# COMP1022Q
## Introduction to Computing with Excel VBA

# Arrays

David Rossiter, Gibson Lam and Oz Lam

# Outcomes

- After completing this presentation, you are expected to be able to:

    1. Use arrays in VBA

    2. Obtain the lower and upper bounds of arrays

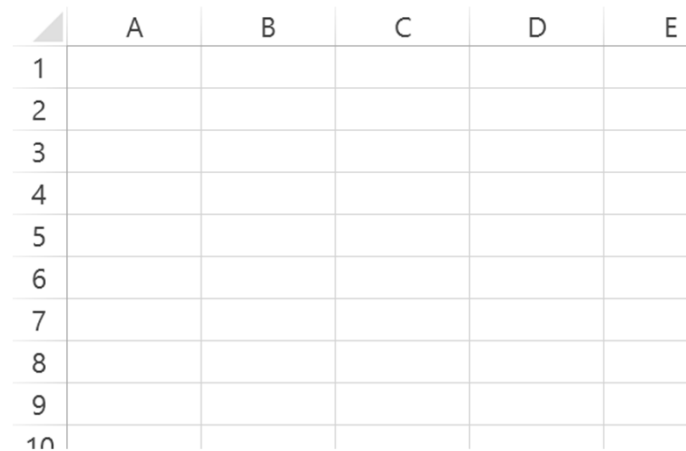    3. Write code using `With…End With`

# What is an Array?

- Perhaps the easiest way to think of an array is a row of boxes, into which you can put things

The indices
of the boxes

| | | | . . . | |
|---|---|---|---|---|
| 0 | 1 | 2 | | n |
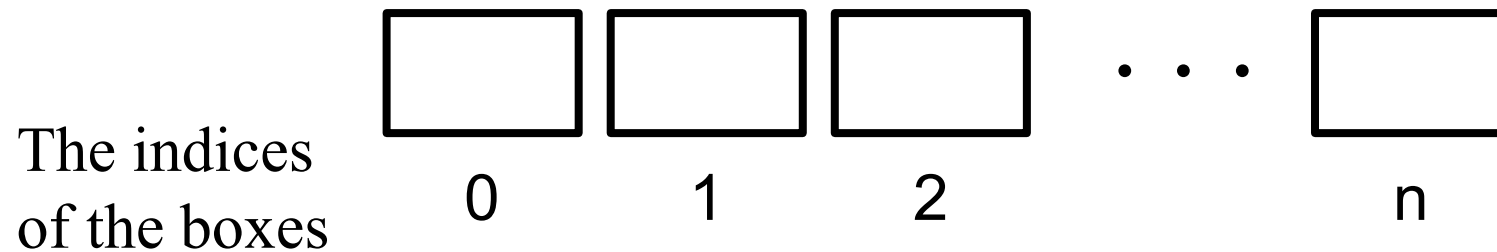
- Each of the boxes has an index so that you can get things from and put things into a box using the name of the array and the index of the box

- Instead of making many variables to store many things, you can do that using one single array

# VBA Arrays in Excel

- You can use a VBA array to store lots of things, e.g. lots of numbers

- However, VBA arrays are not used very often in Excel

- Instead of creating an array, you can simply store lots of things in cells



There are plenty of 'boxes' you can use in Excel!

# Using Range in VBA

- Another reason that VBA arrays are not commonly used in Excel is because of the Range object

- If you have some data in a Range object, you can use one line of VBA code to, for example, sort the data, find something in the data, combine several sets of data, and lots of other useful things

- If you are using arrays it may take many lines of code to do the same thing

# Why Do We Need Arrays?

- Then why do we need arrays?

  - In addition to Excel, you can program VBA in Microsoft PowerPoint and Word – but there are no cells which you can use there!

