**ARRAY WORKBOOK**

This tutorial contains 8 different lessons on array function in VBA.

Lesson1\_Array\_Dimensions.........................................................................................3

1. IS\_1D\_ARRAY\_FUNC ...................................................................................3

2. IS\_2D\_ARRAY\_FUNC....................................................................................5

Lesson2\_Array\_Transpose............................................................................................8

1. MATRIX\_TRANSPOSE\_FUNC.........................................................................9

2. MATRIX\_REVERSE\_FUNC.............................................................................13

Lesson3\_Array\_Redim..................................................................................................19

1. ARRAY\_DIMENSION\_FUNC...........................................................................19

2. MATRIX\_REDIM\_FUNC.................................................................................22

3. MATRIX\_RESIZE\_FUNC..................................................................................28

Lesson4\_Array\_Base......................................................................................................35

1. MATRIX\_CHANGE\_BASE\_ZERO\_FUNC...........................................................35

2. MATRIX\_CHANGE\_BASE\_ONE\_FUNC............................................................38

Lesson5\_Array\_Add........................................................................................................42

1. MATRIX\_ADD\_ROWS\_FUNC .........................................................................42

2. MATRIX\_ADD\_COLUMNS\_FUNC....................................................................46

Lesson6\_Array\_Remove..................................................................................................49

1. MATRIX\_REMOVE\_ROWS\_FUNC ...................................................................49

2. MATRIX\_REMOVE\_COLUMNS\_FUNC..............................................................55

Lesson7\_Array\_Get..........................................................................................................60

1. MATRIX\_GET\_ROW\_FUNC .............................................................................60

2. MATRIX\_GET\_COLUMN\_FUNC.......................................................................64

3. MATRIX\_GET\_SUB\_MATRIX\_FUNC.................................................................67

Lesson8\_Array\_Match.....................................................................................................71

1. ARRAY\_MATCH\_DATA\_FUNC..........................................................................72

2. ARRAY\_REMOVE\_DUPLICATES\_FUNC.............................................................77

**VBA Application and Function Reference**

**Option Explicit** : In VBA an Option Explicit statement should always appear in the declarations section of a module. Option Explicit prevents Visual Basic from making implicit type conversions that may involve loss of data. For your assignments you should always have the Option Explicit statement at the top in your programs.

**Option Base 1** : Used at module level to declare the default lower bound for array subscripts.

**ByRef** : Means By Reference. The memory address of the argument is passed to the Called Procedure and any change to that parameter by the Called Procedure is made to the value in the Calling Procedure. Therefore, the actual value is not passed to the Called Procedure. Since ByRef transfer the memory location, changing the variable will apply to the original variable

**ByVal**: Specifies that an argument is passed in such a way that the called procedure or property cannot change the value of a variable underlying the argument in the calling code. ByVal transfer a number like a constant.

**Long:** Identifier of a value type. Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Note: 64 bits. Big integers, but still no decimal numbers.

**VBA code with Annotation**

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

Option Base 1 'The "Option Base" statement allows to specify 0 or 1 as the default first index of arrays.

**Lesson1\_Array\_Dimensions**

Arrays you create in VBA can have as many as 60 dimensions — although you rarely need more than 2 or 3 dimensions in an array. This lesson contain two functions that design to test if an array is one-dimensional or two-dimensional array : 1. IS\_1D\_ARRAY\_FUNC and 2. IS\_2D\_ARRAY\_FUNC. We assume the array we are testing is either a vector or a matrix. We will not test if the array has more than two dimensions.

**1. IS\_1D\_ARRAY\_FUNC**

This functions will check if the tested array is a one-dimensional (e.g. vector) array or not by check if the array has a upper bound in dimension one and two.

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**NSIZE** : The return of the highest value the subscript of a specified dimension can contain.

**DATA\_VECTOR** : Copies over the content of DATA\_RNG. This is also the array being returned by the main function.

**VBA Application and Function Reference**

**UBOUND(Array, Rank/ Dimension)** : Returns the highest available subscript for the indicated dimension of an array. eg If Array (1 to 100, 1 to 5), then UBound(Array,1)=100. This number may be different from the total number of elements in an array.

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. You can use any line label or line number. eg. ERROR\_LABEL . The specified line must be in the same procedure as the On Error statement. Make sure include a colon ":" after the line label so that VBA recognizes it as a label. e.g. "ERROR\_LABEL:". To end the macro when an error has occurred, place the error handling routine either immediately before the End Sub (or End Function) statement or use the Exit Sub (or Exit Function) statement.

**Process**

**Step 1- Set Up the Error Handler:**

When an error occur, VBA will resumes execution at the line label, ERROR\_LABEL:. Since it is placed immediately before the End Function, VBA will end the macro. The return will be return saved before the error occurred.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

**Step 2 - Make a Copy of the Testing Array:**

We create a new variable named DATA\_VECTOR, which contains all the value of DATA\_RNG. (Technically, we could’ve saved some memory by only using DATA\_RNG if there is no other code to be performed.) If we change DATA\_VECTOR , DATA\_RNG will not change. Then, the output for the IS\_ARRAY\_FUNC is base on theDATA\_VECTOR.

Dim DATA\_VECTOR As Variant

DATA\_VECTOR = DATA\_RNG

**Step 3- Set Up Default Return:**

By default, this function will return False. Unless there is an error occurring in the testing function, Excel will respond with a message. The message will indicate that your code generated a run-time error. Eg. The testing dimension of that array does not exist. It will return error number 9. This indicate "Array is Nothing".

IS\_1D\_ARRAY\_FUNC = False

**Step 4- Test if Array has First Dimension:**

NSIZE is declared as a variable of Long type. NSIZE is set to equal to the size of DATA\_VECTOR's first dimension. Ubound will return a value if that dimension exist and continue run the function, or else it will raise an error. Error label is called and the function will return the value saved before the error which is False. If Ubound returns an integer, then the next line is executed and IS\_1D\_ARRAY\_FUNC returns true. VBA will continue on to the next line of code after until it raises an error or the stop code.

Dim NSIZE As Long

NSIZE = UBound(DATA\_VECTOR, 1)

IS\_1D\_ARRAY\_FUNC = True

**Step 5- Test if Array has Second Dimension:**

At this point, DATA\_VECTOR is known to be one dimensional. UBound(DATA\_VECTOR, 2) tests for the second dimension. The logic is similar to the last step. If DATA\_VECTOR does not have a second dimension, the error label code is executed and the function will return true. If DATA\_VECTOR has the second dimension, this indicates the array is a 2D array or 2+D array. We continue on to the next line and IS\_1D\_Array\_FUNC returns false.

NSIZE = UBound(DATA\_VECTOR, 2)

IS\_1D\_ARRAY\_FUNC = False

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

True if the UBOUND(DATA\_VECTOR, 1) returns an integer AND UBOUND(DATA\_VECTOR, X => 2) raises error. This indicates the array is a one-dimensional array. False if UBOUND(DATA\_VECTOR, 1) raises error or UBOUND(DATA\_VECTOR, 2) returns an integer. This indicates the array tested is not a one-dimensional array. When an error occur, the function will stop running and the return will be the last return value saved.

**VBA Code with Annotation**

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

Dim NSIZE As Long

Dim DATA\_VECTOR As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error 'and look for the line label. Excel will go to the ERROR\_LABEL below and end the function.

DATA\_VECTOR = DATA\_RNG 'Step 2. Transferring DATA\_RNG to DATA\_VECTOR. The change happen to 'DATA\_VECTOR will not apply to DATA\_RNG

IS\_1D\_ARRAY\_FUNC = False 'Step 3. This Function by default will return 'false'.

NSIZE = UBound(DATA\_VECTOR, 1) 'Step 4. We assign NSIZE as the size of the first dimension

'of DATA\_VECTOR.

IS\_1D\_ARRAY\_FUNC = True 'If the first dimension exist, then function return 'true'.

NSIZE = UBound(DATA\_VECTOR, 2) 'Step 5. We are asking for the size of DATA\_VECTOR's 2nd dimension.

'If DATA\_VECTOR only has is a 1 d array, then there would will be an error. We skip to error label.

'The return for this function will remain 'true'. If the array has a second dimension or more, there will not 'be an error and the code will continue to next line.

IS\_1D\_ARRAY\_FUNC = False ' Now, the function will return 'false'.

Exit Function ' Step 6- end function.

ERROR\_LABEL:

End Function

**2. IS\_2D\_ARRAY\_FUNC**

This functions will check if the tested array is a two-dimensional (e.g. matrix) array or not by check if the array has a upper bound in dimension one and two. A few items already present in the previous function and will not be explain in detail here.

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**NSIZE** : The return of the highest value the subscript of a specified dimension can contain.

**DATA\_MATRIX** : Copies over the content of DATA\_RNG. This is also the array being returned by the main function.

**VBA Application and Function Reference**

**UBOUND(Array, Rank/ Dimension)** : Returns the highest available subscript for the indicated dimension of an array.

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. Since the error handling routine is placed immediately before the End Function statement , any error will end the macro.

**Process**

**Step 1- Set Up the Error Handler:**

When an error occur, VBA will raise error and execute ERROR\_LABEL:. Since it is placed immediately before the End Function, VBA will end the macro. The return will be the value saved before the error occurred.

On Error GoTo ERROR\_LABEL

ERROR\_LABEL:

**Step 2 - Make a Copy of the Testing Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. If we change DATA\_MATRIX, DATA\_RNG will not change. Then, the output for the IS\_2D\_ARRAY\_FUNCis base on theDATA\_MATRIX.

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3- Set Up Default Return:**

By default, this function will return false.

IS\_2D\_ARRAY\_FUNC = False

**Step 4- Test if Array has the First and Second Dimension:**

We set NSIZE equal to the size of DATA\_VECTOR's first dimension and then its second dimension. UBOUND will return a value if the dimension exists and the function is continued. Else it will raise an error. If DATA\_VECTOR does not have a first dimension or a second dimension, error label is executed and the IS\_2D\_Array\_FUNC returns False. Else, function continues to run and value return is True.

Dim NSIZE As Long

NSIZE = UBound(DATA\_MATRIX, 1)

NSIZE = UBound(DATA\_MATRIX, 2)

IS\_2D\_ARRAY\_FUNC = True

**Step 5- End Function:**

End the function. We assume the array we are testing is either a vector or a matrix. We will not test if the array has a third dimension.

Exit Function

End Function

**Returns**

True if UBOUND(DATA\_VECTOR, 1) AND UBOUND(DATA\_VECTOR, 2) returns an integer .This indicate the array is a two-dimensional array. False if UBOUND(DATA\_VECTOR, 1) or UBOUND(DATA\_VECTOR, 2) raises an error. When an error occurs, the function will stop.

**VBA Code with Annotation**

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

Dim NSIZE As Long

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and end the function.

DATA\_MATRIX = DATA\_RNG 'Step 2. Transferring DATA\_RNG to DATA\_MATRIX.

IS\_2D\_ARRAY\_FUNC = False 'Step 3. This Function by default will return 'false'.

NSIZE = UBound(DATA\_MATRIX, 1)

NSIZE = UBound(DATA\_MATRIX, 2) 'Step 5. We are asking for the size of DATA\_MATRIX 's 1st and 2nd 'dimension. If DATA\_MATRIX is missing either of the two dimension, then there would will be an error. 'We skip to error label. The return for this function will remain 'false'. If the array has both dimension, 'then the code will continue to next line.

IS\_2D\_ARRAY\_FUNC = True ' Now, the function will return 'true'.

Exit Function ' Step 6- end function.

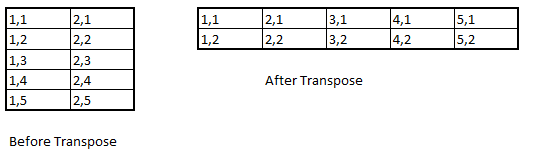
ERROR\_LABEL:

End Function

**Lesson2\_Array\_Transpose**

This lesson contain two functions: 1. MATRIX\_TRANSPOSE\_FUNC and 2. MATRIX\_REVERSE\_FUNC.

MATRIX\_TRANSPOSE\_FUNC transpose an array.



MATRIX\_REVERSE\_FUNC reverse the entries vertically in an array.



**VBA Application and Function Reference**

**UBOUND(Array, Rank/ Dimension)** : Returns the highest available subscript for the indicated dimension of an array. eg If Array (1 to 100, 1 to 5), then UBound(Array,1)=100. This number may be different from the total number of elements in an array. (Also seen in Lesson 1. )

**LBOUND (Array, Rank/ Dimension**): Returns the lowest available subscript for the indicated dimension of an array. eg If Array (1 to 100, 1 to 5), then LBound(Array,1)=1.

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1. )

**IF-Then-Else Statement**:  It returns a value if a specified condition evaluates to TRUE, or another value if it evaluates to FALSE. **Condition\_1** to **condition\_n** are evaluated in the order listed. Once a **condition** is found to be true, the **IF-THEN-ELSE statement** will execute the corresponding code and not evaluate the conditions any further. Result\_1 to result\_n is the code that is executed once a condition is found to be true. If no condition is met, then the Else portion of the IF-THEN-ELSE statement will be executed.

If condition\_1 Then

result\_1

Else

If condition\_n Then

result \_n

End If

**ReDim Statement**: Reallocates storage space for an array variable. You can use the ReDim statement to change the size of one or more dimensions of an array that has already been declared. If you have a large array and you no longer need some of its elements, ReDim can free up memory by reducing the array size. On the other hand, if your array needs more elements, ReDim can add them. The ReDim statement is intended only for arrays. It's not valid on scalars (variables that contain only a single value), collections, or structures. Note that if you declare a variable to be of type Array, the ReDim statement doesn't have sufficient type information to create the new array.

**For...Next Statement:** Repeats a group of statements a specified number of times.

For counter [ As datatype ] = start To end [ Step step ]

[ statements ]

[ Continue For ]

[ statements ]

[ Exit For ]

[ statements ]

Next [ counter ]

**Nesting loop:** Putting one loop within another.

**Counter variable:** Required in the For statement. Numeric variable. The control variable for the loop.

**1. MATRIX\_TRANSPOSE\_FUNC**

This functions transpose an array. We assume the array we are transposing is either a vector or a matrix which means 1D and 2D array, respectively. This function can be use to adjust the direction of the arrays when we print it in Excel. To visualize in Excel, the first dimension can be seen as the rows and second dimension are the columns. 1D arrays are printed horizontally by default unless you transpose it.

This function contains two IF-Then-Else statement because we will transpose 1D and 2D array differently. We use a nesting loop of For...Next Statement to put the content into transposed array.

**Parameters**

**DATA\_RNG** : Array to be transposed. Set of data in array.

**Variable**

**i**: Counter variable for a dimension. If the transpose array is a matrix, then i is the counter of the 2nd dimension of TEMP\_MATRIX and range from SCOLUMN to NCOLUMNS. If the transpose array is a vector , then i is the counter of the 1st dimension of TEMP\_MATRIX and range from SROW to NROWS.

**j:** Counter variable for a dimension in an 2D array. If the transpose array is a matrix, then j is the counter of the 1st dimension of TEMP\_MATRIX and range from SROW to NROWS. 1D array does not require j.

**SROW**: Returns the lowest available subscript for the first dimension of DATA\_MATRIX. It will later use to set the size of the second dimension of the transpose array.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of DATA\_MATRIX. It will later use to set the size of the first dimension of the transpose 2D array.

**NROWS**: The return of the highest value the subscript for DATA\_MATRIX's first dimension can contain. It will later use to set the size of the second dimension of the transpose array.

**NCOLUMN:** The return of the highest value the subscript for DATA\_RNG's second dimension can contain. It will later use to set the size of the transpose array's first dimension if we are transposing a matrix.

**TEMP\_MATRIX**: Temporary holder for the transposed DATA\_MATRIX array.

**DATA\_MATRIX** : Copies over the content of DATA\_RNG. This is also the array being returned by the main function.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_TRANSPOSE\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_TRANSPOSE\_FUNC = Err.Number

**Step 2 - Make a Copy of the Transpose Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. If we change DATA\_MATRIX, DATA\_RNG will not be changed. Then, the output for the IS\_ARRAY\_FUNC is base on theDATA\_MATRIX.

Dim DATA\_MATRIX As Variant

DATA\_ MATRIX = DATA\_RNG

**Step 3.0 - First IF-Then-Else Statement. Transpose a two-dimensional array:**

IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) will return True if the transpose array is a 2-dimensional array, otherwise False. ( Detail of IS\_2D\_ARRAY\_FUNC can be find in lesson 1) If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True then the following lines are executed. If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = False which means the condition is not met, then VBA will executed the Else portion of the IF-THEN-ELSE statement.

If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True Then

...

Else

**Step 3.1 - Define the range of the 2D array:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA\_VECTOR's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA\_VECTOR's first and second dimension, respectively. In general, rows are consider as the first dimension and columns are consider as the second dimension.

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

SROW = LBound(DATA\_MATRIX, 1)

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

**Step 3.2- Transpose Matrix dimension :**

We create a TEMP\_MATRIX variable. We use "ReDim" (re-size) here because we have already declare this array. We set up the size of the TEMP\_MATRIX using the lower and upper bound of DATA\_MATRIX , which is pass by DATA\_RNG. We swap the first and second dimension size information when we create TEMP\_MATRIX. eg. Use S/NCOLUMN to set up the size of first dimension, instead of second dimension. Therefore, we achieved transposing the shape of the matrix. In this step, we have not obtain the content from DATA\_MATRIX .

Dim TEMP\_MATRIX As Variant

ReDim TEMP\_MATRIX(SCOLUMN To NCOLUMNS, SROW To NROWS)

**Step 3.3- For...Next Statement. Input Data into After Transpose Array (TEMP\_MATRIX) :**

We create counter variable i and j variable for the two dimension of the array. Since we are input the content from DATA\_MATRIX into its transpose array, TEMP\_MATRIX. The position of i and j are swap for those two array.

Dim i As Long

Dim j As Long

For j = SCOLUMN To NCOLUMNS

For i = SROW To NROWS

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

Next i

Next j

i is the counter for the second dimension in TEMP\_MATRIX but counter for the first dimension in DATA\_MATRIX. j is used vise versa.

j will run between SCOLUMN and NCOLUMNS. i will run between SROW and NROWS. The first data we will transfer is positioned at j=SCOLUMN and i=SROW. Then we assign value for that point from DATA\_MATRIX to TEMP\_MATRIX. Then we move on to next i , i will equal to SROW+1 and continue to fill the TEMP\_MATRIX. j will remain to be SCOLUMN until i increase to i=NROWS, which means we only move to the next j (j+1) when all the column in the first row are filled. Next j will direct VBA to start filling in the second row. This will keep looping until j eventually equal to NCOLUMNS which means TEMP\_MATRIX is full.

