*Automation, like we do when we record macro, can only go so far. There are always times we wish we could go further, but are limited by Excel’s pre-defined options. Excel allows us to go beyond that by providing a scripting language that we can use to expand on Excel’s capabilities.*

NOTE: This lab has 3 parts labeled A, B, and C. You are expected to complete all three. Only one of which will be marked.

***Note***: I have included a guide, the *VBAScriptingResource* that contains enough information and examples for our purposes here. The VBA Scripting language is quite diverse and can expand Excel’s capabilities exponentially. We only need to focus on a few simple tasks, and the guide is focused to help us. The VBA information starts around page 10.

Contents

[Part A (11 marks) 1](#_Toc417891666)

[Part B (10 marks) 5](#_Toc417891667)

[Part C (5 marks) 6](#_Toc417891668)

[Submission 7](#_Toc417891669)

# Part A (11 marks)

Scripting in Excel uses a language known as Visual Basic for Applications or VBA.

Let’s try an exercise that may help you adjust to the new language by drawing comparisons from work that you’ve already done.

We will use examples of the DOS scripting you did last semester. If you look below you’ll see code that you worked with previously. Please read over this information and review what the code does and what each command is supposed to accomplish.

1. The menu2.scr file contains the following code:

Today's Menu:

1) Display file attributes

2) Display directory structure as a tree

3) Display volume info

E) Exit

2. The menu2.bat file contains the following code:

@ECHO OFF

REM - Batch file to present a menu of options

SETLOCAL

:Menu

cls

TYPE menu2.scr

ECHO.

SET /P KEY=Option [1,2,3,E]:

IF "%KEY%" == "1" GOTO Attrib

IF "%KEY%" == "2" GOTO Tree

IF "%KEY%" == "3" GOTO Vol

IF /i "%KEY%" == "E" GOTO End

ECHO. Dave Kendell COMP1173 Business Applications Support W2014 Lab

ECHO \*\*\*Invalid option, enter 1, 2, 3, or E

pause

GOTO Menu

:Attrib

ECHO Inside Attrib

GOTO Menu

:Tree

ECHO Inside tree

GOTO Menu

:Vol

ECHO Inside Vol

GOTO Menu

:End

ENDLOCAL

Notes:

* SETLOCAL, ENDLOCAL – sets variables for the duration of the batch file
* SET /P – prompts and accepts response from user during batch file
* GOTO – forces flow to a specified label
* :*text* – a label
* ATTRIB – displays and/or sets file attributes: R-read only, H-hidden, S-system file, A-archive (backup bit)
* TREE – display a somewhat graphical representation of the directory structure
* VOL – displays the volume label information for a disk
* IF – allows you to compare two values and execute a command based on the result.
* We did some error handling as well and placed the menu text in its own file.

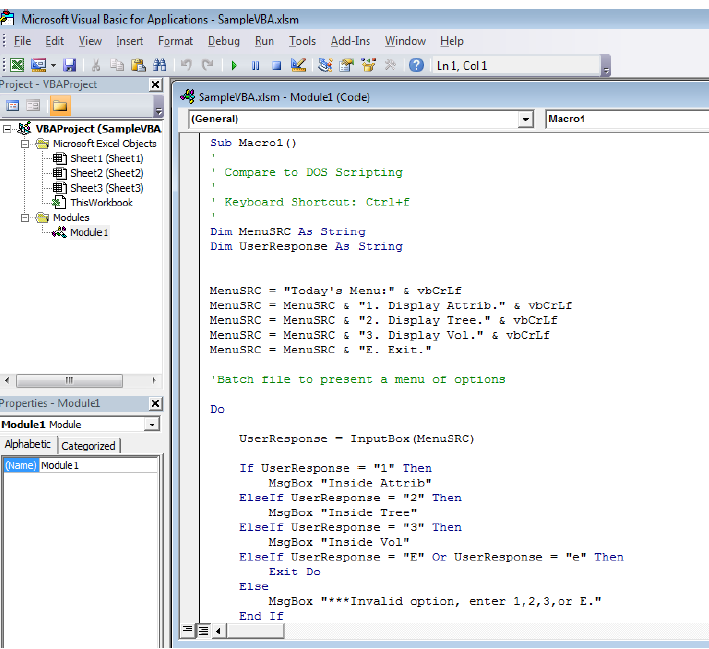
Accompanying these instructions are the two DOS script files. From the command line tool run menu2.bat and re-familiarize yourself with the functionality.

Also included is an Excel workbook titled *SampleVBA*. Open that workbook and run Macro1. You’ll see that the macro works much like the code from the .BAT file did when its run.

**The secret that you need to remember is: *once you’ve learned one scripting language you’ve learned them all. The only difference is the syntax or way things are written.***

1. Open the *SampleVBA* Excel file if it isn’t already.
2. *Allow* the content if Excel is giving you a warning.
3. Display the Developer’s tab if it’s not already showing.
4. On the Developers tab, in the Code pane, click the Visual Basic button. This opens the Visual Basic Editor.

You should see something that looks like the image below:



If you don’t see this:

1. On the left hand side, in the pane titled Project-VBAProject expand the *Modules* folder if necessary and then double click *Module 1*.

