FIT1013 - Week 7 Resources

Date Variables and Repetition Structures

Week 7 Resources

Reference:

- https://msdn.microsoft.com/

1. Objectives

- Use Date and related variables
- Use VBA's date and time functions
- Implement repetition structures in VBA

2. Concept Lesson: Date Variables

The date data type

- Internally stored as IEEE 64-bit (8-byte) floating-point numbers that represent dates ranging from 1 January of the year 0001 through to 31 December 9999 and times from 12:00:00 AM (midnight) through 11:59:59:9999999 PM.
- Use the Date data type to contain date values, time values or date and time values
- The default value of Date is 0:00:00 (midnight) on 1 January, 0001
- A fractional number with no integer part represents a dateless time
- Because the integer portion of a date represents number of days, you can add and subtract days from one date to get another date.
- For more see:
  - https://msdn.microsoft.com/en-us/library/3eaydw6e.aspx

Examples

Internal storage	Represents
567.0	20th July 1901
1299.0	22nd July 1903
0.3	7.12am
0.8	7.12pm
.5692	1.39.39pm
6788.673	1st August 1918, 4.09.07pm

Reserving date variables

- Recall to reserve a procedure level variable:

  Dim VariableName As DataType

Name of variable

Type of data the variable can store

- To reserve a procedure level Date variable:

  Dim VariableName As Date

e.g.

```
Dim dtmStart As Date
Dim dtmBirth As Date
```

Examples of Dim Statements that Reserve Date Variables

- Dim dtmPay as Date
- Dim dtmEmploy as Date
- Dim dtmStart as Date
- Dim dtmEnd as Date
- Dim dtmBirth as Date

Reserving a Procedure-level Date Variable

- When creating a Date variable, datatype is always the keyword **Date**
- The **dtm** ID indicates that the variable is a date variable, which can store date and time information
- Date variables are automatically initialized to the number 0
- After using the Dim statement to both reserve and initialize a Date variable, you can use an assignment statement to assign a different value to the variable

Assigning a value to a date variable

Recall the assignment statement that assigns a value to a variable:

Variablename* = *value

Examples for date variables:

```
dtmBirth = #June 10, 1981#
dtmFinish = #6:48:07 PM#
```

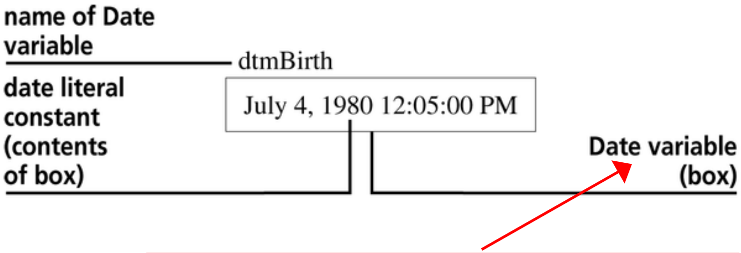
Date literal constants

- A date literal constant is simply a valid date enclosed in # symbols.

  #3:40:03 PM#
  #March 11, 1982#
  #11:05:00 AM#
  #12/31/2002#
- Date literal constants also can include both a date and a time

Using an Assignment Statement to Assign a Value to a Date Variable

- Illustration of date literal constant stored in a date variable



The date variable 'points to' the address of a memory cell which stores the value of the date variable

3. Using VBA’s Date, Time, and Now Functions

In addition to assigning date literal constants to Date variables, you also can assign the value returned by VBA’s Date, Time, and Now functions:

- VBA’s **Date** function returns the system date, which is the date maintained by your computer’s internal clock
- VBA’s **Time** function returns the system time, which is the time maintained by your computer’s internal clock
- VBA’s **Now** function returns both the system date and time

The AssignDisplayDate Procedure

```
Public Sub AssignDisplayDate()
'declare date variables
Dim dtmCurDate As Date
Dim dtmCurTime As Date
Dim dtmCurDateTime As Date
'assign values to date variables
dtmCurDate = Date
dtmCurTime = Time
dtmCurDateTime = Now
'display contents of date variables
MsgBox Prompt:=dtmCurDate & vbNewLine _
  & dtmCurTime & vbNewLine & dtmCurDateTime
End Sub
```

reserves three Date variables named dtmDurDate, dtmCurTime, and dtmCurDateTime

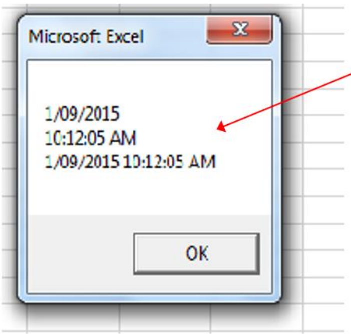
Assign values to the date variables using the **Date**, **Time** and **Now** functions

Use the underscore to indicate the code continues onto the next line

Display the values of the date variables using the MsgBox function

vbNewLine - Visual Basic Constant

Message Box Displayed by the AssignDisplayDate Procedure AssignDisplayDate.xls



Microsoft Excel

1/09/2015
10:12:05 AM
1/09/2015 10:12:05 AM

Each date is displayed on a separate line

Using the Format Function

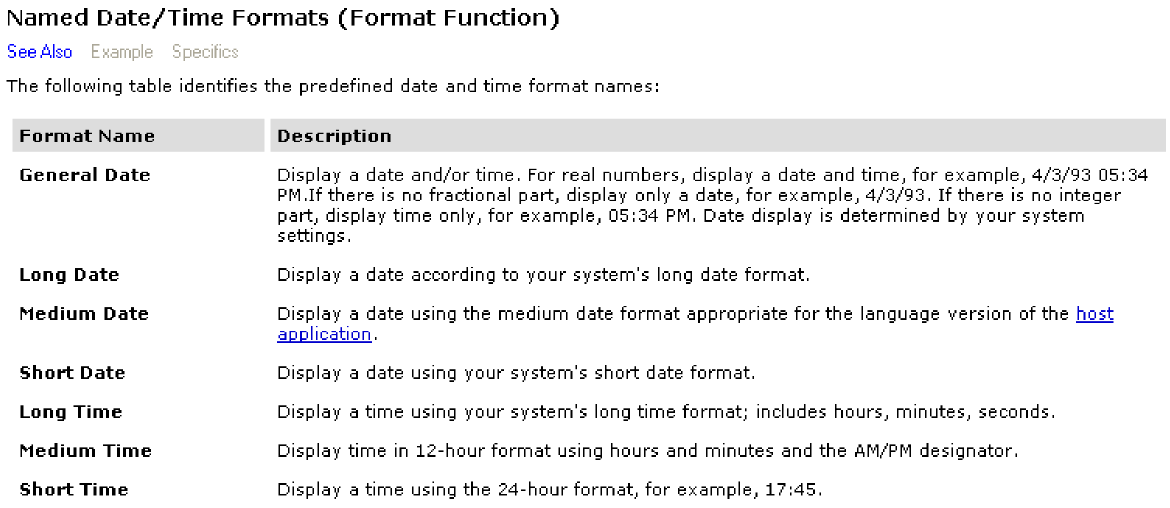
- Use the VBA **Format** function to control the appearance of dates and times
- The syntax of the Format function is:

  Format(Expression:=expression, Format:=format)
- In the syntax, **expression** specifies the number, date, time, or string whose appearance you want to format, and **format** is the name of a predefined VBA format

- E.g.

  Format(Expression:=#1/03/2004#, Format:="short date")

Help Screen Showing the VBA Predefined Date/Time Formats



Named Date/Time Formats (Format Function)

See Also Example Specifics

The following table identifies the predefined date and time format names:

Format Name	Description
General Date	Display a date and/or time. For real numbers, display a date and time, for example, 4/3/93 05:34 PM.If there is no fractional part, display only a date, for example, 4/3/93. If there is no integer part, display time only, for example, 05:34 PM. Date display is determined by your system settings.
Long Date	Display a date according to your system's long date format.
Medium Date	Display a date using the medium date format appropriate for the language version of the host application.
Short Date	Display a date using your system's short date format.
Long Time	Display a time using your system's long time format; includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format, for example, 17:45.

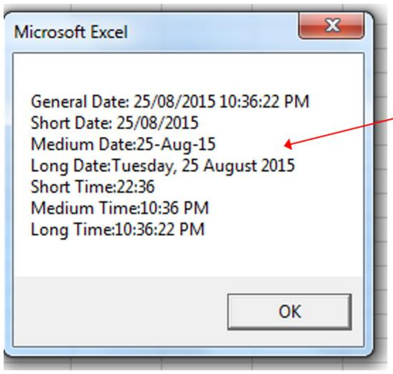
Using the Format Function

E.g. AssignDisplayDate.xls – see dateFormats() procedure

```
Public Sub dateFormats()
'declare date variables
Dim dtmEgDate As Date
'assign values to date variables
dtmEgDate = #2/18/1991 10:36:22 PM#
MsgBox Prompt:= _
   Format(Expression:=dtmEgDate, Format:="General Date") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Short Date") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Medium Date") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Long Date") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Short Time") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Medium Time") & vbNewLine _
   & Format(Expression:=dtmEgDate, Format:="Long Time")
End Sub
```

Note different date formats

Results of Date Format function



```
General Date: 25/08/2015 10:36:22 PM
Short Date: 25/08/2015
Medium Date:25-Aug-15
Long Date:Tuesday, 25 August 2015
Short Time:22:36
Medium Time:10:36 PM
Long Time:10:36:22 PM
```

Note different date formats

4. Using Dates and Times in Calculations

- You may need to include date and time calculations in your procedures
- VBA provides two functions called DateAdd and DateDiff that you can use to perform calculations involving dates and times
- The DateAdd function allows you to add a specified time interval to a date or time, and it returns the new date or time
- The DateDiff function allows you to determine the time interval that occurs between two dates

The DateAdd function

Syntax:

DateAdd(Interval:=*interval*, **Number:=***number*, **Date:=***date*)

Interval specifies the time units: e.g. hours, minutes, years etc..

Number specifies how many time units to add on to the date. Can be positive or negative

Date argument – can be any format

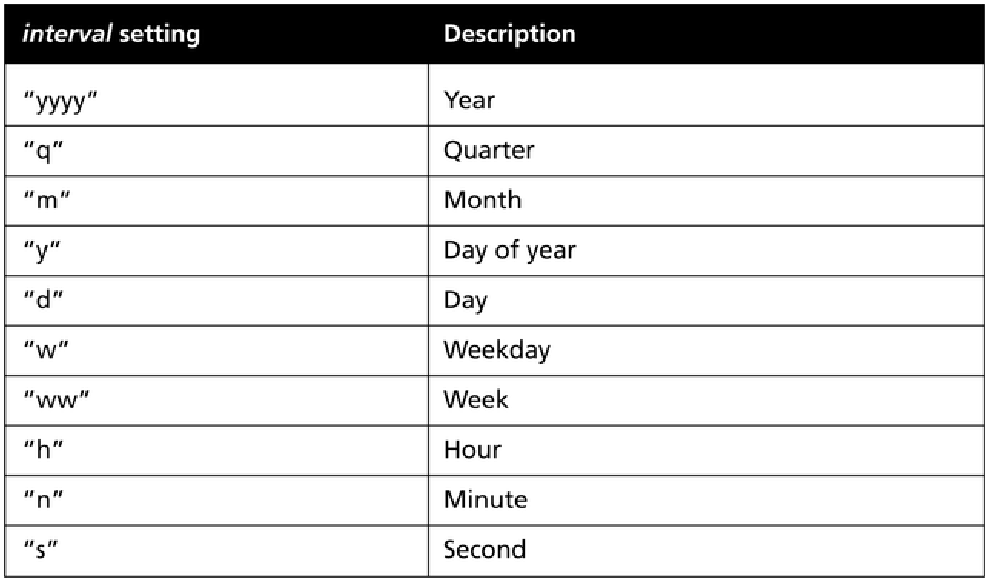
Adds 3 days to the value of the date variable dtmEgDate

E.g.