**Step 4.0- Second IF-Then-Else Statement. Transpose a one-dimensional array:**

VBA will run this part if DATA\_MATRIX is not a two dimensional array. IS\_1D\_ARRAY\_FUNC(DATA\_MATRIX) will return True if the transpose array is a 1-dimensional array and the following lines are executed, otherwise False and VBA will executed the Else portion of the IF-THEN-ELSE statement . ( Detail of IS\_1D\_ARRAY\_FUNC can be find in lesson 1) . As mentioned before this function will only transpose a 1D or 2D array. If the array is not a 1D or 2D array, then it raise an error. VBA will skip to ERROR\_LABEL.

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

Else

GoTo ERROR\_LABEL

End If

**Step 4.1 - Define the range of the 1D array:**

Since this is a 1D array, we will only have 1 set of lower bound and upper bound. Although, 1D array is printed horizontally by default. We do not need the ",#" after DATA\_MTRIX because it is by default to return the value of the only dimension. SROW and NROWS will return the lowest and highest possible subscript of DATA\_MATRIX's first dimension, respectively.

SROW = LBound(DATA\_MATRIX)

NROWS = UBound(DATA\_MATRIX)

**Step 4.2- Transpose Matrix dimension :**

We set up the size of the TEMP\_MATRIX using the lower and upper bound of DATA\_MATRIX. As mentioned before, 1D arrays are printed horizontally unless we transpose it. This means DATA\_MATRIX has the size of( SROW TO SROW, SROW TO NROWS). We can use SROW to SROW here, because 1D array's second dimension element size equal to 1. SROW to SROW returns value of 1. Therefore, we achieved transposing the array from printed horizontally to vertically.

ReDim TEMP\_MATRIX(SROW To NROWS, SROW To SROW)

**Step 4.3- For...Next Statement. Input Data into the After Transpose Array :**

We only need to use counter variable i for the one dimension array. Since we are input the content from DATA\_MATRIX into its transpose array, TEMP\_MATRIX. For TEMP\_MATRIX i will be fix at the first dimension position .

For i = SROW To NROWS

TEMP\_MATRIX(i, SROW) = DATA\_MATRIX(i)

Next i

i will run between SROW and NROWS. The first data we will transfer is positioned at i=SROW. Then we assign value for that point from DATA\_MATRIX to TEMP\_MATRIX. Then we move on to next i , i will equal to SROW+1 and continue to fill the TEMP\_MATRIX. This will keep looping until i eventually equal to NROWS which means TEMP\_MATRIX is full.

**Step 5- Return After Transposed Array:**

Return the after transposed array of DATA\_RNG.

MATRIX\_TRANSPOSE\_FUNC = TEMP\_MATRIX

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which is the after transposed array of DATA\_RNG. When an error raise, the function will stop running and the return will be the error number.

**VBA Code with Annotation**

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

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant 'Some point, we're going to use an array referred to as TEMP\_MATRIX

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2. Transferring DATA\_RNG to DATA\_MATRIX.

If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True Then 'Step 3.0 - Check if it is a 2D array. If it is not a 2D array skip to Else. If it is a 2D array continue execution.

SROW = LBound(DATA\_MATRIX, 1) 'Step 3.1 - Find the lower and upper bounds of the dimensions.

SCOLUMN = LBound(DATA\_MATRIX, 2) 'Rows are the 1st dimension and column are the 2nd dimension.

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

ReDim TEMP\_MATRIX(SCOLUMN To NCOLUMNS, SROW To NROWS) 'Step 3.2- Transpose dimension

'We have to resize the temp array because we're transposing it. Swap the column and row size. eg 'ReDim TEMP\_MATRIX's 1st dimension size transpose to DATA\_MATRIX's second dimension size.

For j = SCOLUMN To NCOLUMNS 'Step 3.3- Input Data into After Transpose Array(TEMP\_MATRIX)

For i = SROW To NROWS

TEMP\_MATRIX(j, i) = DATA\_MATRIX(i, j) 'Assign value for each position from DATA\_MATRIX to TEMP\_MATRIX until all the columns and rows are done.

Next i 'Move to the next i, refer to next row in DATA\_MATRIX and next column in TEMP\_MATRIX.

Next j ' Move to the next j when i=NROWS.

Else 'If DATA\_RNG is not a 2D array.

If IS\_1D\_ARRAY\_FUNC(DATA\_MATRIX) = True Then 'Step 4.0- Check if it is a 1D array. If it is not a 1D 'array skip to Else. If it is a 1D array execution continue .

SROW = LBound(DATA\_MATRIX) 'Step 4.1 - Define the range of the 1D array

NROWS = UBound(DATA\_MATRIX)

ReDim TEMP\_MATRIX(SROW To NROWS, SROW To SROW) 'Step 4.2- Transpose Matrix dimension

For i = SROW To NROWS 'Step 4.3- For...Next Statement. Input Data into the After Transpose Array

TEMP\_MATRIX(i, SROW) = DATA\_MATRIX(i) 'Assign value for each position in TEMP\_MATRIX.

Next i

Else

GoTo ERROR\_LABEL

End If

End If

MATRIX\_TRANSPOSE\_FUNC = TEMP\_MATRIX 'Step 5- Return After Transposed Array

Exit Function 'Step 6- End Function

ERROR\_LABEL:

MATRIX\_TRANSPOSE\_FUNC = Err.Number 'When an error is raised, return Error Number.

End Function

**2. MATRIX\_REVERSE\_FUNC**

This functions reverse the entries in an array, eg. from N X M to M X N. We assume the array we are reversing is either a vector or a matrix which represent 1D and 2D array, respectively.

This function contains two IF-Then-Else statement because we will reverse 1D and 2D array differently. We use a nesting loop of For...Next Statement to input the entries.Some of the logic apply in this function are similar to MATRIX\_TRANSPOSE\_FUNC above.

**Parameters**

**DATA\_RNG** : Array to be reversed. Set of data in array.

**Variable**

**i:** Counter variable for a dimension. If the transpose array is a matrix, then i is the counter of the 2nd dimension of TEMP\_MATRIX and range from SCOLUMN to NCOLUMNS. If the transpose array is a vector , then i is the counter of the 1st dimension of TEMP\_MATRIX and range from SROW to NROWS.

**j**: Counter variable for a dimension in an 2D array. If the transpose array is a matrix, then j is the counter of the 1st dimension of TEMP\_MATRIX and range from SROW to NROWS. 1D array does not require j.

**SROW**: Returns the lowest available subscript for the first dimension of DATA\_RNG.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of DATA\_RNG.

**NROWS**: The return of the highest value the subscript for DATA\_RNG's first dimension can contain.

**NCOLUMN:** The return of the highest value the subscript for DATA\_RNG's second dimension can contain.

**TEMP\_MATRIX**: Temporary holder for the transposed DATA\_MATRIX array.

**DATA\_MATRIX** : Copies over the content of DATA\_RNG. This is also the array being returned by the main function.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_TRANSPOSE\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_REVERSE\_FUNC = Err.Number

**Step 2 - Make a Copy of the Transpose Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. If we change DATA\_MATRIX, DATA\_RNG will not be changed. Then, the output for the MATRIX\_REVERSE\_FUNC is base on theDATA\_MATRIX.

Dim DATA\_MATRIX As Variant

DATA\_ MATRIX = DATA\_RNG

**Step 3.0 - First IF-Then-Else Statement. Transpose a two-dimensional array:**

IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) will return True if DATA\_MATRIX is a 2-dimensional array, otherwise False. ( Detail of IS\_2D\_ARRAY\_FUNC can be find in lesson 1) If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True then the following lines are executed. If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = False which means the condition is not met, then VBA will executed the Else portion of the IF-THEN-ELSE statement.

If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True Then

...

Else

**Step 3.1 - Define the range of the 2D array:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA\_MATRIX's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA\_MATRIX's first and second dimension, respectively. In general, rows are consider as the first dimension and columns are consider as the second dimension.

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

NCOLUMNS = UBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

SCOLUMN = LBound(DATA\_MATRIX, 2)

SROW = LBound(DATA\_MATRIX, 1)

**Step 3.2- Copy Matrix Dimension**

We create a TEMP\_MATRIX variable. We use "ReDim" (re-size) here because we have already declare this array. We set up the size of the TEMP\_MATRIX using the same lower and upper bound of DATA\_MATRIX , which is pass by DATA\_RNG. We only want to reverse the entries in DATA\_MATRIX without, therefore TEMP\_MATRIX need to have the same structure as DATA\_MATRIX. In this step, we have not obtain the content from DATA\_MATRIX .

Dim TEMP\_MATRIX As Variant

ReDim TEMP\_MATRIX(SROW To NROWS, SCOLUMN To NCOLUMNS)

**Step 3.3- For...Next Statement. Input Data Reversely into TEMP\_MATRIX**

We create counter variable i and j variable for the array's two dimension. The position of i and j remain the same for those two array. i is the counter for the first dimension and j is the counter for the second dimension. j will run between SCOLUMN and NCOLUMNS. i will run between SROW and NROWS.

Dim i As Long

Dim j As Long

For j = SCOLUMN To NCOLUMNS

For i = SROW To NROWS

TEMP\_MATRIX(NROWS + SROW - i, j) = DATA\_MATRIX(i, j)

Next i

Next j

Since we are input the entries from DATA\_MATRIX into TEMP\_MATRIX in the vertically reverse order, the entry of the first dimension is flipped for those two array. The entry at (i,j) in DATA\_MATRIX , will be enter at (NROWS + SROW - i, j) in TEMP\_MATRIX. NROWS+SROW -i give you the relative position of the entry if you count i entry from bottom up.

Since the first i=SROW, the first entry in TEMP\_MATRIX is entered at (NROWS, j). After assigning each value, we will move on to the Next i. Next i equal to SROW+1 , and it will continuously loop back to fill the TEMP\_MATRIX . j will remain to be SCOLUMN until i equal to NROWS, which means all the rows in the first column are filled. Next j will direct VBA to start filling in the second column. This will keep looping until j eventually equal to NCOLUMNS which means TEMP\_MATRIX is full.

**Step 4.0- Second IF-Then-Else Statement. Transpose a one-dimensional array**

VBA will run this part if the array is not a two dimensional array. IS\_1D\_ARRAY\_FUNC(DATA\_MATRIX) will return True if the transpose array is a 1-dimensional array and the following lines are executed, otherwise False and VBA will executed the Else portion of the IF-THEN-ELSE statement . ( Detail of IS\_1D\_ARRAY\_FUNC can be find in lesson 1) . As mentioned before this function will only apply to a 1D or 2D array. If the array is not a 1D or 2D array, then it raise an error. VBA will skip to ERROR\_LABEL.

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

Else

GoTo ERROR\_LABEL

End If

**Step 4.1 - Define the range of the 1D array**

SROW and NROWS will return the lowest and highest possible subscript of DATA\_MATRIX's first dimension, respectively.

SROW = LBound(DATA\_MATRIX, 1)

NROWS = UBound(DATA\_MATRIX, 1 )

**Step 4.2- Transpose Array into a Vertical Array**

We set up the size of the TEMP\_MATRIX using the lower and upper bound of DATA\_MATRIX. As mentioned before, 1D arrays are printed horizontally unless we transpose it. Therefore, we need to transpose the array into a vertical array. This means DATA\_MATRIX has the size of( SROW TO SROW, SROW TO NROWS). We can use SROW to SROW here, because 1D array's second dimension has element size equal to 1. SROW to SROW returns value of 1. In this step, we have not obtain the content from DATA\_MATRIX .

ReDim TEMP\_MATRIX(SROW To NROWS, SROW To SROW)

**Step 4.3- For...Next Statement. Input Data Reversely into the Vertical Array**

This has the same logic as Step 3.3. However, we do not need to use j as a counter variable. This is because of the array is a 1D array. After transposing vertically, this array will only have one column .i will run between SROW and NROWS. We will more to Next i after we enter the entry from DATA\_MATRIX to TEMP\_MATRIX. This will keep looping until i eventually equal to NROWS which means TEMP\_MATRIX is full.

For j = SCOLUMN To NCOLUMNS

For i = SROW To NROWS

TEMP\_MATRIX(NROWS + SROW - i, SROW) = DATA\_MATRIX(i)

Next i

Next j

**Step 5- Return Reverse Entries Array**

Return the array with reverse entries of DATA\_RNG.

MATRIX\_TRANSPOSE\_FUNC = TEMP\_MATRIX

**Step 6- End Function**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which has the reverse entries of DATA\_RNG. When an error raise, the function will stop running and the return will be the error number.

**VBA Code with Annotation**

Function MATRIX\_REVERSE\_FUNC(ByRef DATA\_RNG As Variant) ' dont have something to test reverse?

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant 'Some point, we're going to use an array referred to as TEMP\_MATRIX

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2. Transferring DATA\_RNG to DATA\_MATRIX.

If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True Then 'Step 3.0 - Check if it is a two-dimensional array

'If it is not a 2D array skip to Else. If it is a 2D array continue execution.

NCOLUMNS = UBound(DATA\_MATRIX, 2) 'Step 3.1 - Find the lower and upper bounds.

NROWS = UBound(DATA\_MATRIX, 1) 'Rows are in the first dimension and column in the second

SCOLUMN = LBound(DATA\_MATRIX, 2)

SROW = LBound(DATA\_MATRIX, 1)

ReDim TEMP\_MATRIX(SROW To NROWS, SCOLUMN To NCOLUMNS) 'Step 3.2- Copy Matrix Dimension

'We only want to reverse the entries in DATA\_MATRIX. TEMP\_MATRIX has the same dimension size.

For j = SCOLUMN To NCOLUMNS 'Step 3.3- Input Data into After Transpose Array(TEMP\_MATRIX)

For i = SROW To NROWS

TEMP\_MATRIX(NROWS + SROW - i, j) = DATA\_MATRIX(i, j) 'Enter each DATA\_MATRIX's entry into TEMP\_MATRIX at corresponding position until all the columns and rows are done

Next i 'Next i=i+1,nex t row. Moving vertically.

Next j ' Move to the next j ( column )when i=NROWS.

Else 'If DATA\_MATRIX is not a 2D array.

If IS\_1D\_ARRAY\_FUNC(DATA\_MATRIX) Then 'Step 4.0- Check if it is a 1D array. If it is not a 1D

'array skip to Else. If it is a 1D array execution continue .

SROW = LBound(DATA\_MATRIX, 1) 'Step 4.1 - Define the range of the 1D array

NROWS = UBound(DATA\_MATRIX, 1)

ReDim TEMP\_MATRIX(SROW To NROWS, SROW To SROW) 'Step 4.2- Transpose into a Vertical Array

For j = SCOLUMN To NCOLUMNS

For i = SROW To NROWS 'Step 4.3- For...Next Statement. Input Data Reversely TEMP\_MATRIX(NROWS + SROW - i, SROW) = DATA\_MATRIX(i) 'Assign value for each position in TEMP\_MATRIX.

Next i

Next j

Else

GoTo ERROR\_LABEL

End If

End If

MATRIX\_REVERSE\_FUNC = TEMP\_MATRIX 'Step 5- Return Reverse Entries Array

Exit Function ' Step 6- end function.

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_REVERSE\_FUNC = Err.Number

End Function

**Lesson3\_Array\_Redim**

ReDim means change the size of one or more dimensions of an array that has already been declared. In this lesson we look at a function to resize arrays as well as checking their size. This lesson contain three functions: 1. ARRAY\_DIMENSION\_FUNC, 2. MATRIX\_REDIM\_FUNC and 3. MATRIX\_RESIZE\_FUNC

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?

**1. ARRAY\_DIMENSION\_FUNC**

This function returns the number of dimensions of an array. An unallocated dynamic array has 0 dimensions. This function contains a Do...Loop Until Loop.

**VBA Application and Function Reference**

**Do...Loop Until Loop**: The loop continues until a specified condition is met. The condition is tested at the end of the loop. Therefore, at a minimum, the Do-Loop Until loop always results in the body of the loop being executed once.

**On Error Resume Next:** After executing this statement, VBA simply ignores all errors and resumes execution with the next statement.

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**i**: Counter variable for the number represent the dimensions in an array.

**j:** The return of the highest value the subscript of a specified dimension can contain.

**DATA\_VECTOR** : Copies over the content of DATA\_RNG. This is also the array being returned by the main function.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, VBA ignores it and resumes execution with the next statement.

On Error Resume Next

**Step 2 - Make a Copy of the Testing Array:**

First, we create a DATA\_VECTOR variable. Then, we copy over all the values from the DATA\_RNG into DATA\_VECTOR. If we change DATA\_VECTOR, DATA\_RNG will not be changed. Then, the output for the ARRAY\_DIMENSION\_FUNC is base on theDATA\_VECTOR.

Dim DATA\_VECTOR As Variant

DATA\_VECTOR = DATA\_RNG

**Step 3- Step up starting point for i:**

i is declared as a variable of Long type. It is the counter variable for the number represent the different dimensions in an array. We declare i equal to 0, which means the array does not have the first dimension.

Dim i As Long

i = 0

**Step 4- Do...Loop Until Loop. Find the Number of Dimensions in DATA\_VECTOR:**

j is declared as a variable of Long type. It is set to equal to the size of DATA\_VECTOR's "i"th dimension. Ubound will return a value if that dimension exist and continue the loop, or else it will raise an error. If Ubound returns a value, then we will test the next i. This Loop will stop when an error is raised. i was declared to be 0. The first i we run in the loop is equal to 1, since i=i+1=0+1.

Dim j As Long

Do

i = i + 1

j = UBound(DATA\_VECTOR, i)

Loop Until Err.Number <> 0

This step keep on looping and 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.

**Step 5 - Return the Number of Dimension DATA\_VECTOR has :**

The reason we do "i-1" is because the loop stops at "i" which is 1 more number than number of array's dimension exist. The Loop stops when the "i"th dimension of that array does not exist. Therefore, the last dimension the array has is the "i-1" th dimension. For example. For a 2D array, the loop will continue when i=2. When i=3, it will raise an error which will stop the loop and end the function. Since the testing condition is at the end , i is equal to 3 when the loop stopped. The number of the dimension this array has is actually i-1=3-1=2.

ARRAY\_DIMENSION\_FUNC = i - 1

**Step 6- End Function:**

End the function.

End Function

**Returns**

Value of Long type, which is the number of dimension the array has.

**VBA Code with Annotation**

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

Dim i As Long

Dim j As Long

Dim DATA\_VECTOR As Variant

On Error Resume Next 'Step 1- Error Handler

DATA\_VECTOR = DATA\_RNG 'Step 2 - Make a Copy of the Testing Array

i = 0 'Step 3- Step up starting point for i

Do 'Step 4- Do...Loop Until Loop. Find the Number of Dimensions in DATA\_VECTOR

i = i + 1 ' i is the counter use to check the number of dimension

j = UBound(DATA\_VECTOR, i) 'Ubound check if the "i"th dimension exist

' 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.

Loop Until Err.Number <> 0 ' Loop stop condition. The Loop will stop when a error is raised.

' Error raise when the "i"th dimension does not exist

ARRAY\_DIMENSION\_FUNC = i - 1 'Step 5 - Return the Number of Dimension DATA\_VECTOR has

'i-1 because the loop stops at "i" which is 1 more number than number of array's dimension exist.

End Function ' Step 6- end function.

