROWS, 1)) 'Rows.Count > 1

.FormulaArray = "=ARRAY\_TRANSPOSE\_FUNC1(" & NROWS & ")"

Debug.Print .Address

End With

With Range(.Cells(1, 1 + 1), .Cells(1, 1 + NROWS)) 'Rows.Count = 1

.FormulaArray = "=ARRAY\_TRANSPOSE\_FUNC1(" & NROWS & ")"

Debug.Print .Address

End With

With Range(.Cells(NROWS + 2, 1), .Cells(NROWS \* 2 + 1, NROWS)) 'Rows.Count > 1 but why

'it is repeating NROWS times the column array?

.FormulaArray = "=ARRAY\_TRANSPOSE\_FUNC1(" & NROWS & ")"

Debug.Print .Address

End With

End With

End Sub

Function ARRAY\_TRANSPOSE\_FUNC1(NN As Long)

Dim Result() As Long

Dim N As Long

ReDim Result(1 To NN)

For N = 1 To NN

Result(N) = N

Next N

If Application.Caller.Rows.Count > 1 Then

ARRAY\_TRANSPOSE\_FUNC1 = Application.Transpose(Result)

Else

ARRAY\_TRANSPOSE\_FUNC1 = Result

End If

End Function

'You can, of course, forego this and return the array as-is and leave it up to the user to use the TRANSPOSE

'function to properly orient the array.

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

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

'Lesson 3.7.4. Filling an Array

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

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

'Previously we created a VBA function that creates arrays for us. Now let's look at two functions that help us populate them.

'Lets first create a array\_fill method which fills an array with a certain quantity of a given value, starting at a certain

'index position. We will follow this model, however the parameters will be reordered and we will be starting the array at

'whatever index is specified by the existing array.

'First parameter: the array itself

'Second parameter: value to insert in each element

Function ARRAY\_FILL\_FUNC(ByVal DATA\_ARR As Variant, ByVal DATA\_VAL As Variant) As Variant

'This function works with both one and two dimensional arrays.

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

TEMP\_ARR = DATA\_ARR

If IS\_1D\_ARRAY\_FUNC(DATA\_ARR) Then

'The IS\_1D\_ARRAY\_FUNC function mentioned above is located in the module MATRIX\_RC\_VALIDATE\_FUNC

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

TEMP\_ARR(i) = DATA\_VAL

Next i

Else

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

For j = LBound(TEMP\_ARR, 2) To UBound(TEMP\_ARR, 2)

TEMP\_ARR(i, j) = DATA\_VAL

Next j

Next i

End If

ARRAY\_FILL\_FUNC = TEMP\_ARR

End Function

Sub TEST\_ARRAY\_FILL\_FUNC()

'You want an array (two dimensional) with 5 elements (index 1 through 5) filled with "Buy Signal":

Dim TEMP1\_ARR As Variant

TEMP1\_ARR = ARRAY\_CREATE\_FUNC(5, 2, False)

TEMP1\_ARR = ARRAY\_FILL\_FUNC(TEMP1\_ARR, "Buy")

Debug.Print TEMP1\_ARR(5, 2)

'You want an array (one dimensional) with 5 elements (index 0 through 4) filled with "Sell Signal":

Dim TEMP2\_ARR As Variant

TEMP2\_ARR = ARRAY\_CREATE\_FUNC(5, , True)

TEMP2\_ARR = ARRAY\_FILL\_FUNC(TEMP2\_ARR, "Sell")

Debug.Print TEMP2\_ARR(5 - 1)

End Sub

'Filling an Array with a Range of Values

'This method will only work with one dimensional arrays, and only works with numbers. For next lecture you will create

'a function that take strings.

Function ARRAY\_RANGE\_FUNC(ByVal SROW As Long, \_

ByVal NROWS As Long, \_

Optional ByVal k As Long = 1) As Variant

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

If SROW = 0 Then

ReDim TEMP\_ARR(0 To (NROWS / k))

Else

ReDim TEMP\_ARR(0 To ((NROWS / k) - (SROW - 1) - 1))

End If

' value of first array element is starting value

j = SROW

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

'This function wipes out an existing array (if any) and overwrites it with a range of values as specified by

'the input parameters. If you do not specify a step, it increments numbers by one with each iteration.

TEMP\_ARR(i) = j

j = j + k

Next i

ARRAY\_RANGE\_FUNC = TEMP\_ARR

End Function

Sub TEST\_ARRAY\_RANGE\_FUNC()

Dim i As Long

'You want to create an array that holds the numbers 1 through 10.

Dim TEMP1\_ARR As Variant

TEMP1\_ARR = ARRAY\_RANGE\_FUNC(1, 10, 1)

Debug.Print "TEMP1\_ARR -----------------------"

For i = LBound(TEMP1\_ARR) To UBound(TEMP1\_ARR)

Debug.Print TEMP1\_ARR(i)

Next i

'Ex: You want to create an array that holds the numbers 1 through 10, skipping every other number.