DateAdd(interval:="d", Number:=3, Date:=dtmEgDate)

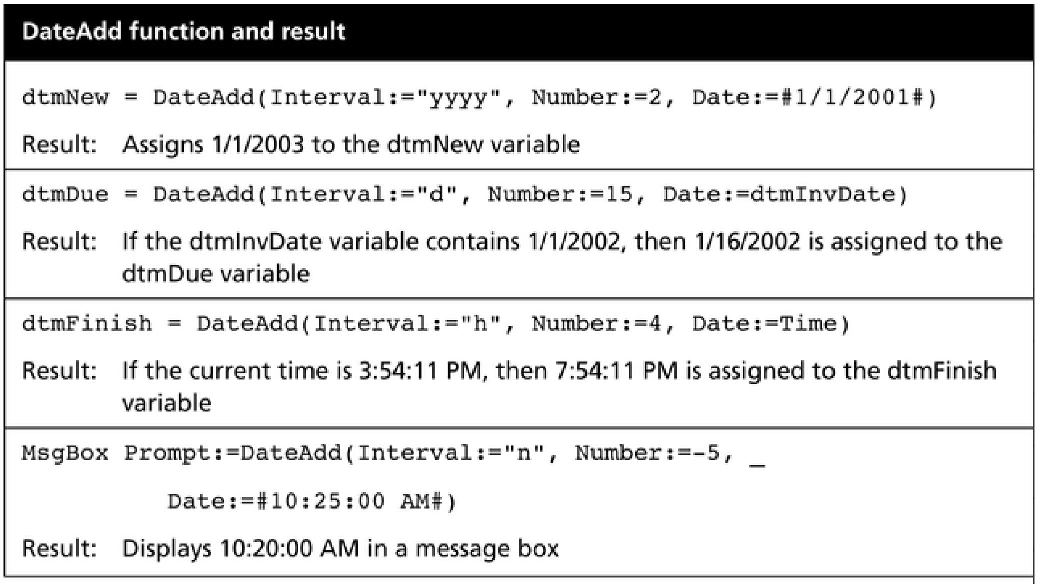
AssignDisplayDate.xls – see DateAddEg() procedure

Valid Settings for the Interval Argument



interval setting	Description
“yyyy”	Year
“q”	Quarter
“m”	Month
“y”	Day of year
“d”	Day
“w”	Weekday
“ww”	Week
“h”	Hour
“n”	Minute
“s”	Second

Examples of the DateAdd Function



DateAdd function and result
`dtmNew = DateAdd(Interval:="yyyy", Number:=2, Date:=#1/1/2001#)` Result: Assigns 1/1/2003 to the dtmNew variable
`dtmDue = DateAdd(Interval:="d", Number:=15, Date:=dtmInvDate)` Result: If the dtmInvDate variable contains 1/1/2002, then 1/16/2002 is assigned to the dtmDue variable
`dtmFinish = DateAdd(Interval:="h", Number:=4, Date:=Time)` Result: If the current time is 3:54:11 PM, then 7:54:11 PM is assigned to the dtmFinish variable

`MsgBox Prompt:=DateAdd(Interval:="n", Number:=-5, _`
`        Date:=#10:25:00 AM#)`
Result: Displays 10:20:00 AM in a message box

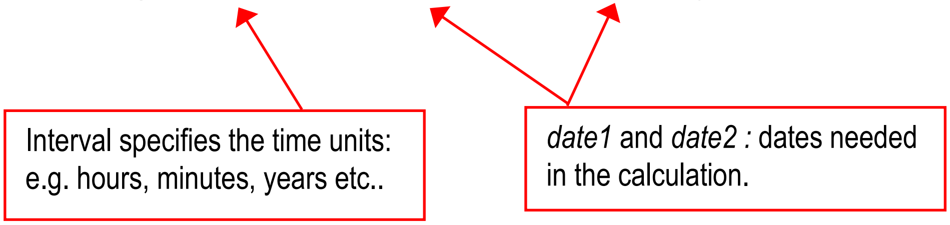
Using Dates and Times in Calculations

- The **DateDiff** function allows you to determine the time interval that occurs between two dates
- Unlike the **DateAdd** function, which returns either a future or past date or time, the **DateDiff** function returns an integer that represents the number of time intervals between two specified dates or times

The DateDiff function

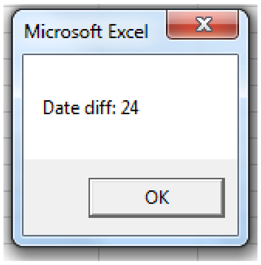
Syntax

DateDiff(Interval:=*interval*, **Date1:=***date1*, **Date2:=***date2*)



Interval specifies the time units: e.g. hours, minutes, years etc..

date1 and *date2 :* dates needed in the calculation.

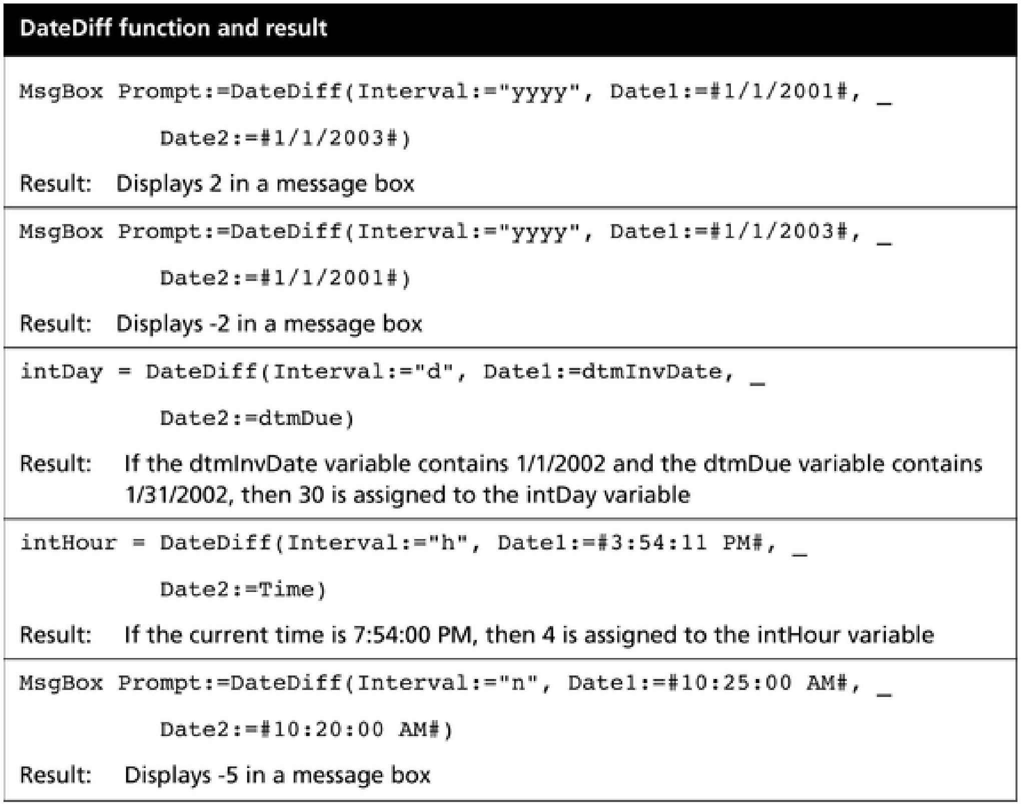


Date diff: 24

E.g.

MsgBox prompt:="Date diff: " & DateDiff("yyyy", #2/18/1991#, #1/27/2015 10:36:22 PM #)

Examples of the DateDiff Function



DateDiff function and result

`MsgBox Prompt:=DateDiff(Interval:="yyyy", Date1:=#1/1/2001#, _`
`Date2:=#1/1/2003#)`
Result: Displays 2 in a message box

`MsgBox Prompt:=DateDiff(Interval:="yyyy", Date1:=#1/1/2003#, _`
`Date2:=#1/1/2001#)`
Result: Displays -2 in a message box

`intDay = DateDiff(Interval:="d", Date1:=dtmInvDate, _`
`Date2:=dtmDue)`
Result: If the dtmInvDate variable contains 1/1/2002 and the dtmDue variable contains 1/31/2002, then 30 is assigned to the intDay variable

`intHour = DateDiff(Interval:="h", Date1:=#3:54:11 PM#, _`
`Date2:=Time)`
Result: If the current time is 7:54:00 PM, then 4 is assigned to the intHour variable

`MsgBox Prompt:=DateDiff(Interval:="n", Date1:=#10:25:00 AM#, _`
`Date2:=#10:20:00 AM#)`
Result: Displays -5 in a message box

5. Converting Strings to Dates

- Before using a string that represents a date
  or time in a calculation, you should use either the VBA **DateValue** function or the **TimeValue** function to convert the string to a date or time, respectively
- The syntax of the **DateValue** function is:
  DateValue(Date:=stringExpression)

stringExpression represents a valid date ranging from January 1, 100 through December 31, 9999

- The DateValue function returns the **date** equivalent of the stringExpression argument

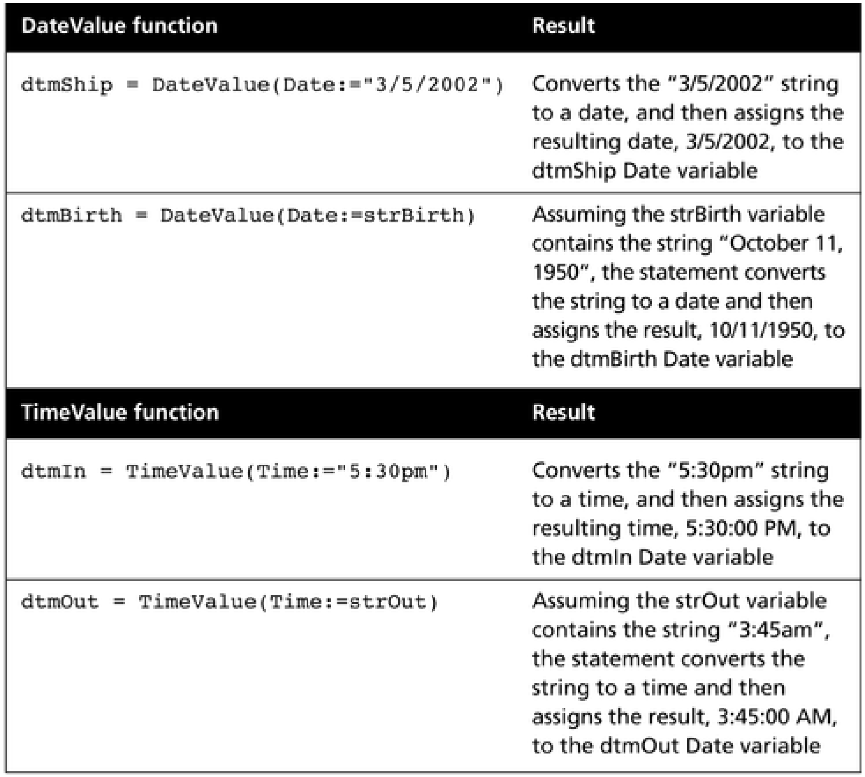
- The syntax of the TimeValue function is
  TimeValue(Time:=stringExpression)

stringExpression represents a valid time ranging from 0:00:00 (12:00:00 AM – start of day) through 23:59:59 (11:59:59 PM)

- The TimeValue function returns the time equivalent of the stringExpression argument

- AssignDisplayDate.xls (Sub Date_Time_Value())

