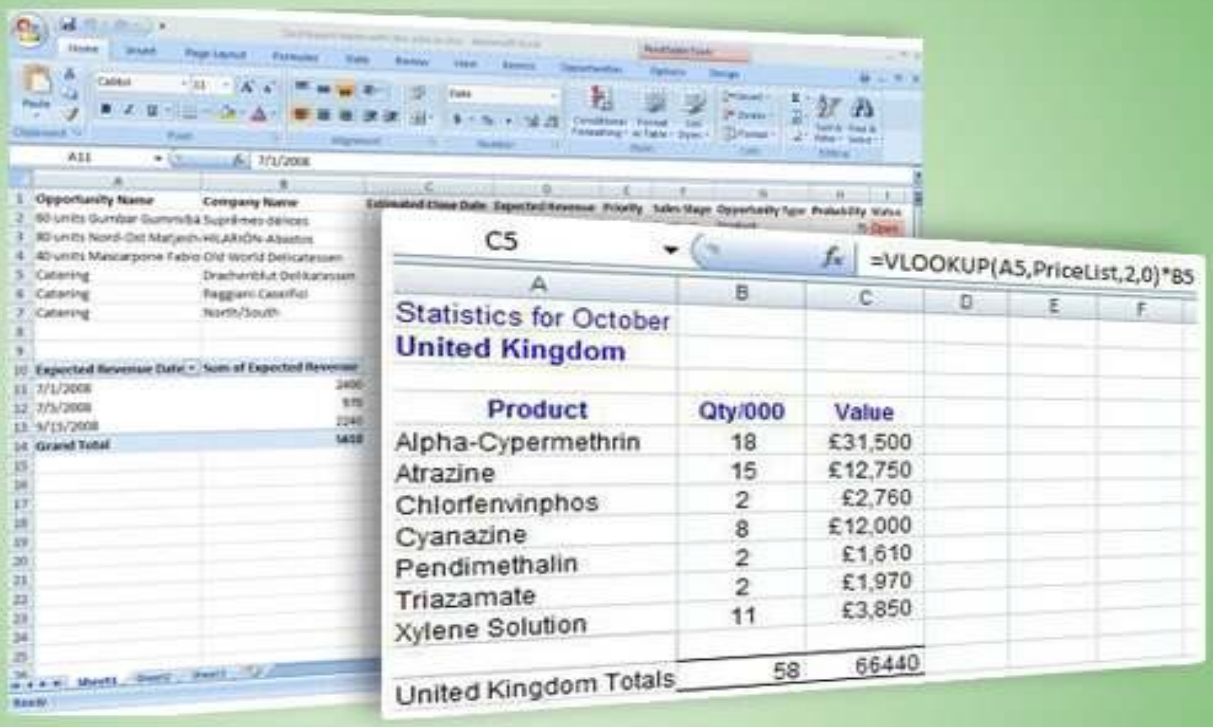
Mastering Excel Macros

Arrays - Book 10



Mark Moore

Mastering Excel Macros

Arrays – Book 10

Mark Moore

Introduction

Welcome to another Mastering Excel lesson. If you have previous lessons, thanks for sticking around If you are new, I hope you enjoy the lesson. The lessons are an easy-going, relaxed, no-nonsense easy to understand. I try my best to explain complex topics in a simple and entertaining way. My goal is that you will finish reading each lesson and have immediately applicable skills you can use at work o home.

If you want to work along the exercises in this lesson (I strongly recommend this) please go to my website and download the follow-along workbook. My website is:
http://markmoorebooks.com/mastering-excel-macros-arrays/

A bit of clarification on how to get the follow-along workbooks. You will input your name and email address. You will receive a confirmation email. Once you confirm, you will receive a second email with the follow-along workbook.

Why do I do this?

I can't package an Excel file with an eBook. Amazon will not allow it. Also, the only thing I do with your email is send you the workbook and periodically send you updates about new lessons that I am working on.

Introduction

Suppose you have thousands of data points in a worksheet that need to be processed. You could populate a single variable with each cell value, process it, then put the new value back in the cell. Thi would work but it is very, very slow. Alternatively, you could create thousands of variables in your macros (i.e. Item1, Item2, Item3, etc.), populate them, process them, then put them back in the worksheet. This is marginally better. However, do you really want to be managing thousands of variables? Of course not.