**2. MATRIX\_REDIM\_FUNC**

ReDim means changing the size of a variable and not the variable type. It is only for adjusting arrays. MATRIX\_REDIM\_FUNC will extend the length or a two-dimensional array and preserving its value.

MATRIX\_REDIM\_FUNC contains a set of Select Case to determine which dimension to extend. It contains a nesting loop of For...Next Statement to enter the entries from the original array.

**VBA Application and Function Reference**

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called. Optional parameters are indicated by the Optional keyword in the procedure definition. The following rules apply: 1. Every optional parameter in the procedure definition must specify a default value. 2. The default value for an optional parameter must be a constant expression. 3. Every parameter following an optional parameter in the procedure definition must also be optional.

**Select Case Structure** : Does any of several things , depending on something's value. The Select Case structure is useful for decisions involving three or more options (although it also works with two options, providing an alternative to the If-Then-Else structure).

Select Case test expression

[Case expression list-n

[statements-n]] . . .

[Case Else

[else statements]]

End Select

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1.)

**Parameters**

**DATA\_RNG** : Array to be extended. Set of data in array.

**k**: Represent the number of rows or columns we are extending. If you do not assign a value to k, k is set to equal to 1 by default.

**VERSION**: Signal which dimension we are resizing. When VERSION=0, the resizing apply to rows (first dimension). When VERSION≠0, the resizing apply to columns (second dimension).If you do not assign a value to VERSION, VERSION is set to 0 by default.

**Variable**

**i**: Counter variable for rows.

**j:** Counter variable for columns.

**NSIZE**: The size of the array's ReDim dimension.

**SROW**: Returns the lowest available subscript for the first dimension of array.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of array.

**NROWS**: The return of the highest value the subscript for array's first dimension can contain.

**NCOLUMNS:** The return of the highest value the subscript for array's second dimension can contain.

**DATA1\_MATRIX** : Copy of DATA\_RNG. This is also the array being returned by the main function.

**DATA2\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_REDIM\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_REDIM\_FUNC = Err.Number

**Step 2 - Make Copies of the Original Array:**

First, we create a DATA1\_MATRIX and DATA2\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA1\_MATRIX. We make another copy of the array in DATA2\_MATRIX. The reason we make two copies, is because we may lose some entries when we resize the array. It is safer if we keep two copy of the original array, one for resizing the structure and one for keeping the entries. At this point, DATA1\_MATRIX and DATA2\_MATRIX are the exact copies of DATA\_RNG.

Dim DATA1\_MATRIX As Variant

Dim DATA2\_MATRIX As Variant

DATA1\_MATRIX = DATA\_RNG

DATA2\_MATRIX = DATA1\_MATRIX

**Step 3 -Define the range of the 2D array:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA2\_MATRIX's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA2\_MATRIX's first and second dimension, respectively. In general, rows are consider as the first dimension and columns are consider as the second dimension.

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

SROW = LBound(DATA2\_MATRIX, 1)

NROWS = UBound(DATA2\_MATRIX, 1)

SCOLUMN = LBound(DATA2\_MATRIX, 2)

NCOLUMNS = UBound(DATA2\_MATRIX, 2)

**Step 4.0- Set Up Select Case Structure:**

The Version variable is being evaluated. The routine is checking for 2 different cases (0 or not 0). Any number of statements can follow each Case statement, and they all are executed if the case is true.

Select Case VERSION

**Step 4.1.0 - Case 0 ReDim Preserve Rows :**

Case 0 will be executed when VERSION=0.

Case 0

**Step 4.1.1 - Define ReDimed Row Amount :**

NSIZE is declared as a variable of Long type. NSIZE is set to equal to the size of DATA1\_MATRIX after ReDim. Since NROWS is the maximum number of rows in DATA2\_MATRIX and k is the number of rows we are adjusting, the array's row size after ReDim is equal to adjusting the original array by k rows. Therefore, we have NSIZE = NROWS + k to represent the adjustment.

Dim NSIZE As Long

NSIZE = NROWS + k

**Step 4.1.2 - Redim Array Size After Extension :**

Since we are only extending the first dimension of the array, the lower bound of the first dimension and the second dimension will not be affected. DATA1\_MATRIX's first dimension is range from SROW to NSIZE and the second dimension remains the same as the original array.

Dim DATA1\_MATRIX As Variant

ReDim DATA1\_MATRIX(SROW To NSIZE, SCOLUMN To NCOLUMNS)

**Step 4.1.3 - For...Next Statement . Input Entries in to After Redim Array :**

We create counter variable i and j variable for the first and second dimension, respectively. We copy and input the entries from DATA2\_MATRIX into the after ReDim array, DATA1\_MATRIX.

Dim i As Long

Dim j As Long

For i = SROW To NROWS

For j = SCOLUMN To NCOLUMNS

DATA1\_MATRIX(i, j) = DATA2\_MATRIX(i, j)

Next j

Next i

SROW and NROWS are the limits for i. SCOLUMN and NCOLUMNS are the limits for j. The first entry we will transfer is positioned at (SROW, SCOLUMN). Then we assign value for that position from DATA2\_MATRIX to DATA1\_MATRIX. Then we move on to next j and j will equal to SCOLUMN+1. Until the next i, VBA will continue fill DATA1\_MATRIX and increase j by 1. i remain to be SROW until j=NCOLUMNS, which means all the columns in the first row are inputted. Next i will direct VBA to start filling in the second row. This will keep looping until i eventually equal to NROWS which means we have finish input all the entries from DATA2\_MATRIX.

**Step 4.2.0 - Case Else ReDim Preserve Column :**

Case Else will be executed when VERSION≠0.

Case Else

**Step 4.2.1 - Define ReDimed Column Amount :**

NSIZE is declared as a variable of Long type. NSIZE is set to equal to the size of DATA1\_MATRIX 's second dimension after ReDim. Since NROWS is the maximum number of columns in DATA2\_MATRIX and k is the number of columns we are adjusting, the array's column size after ReDim is equal to adjusting the original array by k columns. Therefore, we have NSIZE = NCOLUMNS + k to represent the adjustment.

NSIZE = NCOLUMNS + k

**Step 4.2.2 - Redim Extension Array Size :**

Since we are only extending the second dimension of the array, the lower bound of the second dimension and the first dimension will not be affected. DATA1\_MATRIX's second dimension is range from SCOLUMN To NSIZE and the first dimension remains the same as the original array.

ReDim DATA1\_MATRIX(SROW To NROWS, SCOLUMN To NSIZE)

**Step 4.2.3 - For...Next Statement . Input Entries into ReDim Array :**

We create counter variable i and j variable for the first and second dimension, respectively. We copy and input the entries from DATA2\_MATRIX into the after ReDim array, DATA1\_MATRIX.

For i = SROW To NROWS

For j = SCOLUMN To NCOLUMNS

DATA1\_MATRIX(i, j) = DATA2\_MATRIX(i, j)

Next j

Next i

SROW and NROWS are the limits for i. SCOLUMN and NCOLUMNS are the limits for j. The first entry we will transfer is positioned at (SROW, SCOLUMN). Then we assign value for that position from DATA2\_MATRIX to DATA1\_MATRIX. Then we move on to next j and j will equal to SCOLUMN+1. Until the next i, VBA will continue fill DATA1\_MATRIX and increase j by 1. i remain to be SROW until j=NCOLUMNS, which means all the columns in the first row are inputted. Next i will direct VBA to start filling in the second row. This will keep looping until i eventually equal to NROWS which means we have finish input all the entries from DATA2\_MATRIX.

**Step 5- Return ReDim Array:**

Return the ReDim array of DATA\_RNG.

MATRIX\_REDIM\_FUNC = DATA1\_MATRIX

**Step 6- End Function:**

End the function.

End Function

**Returns**

Array of variant type, which is the extended array of DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_REDIM\_FUNC(ByRef DATA\_RNG As Variant, \_

Optional ByVal k As Long = 1, \_

Optional ByVal VERSION As Integer = 0) 'Optional: When u call the function, you don't need to put a 'number for the parameter.

'k is the amount to extend. By default k =1

'VERSION indicate whether we are extend the rows (VERSION =0) or columns (VERSION=1)

Dim i As Long 'Counter variables

Dim j As Long

Dim NSIZE As Long 'The ReDim size of the array's extending dimension.

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DATA1\_MATRIX As Variant

Dim DATA2\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

'Step 2. Make 2 copies of the original array to preserve its value.

DATA1\_MATRIX = DATA\_RNG 'Copy to resize.

DATA2\_MATRIX = DATA1\_MATRIX 'Copy to preserve value

'Step 3 -Define the range of the 2D array:

SROW = LBound(DATA2\_MATRIX, 1) 'Find the lower bound integer for rows

NROWS = UBound(DATA2\_MATRIX, 1) 'Find the upper bound integer for rows

SCOLUMN = LBound(DATA2\_MATRIX, 2) 'Find the lower bound number for columns

NCOLUMNS = UBound(DATA2\_MATRIX, 2) 'Find the upper bound number for columns

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

Select Case VERSION ' Step 4.0- Set Up Select Case Structure

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

Case 0 'Step 4.1.0 - Case 0 ReDim Preserve Rows

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

NSIZE = NROWS + k 'Step 4.1.1 - Define ReDim Row Amount. Change the upper bound limit for rows

ReDim DATA1\_MATRIX(SROW To NSIZE, SCOLUMN To NCOLUMNS) 'Step 4.1.2 - ReDim Array Size After 'Extension. Only change the row size.

'Step 4.1.3 - For...Next Statement . Input Entries in to After ReDim Array

For i = SROW To NROWS 'Set range for i

For j = SCOLUMN To NCOLUMNS 'Set range for j

DATA1\_MATRIX(i, j) = DATA2\_MATRIX(i, j) 'Transfer the original array entries into to ReDim matrix.

Next j

Next i

**VBA Code with Annotation Continue**

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

Case Else ' Step 4.2.0 - Case Else ReDim Preserve Column

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

NSIZE = NCOLUMNS + k 'Step 4.2.1 - Define ReDimed Column Amount

ReDim DATA1\_MATRIX(SROW To NROWS, SCOLUMN To NSIZE) 'Step 4.2.2 ReDim Extension Array Size

For i = SROW To NROWS 'Step 4.2.3 For...Next Statement . Input Entries into ReDim Array

For j = SCOLUMN To NCOLUMNS

DATA1\_MATRIX(i, j) = DATA2\_MATRIX(i, j)

Next j

Next i

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

End Select

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

MATRIX\_REDIM\_FUNC = DATA1\_MATRIX 'Step 5- Return ReDim Array

Exit Function 'Step 6- End Function

ERROR\_LABEL:

MATRIX\_REDIM\_FUNC = Err.Number 'When an error is raised, return Error Number.

End Function

**3. MATRIX\_RESIZE\_FUNC**

MATRIX\_RESIZE\_FUNC will resize an array preserving or cutting its content. 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 This function contains a nesting IF-Then-Else statement to check if it is 1D or 2D array.

**VBA Application and Function Reference**

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called. (Also seen in MATRIX\_REDIM\_FUNC.)

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1.)

**IsMissing Function:** Returns a Boolean value indicating whether an optional Variant argument has been passed to a procedure. Return True if the optional Variant arguments is not provided. Using the missing argument in other code may raise an error. Return False if the optional Variant is provided.

**Parameters**

**DATA\_RNG** : Array to be resized. Set of data in array.

**AROW**: Represent the total number of rows we want to have.

**ACOLUMN**: Represent the total number of columns we want to have. ACOLUMN is optional meaning you do not have to supply a value.

**Variable**

**i**: Counter variable for rows.

**j:** Counter variable for columns.

**ii**: Maximum number of row values to be taken from the original array.

**jj:** Maximum number of column values to be taken from the original array.

**SROW**: Returns the lowest available subscript for the first dimension of array.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of array.

**NROWS**: The return of the highest value the subscript for array's first dimension can contain.

**NCOLUMN:** The return of the highest value the subscript for array's second dimension can contain.

**TEMP\_MATRIX** : Copy of DATA\_RNG. This is also the array being returned by the main function.

**DATA\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_RESIZE\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_RESIZE\_FUNC = Err.Number

**Step 2 - Make a Copy of the Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX.

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3.0 - First IF-Then-Else Statement. Resize Rows:**

IsMissing(ACOLUMN) is True if the optional Variant argument , ACOLUMN is not provide. Then execute the following lines. This also means we do not need to resize the columns. If IsMissing(ACOLUMN) is False, then VBA will executed the Else portion of the IF-THEN-ELSE statement.

If IsMissing(ACOLUMN) = True Then

...

Else

**Step 3.1 - Define the Row Size of Original Array:**

SROW and NROWS are declared as variables of Long type. SROW and NROWS are the lower and upper bound of DATA\_MATRIX's first dimension, respectively .

SROW = LBound(DATA\_MATRIX, 1)

NROWS = UBound(DATA\_MATRIX, 1)

**Step 3.2- Nest IF-Then-Else Statement. Transpose 1D Array:**

First we need to test if the orignal array is vector. NROWS-SROW+1 equal to 1 means NROWS has the same value as SROW. This means the orignal array only has 1 row and it is a vector. This is because 1D array is printed horzontally by default. We assume the user want to change the array size in the same direction as the data entries. Therefore, we need to transpose the array so it will print vertically before we change its size. (MATRIX\_TRANSPOSE\_FUNC can be finde in lesson 2) If NROWS - SROW + 1≠1, then VBA will not transpose the array.

If (NROWS - SROW + 1) = 1 Then

DATA\_MATRIX = MATRIX\_TRANSPOSE\_FUNC(DATA\_MATRIX )

End If

**Step 3.3 - Make a New Copy of the Array to Preserve Data Entries :**

First, we create a TEMP\_MATRIX variable. Then, we copy over all the content from DATA\_MATRIX to TEMP\_MATRIX.

Dim TEMP\_MATRIX As Variant

TEMP\_MATRIX = DATA\_MATRIX

TEMP\_MATRIX is use to preserve the entries from DATA\_MATRIX.

**Step 3.4- Resizing Array :**

Since the user did not input a value for ACOLUMN, we assume he or she does not want to keep a second dimension. Hence, only the data entries in the first column will remain.We resize the first dimension to SROW and AROW. AROW is the total number of rows the user want to have. The second dimension of this array will be erased.

ReDim DATA\_MATRIX(SROW To AROW)

**Step 3.5.0 - Nest IF-Then-Else Statement. Define Data Range for Reduced Size Array:**

ii is declared as a variable of Long type. ii will be used as the maximum number of row values to be taken from the original array. NROWS-SROW+1 gives us the size of the original array's first dimension. AROW is smaller than NROWS-SROW+1 when the user want to reduce the size of the original array. Therefore , VBA will obtain maximum AROW number of row values from the original array.

Dim ii As Long

If AROW < (NROWS - SROW + 1)

ii = AROW

Else

**Step 3.5.1 - Nest IF-Then-Else Statement. Define Data Range for Non- Reduced Size Array :**

VBA excute this portion of the code if the user want to extend the orignal array. The maximum number of row values VBA can obtain will be equal to the total row values the original array has.

ii = (NROWS - SROW + 1)

End If

**Step 3.6.0 - Nest IF-Then-Else Statement. Input Entries for 1D Array:**

IS\_1D\_ARRAY\_FUNC(TEMP\_MATRIX) will return True if TEMP\_MATRIX is a one-dimensional array, otherwise False. ( Detail of IS\_1D\_ARRAY\_FUNC can be find in lesson 1) If IS\_1D\_ARRAY\_FUNC(DATA\_MATRIX) = True then the following lines are executed.

If IS\_1D\_ARRAY\_FUNC(TEMP\_MATRIX) = True Then

For i = SROW To ii

DATA\_MATRIX(i) = TEMP\_MATRIX(i ) Next i

We create counter variable i for the first dimension. SROW and ii are the limits for i. Since it is a 1D array, the first entry we will transfer is positioned at (SROW). We copy and input the entries from TEMP\_MATRIX into DATA\_MATRIX. VBA will then move to Next i and input the next value. It will continue the loop until i equal to ii which is the maximum number of row values the user want to obtain from the orignal array.

**Step 3.6.1 - Nest IF-Then-Else Statement. Input Entries for 2D Array:**

IS\_2D\_ARRAY\_FUNC(TEMP\_MATRIX) will return True if TEMP\_MATRIX is a two-dimensional array, otherwise False. ( Detail of IS\_2D\_ARRAY\_FUNC can be find in lesson 1) If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True then the following lines are executed.

ElseIf IS\_2D\_ARRAY\_FUNC(TEMP\_MATRIX) = True Then

For i = SROW To ii

DATA\_MATRIX(i) = TEMP\_MATRIX(i, 1)

Next i

Else

The logic is similar to Step 3.6.0**-** Nest IF-Then-Else Statement. Input Entries for 1D Array. i is the counter variable runs between SROW and ii. We use TEMP\_MATRIX (i,1) to ensure we are only transfer entries of the first dimension from the original array. VBA will continue looping the code until it obtain the maximum number of row values from the orignal array.

Although Step 3.6.1 is similar to Step 3.6.0, we still need to treat 1D and 2D arrays separately because the output is different. 1D array will return a 1D array as the output and others will returns a 2D array.

**Step 3.6.2 - Raise Error for Non-1D or 2D Array:**

An error will raise if TEMP\_MATRIX is not a 1D or 2D array.

Else

GoTo ERROR\_LABEL

End If

**Step 4.0 - Resize both Rows and Columns:**

This is the Else portion of the IF-THEN-ELSE statement. When ACOLUMN is provided , IsMissing(ACOLUMN) returns False . In that situation , VBA will execute the following lines. This means we will resize both the rows and the columns. Several steps in this portion is simliar to the parts in Step

...

Else

**Step 4.1 - Define the Original Array Size:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA\_MATRIX's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA\_MATRIX's first and second dimension, respectively.

SROW = LBound(DATA\_MATRIX, 1)

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

**'Step 4.2 Make a Separate Copy of the Array to Preserve Data Entries:**

We copy over all the content from DATA\_MATRIX to TEMP\_MATRIX. TEMP\_MATRIX is use to preserve the entries from DATA\_MATRIX.

TEMP\_MATRIX = DATA\_MATRIX

**Step 4.3- Resizing the Array to New Dimension Limit:**

We resize the array dimension to the number of rows and columns the user want. First dimension is range from SROW to AROW, and the second dimension range from SCOLUMN to ACOLUMN. AROW is the total number of rows the user want to have. ACOLUMN is the total number of columns the user want to have.

ReDim DATA\_MATRIX(SROW To AROW, SCOLUMN To ACOLUMN)

**Step 4.4.0 - Nest IF-Then-Else Statement. Define Max Row Values for Reduced Row Size:**

If AROW < (NROWS - SROW + 1)

ii = AROW

Else

Similarly to Step 3.5.0. ii will be used as the maximum number of row values to be taken from the original array. AROW is smaller than NROWS-SROW+1 when the user want to reduce the size of the original array. Therefore , VBA will obtain maximum AROW number of row values from the original array.