Examples of Using the DateValue and TimeValue Functions to Convert Strings to Dates and Times



DateValue function	Result
`dtmShip = DateValue(Date:="3/5/2002")`	Converts the “3/5/2002” string to a date, and then assigns the resulting date, 3/5/2002, to the dtmShip Date variable
`dtmBirth = DateValue(Date:=strBirth)`	Assuming the strBirth variable contains the string “October 11, 1950”, the statement converts the string to a date and then assigns the result, 10/11/1950, to the dtmBirth Date variable
TimeValue function	**Result**
`dtmIn = TimeValue(Time:="5:30pm")`	Converts the “5:30pm” string to a time, and then assigns the resulting time, 5:30:00 PM, to the dtmIn Date variable
`dtmOut = TimeValue(Time:=strOut)`	Assuming the strOut variable contains the string “3:45am”, the statement converts the string to a time and then assigns the result, 3:45:00 AM, to the dtmOut Date variable

Excel Example: Creating the CalcHours Macro Procedure

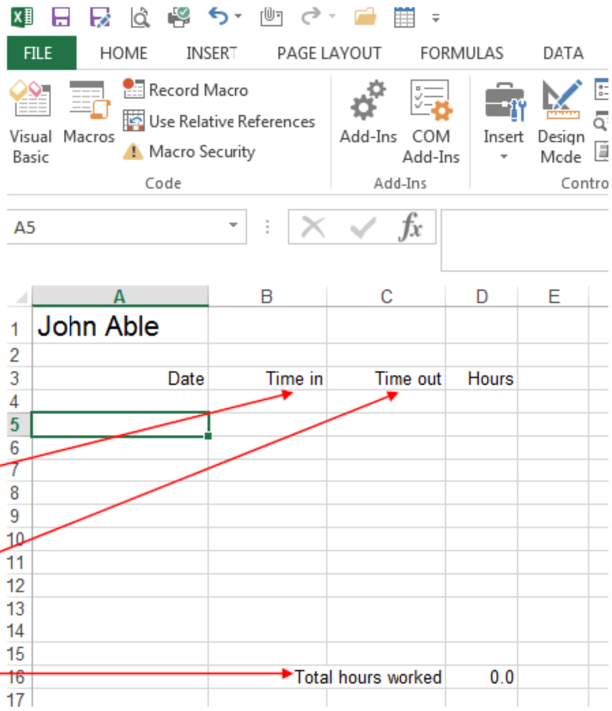
This exercise involves:

- Finding the total number of hours worked each day
- Calculating the total hours worked per fortnight for each employee

Hours Worked.xls

Pseudocode for the CalcHours Procedure

```
1. Use the InputBox function to prompt the user to enter the starting time. Store the response in a string variable named strIn
2. Use the InputBox function to prompt the user to enter the ending time. Store the response in a string variable named strOut
3. Use the TimeValue function to convert the string stored in strIn to a time, then assign the result to a date variable named dtmIn
4. Use the TimeValue function to convert the string stored in strOut to a time, then assign the result to a date variable named dtmOut
5. assign the system date to the active cell in column A
6. assign the starting time (stored in dtmIn ) to the cell located one column to the right of the active cell. I.e. in column B
7. assign the ending time (stored in dtmOut ) to the cell located two columns to the right of the active cell. I.e. in column C
8. use the DateDiff function to calculate the number of hours worked. Assign the result to the cell located three columns to the right
```



	A	B	C	D	E
1	John Able				
2					
3	Date	Time in	Time out	Hours	
4					
5					
16			Total hours worked	0.0	

Creating the CalcHours Macro Procedure

Declare string and object vars, set the object variables:

```
Public Sub CalcHours()
 'declare variables and assign address to object variable
    Dim strIn As String
    Dim strOut As String
    Dim dtmIn As Date
    Dim dtmOut As Date
    Dim rngActive As Range
    Set rngActive = Application.ActiveCell
End Sub
```

User entered times are Stored as strings

The date variables are used to store the actual times in the 'time' format

This range variable stores the active cell Address in the worksheet

ActiveCell Returns a Range object that represents the active cell in the active window

Partially Completed CalcHours Procedure

```
Public Sub CalcHours()
   'declare variables and assign address to object variable
   Dim strIn As String, strOut As String, dtmIn As Date, dtmOut As Date
   Dim rngActive As Range
   Set rngActive = Application.ActiveCell
   'enter starting and ending time
   strIn = InputBox(prompt:="Enter the starting time:", _
      Title:="Start Time", Default:=#9:00:00 AM#)
   strOut = InputBox(prompt:="Enter the ending time:", _
      Title:="End Time", Default:=#5:00:00 PM#)
   'convert strings to times
   dtmIn = TimeValue(Time:=strIn)
   dtmOut = TimeValue(Time:=strOut)
   'assign values to worksheet cells
   rngActive.Value = Date
End Sub
```

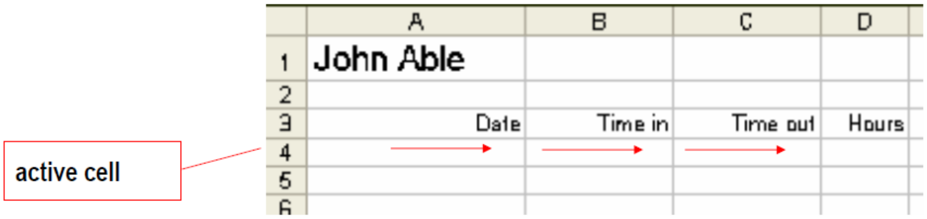
Prompts user for Start/Finish time and stores response in strIn/strOut

Convert the string values to Dates (times)

Assign the System Date to the active cell

6. The Offset Property of the Range object

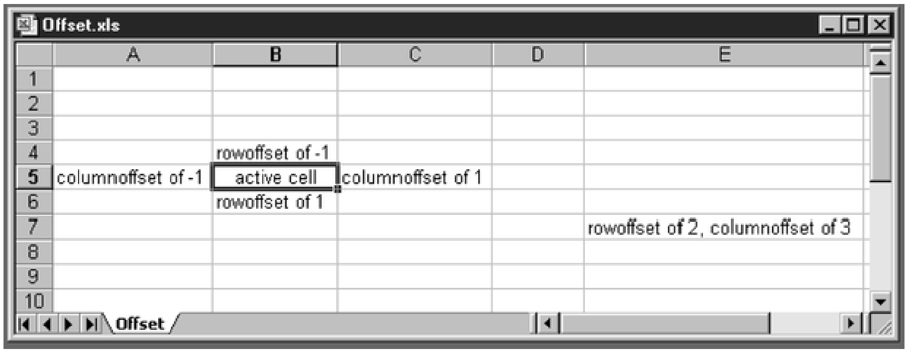
- You can use a Range object's **Offset** property to refer to a cell located a certain number of rows or columns away from the range itself
- The syntax of the Offset property is
-     rangeObject.Offset([rowOffset] [,columnOffset])
- You use a **positive** rowOffset to refer to rows found below the rangeObject, and you use a **negative** rowOffset to refer to rows above the rangeObject
- You use a **positive** columnOffset to refer to columns found to the right of the rangeObject, and you use a **negative** columnOffset to refer to columns to the left of the rangeObject



	A	B	C	D
1	John Able			
2				
3	Date	Time in	Time out	Hours
4				
5				
6				

active cell

Illustration of the Offset Property



	A	B	C	D	E
1					
2					
3					
4		rowoffset of -1			
5	columnoffset of -1	active cell	columnoffset of 1		
6		rowoffset of 1			
7					rowoffset of 2, columnoffset of 3
8					
9					
10					

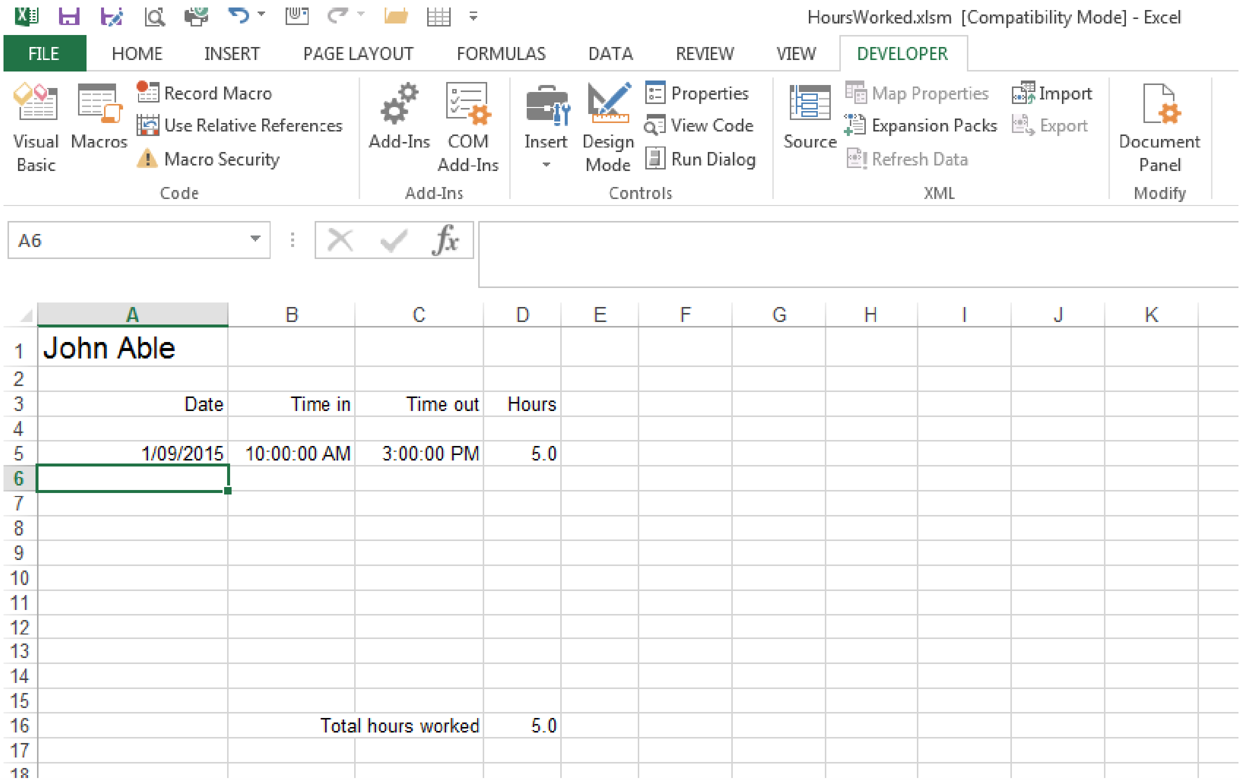
For example:
If rangeObject (I.e. active cell) is B5 then
    rowOffset of 1 refers to B6
    rowOffset of –1 refers to B4
    columnOffset of 1 refers to C5
    columnOffset of –1 refers to A5
What does rangeObject.Offset(2,3) refer to? (E7)

Completed CalcHours Procedure

```
Public Sub CalcHours()
  'declare variables and assign address to object variable
  Dim strIn As String, strOut As String, dtmIn As Date, dtmOut As Date
  Dim rngActive As Range
  Set rngActive = Application.ActiveCell
  'enter starting and ending time
  strIn = InputBox(prompt:="Enter the starting time:", _
    Title:="Start Time", Default:=#9:00:00 AM#)
  strOut = InputBox(prompt:="Enter the ending time:", _
    Title:="End Time", Default:=#5:00:00 PM#)
  'convert strings to times
  dtmIn = TimeValue(Time:=strIn)
  dtmOut = TimeValue(Time:=strOut)
  'assign values to worksheet cells
  rngActive.Value = Date
  rngActive.Offset(columnoffset:=1).Value = dtmIn
  rngActive.Offset(columnoffset:=2).Value = dtmOut
  rngActive.Offset(columnoffset:=3).Value = _
    DateDiff(interval:="n", date1:=dtmIn, date2:=dtmOut) / 60
End Sub
```

**Assigns the time values
To the respective cells
In the worksheet**

Worksheet after running the procedure



	A	B	C	D
1	John Able			
2				
3	Date	Time in	Time out	Hours
4				
5	1/09/2015	10:00:00 AM	3:00:00 PM	5.0
6				
16			Total hours worked	5.0

The Immediate Window and Debug.Print

Immediate window

The Debug.Print statement can be used in your code to display messages or variable values in the Immediate Window.