  - Also, most other computer programming languages are not 'stuck together' with worksheets like Excel VBA

  - For these kinds of situation, arrays are very useful

# Creating a VBA Array

- Let's look at how you can create an array in VBA

- For example, the following code creates an 'Integer' array with three integers:

```
Dim NumberArray(0 To 2) As Integer
```
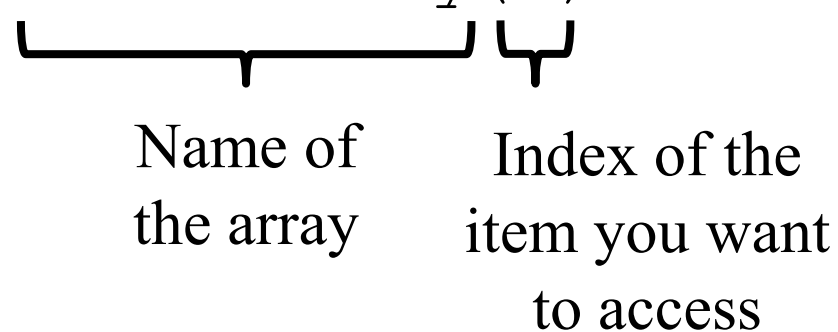
Name of the array     Indices of the array items     Data type of each item

| Index | 0 | 1 | 2 |
|-------|---|---|---|
| Value |   |   |   |

# Accessing Each Array Item

- To read or write an array item you need to use the array name together with the item index

- For example, the following code puts the number 10 into the first 'box' of the array on the previous slide:

```
NumberArray(0) = 10
```

Name of
the array

Index of the
item you want
to access

| Index | 0 | 1 | 2 |
|-------|----|---|---|
| Value | 10 |   |   |

# Example of Using an Integer Array

- Here is some code to put the array content into cells:

```
Dim NumberArray(0 To 2) As Integer
Dim Index As Integer
```

The index starts from 0

```
NumberArray(0) = 10
NumberArray(1) = 14
NumberArray(2) = 3
```

| Index | 0 | 1 | 2 |
|-------|----|----|---|
| Value | 10 | 14 | 3 |

```
For Index = 0 To 2
     Cells(4, Index + 1).Value = NumberArray(Index)
Next
```

The numbers are put in cells A4:C4

| | A | B | C |
|---|----|----|---|
| 4 | 10 | 14 | 3 |

# Array Data Type

- In the last example we made an Integer array

- You can make an array of any data type that VBA knows about

- Here are some examples:

```
Dim MyArray(0 To 2) As Long
Dim MyArray(0 To 2) As Double
Dim MyArray(0 To 2) As Worksheet
Dim MyArray(0 To 2) As Range
```

- However, one array can only contain one single type of data

# Lower and Upper Bound of an Array

- When we create an array we provide the *lower bound* and *upper bound* of the item index, i.e.

```
Dim NumberArray( 0  To  2 ) As Integer
```

