MONASH University

# FIT1013 Digital Futures: IT for Business
# Week 7: Repetition Structures

## On completion of your study this week, you should aim to:

- Implement repetition structures in VBA

GROUP OF EIGHT AUSTRALIA

# Repetition Structures

Programmers use the **repetition structure**, also called **looping** or **iteration**, to direct the computer to repeat one or more instructions either a precise number of times or until some condition is met

| Example 1 | Example 2 |
|---|---|
| Repeat two times:<br>apply shampoo to wet hair<br>lather<br>rinse | Pour 8 ounces of milk into a glass<br>Pour 2 teaspoons of chocolate syrup into the glass<br>Repeat the following until milk and syrup are mixed thoroughly:<br>stir |

# VBA Forms of the Repetition Structure

- Do While
- Do Until
- For Next
  - For…Next
  - For Each…Next

- The With statement

# Repetition: Do Loops

- For repeating an action many times
- **Do While Loop, Do Until Loop**
- 2 versions of each – perform a test at start or a test at end (pretest, posttest)
- **Do While**: Included code executed while condition is true
- **Do Until**: Included code executed while condition is false
- Make sure the condition is such that it will fail eventually – I.e. avoid infinite loops.

# Syntax of the Do While Loops

Start with **Do While**

If condition is true, the loop instructions are performed (pretest loop)

**Do While** *condition*

[*loop instructions*]

[Exit Do]

[*loop instructions*]

**Loop**

End with **Loop**

**Exit Do** forces an exit from loop

Start with **Do**

**Do**

[*loop instructions*]

[Exit Do]

[*loop instructions*]

**Loop While** *condition*

End with **Loop While**

If condition is true, the loop instructions are repeated (posttest loop)

condition must evaluate to true or false
condition can contain variables, constants, properties, functions, mathematical operators, relational operators, and logical operators

# Do While loop (pretest)

Syntax:

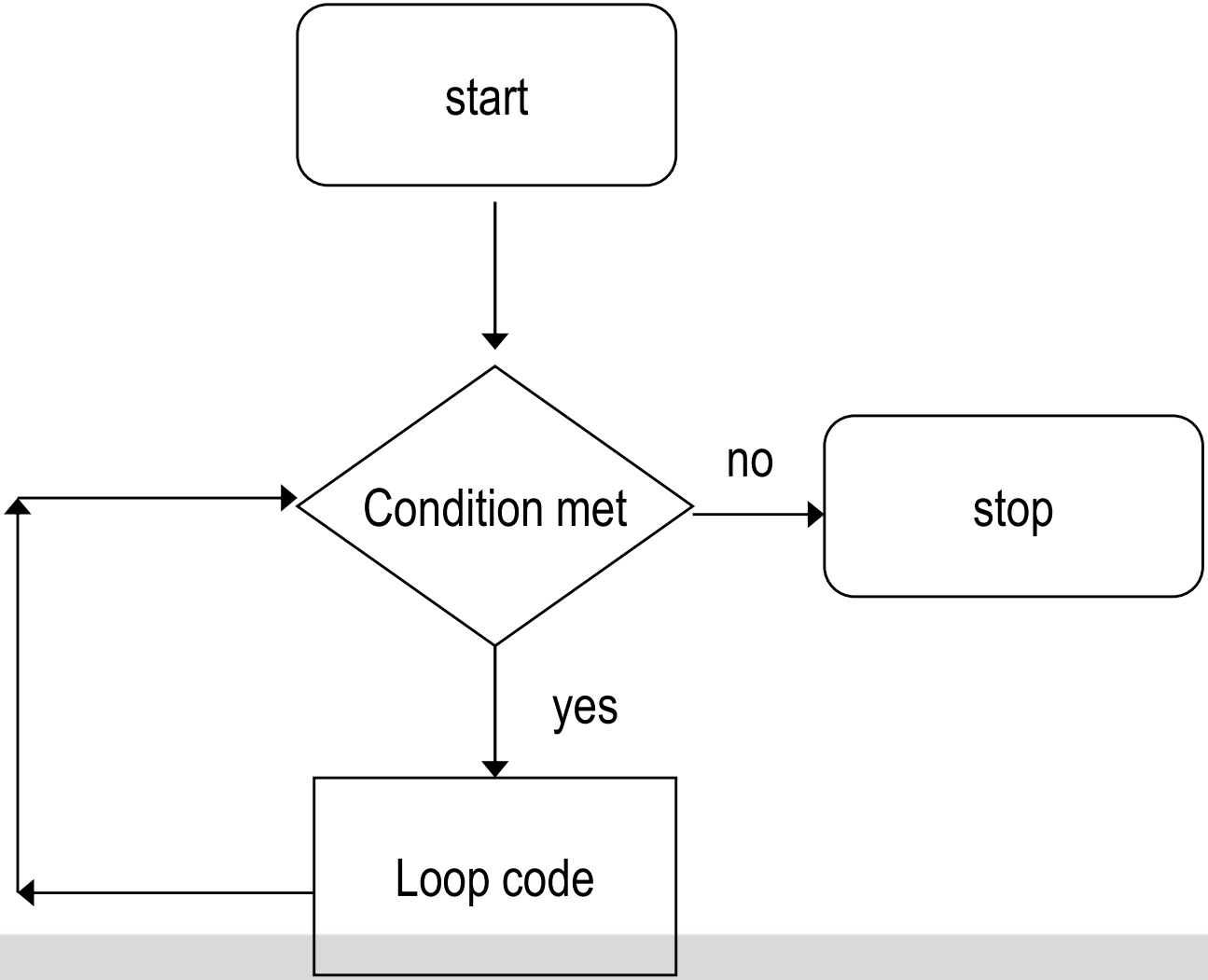
Do While <*condition*>

    VBA code

    [Exit Do]

    VBA code

Loop



# Do While loop E.g.1

Not logical operator

Condition uses the IsEmpty() function to check if the active cell is empty

```
Do While Not IsEmpty(ActiveCell)
    'if the active cell is not empty, put 0 in it, otherwise stop
   ActiveCell.Value = 0
    'then move down one cell
   ActiveCell.Offset(1, 0).Select
Loop
```

Loop section

# Do While loop (posttest)

Syntax:

Do

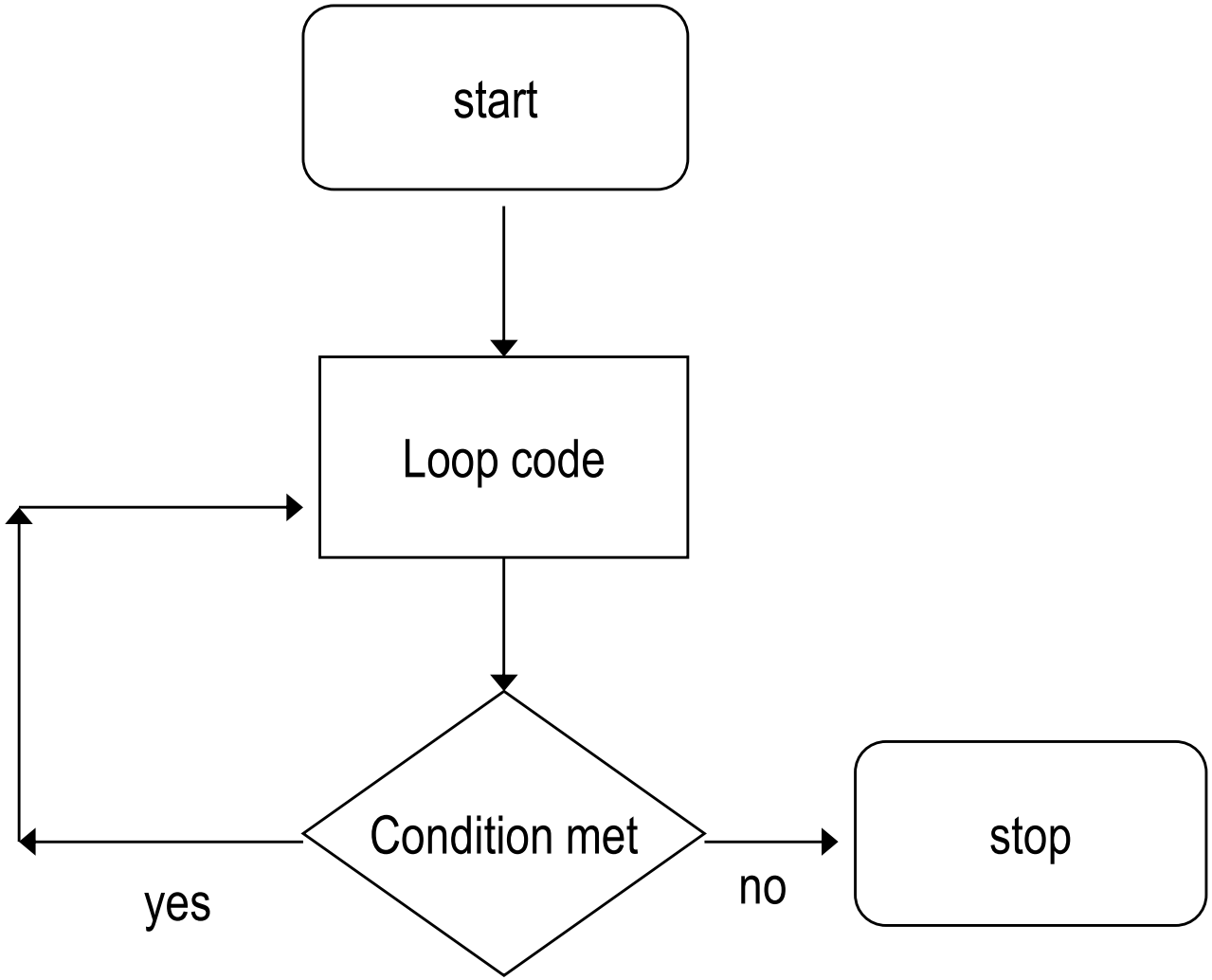
VBA code

[Exit Do]

VBA code

Loop While

<*condition*>



# Do While loop E.g. 2

```
Do
  'Put 0 in the active cell
   ActiveCell.Value = 0
   'then move down one cell
  ActiveCell.Offset(1, 0).Select
   'If the active cell is empty, stop, otherwise continue looping
Loop While Not IsEmpty(ActiveCell)
```

Loop section

condition

Not logical operator

# Syntax of the Do Until Loops

Start with **Do Until**

If condition is false, the loop instructions are performed (pretest loop)

**Do Until** *condition*

[*loop instructions*]

[Exit Do]

[*loop instructions*]

**Loop**

End with **Loop**

**Exit Do** forces an exit from loop

Start with **Do**

**Do**

[*loop instructions*]

[Exit Do]

[*loop instructions*]

**Loop Until** *condition*

End with **Loop Until**

If condition is false, the loop instructions are repeated (posttest loop)

- *condition* must evaluate to true or false
- *condition* can contain variables, constants, properties, functions, mathematical operators, relational operators, and logical operators

# Do Until loop (pretest)

Syntax:

Do Until

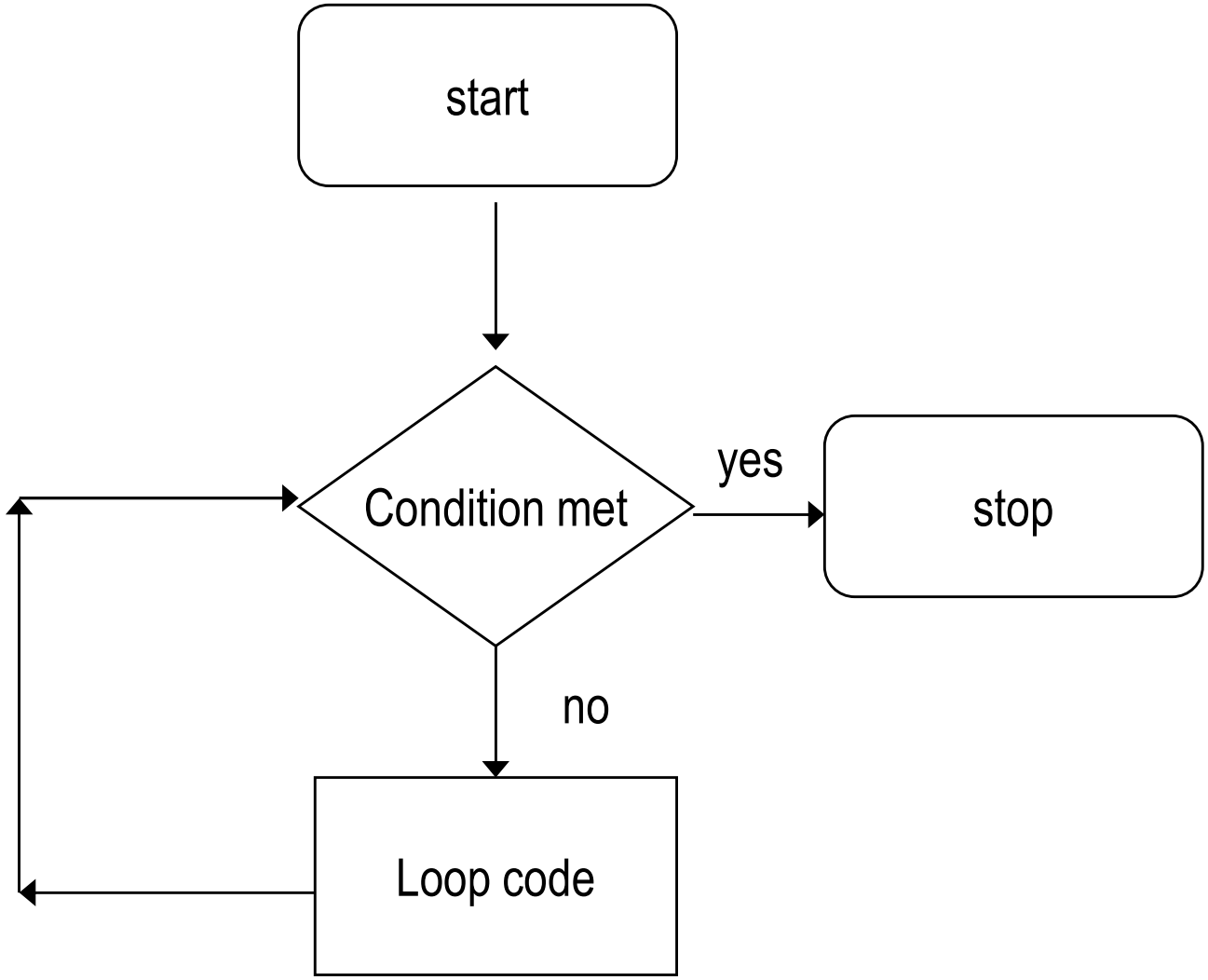
    <*condition*>

    VBA code

    [Exit Do]

    VBA code

Loop



# Example 1 of Do Until (pretest loop)

```
Do Until IsEmpty(ActiveCell)
   ActiveCell.Value = 0
   ActiveCell.Offset(1, 0).Select
Loop
```

condition

Loop section

Loop repeats until the condition is true

# Do Until loop (posttest)

Syntax:

Do

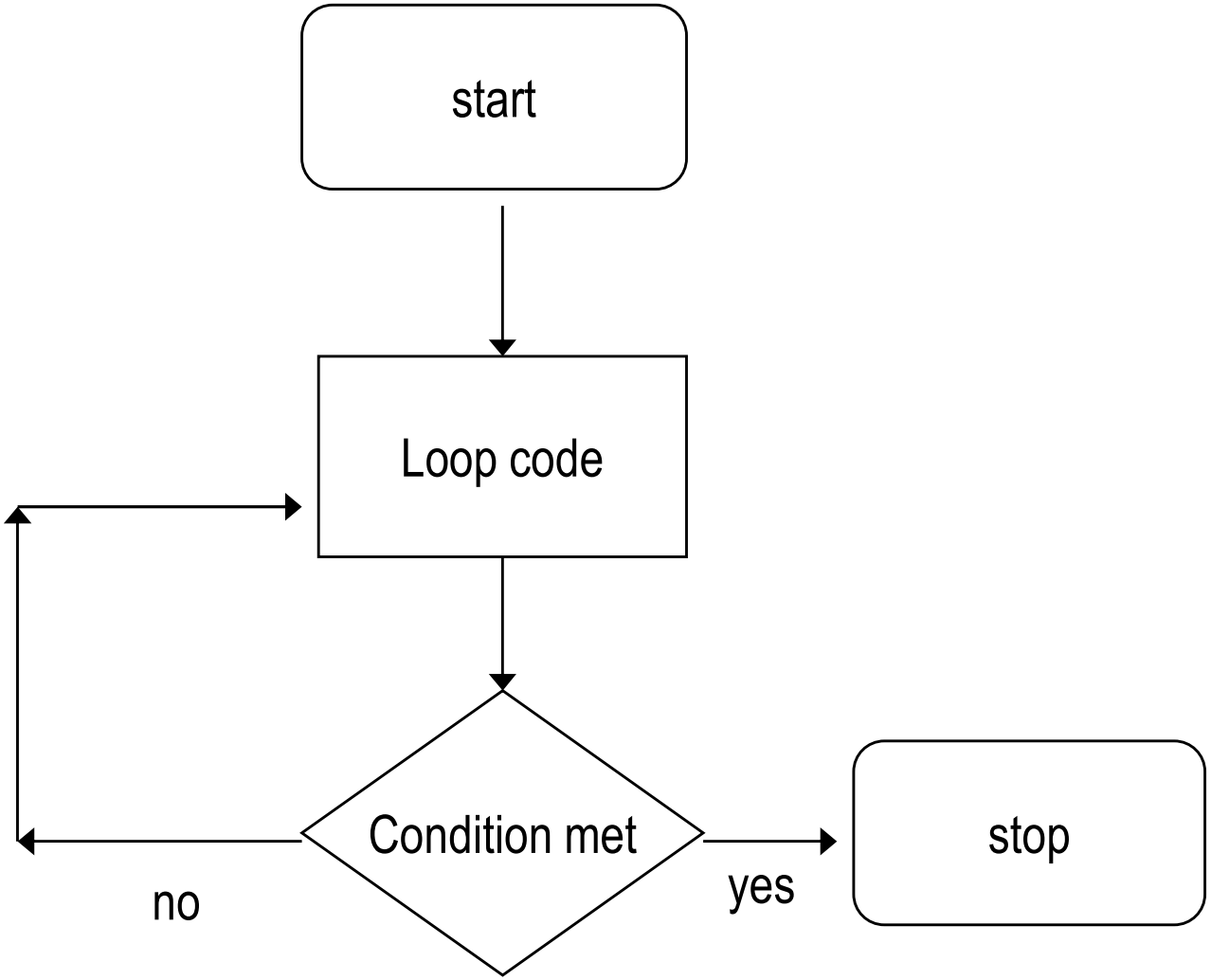
    VBA code

    [Exit Do]

    VBA code

Loop Until

    <*condition*>



# Example 2 of Do Until (posttest loop)

```
Do
    ActiveCell.Value = 0
    ActiveCell.Offset(1, 0).Select
Loop Until IsEmpty(ActiveCell)
```

Loop section

condition

Loop repeats until the condition
is true

# Summary: Do While and Do Until Loops

- In the Do While loop, the instructions are processed only when the condition evaluates to true; the loop stops when the condition evaluates to false
- The condition can be evaluated at the start or the end of the loop

- In the Do Until loop, the instructions are processed only when the condition evaluates to false; the loop stops when the condition evaluates to true
- The condition can be evaluated at the start or the end of the loop

**Evaluating the condition:**

If the condition is evaluated at the start of the loop this is called a **pretest** loop

If the condition is evaluated at the end of the loop this is called a **posttest** loop

# The For…Next Statement

- You can use the VBA **For…Next** statement to include a repetition structure in a procedure
- The **For…Next** statement begins with the **For** clause and ends with the **Next** clause
- You can use the **Exit For** statement to exit the **For…Next** loop prematurely
- You can nest **For…Next** statements, which means that you can place one **For…Next** statement within another **For…Next** statement
- In the syntax, ***counter*** is the name of the numeric variable that will be used to keep track of the number of times the loop instructions are processed