Syntax:

Debug.Print *expression*

Does not affect the operation of your code.

E.g.

Debug.Print strOut

The IsDate() function

To check whether the InputBox function has returned a valid date use the IsDate function.

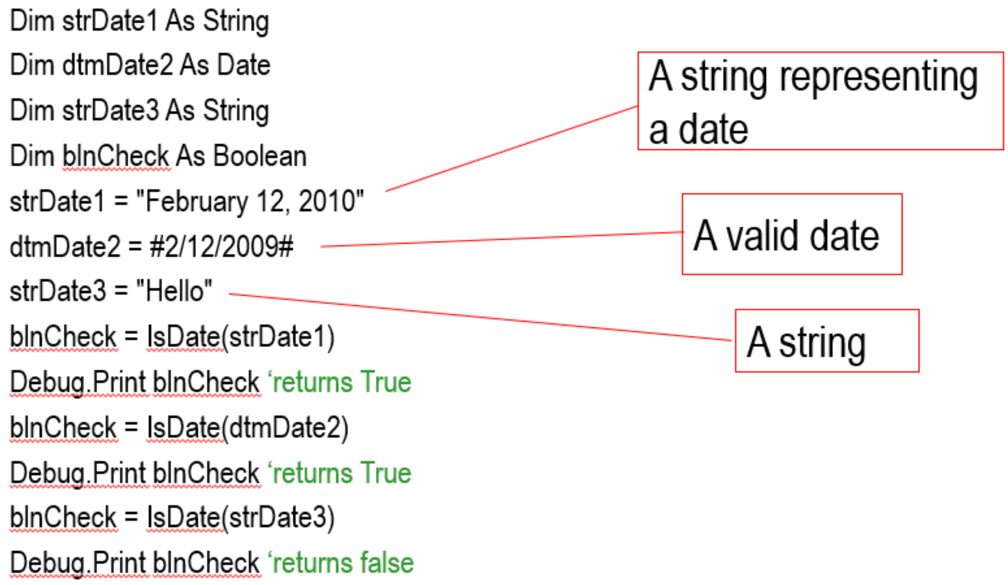
Syntax:

IsDate(*expression*)

The required *expression* argument is a Variant containing a date expression or string expression recognizable as a date or time.

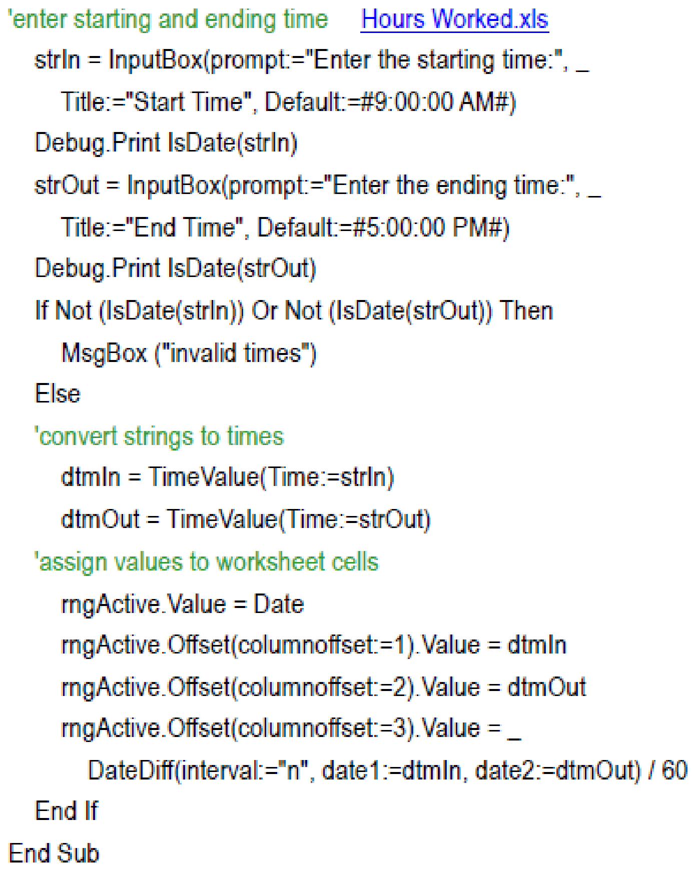
IsDate returns either True or False depending on whether the *expression* represents a valid date.

IsDate() example



```
Dim strDate1 As String
Dim dtmDate2 As Date
Dim strDate3 As String
Dim blnCheck As Boolean
strDate1 = "February 12, 2010"
dtmDate2 = #2/12/2009#
strDate3 = "Hello"
blnCheck = IsDate(strDate1)
Debug.Print blnCheck 'returns True
blnCheck = IsDate(dtmDate2)
Debug.Print blnCheck 'returns True
blnCheck = IsDate(strDate3)
Debug.Print blnCheck 'returns false
```

Updated CalcHours procedure



```
'enter starting and ending time    Hours Worked.xls
   strIn = InputBox(prompt:="Enter the starting time:", _
      Title:="Start Time", Default:=#9:00:00 AM#)
   Debug.Print IsDate(strIn)
   strOut = InputBox(prompt:="Enter the ending time:", _
      Title:="End Time", Default:=#5:00:00 PM#)
   Debug.Print IsDate(strOut)
   If Not (IsDate(strIn)) Or Not (IsDate(strOut)) Then
      MsgBox ("invalid times")
   Else
   'convert strings to times
      dtmIn = TimeValue(Time:=strIn)
      dtmOut = TimeValue(Time:=strOut)
   'assign values to worksheet cells
      rngActive.Value = Date
      rngActive.Offset(columnoffset:=1).Value = dtmIn
      rngActive.Offset(columnoffset:=2).Value = dtmOut
      rngActive.Offset(columnoffset:=3).Value = _
         DateDiff(interval:="n", date1:=dtmIn, date2:=dtmOut) / 60
   End If
End Sub
```

7. The MsgBox Function

- The MsgBox function allows you to display a dialog box that contains a message, one or more command buttons, and an icon
- So far we have used the MsgBox function in the form of a **statement**,

  i.e. MsgBox Prompt:=prompt, Buttons:=buttons, Title:=title

  e.g. MsgBox Prompt:="hello", Buttons:=vbOKOnly, Title:="welcome"
- However it can be used to capture information from the user.
- After displaying the dialog box, both the MsgBox statement and the MsgBox function wait for the user to choose one of the command buttons
- Unlike the MsgBox statement, the MsgBox function **returns an integer value** that indicates which button the user chose

The syntax of the MsgBox function:

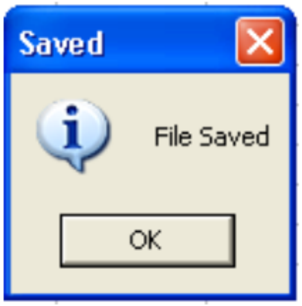
MsgBox (*Prompt, [Buttons], [Title])*

prompt is the message in the dialog box

title is the text in the title bar

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

E.g. MsgBox function:



File Saved

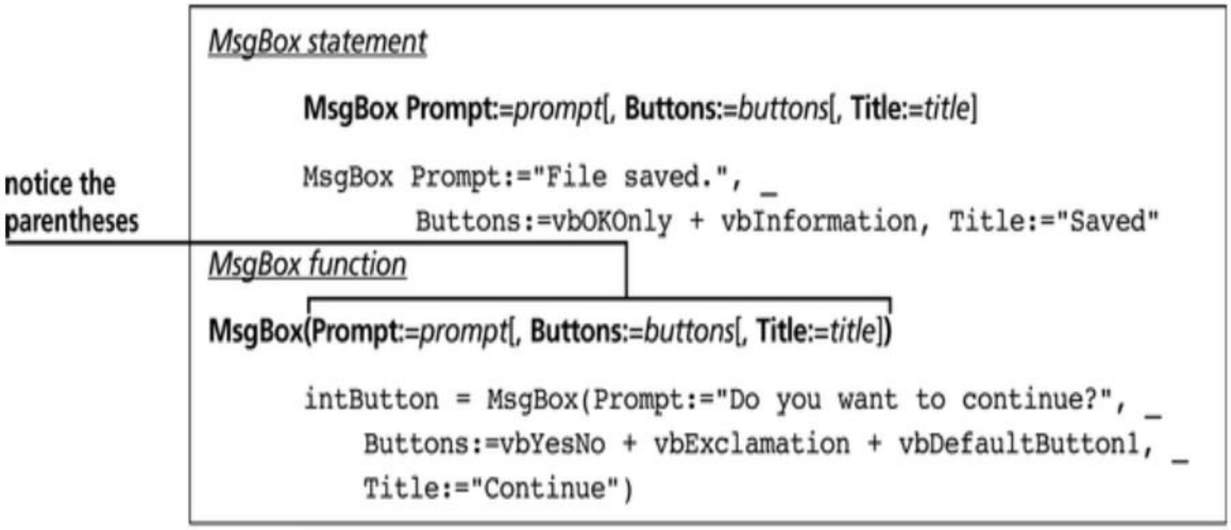
prompt is the message in the dialog box

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"
```

MsgBox function

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

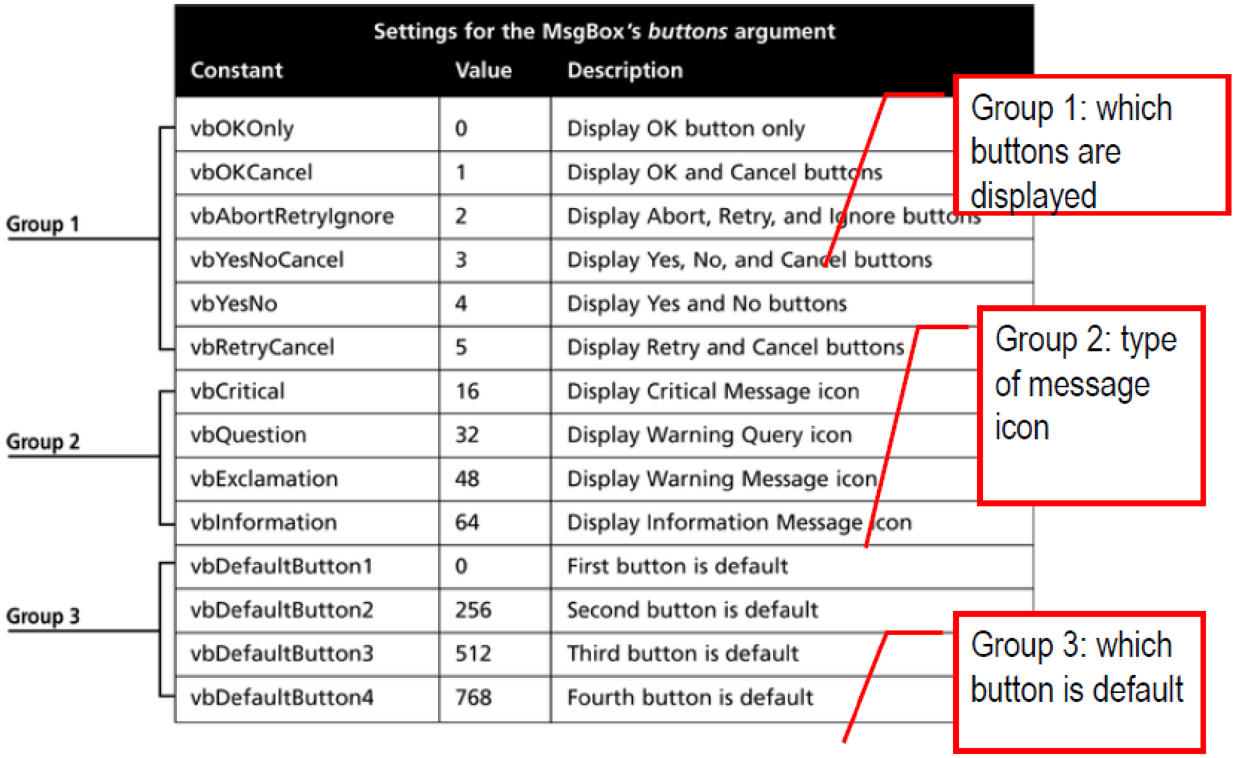
```
intButton = MsgBox(Prompt:="Do you want to continue?", _
   Buttons:=vbYesNo + vbExclamation + vbDefaultButton1, _
   Title:="Continue")
```

notice the parentheses

The Buttons Argument

- The buttons argument is an optional numeric expression that represents the sum of values specifying the number and type of buttons to display in the dialog box, the icon style to use, and the identity of the default button
- If you omit the buttons argument, the dialog box contains an OK button only; it does not contain an icon
- The buttons argument's settings are divided into three groups
- If you do not want to display an icon in the message box, you do not need to include a number from the second group in the buttons argument

Valid Settings for the buttons Argument



Settings for the MsgBox's *buttons* argument

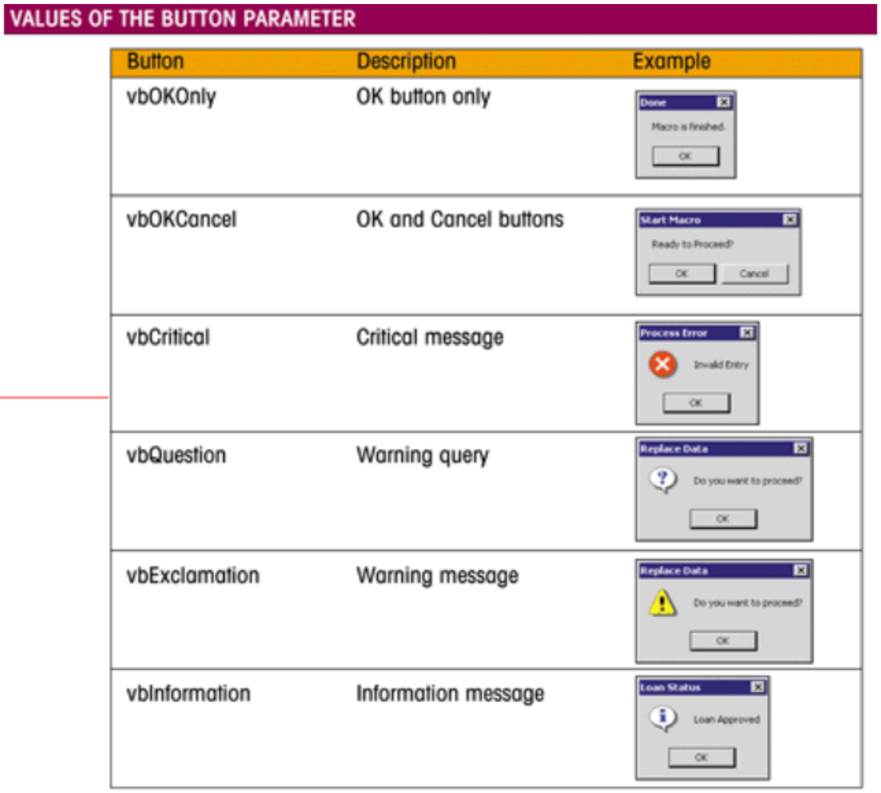
	Constant	Value	Description
Group 1	vbOKOnly	0	Display OK button only
	vbOKCancel	1	Display OK and Cancel buttons
	vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons
	vbYesNoCancel	3	Display Yes, No, and Cancel buttons
	vbYesNo	4	Display Yes and No buttons
	vbRetryCancel	5	Display Retry and 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

Group 1: which buttons are displayed

Group 2: type of message icon

Group 3: which button is default

Message Box Button arguments

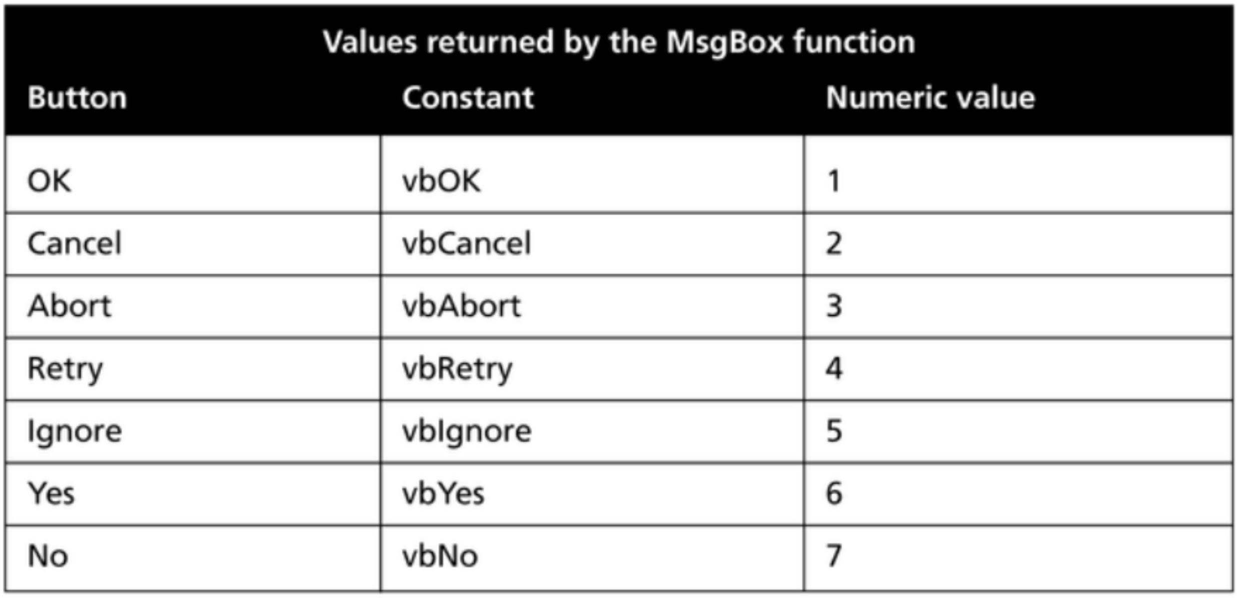


VALUES OF THE BUTTON PARAMETER

Button	Description	Example
vbOKOnly	OK button only	Done Macro is finished. OK
vbOKCancel	OK and Cancel buttons	Start Macro Ready to Proceed? OK Cancel
vbCritical	Critical message	Process Error Invalid Entry OK
vbQuestion	Warning query	Replace Data Do you want to proceed? OK
vbExclamation	Warning message	Replace Data Do you want to proceed? OK
vbInformation	Information message	Loan Status Loan Approved OK

Example Button arguments

MsgBox Function’s Buttons

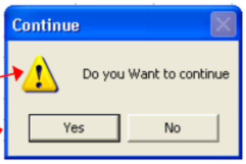


Values returned by the MsgBox function		
Button	**Constant**	**Numeric value**
OK	vbOK	1
Cancel	vbCancel	2
Abort	vbAbort	3
Retry	vbRetry	4
Ignore	vbIgnore	5
Yes	vbYes	6
No	vbNo	7

Values returned by the MsgBox function

Example 7.1

```
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
```



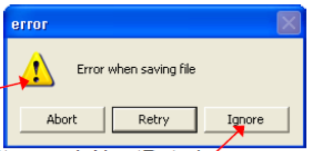
Do you Want to continue

Alternative:
4+48+0

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

MsgBoxEgs.xls

Example 7.2



Error when saving file

```
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
```

Alternative: 2+48+256 Alternative 306

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

MsgBoxEgs.xls

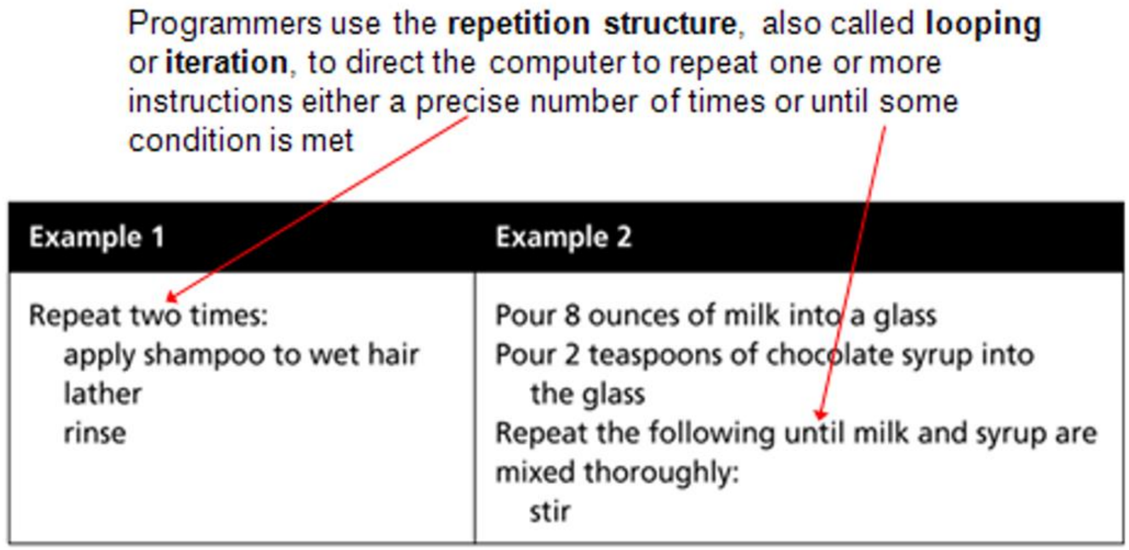
Summary Part A

- To reserve a procedure-level Date variable:
  - Use the Dim statement.
  - The syntax of the Dim statement is ***Dim variablename As datatype***
  - When reserving a Date variable, **datatype** is always the keyword **Date**
- To assign a value to a variable:
  - Use an assignment statement in the following syntax: ***variablename = value***
- To access the current system date and time:
  - Use the VBA Date, Time, and Now functions
- To control the appearance of dates and times:
  - Use the VBA function, the syntax of which **is Format(Expression:=expression, Format:=format)**
- To add a specified time interval to a date or time, and then return the new date or time:
  - Use the VBA **DateAdd** function
- To calculate the number of time intervals between two specified dates or times:
  - Use the VBA **DateDiff** function
- To convert a string to a Date data type:
  - Use the **DateValue** function to return the date equivalent of a string
  - Use the **TimeValue** function to return the time equivalent of a string
- The IsDate function
- The Debug.Print command for displaying messages in the Immediate window
- 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])*

Part B

8. 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: apply shampoo to wet hair lather rinse	Pour 8 ounces of milk into a glass Pour 2 teaspoons of chocolate syrup into the glass Repeat the following until milk and syrup are mixed thoroughly: stir

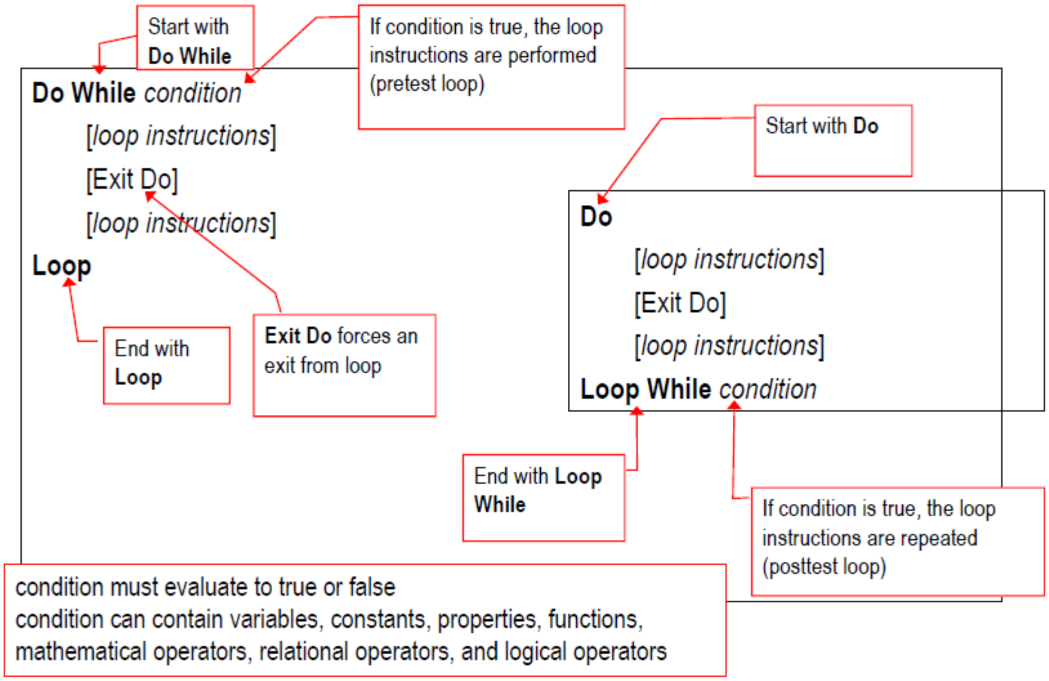
VBA Forms of the Repetition Structure

- Do While
- Do Until
- For Next
  - For…Next
  - For Each…Next
- The With statement

9. 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
```