The *lower bound* of an array is the smallest index of the array

The *upper bound* of an array is the largest index of the array

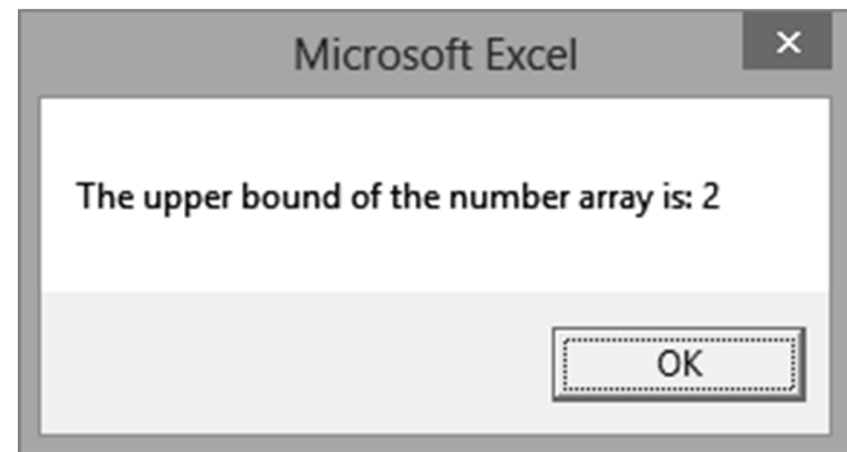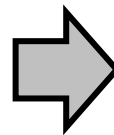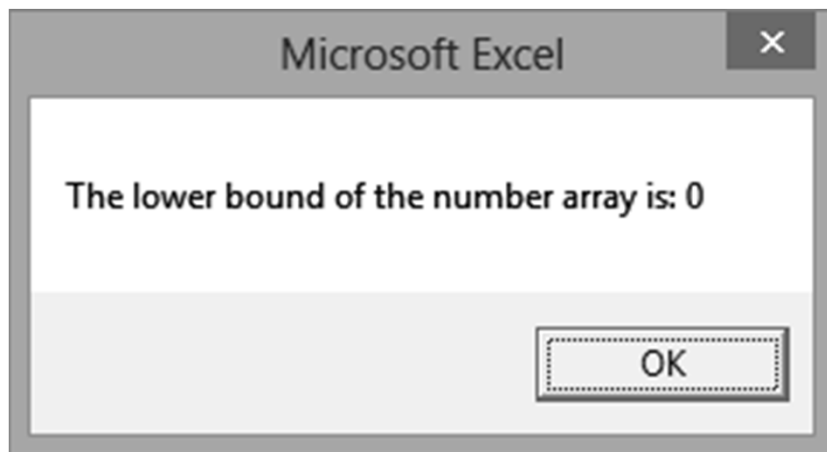| Index | 0 | 1 | 2 |
|-------|---|---|---|
| Value |   |   |   |

# Finding the Lower and Upper Bound

- You can use the following functions to get the values of the lower bound and upper bound of an array

    - `LBound()` returns the lower bound, i.e. the smallest index of the array

    - `UBound()` returns the upper bound, i.e. the largest index of the array

- The example on the next slide shows the lower and upper bound of an array using two message boxes

# Showing the Lower and Upper Bound

```
Dim NumberArray(0 To 2) As Integer

MsgBox "The lower bound of the number" & _
       " array is: " & LBound(NumberArray)
MsgBox "The upper bound of the number" & _
       " array is: " & UBound(NumberArray)
```
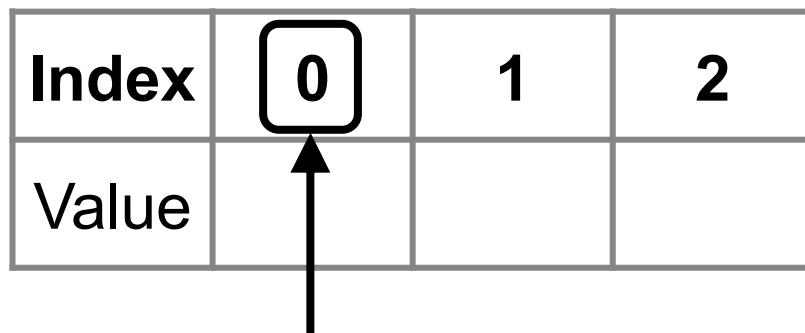
# Default Starting Array Index

- If you do not give the lower bound of an array, it will start the index from 0, for example:

```
Dim NumberArray(2) As Integer
```

| Index | 0 | 1 | 2 |
|-------|---|---|---|
| Value |   |   |   |

0 is the default starting index

This creates **three** items in the array, not two.
This is because the default starting index is zero and in this example the highest index is 2.