**Step 4.4.1 - Nest IF-Then-Else Statement. Define Max Row Values for Non- Reduced Row Size:**

VBA excute this portion of the code if the user want to extend the orignal array. The maximum number of row values VBA can obtain is the maxium row values of the original array. When SROW=1, we can use NROWS and NROWS+SROW-1 interchangablely. If it is base zero, then we have to use NROWS-SROW+1 insteand NROWS.

ii = NROWS

End If

**Step 4.4.2 - Nest IF-Then-Else Statement. Define Column Range for Reduced Column Size:**

jj is declared as a variable of Long type. jj will be used as the maximum number of column values to be taken from the original array. NCOLUMNS - SCOLUMN + 1 gives us the size of the original array's second dimension. ACOLUMN is smaller than NCOLUMNS - SCOLUMN + 1 when the user want to reduce the size of the original array. Therefore , VBA will obtain maximum ACOLUMN number of column values from the original array.

Dim jj As Long

If ACOLUMN < (NCOLUMNS - SCOLUMN + 1) Then

jj = ACOLUMN

Else

**Step 4.4.3 - Nest IF-Then-Else Statement. Define Column Range for Non- Reduced Column Size:**

VBA excute this portion of the code if the user want to keep the orignal dimension size or add more columns. The maximum number of column values VBA can obtain is the maxium number of columns in the original array.

jj = (NCOLUMNS - SCOLUMN + 1)

End If

**Step 4.5 For...Next Statement. Input Entries :**

In this step, we populate the resized array with the existing values.

For i = SROW To ii

For j = SCOLUMN To jj

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

Next j

Next i

We create counter variable i for the first dimension and j for the second dimension. SROW and ii are the limits for i. SCOLUMN and jj are the limits for j. The first entry we will transfer is positioned at (SROW, SCOLUMN).

Then we assign value for that point from TEMP\_MATRIX to DATA\_MATRIX. Then we move on to next j , j will equal to SCOLUMN+1 and continue to fill the TEMP\_MATRIX. i will remain to be SROW until j equals jj, which means all the column in the first row are filled. Next i will direct VBA to start filling in the second row. This will keep looping until i eventually equal to ii which means DATA\_MATRIX obtains all the entries from TEMP\_MATRIX.

**Step 5- Return Resize Array:**

Return the resizing array of DATA\_RNG.

MATRIX\_RESIZE\_FUNC = DATA\_MATRIX

**Step 6- End Function:**

End the function.

End Function

**Returns**

Array of variant type, which is the resizing array of DATA\_RNG.

**VBA Code with Annotation**

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

ByVal AROW As Long, \_

Optional ByVal ACOLUMN As Long) 'ACOLUMN is optional =do not have to supply a value.

Dim i As Long

Dim j As Long

Dim ii As Long 'Maximum number of row values to be taken from the original array.

Dim jj As Long 'Maximum number of column values to be taken from the original array.

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Make a Copy of the Original Array

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

If IsMissing(ACOLUMN) = True Then 'Step 3.0 Did not provide optional Variant argument, ACOLUMN, 'resize rows only.

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

'Step 3.1 - Define the Row Size of Original Array

SROW = LBound(DATA\_MATRIX, 1) 'Find the lower bound integer for rows

NROWS = UBound(DATA\_MATRIX, 1) 'Find the upper bound integer for rows

If (NROWS - SROW + 1) = 1 Then 'Step 3.2- Nest IF-Then-Else Statement.

'Transpose 1D Array because it is printed horizontally.

DATA\_MATRIX = MATRIX\_TRANSPOSE\_FUNC(DATA\_MATRIX) 'MATRIX\_TRANSPOSE\_FUNC lesson 2

End If

TEMP\_MATRIX = DATA\_MATRIX 'Step 3.3 - Make a New Copy of the Array to Preserve Data Entries

ReDim DATA\_MATRIX(SROW To AROW) 'Step 3.4- Resizing Array. Second dimension is removed.

If AROW < (NROWS - SROW + 1) Then 'Step 3.5.0 Define Data Range for Reduced Size Array

ii = AROW

Else

ii = (NROWS - SROW + 1) 'Step 3.5.1-Define Data Range for Non- Reduced Array Size

End If

If IS\_1D\_ARRAY\_FUNC(TEMP\_MATRIX) = True Then 'Step 3.6.0 Input Entries for 1D Array

For i = SROW To ii ' Setup counter variable range

DATA\_MATRIX(i) = TEMP\_MATRIX(i) 'Input Entries

Next i

**VBA Code with Annotation Continue**

ElseIf IS\_2D\_ARRAY\_FUNC(TEMP\_MATRIX) = True Then 'Step 3.6.1 Input Entries for 2D Array

For i = SROW To ii ' Setup counter variable range

DATA\_MATRIX(i) = TEMP\_MATRIX(i, 1) 'Input Entries from the first dimension

Next i

Else

GoTo ERROR\_LABEL 'Step 3.6.2 - Raise Error for Non-1D or 2D Array

End If

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

Else 'Step 4.0 Resize apply to both Rows and Columns.

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

'Step 4.1 Define the Original Array Size.

SROW = LBound(DATA\_MATRIX, 1) 'Find the lower bound integer for rows

SCOLUMN = LBound(DATA\_MATRIX, 2) 'Find the lower bound number for columns

NROWS = UBound(DATA\_MATRIX, 1) 'Find the upper bound integer for rows

NCOLUMNS = UBound(DATA\_MATRIX, 2) 'Find the upper bound number for columns

TEMP\_MATRIX = DATA\_MATRIX 'Step 4.2 Make a Separate Copy of the Array to Preserve Data Entries.

ReDim DATA\_MATRIX(SROW To AROW, SCOLUMN To ACOLUMN) 'Step 4.3 Resizing Array to New Dimension Limit.

If AROW < (NROWS - SROW + 1) Then ' Step 4.4.0 Define Max Row Values for Reduced Row Size.

ii = AROW

Else

ii = NROWS 'Step 4.4.1 Define Row Range for Non-Reduced Row Size.

End If

If ACOLUMN < (NCOLUMNS - SCOLUMN + 1) Then 'Step 4.4.2 Define Reduced Column value number

jj = ACOLUMN

Else

jj = (NCOLUMNS - SCOLUMN + 1) 'Step 4.4.3 Define Column Range for Non- Reduced Column Size

End If

For i = SROW To ii 'Step 4.5 Input Entries

For j = SCOLUMN To jj

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

Next j

Next i

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

End If

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

MATRIX\_RESIZE\_FUNC = DATA\_MATRIX 'Step 5- Return Resize Array

Exit Function 'Step 6- End Function

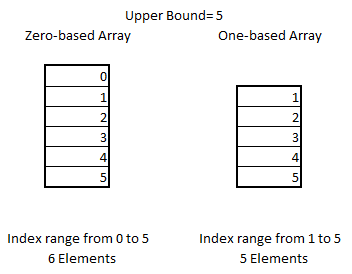
ERROR\_LABEL:

MATRIX\_RESIZE\_FUNC = Err.Number 'When an error is raised, return Error Number.

End Function

**Lesson4\_Array\_Base**

Zero-based array types means the minimum valid value for any index is 0. One-based array types means each index starts at 1. With the same upper bound, zero-based array will have 1 more element ine the indexes than one-based array types.



This lesson contain two functions: 1. MATRIX\_CHANGE\_BASE\_ZERO\_FUNC and 2. MATRIX\_CHANGE\_BASE\_ONE\_FUNC

**1. MATRIX\_CHANGE\_BASE\_ZERO\_FUNC**

This functions change the base of the array to zero for both rows and columns. It contains For...Next Structure to copy entries from the original array to the zero-based array.

**VBA Application and Function Reference**

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1. )

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**i**: Counter variable for the first dimensions in the original dataset.

**j:** Counter variable for the second dimensions in the original dataset.

**SROW**: The lowest available subscript for the first dimension of zero-based array.

**SCOLUMN**: The lowest available subscript for the second dimension of zero-based array.

**NROWS**: The total number of elements in row.

**NCOLUMN:** The total number of elements in column.

**TEMP\_MATRIX** : Copy of DATA\_RNG. This is also the array being changed by the main function.

**DATA\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_CHANGE\_BASE\_ZERO\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3- Define the Total Number of Element in Indexes:**

A index's upper bound subtract lower bound plus 1 give us the total number of elements in the indexes.

Dim NROWS As Long

Dim NCOLUMNS As Long

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1

The graph at the beginning of Lesson 4 description can help you to visualize this. NROWS and NCOLUMS are declared as variables of Long type. They are set to equal to the total number of elements in DATA\_MATRIX's first and second dimension, respectively.

**Step 4- ReDim Zero-based Empty Matrix:**

For zero-based array, the upper bound value for the new array's dimensions will be equal to the total number of elements in that dimension -1. The lower bound for zero-based array will be 0. Therefore, the size of the first dimension is 0 To NROWS -1 and 0 To NCOLUMNS-1 for the second dimension. This step, construct an empty matrix that has the same size as the original array.

ReDim TEMP\_MATRIX(0 To NROWS - 1, 0 To NCOLUMNS - 1)

**Step 5 - Nest IF-Then-Else Statement. Input Entries for Zero-Based Array:**

We create i and j as counter variable for the two dimension of the original data set. Since the original array is not a zero-based array, it will have different upper and lower bound as the zero-based array. The lower bound for zero-based array is 0. Therefore, SROW and SCOLUMN are equal to 0

SROW = 0

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

SCOLUMN = 0

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2)

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j)

SCOLUMN = SCOLUMN + 1

Next j

SROW = SROW + 1

Next i

The entry we will input is copied from j= LBound(DATA\_MATRIX, 2) and i= LBound(DATA\_MATRIX, 1) in the original data set. We will enter is at SROW=0, SCOLUMN=0 in the zero-based array. Then we move on to next column (SCOLUMN+1) in the first row from the zero-based array. We will also move to the next j position correspondingly in the original array to find the next data entry to transfer. We will continue move crossing the first row until j equal to the upper bound of the second dimension.

Since j has reach it's maximum, we will move to the next row for both array. We start to fill in the second row just like how we did for the first row. This will keep looping until i eventually reach its maximum value which means TEMP\_MATRIX is filled.

**Step 6- Return Zero-based Array with Entries:**

Return the zero-based array of DATA\_RNG.

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = TEMP\_MATRIX

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which is the changed zero-base array of DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_CHANGE\_BASE\_ZERO\_FUNC(ByRef DATA\_RNG As Variant)

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 Preserve Entries From Original Array

'Step 3 - Define the Total Number of Element in Indexes

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1 'Total number of elements in row

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1 'Number of elements in column

ReDim TEMP\_MATRIX(0 To NROWS - 1, 0 To NCOLUMNS - 1) 'Step 4 ReDim Zero-based Empty Matrix

'Step 5 - Nest IF-Then-Else Statement. Input Entries for Zero-Based Array:

SROW = 0 'Set up starting position in zero-based array

For i = LBound(DATA\_MATRIX, 1) To UBound(DATA\_MATRIX, 1) ' Set range for i

SCOLUMN = 0 'Set up starting position in zero-based array

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2) ' Set range for j

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j) ' copying value

SCOLUMN = SCOLUMN + 1 'Move horizontally to the next column in zero-based array.

Next j 'Move to next column in the same row in the original data set.

SROW = SROW + 1 'Move to next row after filling all the column in the previous row.

Next i ' Move to next row after filling all the column in the previous row

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = TEMP\_MATRIX 'Step 6- Return Zero-based Array with Entries

Exit Function 'Step 6- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = Err.Number

End Function

**2. MATRIX\_CHANGE\_BASE\_ONE\_FUNC**

This functions change the base of the array to one for both rows and columns. It contains For...Next Structure to copy entries from the original array to the one-based array.

**VBA Application and Function Reference**

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1. )

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**i**: Counter variable for the first dimensions in the original dataset.

**j:** Counter variable for the second dimensions in the original dataset.

**SROW**: The lowest available subscript for the first dimension of zero-based array.

**SCOLUMN**: The lowest available subscript for the second dimension of zero-based array.

**NROWS**: The total number of elements in row.

**NCOLUMN:** The total number of elements in column.

**TEMP\_MATRIX** : Copy of DATA\_RNG. This is also the array being changed by the main function.

**DATA\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_CHANGE\_BASE\_ONE\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3- Define the Total Number of Element in Indexes:**

A index's upper bound subtract lower bound plus 1 gives out the total number of elements in the indexes. The graph at the beginning of Lesson 4 description can help you visualize this. NROWS and NCOLUMS are declared as variables of Long type. They are set to equal to the total number of elements in DATA\_MATRIX's first and second dimension, respectively.

Dim NROWS As Long

Dim NCOLUMNS As Long

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1

**Step 4- ReDim One-based Empty Matrix:**

For one-based array, the upper bound value for the new array's dimensions will be equal to the total number of elements in that dimension. The lower bound for one-based array will be 1. Therefore, the size of the first dimension is 1 To NROWS and 1 To NCOLUMNS for the second dimension. This step, construct an empty matrix that has the same size as the original array.

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

**Step 5 - Nest IF-Then-Else Statement. Input Entries for One-Based Array:**

We create i and j as counter variable for the two dimension of the original data set. Since the original array is not a one-based array, it will have different upper and lower bound as the one-based array. The lower bound for one-based array is 1. Therefore, SROW and SCOLUMN are equal to 1.

SROW = 1

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

SCOLUMN = 1

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2)

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j)

SCOLUMN = SCOLUMN + 1

Next j

SROW = SROW + 1

Next i

The entry we will input is copied from j= LBound(DATA\_MATRIX, 2) and i= LBound(DATA\_MATRIX, 1) in the original data set. We will enter is at SROW=1, SCOLUMN=1 in the one-based array. Then we move on to next column (SCOLUMN+1) in the first row of the one-based array. We will also move to the next j position correspondingly in the original array to find the next data entry to transfer. We will continue move crossing the first row until j equal to the upper bound of the second dimension.

Since j has reach its maximum, we will move to the next row for both array. We start to fill in the second row just like how we did for the first row. This will keep looping until i eventually reach its maximum value which means TEMP\_MATRIX is filled.

**Step 6- Return One-Based Array with Entries:**

Return the one-based array of DATA\_RNG.

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = TEMP\_MATRIX

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which is the changed one-base array of DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_CHANGE\_BASE\_ONE\_FUNC(ByRef DATA\_RNG As Variant)

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 Preserve Entries From Original Array

'Step 3 - Define the Total Number of Element in Indexes

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1 'Total number of elements in row

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1 'Number of elements in column

ReDim TEMP\_MATRIX(1 To NROWS, 1 To NCOLUMNS) 'Step 4 ReDim One-based Empty Matrix

'Step 5 - Nest IF-Then-Else Statement. Input Entries for One-Based Array

SROW = 1 'Set up starting position in One-based array

For i = LBound(DATA\_MATRIX, 1) To UBound(DATA\_MATRIX, 1) ' Set range for i

SCOLUMN = 1 ' Set range for i

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2) ' Set range for j

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j) ' Copying value

SCOLUMN = SCOLUMN + 1 'Move horizontally to the next column in one-based array.

Next j 'Move to next column in the same row in the original data set.

SROW = SROW + 1 'Move to next row after filling all the column in the previous row.

Next i ' Move to next row after filling all the column in the previous row

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = TEMP\_MATRIX 'Step 6- Return One-based Array with Entries

Exit Function 'Step 6- End Function

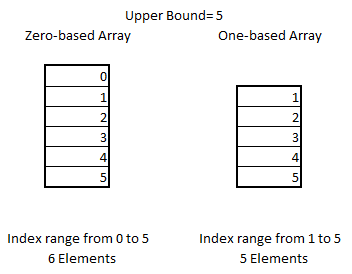
ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = Err.Number

End Function

**Lesson4\_Array\_Base**

Zero-based array types means the minimum valid value for any index is 0. One-based array types means each index starts at 1. With the same upper bound, zero-based array will have 1 more element ine the indexes than one-based array types.



This lesson contain two functions: 1. MATRIX\_CHANGE\_BASE\_ZERO\_FUNC and 2. MATRIX\_CHANGE\_BASE\_ONE\_FUNC

**1. MATRIX\_CHANGE\_BASE\_ZERO\_FUNC**

This functions change the base of the array to zero for both rows and columns. It contains For. Next Structure to copy entries from the original array to the zero-based array.

**VBA Application and Function Reference**

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1. )

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**i**: Counter variable for the first dimensions in the original dataset.

**j:** Counter variable for the second dimensions in the original dataset.

**SROW**: The lowest available subscript for the first dimension of zero-based array.

**SCOLUMN**: The lowest available subscript for the second dimension of zero-based array.

**NROWS**: The total number of elements in row.

**NCOLUMN:** The total number of elements in column.

**TEMP\_MATRIX** : Copy of DATA\_RNG. This is also the array being changed by the main function.

**DATA\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_CHANGE\_BASE\_ZERO\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3- Define the Total Number of Element in Indexes:**

A index's upper bound subtract lower bound plus 1 give us the total number of elements in the indexes.

Dim NROWS As Long

Dim NCOLUMNS As Long

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1

The graph at the beginning of Lesson 4 description can help you to visualize this. NROWS and NCOLUMS are declared as variables of Long type. They are set to equal to the total number of elements in DATA\_MATRIX's first and second dimension, respectively.

**Step 4- ReDim Zero-based Empty Matrix:**

For zero-based array, the upper bound value for the new array's dimensions will be equal to the total number of elements in that dimension -1. The lower bound for zero-based array will be 0. Therefore, the size of the first dimension is 0 To NROWS -1 and 0 To NCOLUMNS-1 for the second dimension. This step, construct an empty matrix that has the same size as the original array.

ReDim TEMP\_MATRIX(0 To NROWS - 1, 0 To NCOLUMNS - 1)

**Step 5 - Nest IF-Then-Else Statement. Input Entries for Zero-Based Array:**

We create i and j as counter variable for the two dimension of the original data set. Since the original array is not a zero-based array, it will have different upper and lower bound as the zero-based array. The lower bound for zero-based array is 0. Therefore, SROW and SCOLUMN are equal to 0.

SROW = 0

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

SCOLUMN = 0

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2)

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j)

SCOLUMN = SCOLUMN + 1

Next j

SROW = SROW + 1

Next i

The entry we will input is copied from j= LBound(DATA\_MATRIX, 2) and i= LBound(DATA\_MATRIX, 1) in the original data set. We will enter is at SROW=0, SCOLUMN=0 in the zero-based array. Then we move on to next column (SCOLUMN+1) in the first row from the zero-based array. We will also move to the next j position correspondingly in the original array to find the next data entry to transfer. We will continue move crossing the first row until j equal to the upper bound of the second dimension.

Since j has reach it's maximum, we will move to the next row for both array. We start to fill in the second row just like how we did for the first row. This will keep looping until i eventually reach its maximum value which means TEMP\_MATRIX is filled.

**Step 6- Return Zero-based Array with Entries:**

Return the zero-based array of DATA\_RNG.