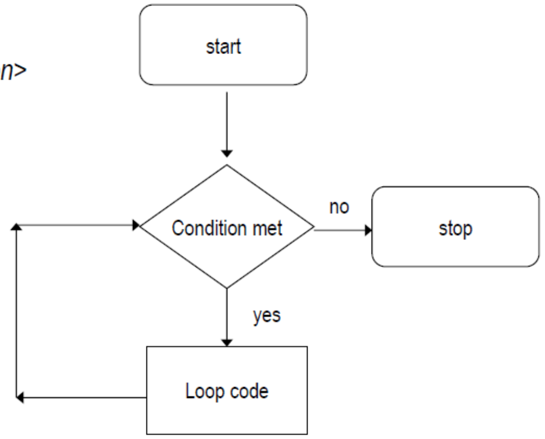
End with **Loop While**

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

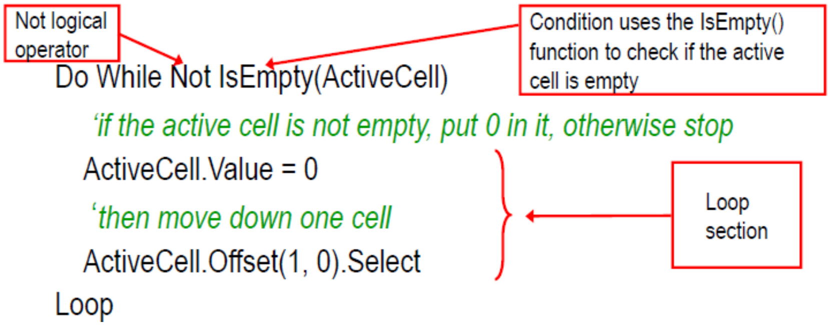
condition must evaluate to true or false
condition can contain variables, constants, properties, functions, mathematical operators, relational operators, and logical operators

Do While loop (pretest)

```
Syntax:
Do While <condition>
    VBA code
    [Exit Do]
    VBA code
Loop
```



Example 9.1: Do While loop



Not logical operator

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

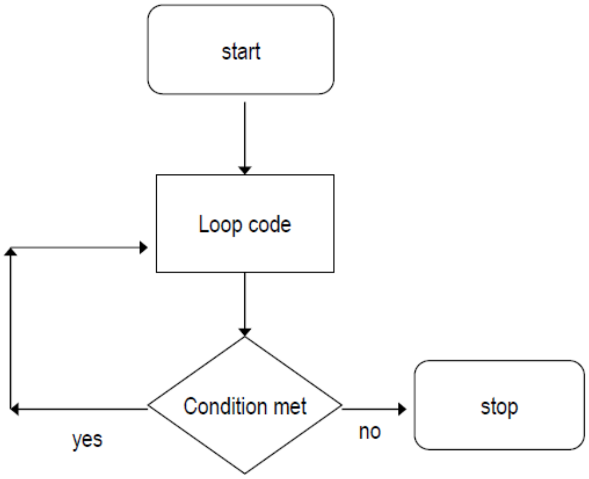
```
Do While Not IsEmpty(ActiveCell)
    'if the active cell is not empty, put 0 in it, otherwise stop
    ActiveCell.Value = 0
    'then move down one cell
    ActiveCell.Offset(1, 0).Select
Loop
```

Loop section

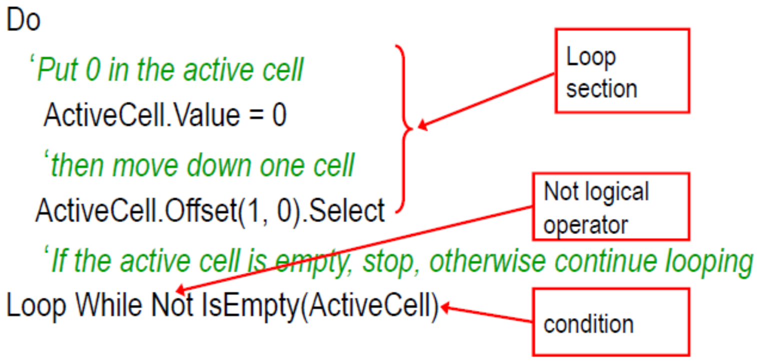
- DO_WHILE and Offset.XLS (macro - DoWhileDemo1())

Do While loop (posttest)

```
Syntax:
Do
    VBA code
    [Exit Do]
    VBA code
Loop While <condition>
```



Example 9.2: Do While loop



```
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)
```

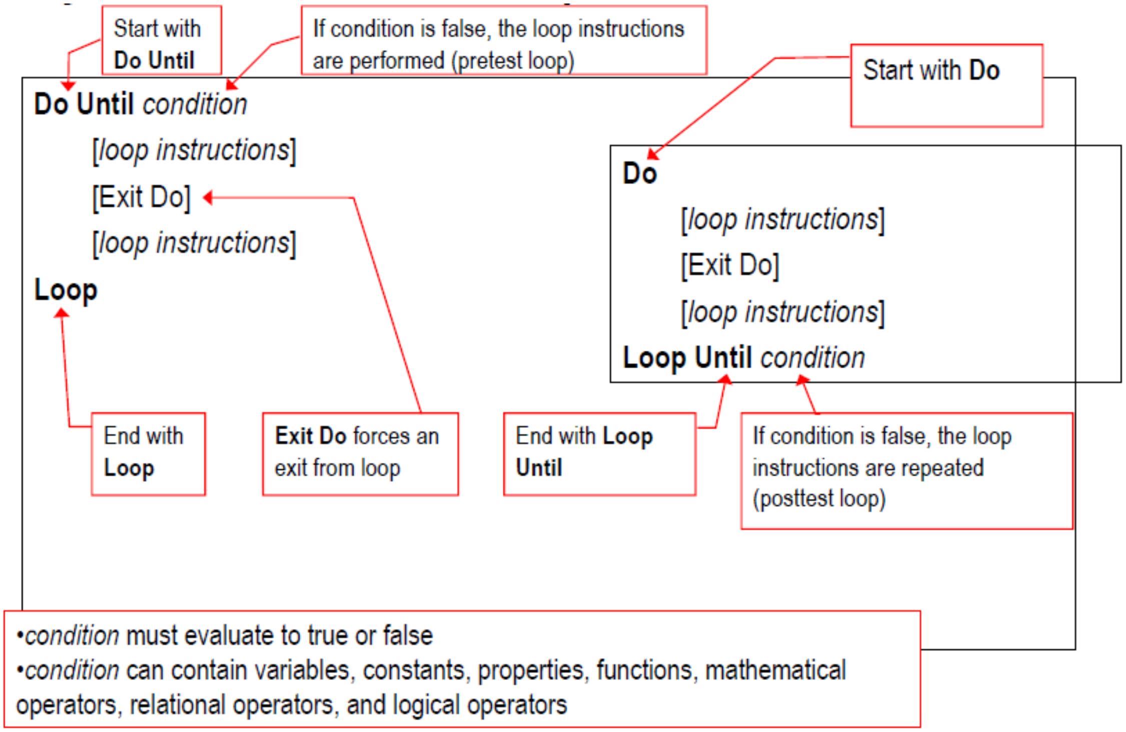
Loop section

Not logical operator

condition

- DO_WHILE and Offset.XLS (macro – DoWhileDemo2())

Syntax of the Do Until Loops



```
Do Until condition
     [loop instructions]
     [Exit Do]
     [loop instructions]
Loop
```

```
Do
     [loop instructions]
     [Exit Do]
     [loop instructions]
Loop Until condition
```

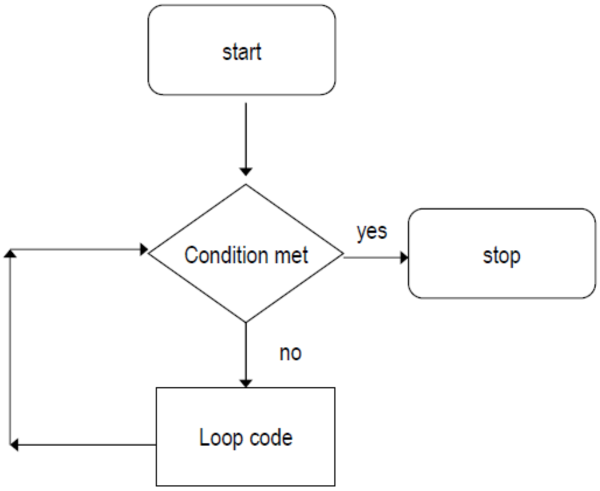
Start with **Do Until**
If condition is false, the loop instructions are performed (pretest loop)
Start with **Do**
End with **Loop**
Exit Do forces an exit from loop
End with **Loop Until**
If condition is false, the loop instructions are repeated (posttest loop)

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

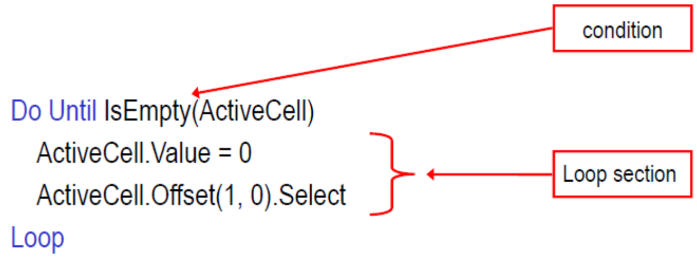
Do Until loop (pretest)