To continue:

1. Enlarge the window SampleVBA.xlsm – Module1 (Code) to fill the window.
2. Review what you see. This is the equivalent code to that found in the batch file.
3. Compare the two sets of code and try to record below the VBA equivalent of the related batch file components (give at least one example):

|  |  |  |
| --- | --- | --- |
| **Batch File command** | **Purpose** | **VBA Command** |
| Example: REM | Remark – used to provide a description of the code in the file. Not run during program execution. |  |
| KEY | Variable – used to temporarily store data being used during program execution. |  |
| ECHO | Used to display text to the screen. |  |
| IF | Used to control program flow based on the result of an expression. |  |
| SETLOCAL | Used to set variables for the duration of the program. |  |
| ENDLOCAL | Used to set variables for the duration of the program. |  |
| SET /P KEY=Option [1,2,3,E]: | Displays to the user possible options to choose and places the user’s choice in KEY. Waits for user input. |  |

There are a few commands in the VBA program that are not used in the batch file. Can you guess at what these commands do?

|  |  |
| --- | --- |
| VBA Command | Use |
| Do |  |
| Loop Until |  |
| Exit Do |  |
| Dim |  |
| Or |  |

# Part B (10 marks)

Always start your scripting by recording a macro, if possible. Macros are simply VBA code that has been created by Excel and there’s no point in redesigning the wheel if we don’t have to. Though, when Excel records the code it can look a little scary, but remember, if you do record a macro first all you’re doing is adding to it, not trying to rewrite it.

Let’s go to the experts for our first scripting example.

1. Open a new workbook.
2. Save the file to a location of your choice, and name it as follows: *your\_full\_Name\_Lab1*.xlsx (example DaveKendellLab1.xlsx).
3. Visit the following website (it says 2010, but the skills are transferable):

<https://msdn.microsoft.com/en-us/library/ee814737.aspx>

1. Read through the material.
2. Do the parts labelled:
   1. On Sheet1 do *Macros and the Visual Basic Editor*
   2. On Sheet2 do *A Real World Example* (until you get to the *More Things You Can Do* part).
3. Save your work as a ***Macro-enabled*** workbook.

# Part C (5 marks)

This portion of the lab is a simple customer record sheet. Our purpose is to set it up so that the person using it can clear the sheet at the click of a button. We will also ask the user if they are sure they want to do it.

On Sheet3:

1. Add the following titles to the range A1:A8

|  |  |
| --- | --- |
| A1 | Customer Info (merged across cells A1:A2) |
| A2 | First |
| A3 | Last |
| A4 | Street |
| A5 | City |
| A6 | Prov |
| A7 | Postal Code |
| A8 | Country |

1. Record a macro called *Empty\_Cells* that will delete whatever is in cells B2:B8.
2. Add a button to the sheet that will execute this macro when clicked.
3. Click the Macros button.
4. Select the macro you just created.
5. Click Edit.

Now we need three things:

* A way to ask the user if they are sure they want to reset the page.
* A variable to hold the user's response.
* A decision structure to test what the user entered.

1. To create the variable we'll add this line of code above the code written by the macro, just under the line starting with Sub:

***Dim answer as VbMsgBoxResult***

*We use VbMsgBoxResult as the data type here. This is a special data type that can hold the VBA types returned by pressing a button on a dialog box.*

1. Under that line, we'll ask the user if they want to by writing this code:

***answer = MsgBox("Are you sure?", vbYesNo,"Checking...")***

*With the Msgbox command we can control the buttons that appear on the screen. In this case the vbYesNo command will show only a yes and no button. When pressed they will return the VBA reserved values vbYes and vbNo respectively.*

1. Finally we need to test to see what they decided. Right above the code created by the macro write:

***If answer = vbYes Then***

1. And to finish off right after the code created by the macro but above the End Sub line write:

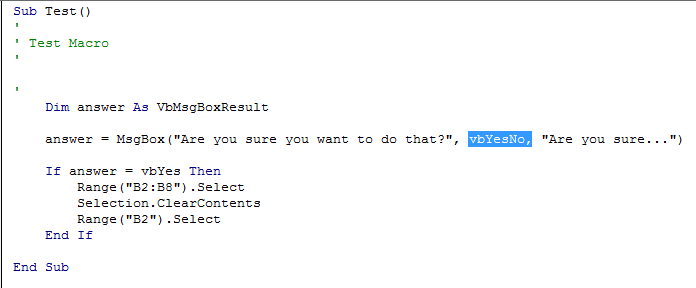
***End if***

*You may be wondering why we're only looking at vbYes when they could click either yes or no.*

*Well, in this case, if they enter ‘no’ we don't want to do anything. The lines of code contained between the if…then and end if line are only executed if the condition between the if…then (in this case answer = vbYes) is true.*

1. Test your code by entering values in to the cells on the worksheet and then running your macro.

***See the image below for a look at the completed code.***

******

1. ***Save your work remembering to choose a macro-enabled book rather than a regular book.***

# Submission

You are required to submit the following files:

* This word document.
* The new, macro-enabled workbook: *your\_full\_Name\_Lab1*.*xlsx*.

Post these files to the link on Blackboard.

See Blackboard for the due date.