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = TEMP\_MATRIX

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which is the changed zero-base array of DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_CHANGE\_BASE\_ZERO\_FUNC(ByRef DATA\_RNG As Variant)

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 Preserve Entries From Original Array

'Step 3 - Define the Total Number of Element in Indexes

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1 'Total number of elements in row

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1 'Number of elements in column

ReDim TEMP\_MATRIX(0 To NROWS - 1, 0 To NCOLUMNS - 1) 'Step 4 ReDim Zero-based Empty Matrix

'Step 5 - Nest IF-Then-Else Statement. Input Entries for Zero-Based Array:

SROW = 0 'Set up starting position in zero-based array

For i = LBound(DATA\_MATRIX, 1) To UBound(DATA\_MATRIX, 1) ' Set range for i

SCOLUMN = 0 'Set up starting position in zero-based array

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2) ' Set range for j

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j) ' copying value

SCOLUMN = SCOLUMN + 1 'Move horizontally to the next column in zero-based array.

Next j 'Move to next column in the same row in the original data set.

SROW = SROW + 1 'Move to next row after filling all the column in the previous row.

Next i ' Move to next row after filling all the column in the previous row

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = TEMP\_MATRIX 'Step 6- Return Zero-based Array with Entries

Exit Function 'Step 6- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_CHANGE\_BASE\_ZERO\_FUNC = Err.Number

End Function

**2. MATRIX\_CHANGE\_BASE\_ONE\_FUNC**

This functions change the base of the array to one for both rows and columns. It contains For...Next Structure to copy entries from the original array to the one-based array.

**VBA Application and Function Reference**

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1. )

**Parameters**

**DATA\_RNG** : Array to be tested. Set of data in array.

**Variable**

**i**: Counter variable for the first dimensions in the original dataset.

**j:** Counter variable for the second dimensions in the original dataset.

**SROW**: The lowest available subscript for the first dimension of zero-based array.

**SCOLUMN**: The lowest available subscript for the second dimension of zero-based array.

**NROWS**: The total number of elements in row.

**NCOLUMN:** The total number of elements in column.

**TEMP\_MATRIX** : Copy of DATA\_RNG. This is also the array being changed by the main function.

**DATA\_MATRIX**: Copy of DATA1\_MATRIX use to preserve the entries from the original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_CHANGE\_BASE\_ONE\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

**Step 3- Define the Total Number of Element in Indexes:**

A index's upper bound subtract lower bound plus 1 gives out the total number of elements in the indexes. The graph at the beginning of Lesson 4 description can help you visualize this. NROWS and NCOLUMS are declared as variables of Long type. They are set to equal to the total number of elements in DATA\_MATRIX's first and second dimension, respectively.

Dim NROWS As Long

Dim NCOLUMNS As Long

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1

**Step 4- ReDim One-based Empty Matrix:**

For one-based array, the upper bound value for the new array's dimensions will be equal to the total number of elements in that dimension. The lower bound for one-based array will be 1. Therefore, the size of the first dimension is 1 To NROWS and 1 To NCOLUMNS for the second dimension. This step, construct an empty matrix that has the same size as the original array.

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

**Step 5 - Nest IF-Then-Else Statement. Input Entries for One-Based Array:**

We create i and j as counter variable for the two dimension of the original data set. Since the original array is not a one-based array, it will have different upper and lower bound as the one-based array. The lower bound for one-based array is 1. Therefore, SROW and SCOLUMN are equal to 1.

SROW = 1

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

SCOLUMN = 1

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2)

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j)

SCOLUMN = SCOLUMN + 1

Next j

SROW = SROW + 1

Next i

The entry we will input is copied from j= LBound(DATA\_MATRIX, 2) and i= LBound(DATA\_MATRIX, 1) in the original data set. We will enter is at SROW=1, SCOLUMN=1 in the one-based array. Then we move on to next column (SCOLUMN+1) in the first row of the one-based array. We will also move to the next j position correspondingly in the original array to find the next data entry to transfer. We will continue move crossing the first row until j equal to the upper bound of the second dimension.

Since j has reach its maximum, we will move to the next row for both array. We start to fill in the second row just like how we did for the first row. This will keep looping until i eventually reach its maximum value which means TEMP\_MATRIX is filled.

**Step 6- Return One-Based Array with Entries:**

Return the one-based array of DATA\_RNG.

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = TEMP\_MATRIX

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, which is the changed one-base array of DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_CHANGE\_BASE\_ONE\_FUNC(ByRef DATA\_RNG As Variant)

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 Preserve Entries From Original Array

'Step 3 - Define the Total Number of Element in Indexes

NROWS = UBound(DATA\_MATRIX, 1) - LBound(DATA\_MATRIX, 1) + 1 'Total number of elements in row

NCOLUMNS = UBound(DATA\_MATRIX, 2) - LBound(DATA\_MATRIX, 2) + 1 'Number of elements in column

ReDim TEMP\_MATRIX(1 To NROWS, 1 To NCOLUMNS) 'Step 4 ReDim One-based Empty Matrix

'Step 5 - Nest IF-Then-Else Statement. Input Entries for One-Based Array

SROW = 1 'Set up starting position in One-based array

For i = LBound(DATA\_MATRIX, 1) To UBound(DATA\_MATRIX, 1) ' Set range for i

SCOLUMN = 1 ' Set range for i

For j = LBound(DATA\_MATRIX, 2) To UBound(DATA\_MATRIX, 2) ' Set range for j

TEMP\_MATRIX(SROW, SCOLUMN) = DATA\_MATRIX(i, j) ' Copying value

SCOLUMN = SCOLUMN + 1 'Move horizontally to the next column in one-based array.

Next j 'Move to next column in the same row in the original data set.

SROW = SROW + 1 'Move to next row after filling all the column in the previous row.

Next i ' Move to next row after filling all the column in the previous row

MATRIX\_CHANGE\_BASE\_ONE\_FUNC = TEMP\_MATRIX 'Step 6- Return One-based Array with Entries

Exit Function 'Step 6- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

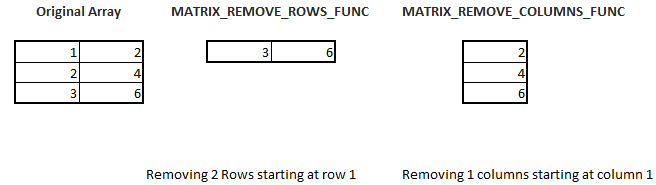
MATRIX\_CHANGE\_BASE\_ONE\_FUNC = Err.Number

End Function

**Lesson6\_Array\_Remove**

This lesson contain two functions: 1. MATRIX\_REMOVE\_ROWS\_FUNC and 2. MATRIX\_REMOVE\_COLUMNS\_FUNC

MATRIX\_REMOVE\_ROWS\_FUNC delete row(s) from an array. MATRIX\_REMOVE\_COLUMNS\_FUNC delete column(s) from an array.



**1. MATRIX\_REMOVE\_ROWS\_FUNC**

This function delete row(s) from an array. This function contains If...Then structure and For..Next statement.

**VBA Application and Function Reference**

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1.)

**If...Then structure**: Conditionally executes a group of statements, depending on the value of an expression. (Also seen in Lesson 2)

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called (Also seen in Lesson 3)

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**Parameters**

**DATA\_RNG** : Array that we want to delete row(s) from. Set of data in array.

**START\_ROW**: Start row position represents the row number to start deleting.

**NO\_ROWS**: Number of rows to remove. If value not provided, NO\_ROWS equal to 1 by default.

**Variable**

**i**: Counter variable for rows in DATA\_MATRIX.

**j:** Counter variable for columns for both DATA\_MATRIX and TEMP\_MATRIX.

**k:** Counter variable for rows in TEMP\_MATRIX.

**SROW**: Returns the lowest available subscript for the first dimension of array.

**NROWS**: Return of the highest value subscript for array's first dimension can contain.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of array.

**NCOLUMN:** Return of the highest value subscript for array's second dimension can contain.

**DATA\_MATRIX** : Copy of DATA\_RNG use to preserve the entries from the original array.

**TEMP\_MATRIX** : The array being returned by the main function.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_REDIM\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_ADD\_ROWS\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3 - If Then Structure. Check If Removing Zero Row:**

First, we need to check if NO\_ROWS equal to zero. NO\_ROWS=0 means we are not removing any rows. Hence, no change apply to the original array. Therefore , MATRIX\_REMOVE\_ROWS\_FUNC will return the original array and end the function.

If NO\_ROWS = 0 Then

MATRIX\_REMOVE\_ROWS\_FUNC = DATA\_MATRIX

Exit Function

End If

**Step 4 - Define Array Dimension Size:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA\_MATRIX's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA\_MATRIX's first and second dimension, respectively.

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

SROW = LBound(DATA\_MATRIX, 1)

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

**Step 5.0- First IF..Then Statement. Check N\_ROWS Value. Adjust Negative N\_ROWS :**

The first If...Then statement check if the NO\_ROWS is negative. We cannot delete negative rows. Therefore, we adjust NO\_ROWS equal to 1. Now we will delete 1 row from the array.

If (NO\_ROWS < 0) Then: NO\_ROWS = 1

**Step 5.1- Second IF...Then Statement. Check N\_ROWS Value. Notify Unreasonable N\_ROWS:**

The second If...Then statement check if NO\_ROWS is larger than TEMP\_MATRIX max number of rows. NROW-SROW+1 gives us the total row number in DATA\_MATRIX.

If NO\_ROWS >= (NROWS - SROW) + 1 Then: GoTo ERROR\_LABEL

VBA cannot delete more rows than the original array's total amount of rows. When this happen, VBA will skip to ERROR\_LABEL and return error number. We should not automatically assume the user want remove the entire array. It is more user friendly to just notify the user about this issue.

**Step 6- ReDim Array Size After Deleting Row(s) :**

TEMP\_MATRIX is declared as variant and is the return array of this function. Since we are only deleting rows, , the second dimension and the lower bound of the first dimension will not be affected. These ranges remain the same as the original array. TEMP\_MATRIX's first dimension's upper bound will decrease because we are removing row(s). The upper bound will be NROW-NO\_ROWS.

Dim TEMP\_MATRIX \_MATRIX As Variant

ReDim TEMP\_MATRIX(SROW To (NROWS - NO\_ROWS), SCOLUMN To NCOLUMNS)

**Step 7.0 - For...Next Statement . Input Entries into Removed Row Array :**

i, j and k are declared as counter variable of Long type. j will be using to located position in the second dimension for both TEMP\_MATRIX and DATA\_MATRIX. k is the counter variable that controls the first dimension of the removed row array. i move along the first dimension in DATA\_MATRIX. SROW and NROWS are the limits for i. SCOLUMN and NCOLUMNS are the limits for j.

Dim i As Long

Dim j As Long

Dim k As Long

For j = SCOLUMN To NCOLUMNS

k = SROW

For i = SROW To NROWS

...

TEMP\_MATRIX(k, j) = DATA\_MATRIX(i, j)

k = k + 1

Next i

...

Next j

The first entry we will input is copied from j=SCOLUMN and i=SROW in DATA\_MATRIX. We will enter it at k=SROW, j=SCOLUMN in TEMP\_MATRIX. Then we move to the next row (k+1) in the first column of TEMP\_MATRIX. We move to the next i position correspondingly in the original array to find the next data entry to transfer. We will continue move down the first column until i is beyond the upper bound of the first dimension.

Since i has reach its maximum, we will move to the next column in both array. We begin to fill in the second column just like last one. k is reset to equal to SROW at the beginning of j loop. VBA will keep running this loop until j reach beyond its maximum value which means TEMP\_MATRIX is filled.

At the beginning of each i loop, we will run two IF...Then statement. They help to control the input of each entry. We will explain these two statement in the following steps.

**Step 7.1 - First IF...Then Statement. Skip Removed Row Entry:**

The first If...Then statement check if the i at the row where we want to remove row(s). When we delete the row(s), it will also erase the data entries in these rows. These entries should not be input into TEMP\_MATRIX. We need to shift down to row (i+NO\_ROWS) in DATA\_MATRIX where we can restart copying data entry . If i does not equal to START\_ROW , then this statement will now affect this loop turn. The colon ":" is the VBA statement separator character. It is use to put the statement on the same line as the Case keyword.

If i = START\_ROW Then: i = i + NO\_ROWS

**Step 7.2 - Second IF...Then Statement. Check if New Array Runs Out of Row Space:**

The second If...Then statement check if i is more than its upper bound or k is larger than TEMP\_MATRIX max number of rows. When one of these situations happen, VBA will skip the following code until it find the line label "1983". It will then execute the code after the line label. i>NROWS indicate there is no more row entries in that column to be transfer. k>UBound(TEMP\_MATRIX,1) indicate there is no more rows in that column to be filled. Therefore, in both situations VBA will more to the next column.

If (i > NROWS) Or (k > UBound(TEMP\_MATRIX, 1)) Then: GoTo 1983

...

1983:

Next j

**Step 8- Return Array After Removing Rows :**

Return the array of DATA\_RNG after deleting rows.

MATRIX\_REMOVE\_ROWS\_FUNC = TEMP\_MATRIX

**Step 9- End Function:**

End the function.

End Function

End Function

**Returns**

Array of variant type, which is the extended array of DATA\_RNG adding number of rows.

**VBA Code with Annotation**

Function MATRIX\_REMOVE\_ROWS\_FUNC(ByRef DATA\_RNG As Variant, \_

ByVal START\_ROW As Long, \_

Optional ByVal NO\_ROWS As Long = 1) 'Optional: You don't need to put a number for the parameter.

'START\_ROW: Start Row Position, the row number to start removing rows. User has to provide a value.

'NO\_ROWS: Number of rows to remove. If value not provided, NO\_ROWS equal to 1 by default.

Dim i As Long

Dim j As Long

Dim k As Long

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim DATA\_MATRIX As Variant

Dim TEMP\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Preserve Entries From Original Array

If NO\_ROWS = 0 Then 'Step 3 - If Then Structure. Check If Removing Zero Row

MATRIX\_REMOVE\_ROWS\_FUNC = DATA\_MATRIX 'Output is the same as the original array.

Exit Function

End If

'Step 4 - Define Array Dimension Size

SROW = LBound(DATA\_MATRIX, 1) 'Rows are the 1st dimension and column are the 2nd dimension

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

If (NO\_ROWS < 0) Then: NO\_ROWS = 1 'Step 5.0 If N\_ROWS is negative then adjust it to 1

If NO\_ROWS >= (NROWS - SROW) + 1 Then: GoTo ERROR\_LABEL ' Step 5.1 Notify Unreasonable N\_ROWS

'Check if we are asking to remove more rows than the total amount there are

'User friendly design. Not to assume the user want to remove the all array set.

'Step 6- ReDim Array Size After Deleting Row(s)

ReDim TEMP\_MATRIX(SROW To (NROWS - NO\_ROWS), SCOLUMN To NCOLUMNS)

' Upper bound equal the original upper bound subtract number of row(s)

**VBA Code with Annotation Continue**

'Step 7.0 - For...Next Statement . Input Entries into Removed Row Array

For j = SCOLUMN To NCOLUMNS ' Set range for j

k = SROW 'Set start point for k

For i = SROW To NROWS ' Set range for i

If i = START\_ROW Then: i = i + NO\_ROWS 'Step 7.1 First IF..Then statement. Skip removed rows entry

If (i > NROWS) Or (k > UBound(TEMP\_MATRIX, 1)) Then: GoTo 1983 'Step 7.2 - Second IF...Then 'Statement. Check if New Array Runs Out of Row Space

TEMP\_MATRIX(k, j) = DATA\_MATRIX(i, j) ' Transfer value

k = k + 1 'Move vertically to the next row in new array.

Next i ' Move to next row

1983: 'GOTO line label

Next j ' Move to next column

MATRIX\_REMOVE\_ROWS\_FUNC = TEMP\_MATRIX 'Step 8- Return Array After Removing Rows

Exit Function 'Step 9- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_REMOVE\_ROWS\_FUNC = Err.Number

End Function

**2. MATRIX\_REMOVE\_COLUMNS\_FUNC**

This function delete column(s) from an array. This function contains If...Then structure and For...Next statement.

**VBA Application and Function Reference**

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1.)

**If...Then structure**: Conditionally executes a group of statements, depending on the value of an expression. (Also seen in Lesson 2)

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called (Also seen in Lesson 3)

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**Parameters**

**DATA\_RNG** : Array that we want to delete row(s) from. Set of data in array.

**START\_COLUMN**: Start column position represents the column number to start deletion.

**NO\_COLUMNS**: Number of columns to remove. If value not provided, NO\_COLUMS equal to 1 by default.

**Variable**

**i**: Counter variable for columns in DATA\_MATRIX.

**j:** Counter variable for rows for both DATA\_MATRIX and TEMP\_MATRIX.

**k:** Counter variable for columns in TEMP\_MATRIX.

**SROW**: Returns the lowest available subscript for the first dimension of array.

**NROWS**: Return of the highest value subscript for array's first dimension can contain.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of array.

**NCOLUMN:** Return of the highest value subscript for array's second dimension can contain.

**DATA\_MATRIX** : Copy of DATA\_RNG use to preserve the entries from the original array.

**TEMP\_MATRIX** : The array being returned by the main function.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_REDIM\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_ADD\_ROWS\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3 - If Then Structure. Check If Removing Zero Column:**

First, we need to check if NO\_COLUMNS equal to zero. NO\_COLUMNS=0 means we are not removing any columnss. Hence, no change apply to the original array. Therefore , MATRIX\_REMOVE\_COLUMNS\_FUNC will return the original array and end the function.

If NO\_COLUMNS = 0 Then

MATRIX\_REMOVE\_COLUMNS\_FUNC = DATA\_MATRIX

Exit Function

End If

**Step 4 - Define Array Dimension Size:**

SROW, SCOLUMN , NROWS and NCOLUMNS are declared as a variable of Long type. SROW and SCOLUMNS are set to equal to the lowest possible subscript of DATA\_MATRIX's first and second dimension, respectively . NROWS and NCOLUMNS are set to equal to the highest possible subscript of DATA\_MATRIX's first and second dimension, respectively.

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

SROW = LBound(DATA\_MATRIX, 1)

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

**Step 5.0- First IF...Then Statement. Check N\_COLUMNS Value. Adjust Negative N\_COLUMNS :**

The first If...Then statement check if the NO\_COLUMNS has negative value. We cannot delete negative columns. Therefore, we adjust NO\_COLUMNS equal to 1. Now we will delete 1 column from the array.

If (NO\_COLUMNS < 0) Then: NO\_COLUMNS = 1

**Step 5.1- Second IF...Then Statement. Check N\_COLUMNS Value. Notify Unreasonable N\_COLUMNS:**

The second If...Then statement check if NO\_COLUMNS is larger than TEMP\_MATRIX max number of columns. NCOLUMNS - SCOLUMN +1 gives us the total column number in DATA\_MATRIX.

If NO\_COLUMNS >= (NCOLUMNS - SCOLUMN) + 1 Then: GoTo ERROR\_LABEL

VBA cannot delete more columns than the original array's total amount of columns. When this happen, VBA will skip to ERROR\_LABEL and return error number. We should not automatically assume the user want remove the entire array. It is more user friendly to just notify the user about this issue.