Syntax:

```
Do Until <condition>
     VBA code
     [Exit Do]
     VBA code
Loop
```



Example 9.3: Do Until (pretest loop)



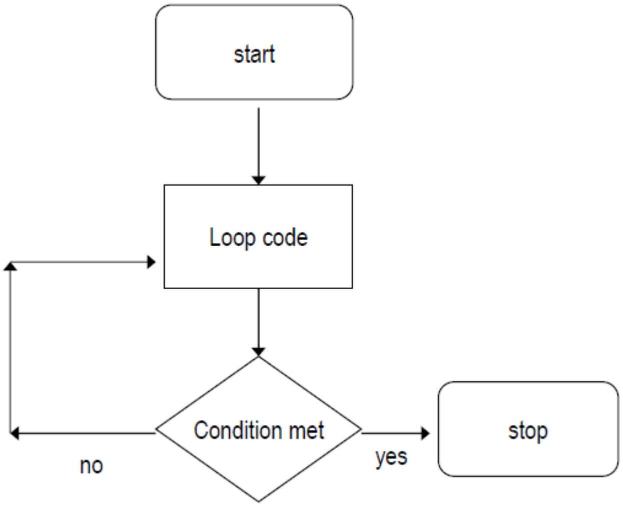
```
Do Until IsEmpty(ActiveCell)
   ActiveCell.Value = 0
   ActiveCell.Offset(1, 0).Select
Loop
```

Loop repeats until the condition is true

Do Until loop (posttest)

Syntax:

```
Do
    VBA code
    [Exit Do]
    VBA code
Loop Until <condition>
```



Example 9.4: 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

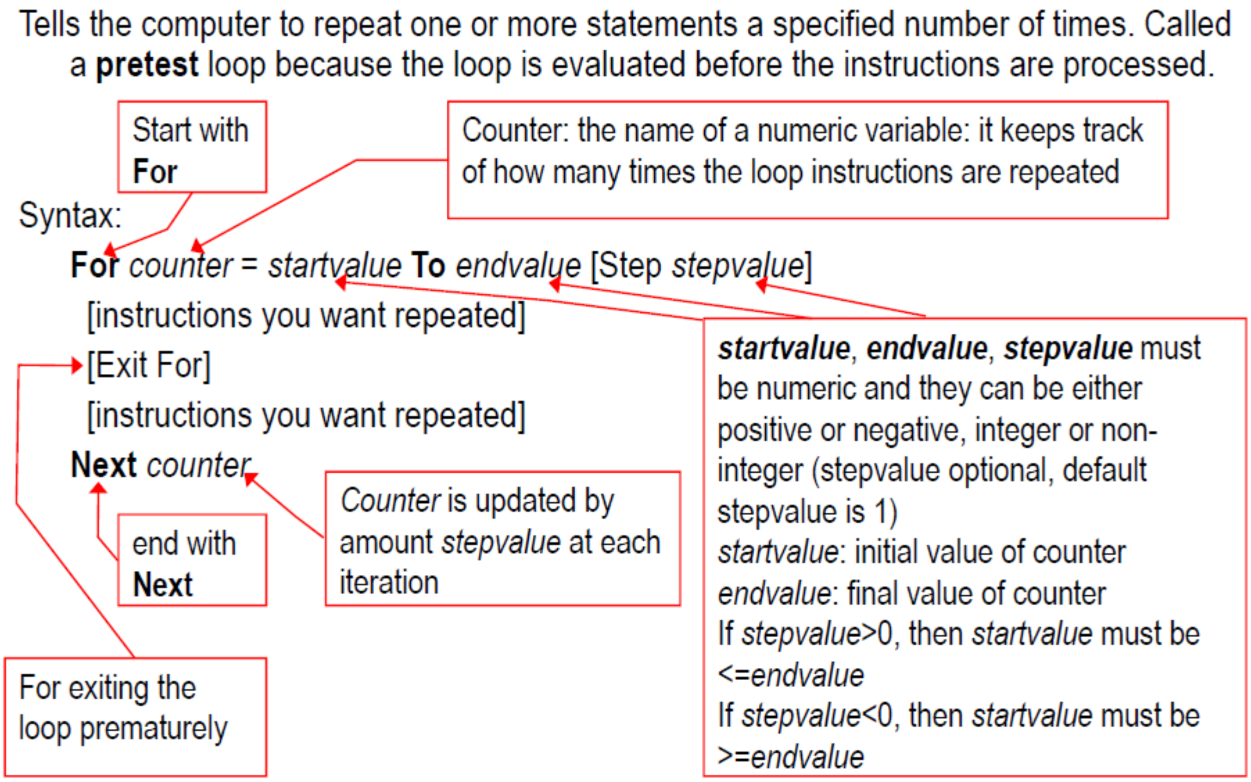
10. The For…Next Statement

- 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
```

Syntax and an Example of the For…Next Statement



Tells the computer to repeat one or more statements a specified number of times. Called a **pretest** loop because the loop is evaluated before the instructions are processed.

Start with **For**

Counter: the name of a numeric variable: it keeps track of how many times the loop instructions are repeated

Syntax:

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

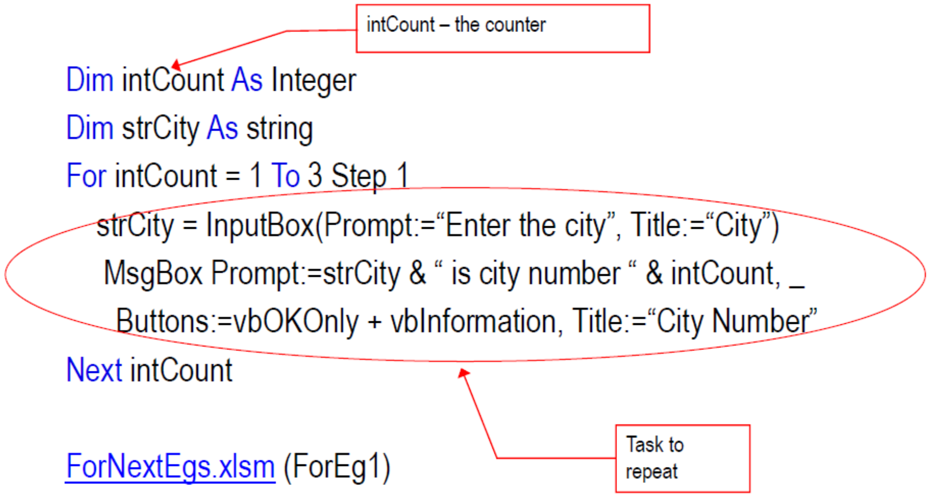
startvalue, ***endvalue***, ***stepvalue*** must be numeric and they can be either positive or negative, integer or non-integer (stepvalue optional, default stepvalue is 1)
startvalue: initial value of counter
endvalue: final value of counter
If *stepvalue*>0, then *startvalue* must be <=*endvalue*
If *stepvalue*<0, then *startvalue* must be >=*endvalue*

Counter is updated by amount *stepvalue* at each iteration

end with **Next**

For exiting the loop prematurely

Example 10.1 of For…Next



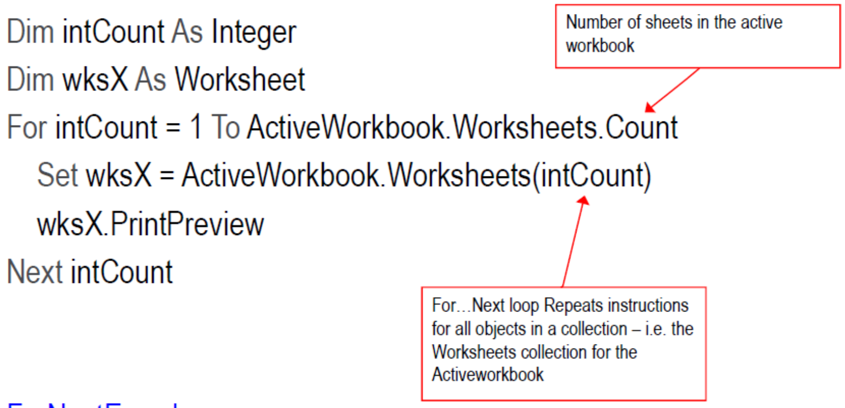
```
Dim intCount As Integer
Dim strCity As string
For intCount = 1 To 3 Step 1
  strCity = InputBox(Prompt:="Enter the city", Title:="City")
   MsgBox Prompt:=strCity & " is city number " & intCount, _
     Buttons:=vbOKOnly + vbInformation, Title:="City Number"
Next intCount
```

intCount – the counter

Task to repeat

ForNextEgs.xlsm (ForEg1)

Example 10.2 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
```

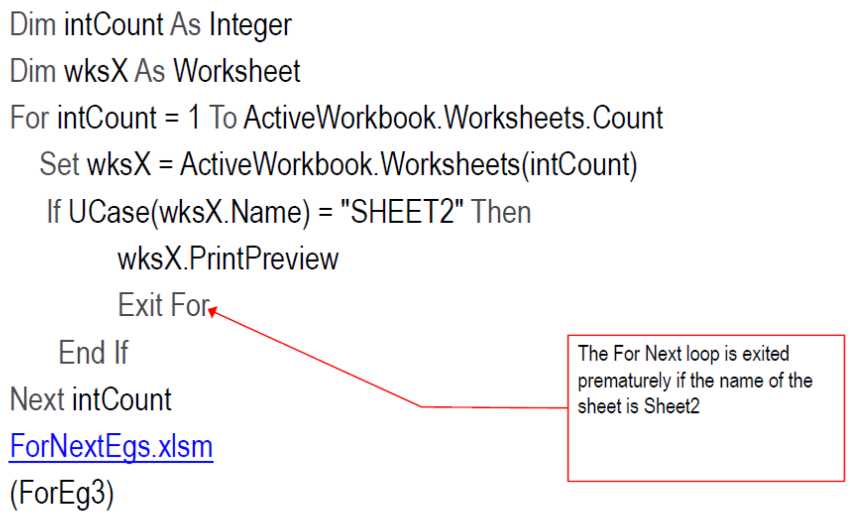
Number of sheets in the active workbook

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

ForNextEgs.xlsm

(ForEg2)