Dim TEMP2\_ARR As Variant

TEMP2\_ARR = ARRAY\_RANGE\_FUNC(1, 10, 2)

Debug.Print "TEMP2\_ARR -----------------------"

For i = LBound(TEMP2\_ARR) To UBound(TEMP2\_ARR)

Debug.Print TEMP2\_ARR(i)

Next i

'The output might not be exactly what you expect. The above returns "1, 3, 5, 7, 9" because if you count by 2s and start

'from 1, the next number in the sequence is 11 which is greater than the upper bound we specified. The function will not

'go beyond the upper bound even if doing so leaves the upper bound out of the array. Obviously, it works best with certain

'number sequences such as

Dim TEMP3\_ARR As Variant

TEMP3\_ARR = ARRAY\_RANGE\_FUNC(0, 100, 10)

Debug.Print "TEMP3\_ARR -----------------------"

For i = LBound(TEMP3\_ARR) To UBound(TEMP3\_ARR)

Debug.Print TEMP3\_ARR(i)

Next i

End Sub

'Example: CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC

Sub TEST\_CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC()

Dim i As Long

Dim j As Long

Dim NSIZE As Long

Dim NROWS As Long

Dim SRC\_ARR As Variant

Dim DST\_ARR() As Double

NROWS = 5

ReDim SRC\_ARR(1 To NROWS, 1 To 1)

j = 1

For i = 1 To NROWS

SRC\_ARR(i, 1) = j

j = j + 1

Next i

NSIZE = CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC(SRC\_ARR, DST\_ARR)

Debug.Print "SRC\_ARR(1 To NROWS, 1 To 1): Size=" & NSIZE

For i = LBound(DST\_ARR, 1) To UBound(DST\_ARR)

Debug.Print "i=" & i & " Value=" & DST\_ARR(i)

Next i

SRC\_ARR = DST\_ARR

Debug.Print

NSIZE = CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC(SRC\_ARR, DST\_ARR)

Debug.Print "SRC\_ARR(1 To NROWS): Size=" & NSIZE

For i = LBound(DST\_ARR, 1) To UBound(DST\_ARR)

Debug.Print "i=" & i & " Value=" & DST\_ARR(i)

Next i

End Sub

'The following routine convert different input types (i.e. horizontal or vertical spreadsheet ranges, or

'1D or 2D arrays of doubles or variants) into a uniform format for processing by the vector functions with

'minimum overhead.

Function CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC(ByRef DATA\_ARR1 As Variant, \_

ByRef DATA\_ARR2() As Double) As Variant

'The CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC function performs the necessary conversion of the

'spreadsheet range into a 1D array of doubles. That is what the code below does:

Dim i As Long

Dim j As Long

Dim k As Long

Dim NSIZE As Long

'On Error GoTo ERROR\_LABEL

'If the input vector (DATA\_ARR1) is a range, convert it into a Variant array

If TypeName(DATA\_ARR1) = "Range" Then DATA\_ARR1 = DATA\_ARR1.Value2

NSIZE = UBound(DATA\_ARR1) - LBound(DATA\_ARR1) + 1

If IS\_2D\_ARRAY\_FUNC(DATA\_ARR1) Then 'check if DATA\_ARR1 is 2 dimension

'If DATA\_ARR1 has 2 dimensions, check the orientation (var k), assuming vertical if it has two or

'more rows, or horizontal if it has 1 row.

k = 1

If NSIZE = 1 Then

NSIZE = UBound(DATA\_ARR1, 2) - LBound(DATA\_ARR1, 2) + 1 'If DATA\_ARR1 is horizontal, find the number of columns

k = 2

End If

End If

ReDim DATA\_ARR2(1 To NSIZE): j = 1 'Redimension the output array, DATA\_ARR2, as a base 1, 1D array.

If IS\_1D\_ARRAY\_FUNC(DATA\_ARR1) Then 'Check if DATA\_ARR1 is one dimension (calling the IS\_1D\_ARRAY\_FUNC)

For i = LBound(DATA\_ARR1) To UBound(DATA\_ARR1)

DATA\_ARR2(j) = DATA\_ARR1(i) 'Assign the values of DATA\_ARR1 to DATA\_ARR2

j = j + 1

Next i

Else 'If IS\_2D\_ARRAY\_FUNC(DATA\_ARR1) Then

'Assign the number of values in DATA\_ARR2 to the function return value

If k = 1 Then

For i = LBound(DATA\_ARR1) To UBound(DATA\_ARR1)

DATA\_ARR2(j) = DATA\_ARR1(i, 1)

j = j + 1

Next i

Else

For i = LBound(DATA\_ARR1, 2) To UBound(DATA\_ARR1, 2)

DATA\_ARR2(j) = DATA\_ARR1(1, i)

j = j + 1

Next i

End If

End If

'Note that the function returns the size of the array DATA\_ARR2, but because DATA\_ARR2 was passed as an empty

'double array by reference, its contents are available to the calling procedure.

CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC = NSIZE

Exit Function

ERROR\_LABEL:

CONVERT\_VECTOR\_TO\_1D\_ARRAY\_FUNC = CVErr(xlErrNA)

End Function