**Step 6- ReDim Array Size After Deleting Column(s) :**

Dim TEMP\_MATRIX \_MATRIX As Variant

ReDim TEMP\_MATRIX(SROW To NROWS, SCOLUMN To (NCOLUMNS - NO\_COLUMNS))

TEMP\_MATRIX is declared as variant and is the return array of this function. Since we are only deleting columns , the first dimension and the lower bound of the second dimension will not be affected. These ranges remain the same as the original array. TEMP\_MATRIX's second dimension's upper bound will decrease because column(s) are removed. The upper bound will be NCOLUMNS - NO\_COLUMNS.

**Step 7.0 - For...Next Statement . Input Entries into Removed Column Array :**

i, j and k are declared as counter variable of Long type. j will be using to located position in the first dimension for both TEMP\_MATRIX and DATA\_MATRIX. k is the counter variable that controls the second dimension of the removed column array. i control the second dimension in DATA\_MATRIX. SROW and NROWS are the limits for j. SCOLUMN and NCOLUMNS are the limits for i.

Dim i As Long

Dim j As Long

Dim k As Long

For j = SROW To NROWS

k = SCOLUMN

For i = SCOLUMN To NCOLUMNS

...

TEMP\_MATRIX(j, k) = DATA\_MATRIX(j, i)

k = k + 1

Next i

...

Next j

The first entry we will input is copied from i=SCOLUMN and j=SROW in DATA\_MATRIX. We will enter it at j=SROW, k=SCOLUMN in TEMP\_MATRIX. Then we move to the next column(k+1) in the first row of TEMP\_MATRIX. We move to the next i position correspondingly in the original array to find the next data entry to transfer. We will continue filling cross the first row until i is beyond its upper bound. Then we will move to the next row in both array. We begin to fill in the second row just like last one. k is reset to equal to SCOLUMN at the beginning of the j loop. VBA will keep looping until j reach beyond its maximum value which means TEMP\_MATRIX is filled.

At the beginning of each i loop, we will run two IF...Then statement. They help to control the input of each entry. We will explain these two statement in the following steps.

**Step 7.1 - First IF...Then Statement. Skip Removed Column Entry:**

The first If...Then statement check if i at the column where we want to start removing column(s). When we delete the column(s), it will also erase the data entries in these columns. These entries should not be input into TEMP\_MATRIX. We need to shift right to column (i+NO\_COLUMNS) in DATA\_MATRIX where we can restart copying data entry . If i does not equal to START\_COLUMN , then this statement will now affect this loop turn. The colon ":" is the VBA statement separator character. It is use to put the statement on the same line as the Case keyword.

If i = START\_COLUMN Then: i = i + NO\_COLUMNS

**Step 7.2 - Second IF...Then Statement. Check if New Array Runs Out of Column Space:**

The second If...Then statement check if i is more than its upper bound or k is larger than TEMP\_MATRIX max number of columns. When one of these situations happen, VBA will skip the following code until it find the line label "1983". It will then execute the code after the line label. i>NCOLUMNS indicate there is no more column entries in that row to be transferred. k>UBound(TEMP\_MATRIX,1) indicate there is no more columns in that row to be filled. Therefore, in both situations VBA will more to the next row.

If (i > NCOLUMNS) Or (k > UBound(TEMP\_MATRIX, 2)) Then: GoTo 1983

...

1983:

Next j

**Step 8- Return Array After Removing Columns :**

Return the array of DATA\_RNG after deleting rows.

MATRIX\_REMOVE\_COLUMNS\_FUNC = TEMP\_MATRIX

**Step 9- End Function:**

End the function.

End Function

End Function

**Returns**

Array of variant type, which is the extended array of DATA\_RNG adding number of rows.

**VBA Code with Annotation**

Function MATRIX\_REMOVE\_COLUMNS\_FUNC(ByRef DATA\_RNG As Variant, \_

ByVal START\_COLUMN As Long, \_

Optional ByVal NO\_COLUMNS As Long = 1) 'Optional: You don't need to put a number for the parameter.

'START\_COLUMN: The column number to start removing columns. User has to provide a value.

'NO\_COLUMNS: Number of columns to be removed. If value not provided, equal to 1 by default.

Dim i As Long

Dim j As Long

Dim k As Long

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim DATA\_MATRIX As Variant

Dim TEMP\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Preserve Entries From Original Array

If NO\_COLUMNS = 0 Then 'Step 3 - If Then Structure. Check If Removing Zero Column

MATRIX\_REMOVE\_COLUMNS\_FUNC = DATA\_MATRIX 'Output is the same as the original array.

Exit Function

End If

'Step 4 - Define Array Dimension Size

SROW = LBound(DATA\_MATRIX, 1) 'Rows are the 1st dimension and column are the 2nd dimension

SCOLUMN = LBound(DATA\_MATRIX, 2)

NROWS = UBound(DATA\_MATRIX, 1)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

If (NO\_COLUMNS < 0) Then: NO\_COLUMNS = 1 'Step 5.0 First If N\_COLUMNS is negative then adjust to 1

'Step 5.1- Notify Unreasonable N\_COLUMNS Value

If NO\_COLUMNS >= (NCOLUMNS - SCOLUMN) + 1 Then: GoTo ERROR\_LABEL

'Check if we are asking to remove more rows than the total amount there are

'User friendly design. Not to assume the user want to remove the all array set.

'Step 6- ReDim Array Size After Deleting Column(s)

ReDim TEMP\_MATRIX(SROW To NROWS, SCOLUMN To (NCOLUMNS - NO\_COLUMNS))

' Upper bound equal the original upper bound subtract number of column(s)

**VBA Code with Annotation Continue**

'Step 7.0 - For...Next Statement . Input Entries into Removed Column Array

For j = SROW To NROWS ' Set range for j

k = SCOLUMN ' Set start point for k

For i = SCOLUMN To NCOLUMNS ' Set range for i

'Step 7.1 - First IF..Then Statement. Skip Removed Column Entry

If i = START\_COLUMN Then: i = i + NO\_COLUMNS

'Step 7.2 - Second IF...Then Statement. Check if New Array Runs Out of Column Space

If (i > NCOLUMNS) Or (k > UBound(TEMP\_MATRIX, 2)) Then: GoTo 1983

TEMP\_MATRIX(j, k) = DATA\_MATRIX(j, i) ' Transfer value

k = k + 1 'Move horizontally to the next column in new array.

Next i ' Move to next column

1983: 'GOTO line label

Next j ' Move to next row

MATRIX\_REMOVE\_COLUMNS\_FUNC = TEMP\_MATRIX 'Step 8- Return Array After Removing Columns

Exit Function 'Step 9- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

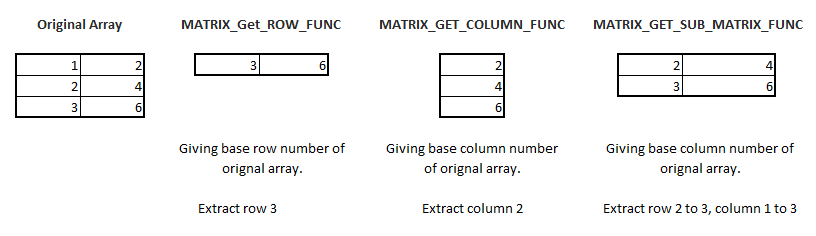
MATRIX\_REMOVE\_COLUMNS\_FUNC = Err.Number

End Function

**Lesson7\_Array\_Get**

This lesson contain three functions to extract entries from an array: 1. MATRIX\_GET\_ROW\_FUNC , 2. MATRIX\_GET\_COLUMN\_FUNC, and 3. MATRIX\_GET\_SUB\_MATRIX\_FUNC.

MATRIX\_GET\_ROW\_FUNC extract row from array. MATRIX\_GET\_COLUMN\_FUNC extract column from array. MATRIX\_GET\_SUB\_MATRIX\_FUNC return the sub matrix of pivot ij.



**VBA Application and Function Reference**

**On Error GoTo <line>** : After executing this statement , VBA resumes execution at the specified line and enables the error-handling routine. (Also seen in Lesson 1.)

**For...Next Statement:** Repeats a group of statements a specified number of times. (Also seen in Lesson 2)

**1. MATRIX\_GET\_ROW\_FUNC**

This function can extract one row from an array. The function contains For..Next statement to extract entries.

**Parameters**

**DATA\_RNG** : Array that we want to extract row from. Set of data in array.

**AROW**: The row number will be extracted.

**BASE\_ROW**: The base number set for the after extraction row array, TEMP\_MATRIX.

**Variable**

**i**: Counter variable for columns in DATA\_MATRIX.

**SCOLUMN**: Returns the lowest available subscript for the second dimension of array.

**NCOLUMN:** Return of the highest value subscript for array's second dimension can contain.

**DATA\_MATRIX** : Copy of DATA\_RNG use to preserve the entries from the original array.

**TEMP\_MATRIX** : The array being returned by the main function. The array will contain the extracted row from original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_GET\_ROW\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_GET\_ROW\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3 - Define Array Second Dimension Size:**

SCOLUMN and NCOLUMNS are declared as a variable of Long type. SCOLUMNS and NCOLUMNS are set to equal to the lowest and highest possible subscript of DATA\_MATRIX's second dimension, respectively .

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

SCOLUMN = LBound(DATA\_MATRIX, 2)

NCOLUMNS = UBound(DATA\_MATRIX, 2)

**Step 4 - ReDim Extracted Row Array :**

TEMP\_MATRIX is declared as variant and is the return array of this function. Since we are only extracting one row , the second dimension will remain the same as the original array. TEMP\_MATRIX's first dimension's lower and upper bound are equal to BASE\_ROW. This is because we are only extract one row from the original array. Therefore, the size of TEMP\_MATRIX's first dimension will be 1.

Dim TEMP\_MATRIX \_MATRIX As Variant

ReDim TEMP\_MATRIX(BASE\_ROW To BASE\_ROW, SCOLUMN To NCOLUMNS)

**Step 5 - For...Next Statement . Input Entries into the Row Array :**

i is declared as counter variable of Long type. i will be used to located position in the second dimension for both TEMP\_MATRIX and DATA\_MATRIX. SCOLUMN and NCOLUMNS are the limits for i.

Dim i As Long

For i = SCOLUMN To NCOLUMNS

TEMP\_MATRIX(BASE\_ROW, i) = DATA\_MATRIX(AROW, i) Next i

The first entry we will input is copied from in DATA\_MATRIX(AROW, SCOLUMN). We will enter it in TEMP\_MATRIX(BASE\_ROW, SCOLUMN) Then we move to the next column (i+1) in the same row for both TEMP\_MATRIX and DATA\_MATRIX. We will continue filling in that row until i is beyond its upper bound. This indicate we have extract all the entries from that row.

**Step 6- Return Extracted Row Array :**

Return the array contain the row extracted from DATA\_RNG.

MATRIX\_GET\_ROW\_FUNC = TEMP\_MATRIX

**Step 7- End Function:**

End the function.

End Function

End Function

**Returns**

Array of variant type, contain the entries from the row that extracted from DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_GET\_ROW\_FUNC(ByRef DATA\_RNG As Variant, \_

ByVal AROW As Long, \_

ByVal BASE\_ROW As Long)

'AROW is the row number that will be extracted.

'BASE\_ROW is the base number for new array contain the extracted row, TEMP\_MATRIX.

Dim i As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim TEMP\_MATRIX As Variant 'Returned array

Dim DATA\_MATRIX As Variant 'Array preserve data

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Preserve Entries From Original Array

SCOLUMN = LBound(DATA\_MATRIX, 2) 'Step 3 - Define Array Second Dimension Size

NCOLUMNS = UBound(DATA\_MATRIX, 2)

'Step 4 - ReDim Extracted Row Array

ReDim TEMP\_MATRIX(BASE\_ROW To BASE\_ROW, SCOLUMN To NCOLUMNS) 'Extract one row.

'Step 5 - For...Next Statement . Input Entries into the Row Array

For i = SCOLUMN To NCOLUMNS

TEMP\_MATRIX(BASE\_ROW, i) = DATA\_MATRIX(AROW, i) ' Input data

Next i 'Move to next column

MATRIX\_GET\_ROW\_FUNC = TEMP\_MATRIX 'Step 6- Return Extracted Row Array

Exit Function 'Step 7- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_GET\_ROW\_FUNC = Err.Number

End Function

**2. MATRIX\_GET\_COLUMN\_FUNC**

This function can extract one column from an array. The function contains For...Next statement to extract entries.

**Parameters**

**DATA\_RNG** : Array that we want to extract the column from. Set of data in array.

**ACOLUMN**: The column number of the extract column in the original array.

**BASE\_COLUMN**: The base number set for the after extraction column array, TEMP\_MATRIX.

**Variable**

**i**: Counter variable for rows in DATA\_MATRIX.

**SROW**: Returns the lowest available subscript for the first dimension of array.

**NROWS:** Return of the highest value subscript for array's first dimension can contain.

**DATA\_MATRIX** : Copy of DATA\_RNG use to preserve the entries from the original array.

**TEMP\_MATRIX** : The array being returned by the main function which contains the extracted column from original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_GET\_COLUMN\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_GET\_COLUMN\_FUNC = Err.Number

**Step 2 - Preserve Entries From Original Array:**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3 - Define Array First Dimension Size:**

SROW and NROWS are declared as a variable of Long type. SROW and NROWS are set to equal to the lowest and highest possible subscript of DATA\_MATRIX's first dimension, respectively .

Dim SROW As Long

Dim NROWS As Long

SROW = LBound(DATA\_MATRIX, 1)

NROWS = UBound(DATA\_MATRIX, 1)

**Step 4 - ReDim Extracted Column Array :**

TEMP\_MATRIX is declared as variant and is the return array of this function. Since we are only extracting one column , the first dimension will remain the same as the original array. TEMP\_MATRIX's second dimension's lower and upper bound are equal to BASE\_ROW and the size of TEMP\_MATRIX's second dimension will be 1.

Dim TEMP\_MATRIX \_MATRIX As Variant

ReDim TEMP\_MATRIX(SROW To NROWS, BASE\_COLUMN To BASE\_COLUMN)

**Step 5 - For...Next Statement . Input Entries into the Column Array :**

i is declared as counter variable of Long type. i will be used to located position in the first dimension for both TEMP\_MATRIX and DATA\_MATRIX. SROW and NROWS are the limits for i.

Dim i As Long

For i = SROW To NROWS

TEMP\_MATRIX(i, BASE\_COLUMN) = DATA\_MATRIX(i, ACOLUMN)

Next i

The first entry we will input is copied from in DATA\_MATRIX(SROW, ACOLUMN). We will enter it in TEMP\_MATRIX(SROW, BASE\_ROW) Then we move to the next row (i+1) in the same column for both TEMP\_MATRIX and DATA\_MATRIX. We will continue filling in that column until i is beyond its upper bound. This indicate we have extract all the entries from that column.

**Step 6- Return Extracted Column Array :**

Return the array contain the column extracted from DATA\_RNG.

MATRIX\_GET\_COLUMN\_FUNC = TEMP\_MATRIX

**Step 7- End Function:**

End the function.

End Function

End Function

**Returns**

Array of variant type, contain the entries from the column that extracted from DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_GET\_COLUMN\_FUNC(ByRef DATA\_RNG As Variant, \_

ByVal ACOLUMN As Long, \_

ByVal BASE\_COLUMN As Long)

'ACOLUMN is the column number that will be extracted.

'BASE\_COLUMN is the base number for new array contain the extracted column, TEMP\_MATRIX.

Dim i As Long

Dim SROW As Long

Dim NROWS As Long

Dim TEMP\_MATRIX As Variant 'Returned array

Dim DATA\_MATRIX As Variant 'Array preserve data

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Preserve Entries From Original Array

'Step 3 - Define Array First Dimension Size

SROW = LBound(DATA\_MATRIX, 1)

NROWS = UBound(DATA\_MATRIX, 1)

'Step 4 - ReDim Extracted Column Array

ReDim TEMP\_MATRIX(SROW To NROWS, BASE\_COLUMN To BASE\_COLUMN) 'Extract one column.

'Step 5 - For...Next Statement . Input Entries into the Column Array

For i = SROW To NROWS

TEMP\_MATRIX(i, BASE\_COLUMN) = DATA\_MATRIX(i, ACOLUMN) 'Input data

Next i

MATRIX\_GET\_COLUMN\_FUNC = TEMP\_MATRIX 'Step 6- Return Extracted Column Array

Exit Function 'Step 7- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_GET\_COLUMN\_FUNC = Err.Number

End Function

**3. MATRIX\_GET\_SUB\_MATRIX\_FUNC**

This function can extract a sub section array from an array. The function contains For...Next statement to extract entries.

**VBA Application and Function Reference**

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called (Also seen in Lesson 3)

**If...Then structure**: Conditionally executes a group of statements, depending on the value of an expression. (Also seen in Lesson 2)

**Parameters**

**DATA\_RNG** : Array that we want to extract the column from. Set of data in array.

**SROW**: The row number where the extraction array begin in DATA\_MATRIX. If you do not assign a value ,it is equal to -1 by default.

**NROWS**: The row number of the last extraction row in DATA\_MATRIX. If you do not assign a value ,it is equal to -1 by default.

**SCOLUMN**: The column number where the extraction array begin in DATA\_MATRIX. If you do not assign a value ,it is equal to -1 by default.

**NCOLUMN:** The column number where the extraction array end in DATA\_MATRIX. If you do not assign a value ,it is equal to -1 by default.

.

**Variable**

**i**: Counter variable for rows in DATA\_MATRIX.

**j:** Counter variable for columns in DATA\_MATRIX.

**ii:** Counter variable for rows in new array, TEMP\_MATRIX.

**jj:** Counter variable for columns in new array, TEMP\_MATRIX.

**DATA\_MATRIX** : Copy of DATA\_RNG use to preserve the entries from the original array.

**TEMP\_MATRIX** : The array being returned by the main function which is the extracted sub matrix from original array.

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then MATRIX\_GET\_SUB\_MATRIX\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

MATRIX\_GET\_SUB\_MATRIX\_FUNC = TEMP\_MATRIX

**Step 2 - Preserve Entries From Original Array :**

First, we create a DATA\_MATRIX variable. Then, we copy over all the values from the DATA\_RNG into DATA\_MATRIX. DATA\_MATRIX is used to preserve the entries from the original array .

Dim DATA\_MATRIX As Variant

DATA\_MATRIX = DATA\_RNG

**Step 3 - If Then Structure. Check IF Extraction Range is Given:**

SROW, SCOLUMN , NROWS and NCOLUMNS are set to equal to -1 if the value of these parameters are not given. -1 is just a label to signal VBA if certain condition has been met.