Syntax:

**For *counter* = *startvalue* To *endvalue* [Step *stepvalue*]**
**[instructions you want repeated]**
**[Exit For]**
**[instructions you want repeated]**
**Next *counter***

# Example 1 of For…Next

intCount – the counter

```
Dim intCount As Integer
Dim strCity As string
For intCount = 1 To 3 Step 1
    strCity = InputBox(Prompt:="Enter the city", Title:="City")
    MsgBox Prompt:=strCity & " is city number " & intCount, _
     Buttons:=vbOKOnly + vbInformation, Title:="City Number"
Next intCount
```

Task to repeat

ForNextEgs.xlsm (ForEg1)

# Example 2 of For…Next

```
Dim intCount As Integer
Dim wksX As Worksheet
For intCount = 1 To ActiveWorkbook.Worksheets.Count
    Set wksX = ActiveWorkbook.Worksheets(intCount)
    wksX.PrintPreview
Next intCount
```

Number of sheets in the active workbook

For…Next loop Repeats instructions for all objects in a collection – i.e. the Worksheets collection for the Activeworkbook

ForNextEgs.xlsm

(ForEg2)

# Example 3 of For…Next

```
Dim intCount As Integer
Dim wksX As Worksheet
For intCount = 1 To ActiveWorkbook.Worksheets.Count
    Set wksX = ActiveWorkbook.Worksheets(intCount)
    If UCase(wksX.Name) = "SHEET2" Then
        wksX.PrintPreview
        Exit For
    End If
Next intCount
```

The For Next loop is exited prematurely if the name of the sheet is Sheet2

# The For Each…Next Statement

- You can also use the VBA **For Each…Next** statement to repeat a group of instructions for each **object in a collection**
- In the syntax, ***element*** is the name of the object variable used to refer to each object in the collection, and ***group*** is the name of the collection in which the object is contained
- The **For Each** clause first verifies that the ***group*** contains at least one object

# Example 4: For Each

For Each loop Repeats instructions for all objects in a collection – i.e. the Worksheets collection for the Activeworkbook

```
Dim wksX As Worksheet
For Each wksX In ActiveWorkbook.Worksheets
    wksX.PrintPreview
Next wksX
```

The **For Each** clause:

- Checks to see the group contains at least one object
- If none, loop instructions are skipped
- If at least one object is in the group:

1. The address of the object is assigned to the object variable and the loop instructions are processed
2. The Next clause checks to see if there is another object in the group; if so 1. is repeated

- 2. is repeated until all objects are processed
- The loop can be exited prematurely using **Exit For**

# Comparison between For…Next and For Each

For…Next loop

```
Dim intCount As Integer
Dim wksX As Worksheet
For intCount = 1 To _
    ActiveWorkbook.Worksheets.Coun
    t
    Set wksX =
    ActiveWorkbook.Worksheets(intCo
    unt)
    wksX.PrintPreview
Next intCount
```

For Each loop
Same task

```
Dim wksX As Worksheet
For Each wksX In
    ActiveWorkbook.Worksheets
    wksX.PrintPreview
Next wksX
```

# Example 5: For Each

```
Dim wksX As Worksheet
For Each wksX In ActiveWorkbook.Worksheets
    If UCase(wksX.Name) = "SHEET2" Then
        wksX.PrintPreview
        Exit For
    End If
Next wksX
```

For Each loop including Exit For statement

# Comparison between For…Next and For Each

For…N loop

```
Dim intCount As Integer
Dim wksX As Worksheet
For intCount = 1 To
    ActiveWorkbook.Worksheets.Count
  Set wksX =
    ActiveWorkbook.Worksheets(intCount)
    If UCase(wksX.Name) = "SHEET2" Then
            wksX.PrintPreview
            Exit For
      End If
Next intCount
```

For Each loop more efficient:

- Less variables
- Set statement not necessary

For Each loop Same task

```
Dim wksX As Worksheet
For Each wksX In ActiveWorkbook.Worksheets
  If UCase(wksX.Name) = "SHEET2" Then
      wksX.PrintPreview
      Exit For
  End If
Next wksX
```