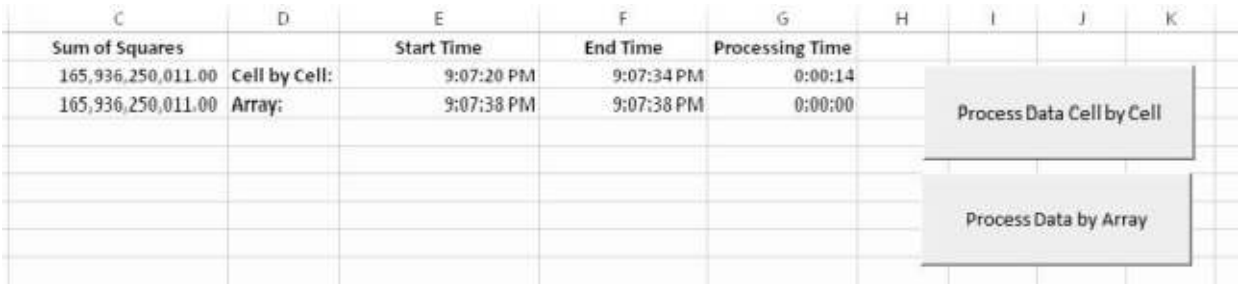
VBA has special types of variables that let you manage large amounts of data easily. They are Arrays Collections and the Dictionary. This lesson will focus on how to use these objects to greatly speed up your macros.

As an example, let's look at the follow-along workbook, Speed.xlsx. This workbook has about 5,000 data points in column A. I need to calculate the sum of squares for column A. In other words, I need t raise the number to the power of 2 and then take the sum of all the squares.

There are two macros in the workbook. They each take the value from a cell in column A, square it and store the value. One macro does this cell by cell, the second macro does it by using an array.

Look at the performance improvement just by using arrays! It took 14 seconds to process the data cel by cell but just milliseconds to use an array.

If you don't believe me, go ahead and run the macros to see for yourself.



Why is the improvement so drastic? Because array are stored in the computer's memory. It is exponentially faster for a computer to use memory space for calculations versus having to constantly go back and read the data cell by cell.

If you are dealing with large amounts of data and calculations, you are better off using arrays, collections or dictionaries.

Arrays

An array is a variable that can store multiple values of the same type. You can think of an array as a shoe rack. Each pair of shoes has a location where you can store the shoes until you need them. An array is similar except that instead of shoes you are storing data **of the same type**. Shoe racks are onl for shoes; you can't stick a coat in there (well you could, but it would get all wrinkled). In an array,

you have to declare the data type and then put data in it that matches the data type. For example, if yo create an array to store integers, you can't store text in it.

Arrays are excellent for when you need to:

- Store many numbers and use them in calculations
- Change the numbers in the array to other numbers

Arrays are not great when you need to:

- Find one particular number
- Sort the numbers

I want to set the expectations for these exercises before I start. These exercises are designed to be simple on purpose. Almost everything you are going to do in this lesson can be done using formulas. The goal here is not to show you the best way to perform a calculation in Excel, the goal is to show you how to do it in VBA.

If you find yourself thinking, 'Why would I do this task this way when a simple formula would work then suppose you are working on a very large, complex workbook that takes 10 or more minutes to calculate. Now the convenience of doing the calculation in a macro versus waiting for 10 minutes becomes apparent.

Types of Arrays

Arrays can be *static* or *dynamic*. Static arrays are fixed in size; you know exactly how many items yo are going to store. Dynamic arrays are variable and can be resized as needed.

Array Dimensionality

Arrays have the concept of a dimension. You can think of a dimension as a field. A one-dimensional array will have one field available to store data. A two-dimensional array will have two fields to stor data. You can have as many dimensions as you like in an array, but anything past two dimensions get really tricky to work with. This lesson will not go into the topic of arrays with more than two dimensions.

Let's begin with a hands-on exercise. I will explain new topics as you work through the exercise.

The goal of this first exercise is to create a one-dimensional array, calculate the average score of all the students, and put the average score in a cell in the worksheet.

1. Open the follow-along workbook, One-dimensional Array.xlsm.

This is an image of the sample data you will be working with.

	A	B	C	D	E	F	G
1	Student Name	Test Score					
2	Ji Kratz	86			Average:		
3	Anderson Fentress	92					
4	Terina Milholland	98					
5	Leila Allyn	53					
6	Leonarda Scruggs	70					
7	Shawnta Barbeau	79					
8	Adelaide Mccalla	82					

2. Insert a new VBA module (Developer>Visual Basic).
3. Create a new macro called CalcAverage.

```
Sub CalcAverage()


End Sub
```

Declaring Arrays

The first thing you have to do when using array is to declare (i.e. create) it. When you declare an array, Excel sets apart a portion of memory to handle that data.

The data has 50 students. You are going to create a 50-item static array to process the data.

Data Types

I've covered data types before, but let's have a quick refresher. Specifying the exact type of data that the array will store will make the array more efficient. Excel will store just enough memory to handl the data. You can use a 'bigger' data type that is needed and it won't break anything. Honestly, you might never see any performance improvements depending on how complex your code or calculation are.

VBA also has a data type called Variant. This is a catch-all data type that can store both text and numbers. However, it is not as efficient as the other data types.