If SROW = -1 Then: SROW = LBound(DATA\_MATRIX, 1)

If NROWS = -1 Then: NROWS = UBound(DATA\_MATRIX, 1)

If SCOLUMN = -1 Then: SCOLUMN = LBound(DATA\_MATRIX, 2)

If NCOLUMNS = -1 Then: NCOLUMNS = UBound(DATA\_MATRIX, 2)

It is more user friendly to assume the user want to keep as many entries as possible, instead of not extracting the values. When the user did not input the starting dimension point of the extraction, VBA will assume the user would like to extract the entries from the lower bound of that dimension. When the user did not input the ending dimension point of the extraction, VBA will assume the user would like to extract entries up to upper bound of that dimension. Therefore, MATRIX\_GET\_SUB\_MATRIX\_FUNC will return the original array if the user did not input any of those four parameters.

**Step 4 - ReDim Sub Matrix Array :**

Dim TEMP\_MATRIX As Variant

ReDim TEMP\_MATRIX(1 To NROWS - SROW + 1, 1 To NCOLUMNS - SCOLUMN + 1)

TEMP\_MATRIX is declared as variant and is the return array of this function. Although we are extracting a sub matrix from the original array, the return of MATRIX\_GET\_SUB\_MATRIX\_FUNC will be one-based array. the lower bound of the first and second dimension will be set at 1. The upper bound of TEMP\_MATRIX's first dimension is NROW-SROW+1 and NCOLUMNS-SCOLUMN+1 for the second dimension. You can visualize this in the example provided at the beginning up Lesson 7 description.

**Step 5 - For...Next Statement . Input Entries into the Sub Matrix Array :**

i and j are counter variable for DATA\_MATRIX , are used to located position in the first and second dimension, respectively. ii and jj are counter variable for TEMP\_MATRIX , are used to located position in the first and second dimension, respectively. SROW and NROWS are the lower and upper limits for i. SCOLUMN and NCOLUMNS are the limits for i. ii and jj starts at 1 because TEMP\_MATRIX is a one-based array.

Dim i As Long

Dim j As Long

Dim ii As Long

Dim jj As Long

ii = 1: jj = 1

For j = SCOLUMN To NCOLUMNS

ii = 1

For i = SROW To NROWS

TEMP\_MATRIX(ii, jj) = DATA\_MATRIX(i, j)

ii = ii + 1

Next i

jj = jj + 1

Next j

The first entry we will input is copied from DATA\_MATRIX(SROW, NCOLUMNS). We will enter it in TEMP\_MATRIX(1, 1). Then we stay in the same column and move to the next row for both TEMP\_MATRIX (ii+1) and DATA\_MATRIX (i+1). We will continue filling in that column until i is beyond its upper bound. This indicate we have extract all the entries from that column. At this point, we will move to the next column in both TEMP\_MATRIX (jj+1) and DATA\_MATRIX (j+1). ii and i will restart from the lower bound. We start to fill in the second column just like how we did for the first one. This will keep looping until j eventually reach its maximum value which means we have complete extract the sub matrix from DATA\_MATRIX.

The colon ":" is the VBA statement separator character. It is use to put the statement on the same line as the Case keyword.

**Step 6- Return Extracted Sub Matrix Array :**

Return the array contain the sub matrix extracted from DATA\_RNG.

MATRIX\_GET\_SUB\_MATRIX\_FUNC = TEMP\_MATRIX

**Step 7- End Function:**

End the function.

End Function

End Function

**Returns**

Array of variant type, contain the entries from the sub matrix that extracted from DATA\_RNG.

**VBA Code with Annotation**

Function MATRIX\_GET\_SUB\_MATRIX\_FUNC(ByRef DATA\_RNG As Variant, \_

Optional ByVal SROW As Long = -1, \_

Optional ByVal NROWS As Long = -1, \_

Optional ByVal SCOLUMN As Long = -1, \_

Optional ByVal NCOLUMNS As Long = -1)

'The row and column numbers outline the dimension of the extracted sub matrix

'If no value being provided, then the parameter will equal to -1 by default.

Dim i As Long

Dim j As Long

Dim ii As Long

Dim jj As Long

Dim TEMP\_MATRIX As Variant

Dim DATA\_MATRIX As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

DATA\_MATRIX = DATA\_RNG 'Step 2 - Preserve Entries From Original Array

'Step 3 - If Then Structure. Check IF Extraction Range is Given.

If SROW = -1 Then: SROW = LBound(DATA\_MATRIX, 1)

If NROWS = -1 Then: NROWS = UBound(DATA\_MATRIX, 1)

If SCOLUMN = -1 Then: SCOLUMN = LBound(DATA\_MATRIX, 2)

If NCOLUMNS = -1 Then: NCOLUMNS = UBound(DATA\_MATRIX, 2)

'-1 a label to signal VBA if certain condition has been met. User friendly to extract entries as many entries 'as possible. Min from the lower bound and max to the upper bound.

'Step 4 - ReDim Sub Matrix Array

ReDim TEMP\_MATRIX(1 To NROWS - SROW + 1, 1 To NCOLUMNS - SCOLUMN + 1) 'Sub matrix size

'Step 5 - Set Counter Variable Starting Point for Sub Matrix Array

ii = 1: jj = 1 'Start ii and jj at TEMP\_MATRIX lower bound

' ":" is the VBA statement separator character. It is use to put the statement with the Case keyword.

For j = SCOLUMN To NCOLUMNS ' Set range for j

ii = 1 'Start ii at TEMP\_MATRIX lower bound

For i = SROW To NROWS ' Set range for i

TEMP\_MATRIX(ii, jj) = DATA\_MATRIX(i, j) 'Input data

ii = ii + 1 ' Move to next row

Next i ' Move to next row

jj = jj + 1 ' Move to next column

Next j ' Move to next column

MATRIX\_GET\_SUB\_MATRIX\_FUNC = TEMP\_MATRIX 'Step 6- Return Extracted Sub Matrix Array

Exit Function 'Step 7- End Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

MATRIX\_GET\_SUB\_MATRIX\_FUNC = Err.Number

End Function

**Lesson8\_Array\_Match**

This lesson contain two functions to compare entries in arrays: 1. ARRAY\_MATCH\_DATA\_FUNC and 2. ARRAY\_REMOVE\_DUPLICATES\_FUNC

ARRAY\_MATCH\_DATA\_FUNC compare two arrays and identify entries that appear in both array. extract row from array. ARRAY\_REMOVE\_DUPLICATES\_FUNC remove duplicates in an array.

**VBA Application and Function Reference**

**Collections :** In general terms, a collection is an object used for grouping and managing related objects. For example, every Form has a collection of controls. A collection can be zero-based or one-based, depending on what its starting index is. The former means that the index of the first item in the collection is 0, and the latter means that it is 1. Instances of the Visual Basic Collection class allow you to access an item using either a numeric index or a String key. You can add items to Visual Basic Collection objects either with or without specifying a key. If you add an item without a key, you must use its numeric index to access it.

**New Operator**: Introduces a New clause to create a new object instance, specifies a constructor constraint on a type parameter, or identifies a Sub procedure as a class constructor. In a declaration or assignment statement, a New clause must specify a defined class from which the instance can be created. This means that the class must expose one or more constructors that the calling code can access.

When the statement runs, it calls the appropriate constructor of the specified class, passing any arguments you have supplied. The following example demonstrates this by creating instances of a Customer class that has two constructors, one that takes no parameters and one that takes a string parameter.

**CSTR Function**: Expression is the value to convert to a string.

CStr(expression)

**Add Method:** Adds a member to a Collection object. Object and item are required. An object expression that evaluates to an object in the Applies To List. Item is an expression of any type that specifies the member to add to the collection. key, before and after are optional. Key is a unique string expression that specifies a key string that can be used, instead of a positional index, to access a member of the collection. Before and after are the expressions that specifies a relative position in the collection. An error also occurs if a specified key duplicates the key for an existing member of the collection.

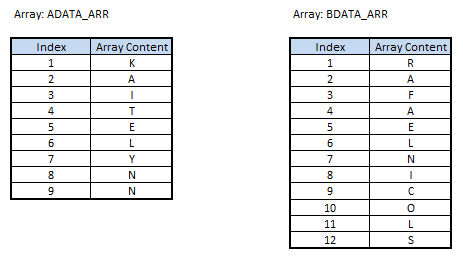
*object***.Add *item*,** ***key*,** ***before*,** ***after***

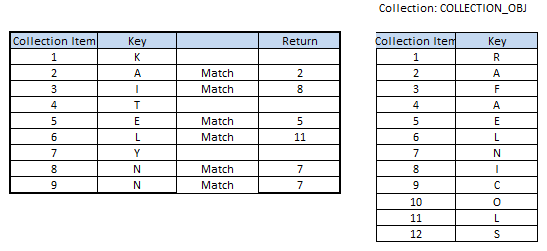
**ReDim Preserve Statement**: ReDim reallocates storage space for an array variable. Preserve is a modifier used to preserve the data in the existing array when you change the size of only the last dimension. Name is the name of the array variable. Boundlist is the list of bounds of each dimension of the redefined array.

ReDim [ Preserve ] name(boundlist) [ ,  name(boundlist) [, ... ] ]

**1. ARRAY\_MATCH\_DATA\_FUNC**

This function use two custom classes that allow you to store and look up multiple data items quickly. It can identify entries that appear in both classes. For simplicity, we call all the entries that present in both arrays "matching entry". It will return these entries' index number in second class. We can use index number to get the matching entries value.





**Parameters**

**ADATA\_RNG** : First one-dimensional array to store and look up data.

**BDATA\_RNG** : Second one-dimensional array to store and look up data.

**Variable**

**i**: Counter variable for the array size of ADATA\_ARR and BDATA\_ARR.

**j**: Counter variable for the number of matching entries .

**k:** Counter variable for the location of entries.

**DATA\_OBJ**: It is declare as a collection to store BDATA\_ARR.

**TEMP\_VAL**: Use to check if ADATA\_ARR has the same entry as BDATA\_ARR.

**TEMP\_ARR:** Array stores matching entries' index number in BDATA\_ARR. This is the array being returned by the main function

**Process**

**Step 1- Set Up the Error Handler:**

When an error is raised, execution transfer to the line label, ERROR\_LABEL:. Then ARRAY\_MATCH\_DATA\_FUNC will return the error number.

On Error GoTo ERROR\_LABEL

...

ERROR\_LABEL:

ARRAY\_MATCH\_DATA\_FUNC = Err.Number

**Step 2 - Create Collection for BDATA:**

First, we create a collection called DATA\_OBJ . k is the counter variable where the array sequence is located in the array. i is the counter variable for BDATA\_ARR. Therefore, i has the same limit between BDATA\_ARR's lower and upper bound. In an array, we can use i to look up the array entry location. In a collection, we use the key to lookup the collection member. Remember we use ADATA\_ARR as the reference array, and check if BDATA\_ARR have any value present in ADATA\_ARR. In a collection, each member will have its specific key. In collection DATA\_OBJ, k is the collection member . The entry in BDATA\_ARR positioned correspond to k will be the key for that k. Remember collection member and key have to be string type. We use Cstr function convert the value to a string.

Dim DATA\_OBJ As New Collection

Dim k As Long

Dim i As Long

k = 1

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

DATA\_OBJ.Add CStr(k), CStr(BDATA\_ARR(i))

k = k + 1

Next i

We start counting k at 1 and i at the lower bound of BDATA\_ARR. If BDATA\_ARR is a one-based array, the first i will also equal to 1. The first member add to collection DATA\_OBJ will be 1 and its key is the value of the first entry in BDATA\_ARR. Then we will increase k by 1 and move the next i. This will keep looping until all the entries in BDATA\_ARR has been add into the collection.

**Step 3- Set Up the Error Trapping:**

When an error is raised, VBA ignores it and resumes execution with the next statement. This error handler apply to the steps below.

On Error Resume Next

**Step 4.0 - Search BDATA\_ARR has Same Entries in ADATA\_ARR:**

In this step, VBA will go through all the entries in array ADATA\_ARR and check if BDATA\_ARR has the same entries.

Dim TEMP\_VAL As Variant

ReDim TEMP\_ARR(1 To 1)

j = 0: k = 1

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

TEMP\_VAL = DATA\_OBJ(CStr(ADATA\_ARR(k)))

...

k = k + 1

Next i

Temp\_ARR is declared as an array that can holds one element. It is the array that will hold all matching entries' position number in BDATA.

j is counter variable that help us to size the TEMP\_ARR. j starts at 0 and k starts at 1. i is the counter variable for ADATA\_ARR, so it has the same lower and upper bound as ADATA\_ARR.

ADATA\_ARR(k) will return the entry located at k position in ADATA\_ARR. We use CStr to cover it to a sting. We can access an item using its string key. CStr(ADATA\_ARR(k) will be the string key. If the key exist in collection DATA\_OBJ, then TEMP\_Val returns the item associate with that key, matching entry's position number in BDATA\_ARR. If it is not a key, there will be an error . Since we already set On Error Resume Next, VBA will continue to the next statement. This will keep looping until i reach its max. This means VBA has checked all the entries in ADATA\_ARR.

On There will be two situations depends whether or not the collection has that key. We will explain them in detail below.

**Step 4.1 - BDATA\_ARR has the Same Entry as ADATA\_ARR :**

Err=0 means ADATA\_ARR(k) is a key in DATA\_OBJ. Therefore, we find an entry that present both in ADATA\_ARR and BDATA\_ARR. Since we find a new match, we need to adjust the size of TEMP\_ARR accordingly. When we increase the size of TEMP\_ARR, we do not want to lose the matching data entry information we already found. We use ReDim Preserve to preserve the data in the existing array since we are only change the size of the only dimension. j is the counter variable for the number of matching array we find. It is also used to define the size of TEMP\_ARR. We will add the matching entry into TEMP\_ARR at the j position.

If Err = 0 Then

j = j + 1

ReDim Preserve TEMP\_ARR(1 To j )

TEMP\_ARR(j) = TEMP\_VAL

**Step 4.1 - Non-Matching Entry:**

An error will raised if the key we are testing is not in the collection. Error number will not equal to 0. Since we did not find a match, we will reset the error number and look for the next key.

Else

Err = 0

End If

**Step 5 - Return Matching Entries Array:**

Return the array contain all matching entries' index number in BDATA\_ARR. We can use these index numbers to get the matching entries' value.

ARRAY\_MATCH\_DATA\_FUNC = TEMP\_ARR

**Step 7- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, contain all entries present in both ADATA\_ARR and BDATA\_ARR

**VBA Code with Annotation**

Function ARRAY\_MATCH\_DATA\_FUNC(ByRef ADATA\_ARR As Variant, \_

ByRef BDATA\_ARR As Variant)

Dim i As Long

Dim j As Long

Dim k As Long

Dim DATA\_OBJ As New Collection '"As New" : way to declare collection. "Collection": name of the object.

'Collections are very useful, but you can only store one item, and you can't store 'types'.

Dim TEMP\_VAL As Variant

Dim TEMP\_ARR() As Variant

On Error GoTo ERROR\_LABEL 'Step 1- Error Handler. If there is an error, tell Excel not to report the error

'and look for the line label. Excel will go to the ERROR\_LABEL below and return error number.

'Step 2 - Create Collection for BDATA

k = 1 'k is the location of the data in BDATA\_ARR

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

DATA\_OBJ.Add CStr(k), CStr(BDATA\_ARR(i)) 'Creating collection

'CStr coverts value to string. In collection, both item and its key are strings.

'First item is the collection member, its value is the location of the array sequence in BDATA\_ARR.

'Second item the item's key, its value is the array entry.

'Array: Array(location number) gives you the array sequence.

'Collection: Collection (key) Use sting key to find the item

k = k + 1

Next i

On Error Resume Next 'Step 3- Set Up the Error Trapping. When an error is raised, VBA ignores it and resumes execution with the next statement.

'Step 4.0 - Search BDATA\_ARR has Same Entries in ADATA\_ARR

ReDim TEMP\_ARR(1 To 1)

j = 0: k = 1 'the syntax ':' allows two lines of code to be placed onto one line

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

'Next line looks up the item from ADATA\_ARR in the collection.

'If a entry in ADATA\_ARR exist in BDATA\_ARR, then TEMP\_VAL returns the value of that item.

'This value is the location of that entry in BDATA\_ARR. Error=0

'If an error raise then we did not find a match.

TEMP\_VAL = DATA\_OBJ(CStr(ADATA\_ARR(k)))

'Step 4.1 - BDATA\_ARR has the Same Entry as ADATA\_ARR

If Err = 0 Then 'No error, we have a match.

'If ADATA\_ARR(k) exist as a key in the collection, then this string also exist in BDATA\_ARR.

'There will be no error, and TEMP\_VAL return the array sequence/index in BDATA\_ARR

**VBA Code with Annotation Continue**

j = j + 1

ReDim Preserve TEMP\_ARR(1 To j)

'Every time we find a match. We increase j by 1, resize TEMP\_ARR to j to accommodate the new match.

'ReDim Preserve allows you to keep the data in an array after adjusting its size.

TEMP\_ARR(j) = TEMP\_VAL 'Adding new match to the return array

'TEMP\_VAL is the collection member, not the key of the collection DATA\_OBJ

'Therefore, this function is returning the location of the matching entry in BDATA\_ARR

'Step 4.1 - Non-Matching Entry

Else 'No match, reset error

Err = 0

End If

k = k + 1

Next i

'Step 5 - Return Matching Entries Array

ARRAY\_MATCH\_DATA\_FUNC = TEMP\_ARR

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

Exit Function 'Step 7- End Function

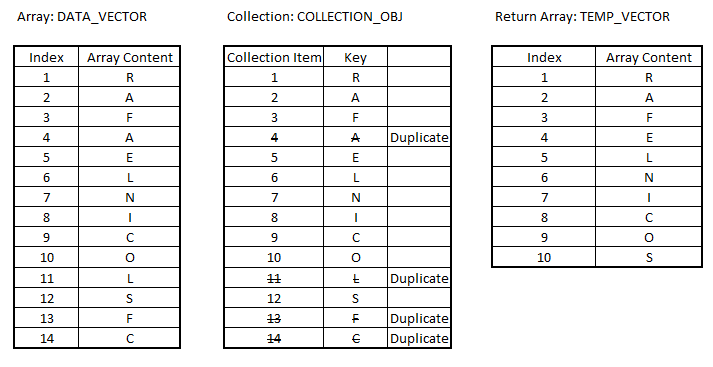
ERROR\_LABEL:

ARRAY\_MATCH\_DATA\_FUNC = Err.Number ''When an error is raised, return Error Number.

End Function

**2. ARRAY\_REMOVE\_DUPLICATES\_FUNC**

This function ultimately removes duplicate rows from a vector. It will input all the array sequence into a collection. It leverage the unique property of collection to identify and remove the duplicate entry. In order to compare array sequences, it is essential to standardize their format. This function will modify and remove the spacing in the entries.



**VBA Application and Function Reference**

**Optional Parameters:** You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called (Also seen in Lesson 3)

**IsArray Function**: Returns a Boolean value indicating whether a variable is an array. IsArray returns True if the variable is an array. otherwise, it returns False

IsArray(varname)

**Collection.Count Property:** Returns an integer containing the number of elements in a collection.

**Trim Function:** Returns a string that contains a copy of string without leading or trailing spaces. There is also LTrim ( without leading spaces) and RTrim ( without trailing spaces).