# Using For Each…. To access all cells in a range e.g.1

Declare 2 Range variables, one points at the range of interest, the other is used to access all cells in the range

```
Public Sub ForEachCell()
Dim rngCell As Range
Dim rngNumbers As Range
Set rngNumbers = Application.ActiveWorkbook.Worksheets("Sheet1")._
      Range("Number_Area")
For Each rngCell In rngNumbers
   rngCell.Value = 1
Next rngCell
End Sub
```

Looks at every cell in a range

ForNextEgs.xlsm (ForEachCell())

# Using For Each…. To access all cells in a range e.g.2

```
Public Sub ForEachCell_2()
Dim rngCell As Range
Dim rngNumbers As Range
Set rngNumbers = Application.ActiveWorkbook.Worksheets("Sheet1")._
    Range("Number_Area")
For Each rngCell In rngNumbers
   If rngCell.Value = 1 Then
   MsgBox "address = " & rngCell.Address
   End If
Next rngCell
End Sub
```

Looks at every cell in a range. Provides the address if the entry = 1

ForNextEgs_2.xlsm (ForEachCell2())

# The With Statement

The **With** statement provides a convenient way of accessing the properties and methods of a single object

Syntax:

Start with **With**

**With** *object*

*[statements]*

**End With**

*object* is the name of the object whose properties or methods you want to access

finish with **End With**

# Example 6: With Statement

Accessing the properties and methods of wksJuly (an Excel worksheet object)

worksheet object variable

```
Dim wksJuly As Worksheet
Set wksJuly = Application.Workbooks(1).Worksheets(1)
With wksJuly
    .Range("B1").Value = "Bonus"
    .Range("B2:B29").Formula = "=A2 * .1"
    .Name = "July Bonus"
    MsgBox prompt:="The sheet's name is " & .Name, _
        Buttons:=vbOKOnly + vbInformation, Title:="Name"
    .PrintPreview
End With
```

ForNextEgs.xlsm

Inserts "Bonus" into cell B1 of wksJuly

Inserts formula into cells B1:B29 of wksJuly

Note properties and methods are prefaced with ".", but the name of the object variable is not needed

Changes the name of wksJuly to "July Bonus"

Applies PrintPreview method to wksJuly

Provides a message with the name of wksJuly

# The MsgBox Function

The syntax of the MsgBox  function:

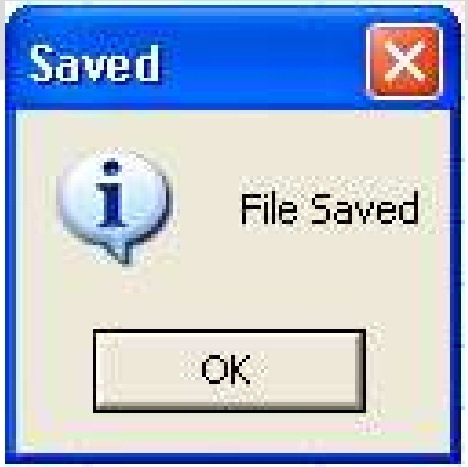
MsgBox (*Prompt, [Buttons], [Title]*)

***prompt*** is the message in the dialog box

***Buttons*** is the type of button that appears on the message box

***title*** is the text in the title bar

# The MsgBox Function




E.g. MsgBox function:

***prompt*** is the message in the dialog box

```
intButton = MsgBox (Prompt:= “File Saved”,_
Buttons:=vbOKOnly+vbInformation, Title:=“Saved”)
```

***Buttons*** determines the type/s of button/s, appearance of icon and default button that appears on the message box

***title*** is the text in the title bar

# Syntax and Examples of the MsgBox Statement and the MsgBox Function

*MsgBox statement*

**MsgBox Prompt:=***prompt*[, **Buttons:=***buttons*[, **Title:=***title*]

```
MsgBox Prompt:="File saved.", _
       Buttons:=vbOKOnly + vbInformation, Title:="Saved"
```

notice the parentheses

*MsgBox function*

**MsgBox(Prompt:=***prompt*[, **Buttons:=***buttons*[, **Title:=***title*])

```
intButton = MsgBox(Prompt:="Do you want to continue?", _
    Buttons:=vbYesNo + vbExclamation + vbDefaultButton1, _
    Title:="Continue")
```

