

FIT1013 – Digital Futures: IT for Business

Tutorial 7 – Date Variables and Repetition Structures

Objectives:

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

Reminder:

1. Ensure the Developer tab is visible on the Ribbon (If not, go to **File->Options->Customize Ribbon**, and select **Developer** in the **Main Tabs** list).
2. Save your file as **Tute7** (with Save as type: *Excel Macro-Enabled Workbook (*.xlsm)*).
3. Open the VBE (Visual Basic editor) by selecting the **Developer** tab, then clicking the **Visual Basic** button on the **Code group** (or by pressing <Alt> and <F11>).

Part 1:

Exercise 1: Using the range **Offset** property and the **Inputbox** function

Create a procedure called **AddNames** which performs the following tasks:

- Prompts the user for their given name
- Enters the name in cell B2 of "Sheet1"
- Prompt the user for their family name
- Move one cell across
- Enters the family name in the active cell

Exercise 2: Using a **Do until loop**

Create a procedure that performs the following tasks:

- Start in cell A1 of "Sheet1" and move down column A one cell at a time **until** an empty cell is reached
- Then perform the data entry related to Exercise 1 (i.e. repeat the code as Exercise 1)

Exercise 3 (extension of Exercise 2):

Extend the procedure in **Exercise 2** so that once the given/family names have been entered, ask user whether s/he wishes to continue entering names (using the MsgBox function – details provided on page 4 and 5).

- If the user wishes to continue entering names, move down to the next row and repeat the process that asks the user for the given/family names. Otherwise exit the loop. (See examples on page 5 and 6 for prompting user to continue or not)

Part 2:

Exercise 1:

Download the workbook **PrintWorksheets.xlsm** from the unit site. Insert a new Public procedure called **PrintWorksheets** that prints all worksheets in the workbook (in print preview mode). Use the following pseudocode:

1. declare 1 integer variable called intCount
2. declare 1 Workbook object variable called wkbHours
3. declare 1 Worksheet object variable called wksCurrent
4. point wkbHours to the workbook PrintWorksheets.xls
5. initialise intCount to 1
6. present to the user each worksheet in the workbook in Print Preview mode.

Hint:

Find the total number of worksheets in the workbook by using the Worksheets Count property. Then use a *Do While...Loop* to cycle through each worksheet, presenting it in Print Preview mode. Increment intCount at each iteration of the Loop. Stop once intCount has exceeded the number of worksheets in the workbook.

Exercise 2:

Perform the same task using a *For Each...Next* structure. In this case you will require fewer variables.

Part 3: The **Select Case** statement and the **Do (While/Until) Loop**

Exercise 1:

Stan owns several delicatessens in suburban. His mark up on imported French cheeses varies depending on the suburb in which it is sold. The suburbs and mark up percentages is as follows:

Elsternwick	120%
Knox	90%
Glen Waverley	65%
Hawthorn	130%
Richmond	130%

The workbook CheeseSales.xlsm contains a worksheet (“Cheese Sales”) which lists the cost price of the sales of imported cheeses for the past 2 months. Write a public procedure which includes a **Select case statement which compares each of the entries in the Suburb column with the list above** and updates the Total with Markup column to reflect the markup percentages provided above. (Note you can use the Cells property of the Range object to do this).

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.
- After displaying the dialog box, the MsgBox function waits for the user to choose one of the command buttons.
- 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

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

Title is the text in the title bar

- 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
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
vbCritical	16	Display Critical Message icon
vbQuestion	32	Display Warning Query icon
vbExclamation	48	Display Warning Message icon
vbInformation	64	Display Information Message icon
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

Valid settings for the *buttons* argument

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

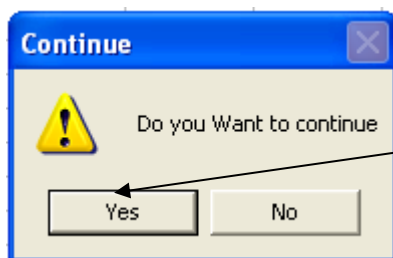
MsgBox function's buttons

e.g. 1

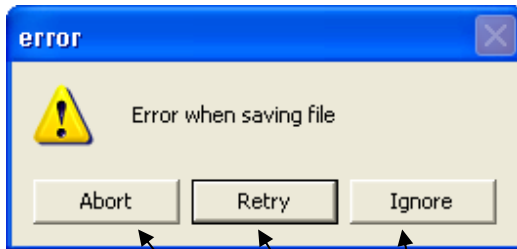
```
Public Sub MsgBoxEg1()
    Dim intResponse As Integer
    intResponse = MsgBox(Prompt:="Do you Want to continue", Buttons:=vbYesNo +
vbExclamation + vbDefaultButton1, Title:="Continue")
    If intResponse = vbYes Then
        MsgBox Prompt:="Yes button was selected"
    Else
        MsgBox Prompt:="No button was selected"
    End If
End Sub
```

Displays the Yes button and the No button

If the Yes button is pressed



Note that in this example the **MsgBox function** is being used to obtain information from the user – i.e. which button did they click. The **MsgBox statement** is used to provide information to the user.



e.g. 2

```
Public Sub MsgBoxEg2()
    Dim intButton As Integer
    intButton = MsgBox(Prompt:="Error when saving file",
        Buttons:=vbAbortRetryIgnore + vbExclamation + vbDefaultButton2, Title:="error")
    Select Case intButton
        Case vbAbort
            MsgBox Prompt:="vbAbort button was selected"
        Case vbRetry
            MsgBox Prompt:="[vbRetry button was selected]"
        Case vbIgnore
            MsgBox Prompt:="[vbIgnore button was selected]"
    End Select
End Sub
```

Extra Notes: The Cells property of the Range object (from VBA Help)

You can use **Cells(row, column)** where *row* is the row index and *column* is the column index, to return a single cell.

The following example sets the value of cell A1 to 24.

```
Application.Workbooks(1).Worksheets(1).Cells(1, 1).Value = 24
```

The following example sets the formula for cell A2.

```
Application.Workbooks(1).Worksheets(1).Cells(2, 1).Formula = "=sum(B1:B5)"
```

Although you can also use **Range("A1")** to return cell A1, there may be times when the **Cells** property is more convenient because you can use a variable for the row or column. NB after a worksheet has been activated, the **Cells** property can be used without an explicit sheet declaration (it returns a cell on the active sheet).

e.g. code extract that uses the Cells property of the Range object

```
Sub Add_Values()
    dim intNum1, intNum2 as Integer
    intNum1 = 234
    intNum2 = 45
    Application.Workbooks(1).Worksheets(1).Cells(1, 1) = intNum1 + intNum2
End Sub
```