Example 10.3 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
```

ForNextEgs.xlsm

(ForEg3)

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

The For…Next Statement summary

- The ***startvalue, endvalue,*** and ***stepvalue*** items control how many times the loop instructions should be processed
- The ***startvalue*** tells the loop where to begin
- The loop initializes the counter to the ***startvalue*** (done only once, at the beginning of the loop).
- The ***endvalue*** tells the loop when to stop
- the ***stepvalue*** tells the loop how much to add to (or subtract from if the ***stepvalue*** is a negative number) the counter each time the loop is processed
- If the ***stepvalue*** is positive (negative), the loop checks if the value in counter is greater than (less than) the ***endvalue***. If it is, the loop stops; otherwise the instructions within the loop are processed and the next task is performed
- The For clause's ***startvalue, endvalue***, and ***stepvalue*** values must be numeric and they can be either positive or negative, integer or non-integer

Syntax:

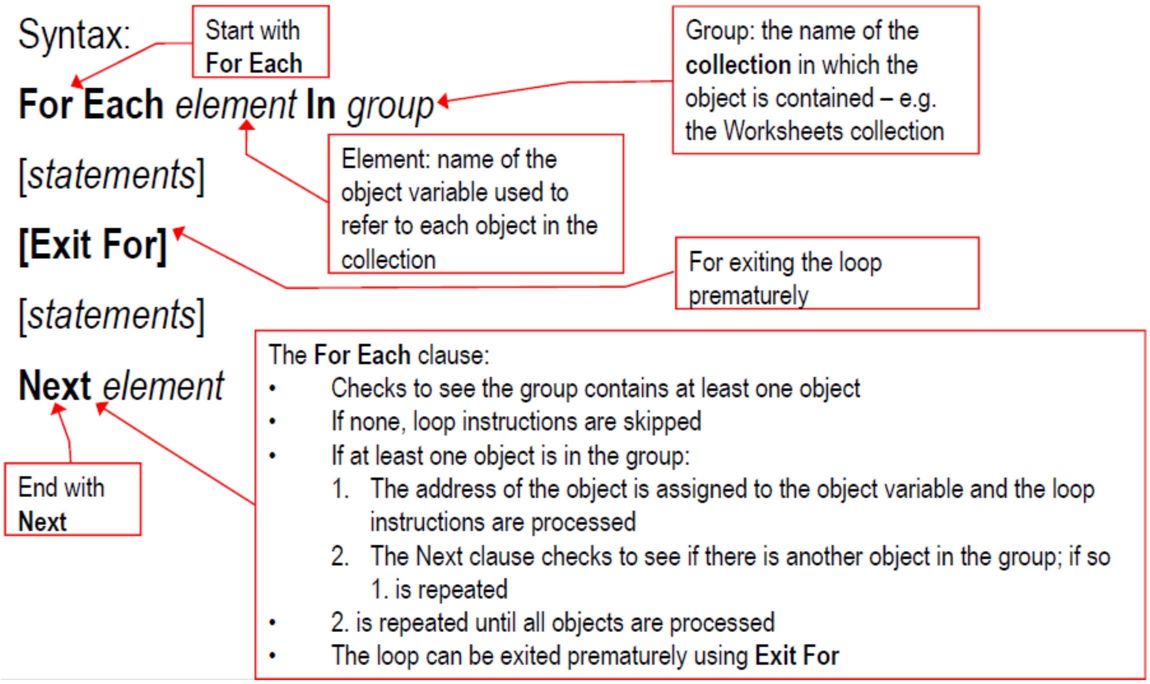
```
For counter = startvalue To endvalue [Step stepvalue]
  [instructions you want repeated]
  [Exit For]
  [instructions you want repeated]
Next counter
```

11. 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

The For Each…Next Statement



Syntax:

For Each *element* **In** *group*

[*statements*]

[Exit For]

[*statements*]

Next *element*

Start with **For Each**

Group: the name of the **collection** in which the object is contained – e.g. the Worksheets collection

Element: name of the object variable used to refer to each object in the collection

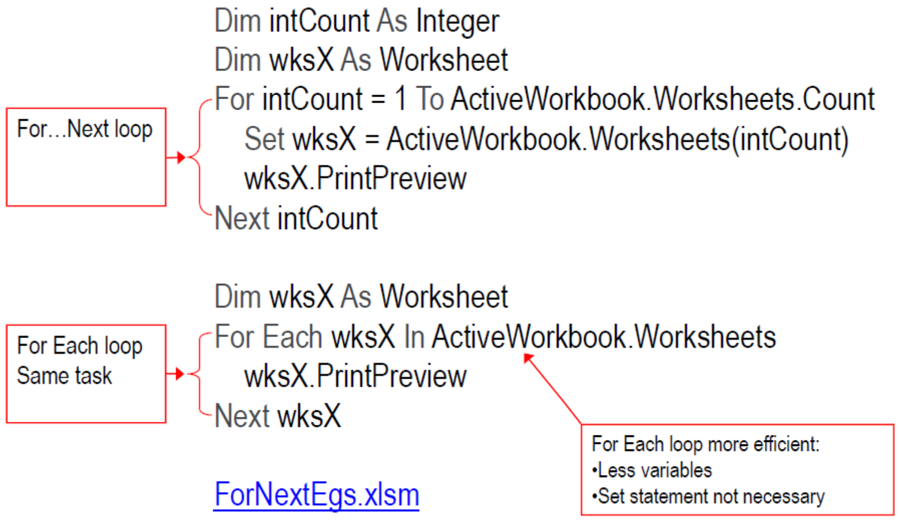
For exiting the loop prematurely

End with **Next**

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



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

For…Next loop

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

For Each loop
Same task

For Each loop more efficient:
•Less variables
•Set statement not necessary

ForNextEgs.xlsm

Example 11.1: 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

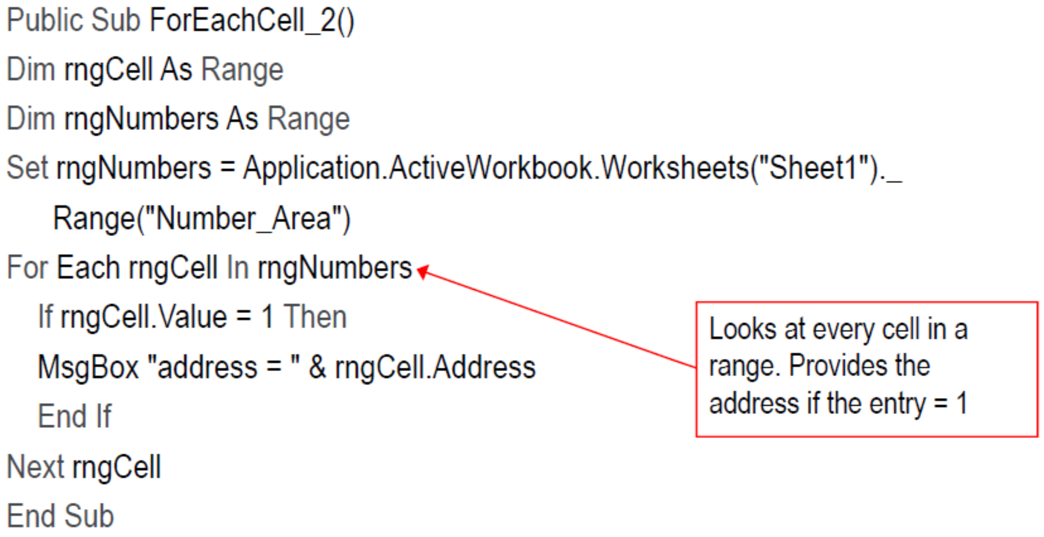
- Declare 2 Range variables, one points at the range of interest, the other is used to access all cells in the range

```
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

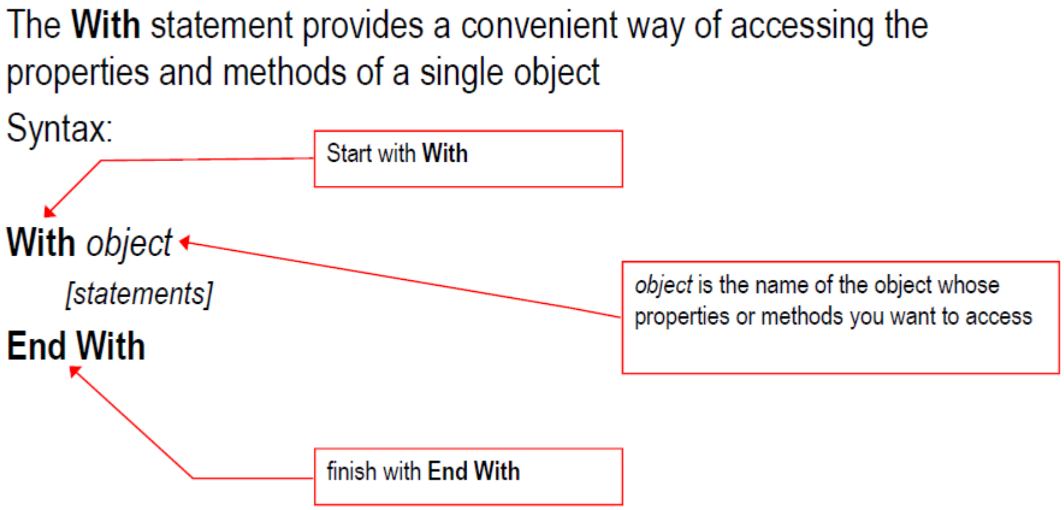


```
Public Sub ForEachCell_2()
Dim rngCell As Range
Dim rngNumbers As Range
Set rngNumbers = Application.ActiveWorkbook.Worksheets("Sheet1")._
     Range("Number_Area")
For Each rngCell In rngNumbers
   If rngCell.Value = 1 Then
   MsgBox "address = " & rngCell.Address
   End If
Next rngCell
End Sub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

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

ForNextEgs_2.xlsm (ForEachCell2())

12. The With Statement



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

Syntax:

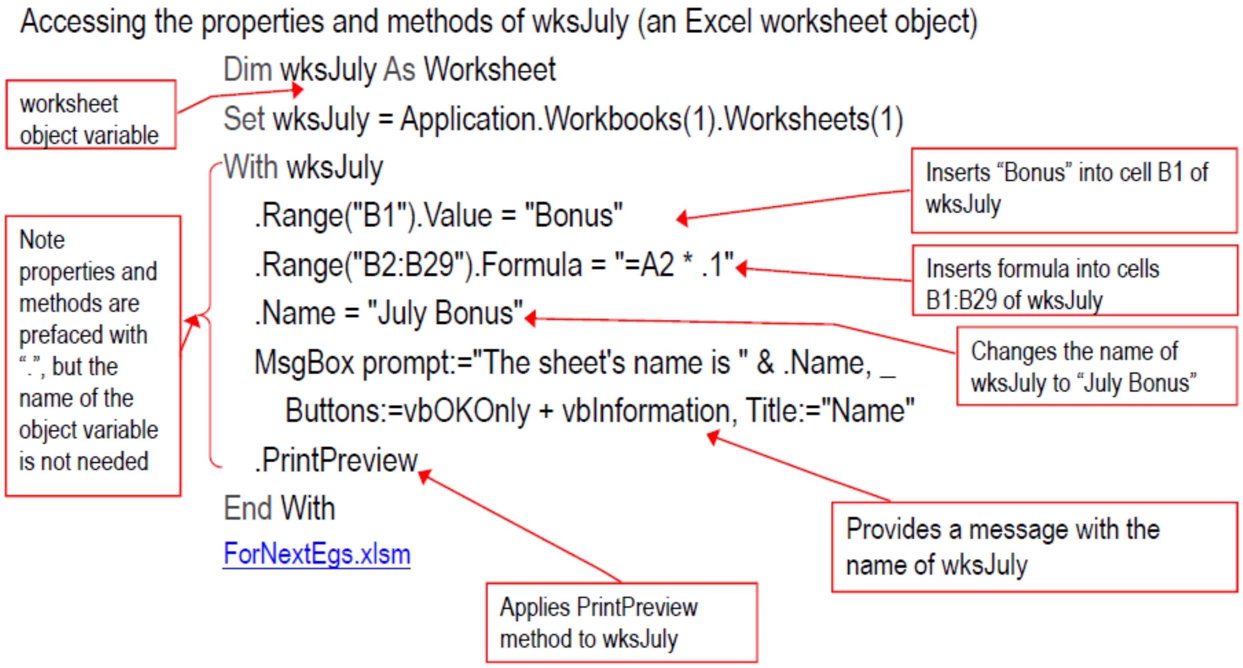
Start with **With**

```
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 12.1: With Statement



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

```
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

worksheet object variable

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

Inserts "Bonus" into cell B1 of wksJuly

Inserts formula into cells B1:B29 of wksJuly

Changes the name of wksJuly to "July Bonus"

Provides a message with the name of wksJuly

Applies PrintPreview method to wksJuly

13. Practice and Apply

- Understanding how to use Date and related variables
- Understanding how to use VBA's date and time functions
- Understanding how to implement repetition structures in VBA
- Complete Tutorial 7 Exercises