Introduction to VBA for Excel-Tutorial 6

In this tutorial, you will learn deal with arrays. We will first review how to declare the arrays, then how to pass data in and how to output arrays to Excel environment. First, what is an array? In simple words, an array is a group of values referenced as one variable name. An example can be demonstrated by considering time range in a projectile motion. So instead of referring to t_0, t_1, t_2 …… , t_n, we can define one time array and collect all these values in that one variable. Arrays are used a lot in all disciplines of engineering and if you understand the concept, the knowledge can be easily adapted to other programming languages.

Declaring an array:

The syntax to declare an array starts by using the Dim statement, such that:

Dim *aname*(*No_of_elements*) As *type*

where, *aname* is the array variable name, *No_of_elements* is the number of values that can be assigned to the array, and *type* is the data type which are similar to those types discussed before. Examples:

```
Dim time(5) As single

Dim velocity(1 To 100) As Double

Dim displacement(0 To 100) As Long
```

Now, do you remember the difference between the second and third statement above? You should remember from tutorial # 4. Read the following carefully: In the second statement, you defined an array with 100 elements, where the start index (sometimes referred to as the lower index) is 1 and the end index (sometimes referred to as higher index) is 100. In the third statement, you defined an array with 101 elements, where the start index is 0 and the end index is 100. To elevate this confusion and resolve this issue once and for all, we will force VBA to start all arrays with 1 as the lower index. Add the following statement following the Option Explicit statement as we discussed in tutorial #4.

Example 1: Let's say you want to create a time array which values range from 0-100 and then output the created array to Excel environment.

```
Option Explicit
Option Base 1

Sub testarrary()
'declaring the variables
Dim time(100) As Integer
Dim i As Integer
'looping to create the array
For i = 1 To 100 Step 1
time(i) = i
Cells(i, 1).Value = i 'output
Next i
End Sub
```

Example 2: Let's say you want to perform the following algebraic equation

$$y = 3x_1 + 4x_2 - 7x_3$$

Thus, you expect the user to enter three values. For sake of demonstration, let's say that these values are stored in A1:A3. The code would be

```
Option Explicit
Option Base 1

Sub testarrary()
'declaring the variables
Dim x(3) As Double
Dim y As Double
Dim i As Integer
'looping to create the array
For i = 1 To 3 Step 1
x(i) = Cells(i, 1).Value
Next i
'calculating y from the given equation
y = 3 * x(1) + 4 * x(2) - 7 * x(3)
'output
MsgBox "The answer is " & y
End Sub
```

Well, if you noticed we defined the size of the array a prior to starting the code. This luxury of knowing before hand is not always available to us. Therefore two questions must be answered (1) Can we resize the array after the initial declaration? And (2) Can we define the size of an array based on the number of values on the worksheet (i.e. A1:An, where n is not known)? Two good questions and the answers for both are Yes!

Array Resizing:

We are required to declare all variables before coding by Option Explicit statement and also to have good budget on the memory. However, sometimes the size of the array needs to change or need to be decided later. This method will give you the ability to change the size of the array as needed. The syntax for declaring the array is:

```
Dim time() As Single
```

Here, we declared *time* as an array without specific size; therefore it is assigned the biggest size possible and allowed by VBA. Then, the syntax for resizing or assigning a size:

```
ReDim time(5)
```

Here, we made the size of the *time* array equals to five. You can resize the array as many times as you see needed. It is also known as "dynamic sizing." Remember, you cannot use ReDim statement without declaring the variable first.

Example: If you want to write a program that computes the average of class grade the high and the low to be used for multiple classes and thus the number of students will be different for one class to the other. The code would be:

```
Option Explicit
Option Base 1
```

```
Sub gradecalcs()
Dim grade() As Single 'array to keep every student grade
Dim low As Integer, high As Integer, average As Double, sum1 As Double
Dim n As Integer 'number of students
Dim i As Integer 'For loop counter
Dim Another_Average As Double
Dim Another_sum As Double
n = Val(InputBox("Enter Number of students"))
ReDim grade(n) 'resize grade array based on the number of students
sum1 = 0
low = 100
high = 0
For i = 1 To n
  grade(i) = Cells(i + 1, 1).Value
  sum1 = sum1 + grade(i)
  If grade(i) > high Then
  high = grade(i)
  End If
  If grade(i) < low Then
  low = grade(i)
  End If
Next i
'the following is another way to calculate the sum using worksheet functions
Another_sum = Application.WorksheetFunction.sum(grade())
average = sum1 / n
'the following is another way to calculate the average using worksheet functions
Another_Average = Application.WorksheetFunction.average(grade())
Cells(1, 2).Value = "average": Cells(1, 3).Value = average
Cells(2, 2).Value = "Another_Average": Cells(2, 3).Value = Another_Average
Cells(3, 2).Value = "Low": Cells(3, 3).Value = low
Cells(4, 2).Value = "High": Cells(4, 3).Value = high
End Sub
```

Sizing based on Excel Input:

Rather than specifying the size of the array by the user's input or by hard-coding the number as in the previous examples. Let's have the program count how many cells contain data and pass this as the size of the array.

1- Using cells properties: the following code demonstrate how you can use the active cell row number and relate that to the size of the array.

```
Option Explicit
Option Base 1
Sub demo()
Dim asize As Integer
Dim adata() As Long
'determine the size of the array
Sheets("sheet1").Select
Range("a1").Select
asize = ActiveCell.Row
Selection.End(xlDown).Select
asize = ActiveCell.Row - asize + 1
ReDim adata(asize)
'output
MsgBox "Array Size" & asize
End Sub
```

Another way to accomplish the same task:

```
Option Explicit
Option Base 1

Sub demo2()
Dim in_array() As Integer
Dim n As Integer
  Sheets("sheet1").Select
  Range("a1").Select
  Range(Selection, Selection.End(xlDown)).Select
  n = Selection.Count
  ReDim in_array(n)
  MsgBox n
End Sub
```

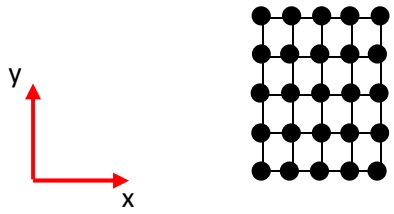
2- Using Loops: In this method, we will go though each cell and check if the cells is full or empty, once we find an empty cell the loop will stop and report the size of the array.

```
Option Explicit
Option Base 1

Sub demo3()
Dim in_array() As Integer
Dim i As Integer
Dim n As Integer
Dim k As Integer
i = ActiveCell.Row
k = 0
Do While Cells(i + 1, 1) <> Empty
k = k + 1
i = i + 1
Loop
ReDim in_array(k)
MsgBox k
End Sub
```

Multidimensional Arrays:

Thus far, we have been practicing on 1D array or vectors, which can used to represent time, distance in one direction, velocity etc. However, higher order arrays are used as frequent in engineering as 1D array. Let's say that want to enter the temperature distribution in a plate, which is defined by 5 nodes in the x-direction and 5 nodes in y-direction.



The syntax to declare the array:

```
Dim temp(5,5) As Single

Dim temp(1 To 5, 1 To 5) As Single
```

Now to fill this 2D array, you have to use nested for loops such as:

```
For i = 1 To 5
  For j = 1 To 5
  temp(i, j) = Cells(i, j).Value
  Next j
Next i
```