For your reference, here are the various data types you can use:

Data Type	Range of Values
Boolean	True or False
Integer	-32,768 to 32767
Long	-2,147,483,648 to 2,147,483,647
Single	-3.402823E38 to 1.401298E45
Double (negative)	-1.79769313486232E308 to -4.94065645841247E-324
Double (positive)	4.94065645841247E-324 to 1.79769313486232E308
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Date	1/1/100 to 12/31/9999
String	Varies
Object	Any defined object
Variant	Any data type

4. Declare a new 50-item static array of type Integer. Name the array Scores.

```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer


End Sub
```

A one-dimensional array is really just a list of numbers. That's it. In this case, you created a 50-item list of numbers. This array starts a 1 and extends up to 50 items. As you get exposed to arrays, you will notice that some arrays start at 0, not 1. The default behavior is for an array to start at 0. It's not that big of a deal, just remember that if you start at 0, you will need one less item. For example, if I started at 0, the array would be 0 to 49, since the first score would be in the 0 spot.

The 'As Integer' tells Excel that each one of those 50 spots will store an integer. I hope that now it makes sense when I initially told you that arrays store data of the same type. If you must mix and match data types, then use the Variant data type. That one can store both types but then you will have to write more code to test which data type is currently being processed.

Populating the Array

Now you have to get the data from Excel into the array. I'm going to show you two ways to do this.

Populating by Looping

The first method of populating an array is by writing a loop. You know there are 50 students, the arra is 50 items large, therefore the loop has to process 50 cells.

5. Write code to select cell A1.
6. Write a 50 item For Each loop.

```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer

    'Select cell A1
    Range("A1").Select

    ' Loop 50 times
    For i = 1 To 50

    Next i


End Sub
```

To get the data from Excel into the array, you're probably thinking, "I'm going to move to the first cell, load that data, move to the second cell, load that data and do that 50 times." Yes, that would wo but it is way slow. Imagine you have 1 million rows, moving from cell to cell will take several hours

Excel knows where the cells are. You don't have to move to the cell to get the data. I am going to show you one way to get the data in without having to move the Active Cell from A1.

Loading data into an array is done by position. For example, Scores(1) = 86 will load 86 into the first spot in the array. Scores(1) = ActiveCell.Value will load the contents of the ActiveCell into spot 1 of the array.

You are not going to use the ActiveCell property to load data, that would be the equivalent of moving from cell to cell. Instead you are going to use the OFFSET function.

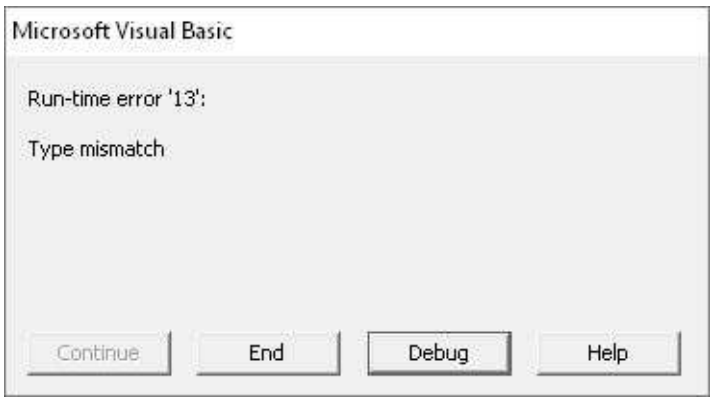
7. Load data into the array using the OFFSET function.

```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer

    'Select cell A1
    Range("A1").Select

    ' Loop 50 times
    For i = 1 To 50
        Scores(i) = ActiveCell.Offset(i, 0).Value
    Next i


End Sub
```

i is the variable that I used to count from 1 to 50.
The i variable is doing double duty, I'm also using it to move from spot to spot in the array.
The OFFSET function returns data from a cell that is x rows and y columns away from the ActiveCel I'm making the i variable do triple duty by using it as the offset to return data from the cell that is i rows and 0 columns away. i rows and 0 columns away means down the same column as the active cel (column A in this case).

8. Run the macro.

What happened? Did you get this error?



Although the error message is not that descriptive, looking at the data you can figure out what is wrong. You selected the start point as A1. Column A has the names of the students as text, the array i expecting integers since we are trying to load the numbers.

It's an easy fix. Instead of selecting cell A1, select cell B1. That's the column where the numbers are

```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer

    'Select cell A1
    Range("B1").Select

    ' Loop 50 times
    For i = 1 To 50
        Scores(i) = ActiveCell.Offset(i, 0).Value
    Next i


End Sub
```

You can run the macro now and there should be no errors. However, it doesn't 'do' anything other than load the array. You can't even see if it did it correctly.