- Some programmers like to start the index at 0
- Some other programmers like to start the index at 1
- Both are OK

# Making a Wave

- In the next example, we will make a wave motion using some circle shapes created on a worksheet

- Before we look at the code of the example, we will first learn how to use `With…End With`
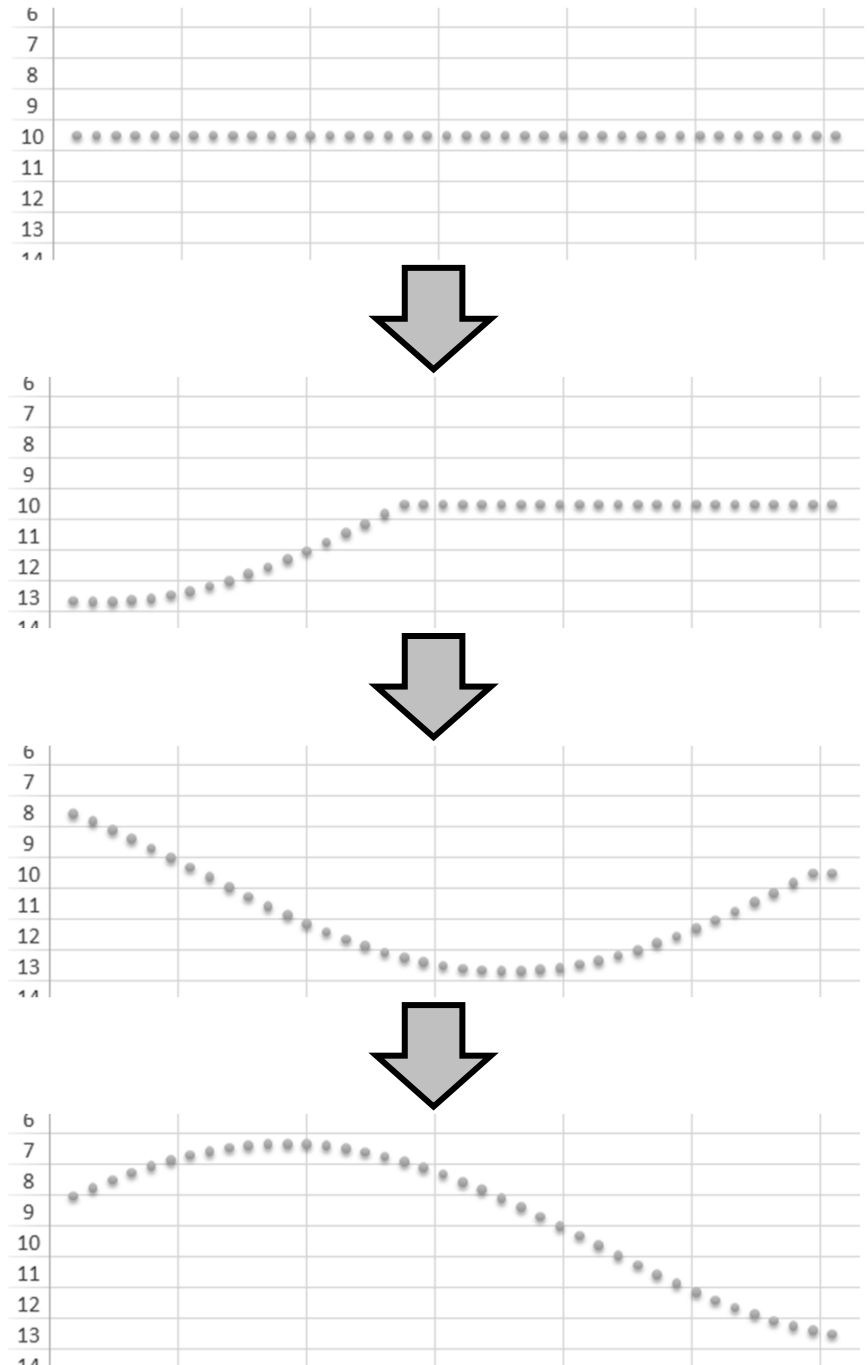
# Using With…End With  1/2

- `With…End With` is useful when you want to change many properties of the same object

- For example, the following code changes some properties of the same cell B5:

```
Range("B5").Font.Bold = True
Range("B5").Font.Size = 30
Range("B5").Color = vbRed
```

- You need to repeat `Range("B5")` three times when you change the properties of the cell

# Using With…End With  2/2

- Instead of writing the same object in the code multiple times, you can write the object only once using `With...End With`