**CLng Function:** Coverts a value to a long integer.

CLng(expression)

**InStr Function**: Returns the position of the first occurrence of a substring in a string. Start is optional. It is the starting position for the search. If not providing, the search will start at position 1 by default. Compare is also optional. It is the type of comparison to perform. There are 4 values and each of them represent a VBA Constant:

InStr([start], string, substring, [Compare])

|  |  |  |
| --- | --- | --- |
| **VBA Constant** | **Value** | **Explanation** |
| vbUseCompareOption | -1 | Uses option compare |
| vbBinaryCompare | 0 | Binary comparison |
| vbTextCompare | 1 | Textual comparison |
| vbDatabaseCompare | 2 | Comparison based on your database |

**Replace Function**: Replaces a sequence of characters in a string with another set of characters. String is the string we will researching the replacing substring. Find is the substring we will be replaced. Replacement is the sting use to replace find sting. Similar to InStr Function, Replace Function also has Start and Compare. Start is optional. It is the starting position in string1 for the search. If not providing, the search will start at position 1 by default. Compare is also optional. It is the type of comparison to perform and its four values are listed above. Count is also optional and it is use to limit the number of replacement. When count if not given, VBA will replace all find string it can found.

Replace (string1, find, replacement, [start,[count,[compare]]])

**REMOVE\_EXTRA\_SPACES\_FUNC**: This function remove single space in a string, TEMP\_STR. It use InStr Function to locate double space inside the TEMP\_STR. 0 at the end of the InStr Function indicate VBA should perform a binary comparison when comparison functions are called. i is the location of double space in the TEMP\_STR. In order to remove the extra spaces, we use replace function to replace double space with single space. i will equal 0 when VBA cannot find any double space and we will stop the replacement loop. At this point, we can return TEMP\_STR since we have removed all the double space.

**Function** **REMOVE\_EXTRA\_SPACES\_FUNC**(**ByVal** TEMP\_STR **As** **String**) *'DATA\_STR*

**Dim** i **As** **Long**

*'Dim TEMP\_STR As String*

**On** **Error** **GoTo** ERROR\_LABEL

*'TEMP\_STR = DATA\_STR*

i **=** InStr(1, TEMP\_STR, Space(2), 0)

**Do** Until i **=** 0

    TEMP\_STR **=** Replace(TEMP\_STR, Space(2), Space(1), 1, **-**1, 0)

    i **=** InStr(1, TEMP\_STR, Space(2), 0)

**Loop**

REMOVE\_EXTRA\_SPACES\_FUNC **=** TEMP\_STR

**Exit** **Function**

**ERROR\_LABEL**:

REMOVE\_EXTRA\_SPACES\_FUNC **=** Err.number

**End** **Function**

**Parameters**

**DATA\_VECTOR** : One or two-dimensional array we want to remove duplicates from.

**VERSION** : Signal whether or not to recognize single space. When VERSION=0,recognize single space. When VERSION=1, do not recognize single space.

**Variable**

**i**: Counter variable for array sequence location in DATA\_VECTOR and COLLECTION\_OBJ .

**j**: Counter variable for DATA\_VECTOR.

**NROWS:** Size of the DATA\_VECTOR and COLLECTION\_OBJ.

**TEMP\_STR:** Hold individual array sequence as a string.

**TEMP\_VECTOR**: Returned array contain standardize and non-duplicate array sequence.

**COLLECTION\_OBJ**: The collection we use to organize the array sequence.

**Process**

**Step 1- Set Up the Error Handler:**

ARRAY\_REMOVE\_DUPLICATES\_FUNC remove duplicates rows from a vecto. The function cannot proceed if DATA\_VECTOR is not an array.

If IsArray(DATA\_VECTOR) = False Then: GoTo ERROR\_LABEL

...

ERROR\_LABEL:

ARRAY\_REMOVE\_DUPLICATES\_FUNC = Err.Number

IsArray returns True for arrays. IsArray(DATA\_VECTOR) = False, meaning DATA\_VECTOR is not a vector. When this happen, execution transfer to the line label, ERROR\_LABEL:. Then ARRAY\_MATCH\_DATA\_FUNC will return the error number.

**Step 2- Set Up the Error Trapping:**

When an error is raised, VBA ignores it and resumes execution with the next statement. This error handler apply to the steps below.

On Error Resume Next

**Step 3 -Setup Upper Bound limit :**

NROWS is declared as a variable of Long type. NSIZE is set to equal to the size of DATA\_VECTOR's first dimension.

Dim NROWS As Long

NROWS = UBound(DATA\_VECTOR)

**Step 4.0.0- Separate Treatment for 2D and non-2D Array :**

We have separate the following code in two parts depend on if DATA\_VECTOR is a two-dimensional array. IS\_2D\_ARRAY\_FUNC will return True if DATA\_VECTOR is a two-dimensional array, otherwise False. ( Detail of IS\_2D\_ARRAY\_FUNC can be find in lesson 1) We

If IS\_2D\_ARRAY\_FUNC(DATA\_VECTOR) Then

...

Else

...

End If

**Step 4.1.0 - Remove Duplicates for 2D Array :**

If IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = True then the following lines are executed. i is the counter variable where an array sequence is located in DATA\_MATRIX and COLLECTION\_OBJ. i has a range from 1 to NROWS. First, TEMP\_STR is declared as a variable of String type.

Dim i As Long

Dim TEMP\_STR As String

Dim COLLECTION\_OBJ As New Collection

If IS\_2D\_ARRAY\_FUNC(DATA\_VECTOR) Then

For i = 1 To NROWS

TEMP\_STR = DATA\_VECTOR(i, 1)

...

Call COLLECTION\_OBJ.Add(CStr(i), TEMP\_STR)

DATA\_VECTOR(i, 1) = TEMP\_STR

If Err.Number <> 0 Then: Err.Number = 0

Next i

TEMP\_STR will hold the array sequence at first column and i row in DATA\_VECTOR.

The same array data may appear differently because they may have different spacing in the sequence. It is essential for us to standardize all the array sequence. It will make the process to remove duplicates easier. We will standardize all the array sequence in this section. The If...then...else statement separates standardization into two condition. VBA will execute different codes depends on the value of VERSION. VERSION controls whether or not VBA will recognize single space. We will explain these two situation in the next two steps.

COLLECTION\_OBJ is declared as a collection. We will add all the standardized array sequences into the collection. In a collection, each member will have its specific key. In COLLECTION\_OBJ, i (index number) is the collection member . TEMP\_STR, the entry in DATA\_VECTOR positioned correspond to that i will be the key. Remember collection member and key have to be string type. We use Cstr function convert i to a string. Since TEMP\_STR is already a string, we do not need to apply CStr function. We will also return the standardized array sequences into its corresponding position in DATA\_VECTOR.

In a collection, each collection member will have its own key. An error will raise if two or more collection member have the same key. In this case, it indicate we have found a duplicate. When errors raise (ERR.Number<>0), VBA simply reset the error number to 0 and continue to the next line. The duplicate will not be added to COLLECTION\_OBJ. It will keep the duplicate in DATA\_VECTOR. Next i simply means we will apply this procedure to each array sequence one by one.

**Step 4.1.0.1 - Array Standardization Recognize Single Space:**

When VERSION=0 indicate we want VBA to recognize single space. First, REMOVE\_EXTRA\_SPACES\_FUNC to replace all the double spaces in the TEMP\_STR with single space. Then, Trim removes spaces from both ends of the newly modified TEMP\_STR. Finally, we replace the original TEMP\_STR with the modified TEMP\_STR.

If VERSION = 0 Then

TEMP\_STR = Trim(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR))

**Step 4.1.0.2 -Standardization Not Recognize Single Space:**

VERSION is not 0 indicate we do not want VBA to recognize single space. First, REMOVE\_EXTRA\_SPACES\_FUNC to replace all the double spaces in the TEMP\_STR with single space. Then, Replace function replace all the single space into no space. Next, Trim removes spaces from both ends of the newly modified TEMP\_STR. Finally, we replace the original TEMP\_STR with the modified TEMP\_STR.

Else

TEMP\_STR = Trim(Replace(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR), " ", ""))

End If

**Step 4.1.1 -Adjust Upper Bound:**

COLLECTION\_OBJ.Count returns the number of elements in the collection. Since we have removed the duplicate array sequence, we need to adjust NROWS.

NROWS = COLLECTION\_OBJ.Count

**Step 4.1.2 - ReDim TEMP\_VECTOR :**

TEMP\_VECTOR is the array return of ARRAY\_REMOVE\_DUPLICATES\_FUNC. We need to adjust TEMP\_VECTOR according to the size of after-remove-duplicates COLLECTION\_OBJ.

Dim j As Long

Dim TEMP\_VECTOR As VariantReDim TEMP\_VECTOR(1 To NROWS, 1 To 1)

For i = 1 To NROWS

j = CLng(COLLECTION\_OBJ.Item(i))

TEMP\_VECTOR(i, 1) = DATA\_VECTOR(j, 1)

Next i

TEMP\_VECTOR is resized to 1 to NROWS and 1 to 1 for the first and second dimension, respectively. i is the counter variable for location of the collection after remove the duplicates. Value of i for a certain array sequence may be different from the array sequence's index number in the original array. j is declared as the counter variable represent the location of the array sequence in the original array. COLLECTION\_OBJ.Item(i) returns the collection item member i, which is the string of the value the original array index number. We will return array sequence from DATA\_VECTOR into TEMP\_VECTOR. It will continue looping until we have input all the none duplicate standardized array sequence into TEMP\_VECTOR.

**Step 4.2.0 - Remove Duplicates for 1D Array :**

When IS\_2D\_ARRAY\_FUNC(DATA\_MATRIX) = False, the following lines are executed. Since DATA\_VECTOR is a one-dimensional array, we will obtain array sequence from its own dimension. We do not need to worry about transposing the array.

For i = 1 To NROWS

TEMP\_STR = DATA\_VECTOR(i)

....

Call COLLECTION\_OBJ.Add(CStr(i), TEMP\_STR)

DATA\_VECTOR(i) = TEMP\_STR

If Err.Number <> 0 Then: Err.Number = 0

Next i

The rest of the procedure will be similar to Step 4.1.0.

i is the counter variable where an array sequence is located in DATA\_VECOTR and then COLLECTION\_OBJ. i has a range from 1 to NROWS. First, TEMP\_STR is declared as a variable of String type. TEMP\_STR will hold the array sequence at i in DATA\_VECTOR.

The same array data may appear differently because they may have different spacing in the sequence. It is essential for us to standardize all the array sequence. It will make the process to remove duplicates easier. We will standardize all the array sequence in this section. The If...then...else statement separates standardization into two condition. VBA will execute different codes depends on the value of VERSION. VERSION controls whether or not VBA will recognize single space. We will explain these two situation in the next two steps.

COLLECTION\_OBJ is declared as a collection. We will add all the standardized array sequences into the collection. In a collection, each member will have its specific key. In COLLECTION\_OBJ, i (index number) is the collection member . TEMP\_STR, the entry in DATA\_VECTOR positioned correspond to that i will be the key. Remember collection member and key have to be string type. We use Cstr function convert i to a string. Since TEMP\_STR is already a string, we do not need to apply CStr function. We will also return the standardized array sequences into its corresponding position in DATA\_VECTOR.

In a collection, each collection member will have its own key. An error will raise if two or more collection member have the same key. In this case, it indicate we have found a duplicate. When errors raise (ERR.Number<>0), VBA simply reset the error number to 0 and continue to the next line. The duplicate will not be added to COLLECTION\_OBJ. It will keep the duplicate in DATA\_VECTOR. Next i simply means we will apply this procedure to each array sequence one by one.

**Step 4.2.0.1 - Array Standardization Recognize Single Space:**

This step is the same as Step 4.1.0.1. When VERSION=0 indicate we want VBA to recognize single space. First, REMOVE\_EXTRA\_SPACES\_FUNC to replace all the double spaces in the TEMP\_STR with single space. Then, Trim removes spaces from both ends of the newly modified TEMP\_STR. Finally, we replace the original TEMP\_STR with the modified TEMP\_STR.

If VERSION = 0 Then

TEMP\_STR = Trim(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR))

**Step 4.2.0.2 -Standardization Not Recognize Single Space:**

**This step is the same as Step 4.1.0.2.**

VERSION is not 0 indicate we do not want VBA to recognize single space. First, REMOVE\_EXTRA\_SPACES\_FUNC to replace all the double spaces in the TEMP\_STR with single space. Then, Replace function replace all the single space into no space. Next, Trim removes spaces from both ends of the newly modified TEMP\_STR. Finally, we replace the original TEMP\_STR with the modified TEMP\_STR.

Else

TEMP\_STR = Trim(Replace(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR), " ", ""))

End If

**Step 4.2.1-Adjust Upper Bound:**

This step is the same as Step 4.1.1. COLLECTION\_OBJ.Count returns the number of elements in the collection. Since we have removed the duplicate array sequence, we need to adjust NROWS.

NROWS = COLLECTION\_OBJ.Count

**Step 4.2.2 - ReDim TEMP\_VECTOR :**

This step is similar as Step 4.1.2. TEMP\_VECTOR is the array return of ARRAY\_REMOVE\_DUPLICATES\_FUNC. We need to adjust TEMP\_VECTOR according to the size of after-remove-duplicates COLLECTION\_OBJ.

ReDim TEMP\_VECTOR(1 To NROWS)

For i = 1 To NROWS

j = CLng(COLLECTION\_OBJ.Item(i))

TEMP\_VECTOR(i) = DATA\_VECTOR(j)

Next i

TEMP\_VECTOR is resized to 1 to NROWS. i is the counter variable for location of the collection after remove the duplicates. Value of i for a certain array sequence may be different from the array sequence's index number in the original array. j is declared as the counter variable represent the location of the array sequence in the original array. COLLECTION\_OBJ.Item(i) returns the collection item member i, which is the string of the value the original array index number. We will return array sequence from DATA\_VECTOR into TEMP\_VECTOR. It will continue looping until we have input all the none duplicate standardized array sequence into TEMP\_VECTOR.

**Step 5 - Return Matching Entries Array:**

Return the array contain the standardize and none duplicated array sequence of the original array.

ARRAY\_REMOVE\_DUPLICATES\_FUNC = TEMP\_VECTOR

**Step 6- End Function:**

End the function.

Exit Function

End Function

**Returns**

Array of variant type, contain all the standardize non-duplicated array sequence from DATA\_VECTOR.

**VBA Code with Annotation**

Function ARRAY\_REMOVE\_DUPLICATES\_FUNC(ByRef DATA\_VECTOR As Variant, \_

Optional ByVal VERSION As Integer = 0)

'VERSION 0 --> Recognize single space. VERSION 1 --> Do not recognize single space

Dim i As Long 'Counter for COLLECTION\_OBJ

Dim j As Long 'None duplicated array location of in DATA\_VECTOR

Dim NROWS As Long

Dim TEMP\_STR As String

'If you want to use this function inside excel instead of calling the main routine

'you must change the input DATA\_VECTOR for DATA\_RNG and declare DATA\_VECTOR again

'inside the function. This is a trick to save memory, so we don't have two arrays declared.

Dim TEMP\_VECTOR As Variant

Dim COLLECTION\_OBJ As New Collection '"As New" : declare collection. "Collection": name of the object.

'Collection: Collection (key) Use string key to find the item.

If IsArray(DATA\_VECTOR) = False Then: GoTo ERROR\_LABEL 'Step 1- Set Up the Error Handler

'If DATA\_VECTOR is not a array, then return an error because there are no duplicate, in fact,

'there is only a single value

On Error Resume Next 'Step 2- Set Up the Error Trapping. When an error is raised, VBA ignores it and 'resumes execution with the next statement.

NROWS = UBound(DATA\_VECTOR) 'Step 3 -Setup Upper Bound limit

'Step 4.0.0- Separate Treatment for 2D and non-2D Array

If IS\_2D\_ARRAY\_FUNC(DATA\_VECTOR) Then 'Step 4.1.0 - Remove Duplicates for 2D Array

For i = 1 To NROWS

TEMP\_STR = DATA\_VECTOR(i, 1) 'Covert array sequence as a string

If VERSION = 0 Then 'Step 4.1.0.1 - Array Standardization Recognize Single Space:

TEMP\_STR = Trim(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR)) 'Remove double spaces and ending

Else 'Step 4.1.0.2 -Standardization Not Recognize Single Space

TEMP\_STR = Trim(Replace(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR), " ", "")) 'Remove all spaces and ending

End If

Call COLLECTION\_OBJ.Add(CStr(i), TEMP\_STR) 'Creating collection

'CStr coverts value to string. In collection, both item and its key are strings.

'First item is the collection member, its value is the location of the array sequence in COLLECTION\_OBJ.

'Second item the item's key, its value is the array entry.

DATA\_VECTOR(i, 1) = TEMP\_STR

If Err.Number <> 0 Then: Err.Number = 0 'Found Duplicate

Next i

**VBA Code with Annotation Continue**

NROWS = COLLECTION\_OBJ.Count 'Step 4.1.1 -Adjust Upper Bound

'Step 4.1.2 - ReDim TEMP\_VECTOR

ReDim TEMP\_VECTOR(1 To NROWS, 1 To 1)

For i = 1 To NROWS

j = CLng(COLLECTION\_OBJ.Item(i)) 'CLng convert string to Long

TEMP\_VECTOR(i, 1) = DATA\_VECTOR(j, 1) 'Input non-duplicate standardize array sequence into return array.

Next i

Else 'Step 4.2.0 - Remove Duplicates for 1D Array

For i = 1 To NROWS

TEMP\_STR = DATA\_VECTOR(i)

If VERSION = 0 Then 'Step 4.2.0.1 - Array Standardization Recognize Single Space

TEMP\_STR = Trim(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR))

Else 'Step 4.2.0.2 -Standardization Not Recognize Single Space

TEMP\_STR = Trim(Replace(REMOVE\_EXTRA\_SPACES\_FUNC(TEMP\_STR), " ", ""))

End If

Call COLLECTION\_OBJ.Add(CStr(i), TEMP\_STR)

DATA\_VECTOR(i) = TEMP\_STR

If Err.Number <> 0 Then: Err.Number = 0 'Found Duplicate

Next i

'Step 4.2.1-Adjust Upper Bound

NROWS = COLLECTION\_OBJ.Count

'Step 4.2.2 - ReDim TEMP\_VECTOR

ReDim TEMP\_VECTOR(1 To NROWS)

For i = 1 To NROWS

j = CLng(COLLECTION\_OBJ.Item(i))

TEMP\_VECTOR(i) = DATA\_VECTOR(j)

Next i

End If

'Step 5 - Return Matching Entries Array

ARRAY\_REMOVE\_DUPLICATES\_FUNC = TEMP\_VECTOR

'Step 6- End Function

Exit Function

ERROR\_LABEL: 'When an error is raised, return Error Number.

ARRAY\_REMOVE\_DUPLICATES\_FUNC = Err.Number

End Function