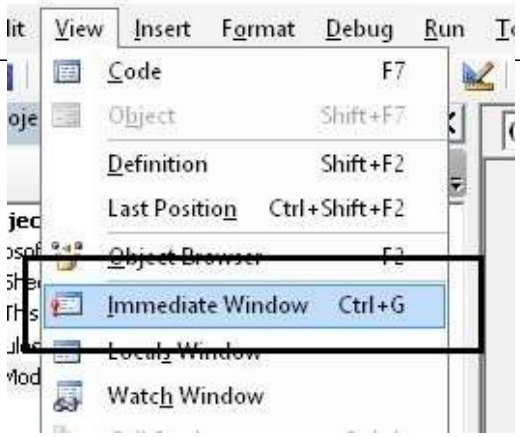
Show Specific Values of an Array

Once the macro finishes running, the array is destroyed. You need to create a breakpoint in the code and then use the Immediate window to see the individual values. Let's do that now.

9. Click on the row header at End Sub to create a breakpoint in the code.

Now when you run the macro, the code will stop right before the macro ends. At this point, the array has been completely loaded.

10. Show the Immediate window.

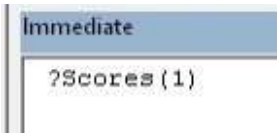


11. Run the macro.

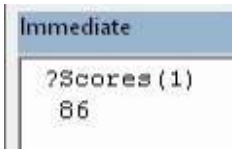
```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer

    'Select cell A1
    Range("B1").Select

    ' Loop 50 times
    For i = 1 To 50
        Scores(i) = ActiveCell.Offset(i, 0).Value
    Next i


End Sub
```

The yellow means the macro has paused at the breakpoint line.

12. Type this in the immediate window to see the value of Scores(1).



13. Press Enter to evaluate the expression.



You can change the number in the parenthesis to check a few numbers, if you'd like.

Keep in mind that the order of the numbers is irrelevant. Also note that you cannot write code that retrieves the data for a specific student; this array is just a list of numbers.

14. Click stop to stop the macro.

Now you just have to finish the macro by adding lines that calculate the average. Unfortunately, there

is no built-in formula to calculate the average of an array, you will have to do it yourself. The averag of a set of numbers is calculated as the sum of all the numbers divided by the count of the numbers. I this case it would be the sum of all the numbers / 50.

After loading the data to the array, you can create a new variable to aggregate the new values. After the loop is finished, you will put the result into cell F2. It will make much more sense when you see the example.

15. Click the red dot in the breakpoint line to remove the breakpoint.
16. Add lines to calculate the average and put the result in cell F2.

```
Sub CalcAverage()
    Dim Scores(1 To 50) As Integer

    'Select cell A1
    Range("B1").Select

    ' Loop 50 times
    For i = 1 To 50
        Scores(i) = ActiveCell.Offset(i, 0).Value
        Total = Total + Scores(i)
    Next i
    Range("F2").Value = Total / 50

End Sub
```

When you run the macro, the array will be populated; the average will then be calculated and put in cell F2.

Populate by Range

Now, I'm going to show you another way to load data into an array where you don't have to process c point to each cell. As long as the source data size matches the array size, you can load a range into an array. Excel is intelligent enough to know that each cell corresponds to a lot in the array.

For example, you can use this line to populate the array:

```
Scores = Range("B2:B51").Value
```

The above line would replace the `Scores(i) = ActiveCell.Offset(i,0).Value` and it would go outside the loop. Yo would still need to loop through each item to calculate the average.

The trick to this method is that the data type of the array must be Variant. You will get an error you use another data type.

Dynamic Arrays

What about if you don't know how many students are going to be in the worksheet? For this scenario a static array won't work. Instead you need to use a dynamic array. Dynamic arrays work just like static arrays except you have the additional task of writing a few extra lines of code telling Excel how large the array is going to be.

17. Create a new macro called CalcAverageDynamic.

```
Sub CalcAverageDynamic()


End Sub
```

To declare a dynamic array, you don't put a size in between the parentheses. For example, Scores () as Integer.

18. Declare the Scores array as a dynamic array.

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

End Sub
```

19. Add the line to select cell B1.

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

End Sub
```

How many items are going to be in the loop? You need to figure that out. In a worksheet you would use the COUNTA function to count the number of non-blank cells in column B. Guess what? You can use Excel formulas in a macro. You just have to use them by using the WorksheetFunction.

To count the number of non-blank cells in column B the macro code would be:

```
arrSize = WorksheetFunction.CountA(Range("B:B"))
```

20. Add the line that calculates the number of data rows (which is equal to the array size).

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

    'Calculate array size
    arrSize = WorksheetFunction.CountA(Range("B:B"))

End Sub
```

The ReDim statement resizes an array. The syntax is the same as the Dim statement. In this example you are resizing the array once, however, you can resize the array as many times as necessary.

21. Resize the array to 50.

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

    'Calculate array size
    ArrSize = WorksheetFunction.CountA(Range("B:B"))

    'resize the array
    ReDim Scores(1 To ArrSize)

End Sub
```

22. Populate the array (use the shortcut way with the range).

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

    'Calculate array size
    ArrSize = WorksheetFunction.CountA(Range("B:B"))