- Here is
  an example:

```
With Range("B5")
     .Font.Bold = True
     .Font.Size = 30
     .Color = vbRed
End With
```

- The above code is equivalent to the code shown on the previous slide

# The Circles in the Example

- The 'wave' in the example is composed of 40 circles

- We create the circles using Excel shapes

- The Excel shapes are stored in an array, which has been created using this line of code:
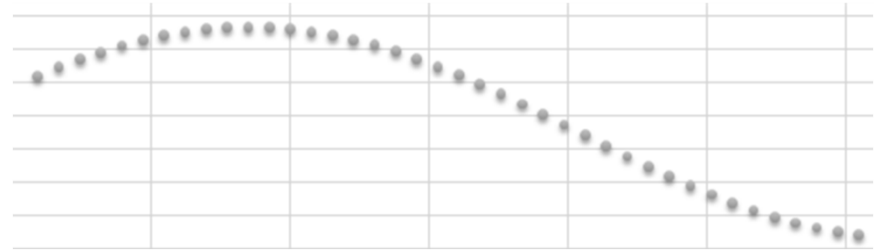
```
Dim Circles(1 To 40) As Shape
```

There are 40 items in the array

Each item is an Excel shape

# Creating the Circles

- The example uses the following loop to create the circles:

```
For Index = LBound(Circles) To _
            UBound(Circles)
    Set Circles(Index) = _
            ActiveSheet.Shapes.AddShape( _
                msoShapeOval, X, Y, _
                Size, Size)
```

The loop runs through the entire array

*...Setting the properties of the newly created circle...*

Each circle has a diameter of `Size`

```
    X = X + 10
Next Index
```

Move the next circle a little bit to the right

# Setting the Properties of a Circle

- The loop changes the name and colour of the newly created circle using this code:

```
With Circles(Index)
    .Name = "Circle" & Index
    .Line.Visible = False
    .Fill.ForeColor.RGB = _
        RGB(50, 170, 250)
End With
```

The name of the shape is changed to start with the text 'Circle'; it will be used later in another part of the code

- `With…End With` is used so that there is no need to write `Circles(Index)` three times

# Making a Wave Motion

- Initially the circles form a line in the worksheet

- To create a wave motion, the code moves the first circle based on the value of a sine wave

- Each of the rest of the circles follows the movement of the circle immediately before itself on the left

# Moving the First Circle

- The following code moves the first circle based on a sine wave:

This returns the first circle from the `Circles` array

```
Circles(LBound(Circles)).Top = _
        Sin(Radian) * 50 + 200
```

- The `Top` property adjusts the vertical position of an Excel shape

- Therefore, the first circle moves up and down based on a sine wave, which is returned by the `Sin` function and a `Radian` value

# The Loop

- The first circle is continuously moved inside a loop, which keeps on increasing `Radian`, as follows:

```
Radian = 0
Do

    . . .move the rest of the circles . . .


    Circles(LBound(Circles)).Top = _
                    Sin(Radian) * 50 + 200


    Radian = Radian + 0.1



    DoEvents
Loop Until Stopped
```

The loop runs until `Stopped` is set to `True` by pressing 'Stop'

# Moving the Rest of the Circles

- The rest of the circles can be moved by going through the array from the last circle to the second circle

```
For Index = UBound(Circles) To _
            LBound(Circles) + 1 Step -1
    Circles(Index).Top = _
            Circles(Index - 1).Top
Next Index
```

- Each circle is moved to the position of the circle on the left, i.e. `Circles(Index - 1)`

# Deleting the Circles



- The wave can be stopped and restarted

- Before restarting, a for each loop deletes the circles by going through all shapes in the worksheet and deleting the ones with a name started with the text 'Circle', like this:

```
For Each Shape In ActiveSheet.Shapes
    If Left(Shape.Name, 6) = "Circle" Then
        Shape.Delete
    End If
Next Shape
```

This line of code removes the shape from the worksheet