# Valid Settings for the buttons Argument

Group 1: which buttons are displayed

Group 2: type of message icon

Group 3: which button is default

| Settings for the MsgBox's *buttons* argument | | | |
|---|---|---|---|
| | Constant | Value | Description |
| Group 1 | vbOKOnly | 0 | Display OK button only |
| | vbOKCancel | 1 | Display OK and Cancel buttons |
| | vbAbortRetryIgnore | 2 | Display Abort, Retry, and Ignore buttons |
| | vbYesNoCancel | 3 | Display Yes, No, and Cancel buttons |
| | vbYesNo | 4 | Display Yes and No buttons |
| | vbRetryCancel | 5 | Display Retry and Cancel buttons |
| Group 2 | vbCritical | 16 | Display Critical Message icon |
| | vbQuestion | 32 | Display Warning Query icon |
| | vbExclamation | 48 | Display Warning Message icon |
| | vbInformation | 64 | Display Information Message icon |
| Group 3 | vbDefaultButton1 | 0 | First button is default |
| | vbDefaultButton2 | 256 | Second button is default |
| | vbDefaultButton3 | 512 | Third button is default |
| | vbDefaultButton4 | 768 | Fourth button is default |

# Message Box *Button* arguments

Example Button arguments

**VALUES OF THE BUTTON PARAMETER**

| Button | Description | Example |
|---|---|---|
| vbOKOnly | OK button only |   |
| vbOKCancel | OK and Cancel buttons |   |
| vbCritical | Critical message |   |
| vbQuestion | Warning query |   |
| vbExclamation | Warning message |   |
| vbInformation | Information message |   |

# MsgBox Function's Buttons

| Values returned by the MsgBox function | | |
|---|---|---|
| Button | Constant | Numeric value |
| OK | vbOK | 1 |
| Cancel | vbCancel | 2 |
| Abort | vbAbort | 3 |
| Retry | vbRetry | 4 |
| Ignore | vbIgnore | 5 |
| Yes | vbYes | 6 |
| No | vbNo | 7 |

Values returned by the MsgBox function

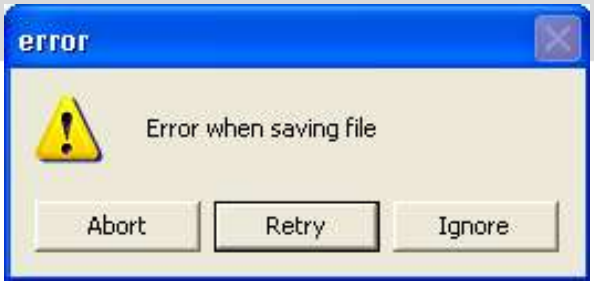
# Example 1


```
Dim intResponse As Integer
intResponse = MsgBox(Prompt:="Do you Want to continue", _
Buttons:=vbYesNo + vbExclamation + vbDefaultButton1, _ Title:="Continue")
If intResponse = vbYes Then
   [instructions to process when Yes button is selected]
Else
   [instructions to process when No button is selected]
End If
```

MsgBoxEgs.xls

- If the user selects the Yes button, the MsgBox function returns the integer 6, represented by the intrinsic constant vbYes

# Example 2




```
Dim intButton As Integer
intButton = MsgBox(prompt:="Error when saving file", Buttons:=vbAbortRetryIgnore +
    vbExclamation + vbDefaultButton2, Title:="error")
Select Case intButton
Case vbAbort
   [instructions to process when vbAbort button is selected]
Case vbRetry
   [instructions to process when vbRetry button is selected]
Case vbIgnore
   [instructions to process when vbIgnore button is selected]
End Select
```

e.g.If the user selects the Retry button, the MsgBox function returns the integer 4, represented by the intrinsic constant vbRetry

MsgBoxEgs.xls

# Summary

To display VBA's predefined message box, and then return a value that indicates which button was selected in the message box:

Use the MsgBox function:

MsgBox (*Prompt, [Buttons], [Title])*