    'Resize the array
    ReDim Scores(1 To ArrSize)

    'Populate array
    Scores = Range("B2:B51").Value

End Sub
```

Now you have a choice, you need to loop through the array and calculate the average. You could do something like:

For i = 1 to arrSize

Next i

That would work fine. However, let me show you another way. Instead of using a For Loop, you can use a For Each loop. The For Each loop will cycle through all objects in a collection. An array can be thought of as a collection; it is a grouping of similar items, and the For Each loop is well suited for th task.

23. Use a For Each loop to loop through each item in the array.

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

    'Calculate array size
    ArrSize = WorksheetFunction.CountA(Range("B:B"))

    'Resize the array
    ReDim Scores(1 To ArrSize)

    'Populate array
    Scores = Range("B2:B51").Value

    'Loop through each item
    For Each c In Scores
        Total = Total + c
    Next c

End Sub
```

Note: In case you forgot, the c in the loop is a variable I created. It doesn't have to be c. You can call it b, Bobby, Santa, etc.

For Each Loop Benefits

- You don't need to know how large the array is. The loop will process every element in th array.

For Each Loop Drawbacks

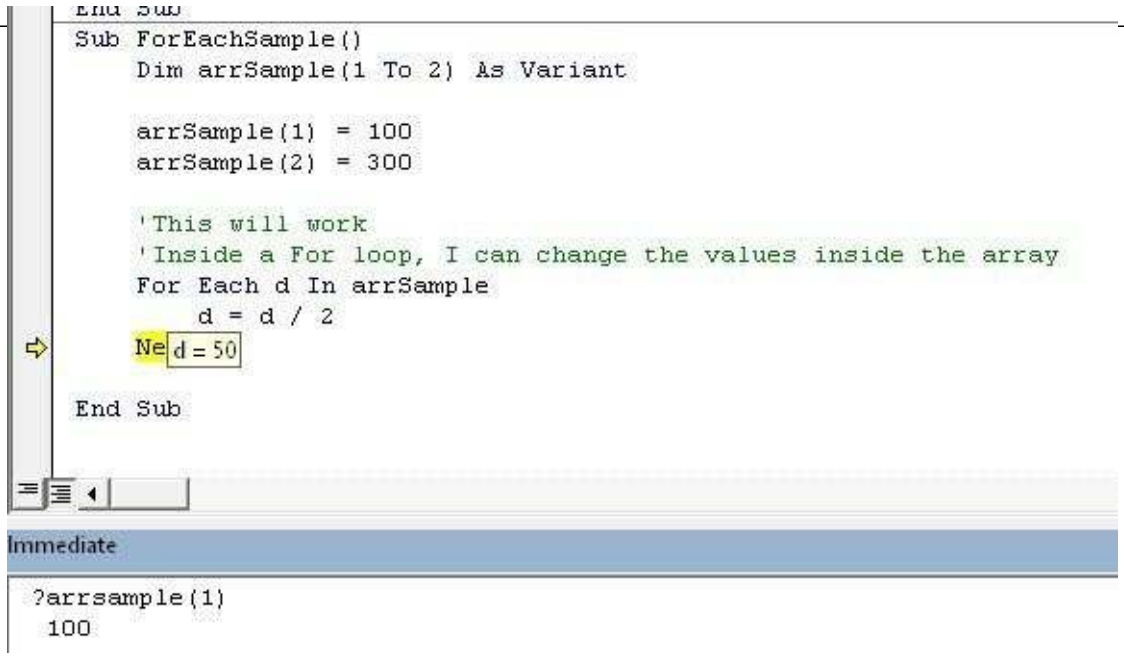
- It is read only. You cannot change the elements in the array in this loop.(You CAN chang the elements in the For Loop).

This last point is important. I want to make sure you understand it.

This loop will process and change the values that are stored in the array. In this example, I am taking the value, dividing it by 2 and putting it back into the array.

```
Sub ForSample()
    Dim arrSample(1 To 2) As Variant

    arrSample(1) = 100
    arrSample(2) = 300

    'This will work
    'Inside a For loop, I can change the values inside the array
    For i = 1 To 2
        arrSample(i) = arrSample(i) / 2
    Next i

End Sub
```

The next example WILL NOT WORK. Look at the image below to see how tricky this is. Variable d did get divided by 2 but the value in the array, arrSample(1) did not get changed. In a For Each loop, the variable (in my case, d) is a copy of the value in the array. I can do whatever I want with variable but I cannot change the value of the actual item inside the array.



In summary, if you need to change the values inside the array, use a For Loop.

24. Finish the macro by calculating the average and putting it in cell F2.

```
Sub CalcAverageDynamic()
    Dim Scores() As Variant

    'Select cell B1
    Range("B1").Select

    'Calculate array size
    ArrSize = WorksheetFunction.CountA(Range("B:B"))

    'Resize the array
    ReDim Scores(1 To ArrSize)

    'Populate array
    Scores = Range("B2:B51").Value

    'Loop through each item
    For Each c In Scores
        Total = Total + c
    Next c

    Range("F2").Value = Total / 50

End Sub
```

Yet Another Way to Loop

Ok, now that you have the basics of looping, I'm going to show you another way to use array functio
to loop. Excel has a few functions that are designed to be used with arrays. I'm going to show you tw
of them now.
UBound() - Ubound returns the largest subscript of the array.
LBound() - LBound returns the smallest subscript of the array.

Subscript refers to the slot number of the array. In essence, UBound tells you how large the array is.

You can use UBound and LBound as a way to ensure that your loop will always start at the lowest slo
end in the highest slot, and you will have the ability to edit the values in the array.

Look at this example:

```
Sub UsingBounds()
    Dim arrSample(1 To 5) As Variant

    For i = LBound(arrSample) To UBound(arrSample)
        'do stuff here
    Next i

End Sub
```

This code will work regardless of the size of the array. You will never have to change your looping code to accommodate the array.

SPLIT Function

The SPLIT function is a very useful function that splits a text string by a delimiter into a zero-based, one-dimensional string array. Sounds complicated right? Not really. You already know all this stuff.

Zero based - The array starts at 0 not 1.
One-dimensional array - An array of one dimension. Just like the student scores you have been working with.
String array - an array of string data type.

So, what the SPLIT function does is split the text you give it, at the delimiter you specify and puts each part in its own array slot.

The syntax for the SPLIT function is:

SPLIT (String, Delimiter, Limit, Compare)

String - The text to be processed.
Delimiter - The character that determines the substring.
Limit - The maximum number of times the string should be split. This is optional.
Compare - Numeric value indicating which type of compare to use. This is optional.

You will mostly use only the first two parameters.

This example will help you understand. I am just going to show you the example, you are not going t
build it here. The example is in the follow-along workbook. SPLIT is a fantastic way to parse text tha
you get from other systems or long text values in Excel.

```
Sub UsingSplit()
    Dim Fruits() As String

    strText = "Orange, Banana, Apple, Kiwi, Grape"
    Fruits = Split(strText, ",")

    For i = LBound(Fruits) To UBound(Fruits)
        MsgBox Fruits(i)
    Next i
End Sub
```

This example parses the string of fruits by each comma. The text in between each comma gets loaded into the Fruits array. The For loop displays a message box with each individual fruit.

Erasing an Array

If you want to remove all the items from an array you use the Erase command. Using the example above, to erase the Fruits array the command would be:

```
Erase Fruits
```

Using Erase on a static array will remove all the items and the array will still exist. Using Erase on a dynamic array will delete the array. You will need to use ReDim to recreate and resize the array.

Two-Dimensional Arrays

A two-dimensional array can be thought of as having rows and columns, much like a spreadsheet. To loop through a two-dimensional array, you need to use two nested loops; one to loop through the row and a second one to loop through the columns.
The loops always go through the rows. In other words, the loop will get to row 1, loop through all th columns in row 1, move to row 2, loop through all the columns in row 2, etc.

1. Open the workbook Two Dimensional Array.xlsm.

This is the data you will be working with:

	A	B	C
1	Student Name	Test Score 1	Test Score 2
2	Ji Kratz	86	48
3	Anderson Fentress	92	45
4	Terina Milholland	98	40
5	Leila Allyn	53	38
6	Leonarda Scruggs	70	55
7	Shawnta Barbeau	79	74
8	Adelaide Mccalla	82	44
9	Tambra Beaty	99	73
10	Henriette Dillard	88	72

You are going to declare a two-dimensional array, populate it and then print out the values in the immediate window.

2. Create a new macro name it TwoDim.

```
Sub TwoDim()


End Sub
```

When you declare a two-dimensional array, you need to specify how many rows and how many columns in this format (1 to 10, 1 to 2). The next image will show you how to write the declaration statement.

3. Declare a dynamic array of Variant data type called arrScores.

```
Sub TwoDim()
    Dim arrScores() As Variant


End Sub
```

You can use the shortcut range method to populate the array.

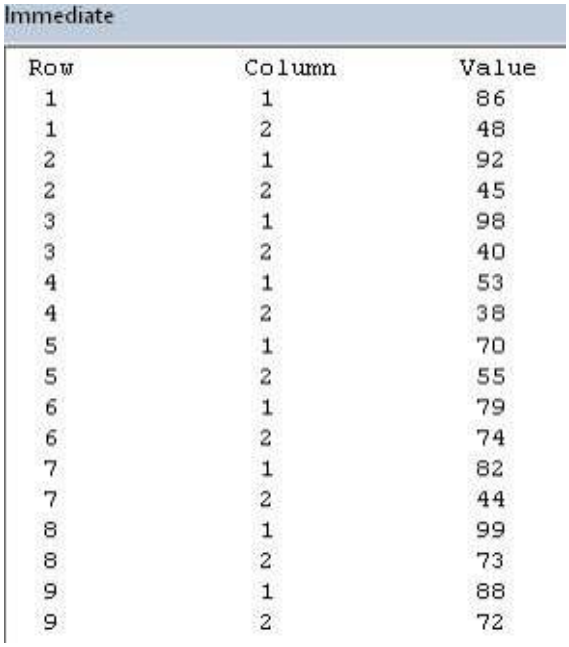
4. Populate the array with this line: arrScores = Range("B2:C10").Value.

```
Sub TwoDim()
    Dim arrScores() As Variant

    'Populate array
    arrScores() = Range("B2:C10").Value

End Sub
```

Now you are going to write two nested loops to print out the results to the Immediate window. Why the Immediate window? Because when you are developing code, you don't need to write to the worksheet. The Immediate window can serve as a scratchpad to see your results without interfering with the Excel file.

```
Sub TwoDim()
    Dim arrScores() As Variant

    'Populate array
    arrScores() = Range("B2:C10").Value

    'Print headers
    Debug.Print "Row", "Column", "Value"

    'Loop through each row
    For i = LBound(arrScores) To UBound(arrScores)
        'loop through each column
        'the 2 indicates the second dimension
        For k = LBound(arrScores, 2) To UBound(arrScores, 2)
            Debug.Print i, k, arrScores(i, k)
        Next k
    Next i

End Sub
```

5. Run the macro.
6. Display the Immediate window (if it is not visible) to see your results.


Immediate

Row	Column	Value
1	1	86
1	2	48
2	1	92
2	2	45
3	1	98
3	2	40
4	1	53
4	2	38
5	1	70
5	2	55
6	1	79
6	2	74
7	1	82
7	2	44
8	1	99
8	2	73
9	1	88
9	2	72

This ends the section on arrays. This has not been an exhaustive and complete lesson on arrays. There are many, many more things you can do with arrays but this information will give you a solid foundation to start using this feature that you can build upon on your own.

Now, we move on to Collections...

Collections

Collections are similar to arrays in that they store data of the same type. Arrays get all the glory (because arrays exist in almost all programming languages), but collections are actually easier to use than arrays.

Many of the skills you just practiced with arrays (For Each loops, For Loops, etc.) are also applicable to collections. In fact, you use them the same way. You need to use the loops to cycle through each element. One BIG difference between collections and arrays is that **you cannot change the item in a collection. Collections are read only.** If you need to change the item in the collection, then you need to use an array instead.

Ok, let's write a macro so you can see how collections are easier to work with than arrays.

1. Open the follow-along workbook Collections.xlsm.

This has a small data set of students.

	A	B
1	Student Name	Test Score
2	Ji Kratz	86
3	Anderson Fentress	92
4	Terina Milholland	98
5	Leila Allyn	53
6	Leonarda Scruggs	70
7	Shawnta Barbeau	79
8	Adelaide Mccalla	82
9	Tambra Beaty	99
10	Henriette Dillard	88
11		

2. Insert a new macro. Call it myCollection.

```
Sub myCollection()


End Sub
```

You can create the collection in one line with the Dim keyword. You can add the collection to Excel with the New keyword.

3. Create a new collection called Students.

```
Sub myCollection()
    Dim Students As New Collection

End Sub
```

Notice that unlike arrays, there is no sizing parameters needed with collections. **Notice that no data type is needed.**

Before you start writing loops to load the collection, let's work through a few simple exercises that will show you how convenient collections are. You have an empty collection. You need to add a few items to it.

You can add individual items to a collection by using the Add method. The syntax is [collection name].Add "value to add"

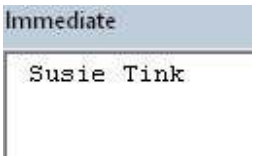
4. Add the student Susie Tink to the collection.

```
Sub myCollection()
    Dim Students As New Collection

    Students.Add "Susie Tink"

End Sub
```

5. Add a new line that displays item #1 in the Immediate window.

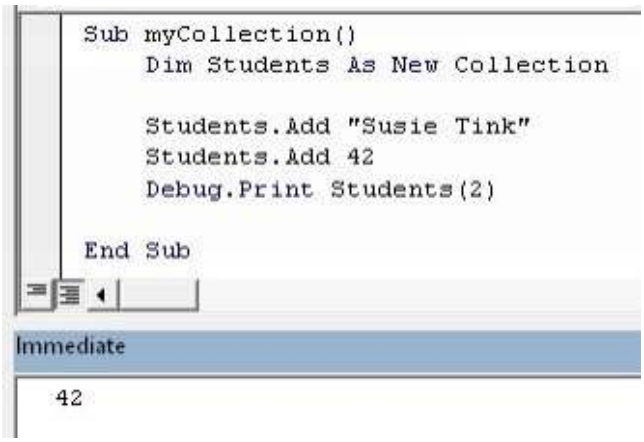
```
Sub myCollection()
    Dim Students As New Collection

    Students.Add "Susie Tink"
    Debug.Print Students(1)

End Sub
```

6. Run the macro and confirm that Susie Tink appears in the Immediate window (you might have to display the window if it is not visible).



Remember a few steps ago, I told you that you did not need to declare a data type when creating a collection? In collections, you can mix data types! Add a new item that is numeric.

7. Add this line: Students.Add 42.
8. Change the Print line to display the second item.
9. Run the macro to see the second item.



Wait. Maybe you messed up and in this case, order matters. You need to add a new student before Susie. In an array, you would be out of luck. Thankfully, you are working with a collection. In a collection, it is easy to add an items in a specific spot in the collection.

10. Add this line to add item James Polk before Susie Tink: Students.Add “James Polk”, Before:=1.
11. Change the order of the macro code to match the image below. This will let yo check that James is really the first item in the collection.

```
Sub myCollection()
    Dim Students As New Collection

    Students.Add "Susie Tink"
    Students.Add 42
    Students.Add "James Polk", Before:=1
    Debug.Print Students(1)

End Sub
```

```
Immediate
James Polk
```

Ok, we made a mistake. That 42 should not be in there. You need to get rid of it. Unfortunately, the code has moved items around and you aren't sure where the 42 is in the collection.

Similarly to what you did for arrays, you are going to write a For loop to display the item number and the item for each item in the collection. Arrays had the issue where you had to calculate the size of the array and use that in the For i = 1 to x line in the macro (where x was the size of the array). Collections are easier. The Count property returns the number of items. Students.Count is what you will use.

12. Add a For loop to display each item and its index.

If the Immediate window has previous values in there, you can highlight them and press the DELETE key to clean it up.

```
Sub myCollection()
    Dim Students As New Collection

    Students.Add "Susie Tink"
    Students.Add 42
    Students.Add "James Polk", Before:=1
    'Debug.Print Students(1)

    For i = 1 To Students.Count
        Debug.Print i, Students(i)
    Next i

End Sub
```

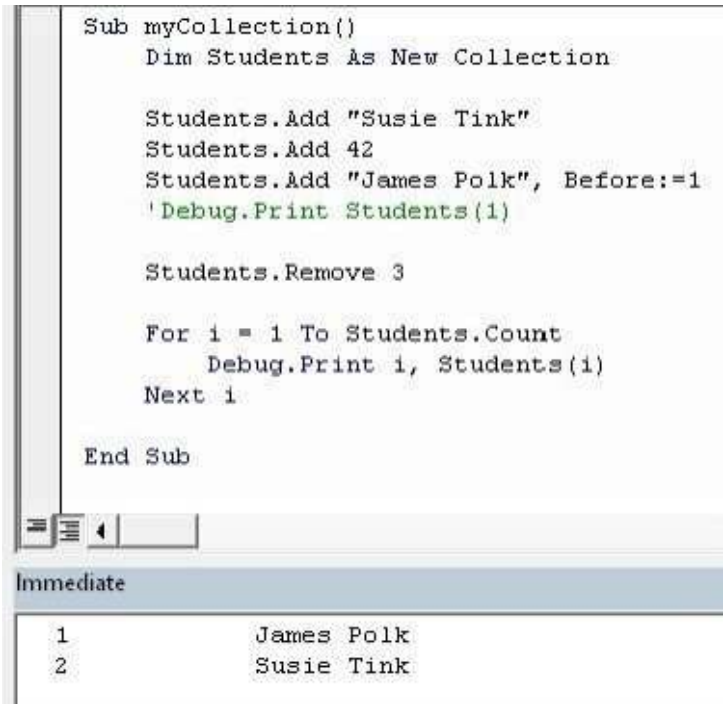
```
Immediate
 1            James Polk
 2            Susie Tink
 3             42
```

Which one of those items is not like the others? 42. In this case, 42 is not the answer. You need to remove that item. You know that it is item 3 from the code you just ran. You need to use the collection's remove method.

Students.Remove 3

13. Add the remove method to remove items #3.

Note: Put the remove method before the loop so you can see that 42 is actually removed.



In cases where you need to delete all the items from a collection, you would set the collection to Nothing, like this:

Set Students = Nothing

Using Keys in Collections

I put the heading for this section in big blue letter because (in my humble opinion) keys are a MAJO advantage of collections over arrays and I wanted you to pay attention to it.

When you add an item to a collection, you can assign it a key and then retrieve it using the key. **The one condition is that the key must be unique.**

For example, if you wanted to add a key value pair, you could use this line:

Students.Add Item:=42, Key:=”James”

Then when you want to retrieve the value, you would use:

Debug.Print Students(“James”)

Let’s work with this feature so you can see how great it is. You are going to load the data in the worksheet into a new collection.

1. Create a new macro called UsingKeys.