

Getting Started with Excel and VBA in the Laboratory

William Neil and Martin Echols

Course Examples

- During the course we will be working on a series of programming examples.
- Each example is a continuation of the previous example.
- Each person should have an *Examples* folder on their C: drive and a *Completed Examples* folder.
- If you are not able to complete one example, copy its completed version from the Completed Folder to the Examples folder and use to continue with the next example.

What We'll Learn ...

- How to load data into Excel
- How to process data in Excel with the Visual Basic for Applications (VBA) programming language
- How to display data graphically with Excel
- How to modify Excel using VBA
- ... and much more

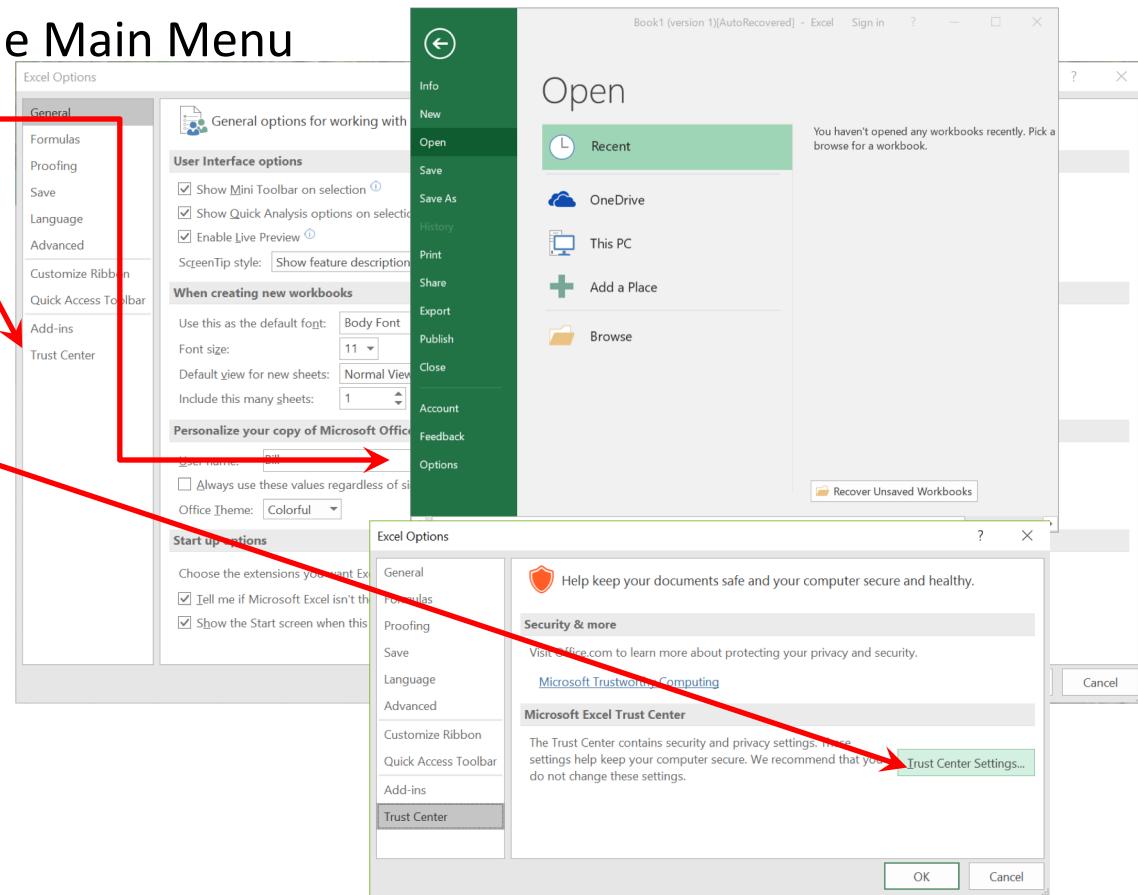
The Trust Center

- Excel Workbooks are vulnerable to malicious code that can be hidden within macros.
- Microsoft has created the Trust Center to give you control over which Workbooks can run macros on your computer.
- By default, ALL macros are disabled.
- We will use the Trust Center to enable macros in the Workbooks in our Examples folder.

The Trust Center (cont.)

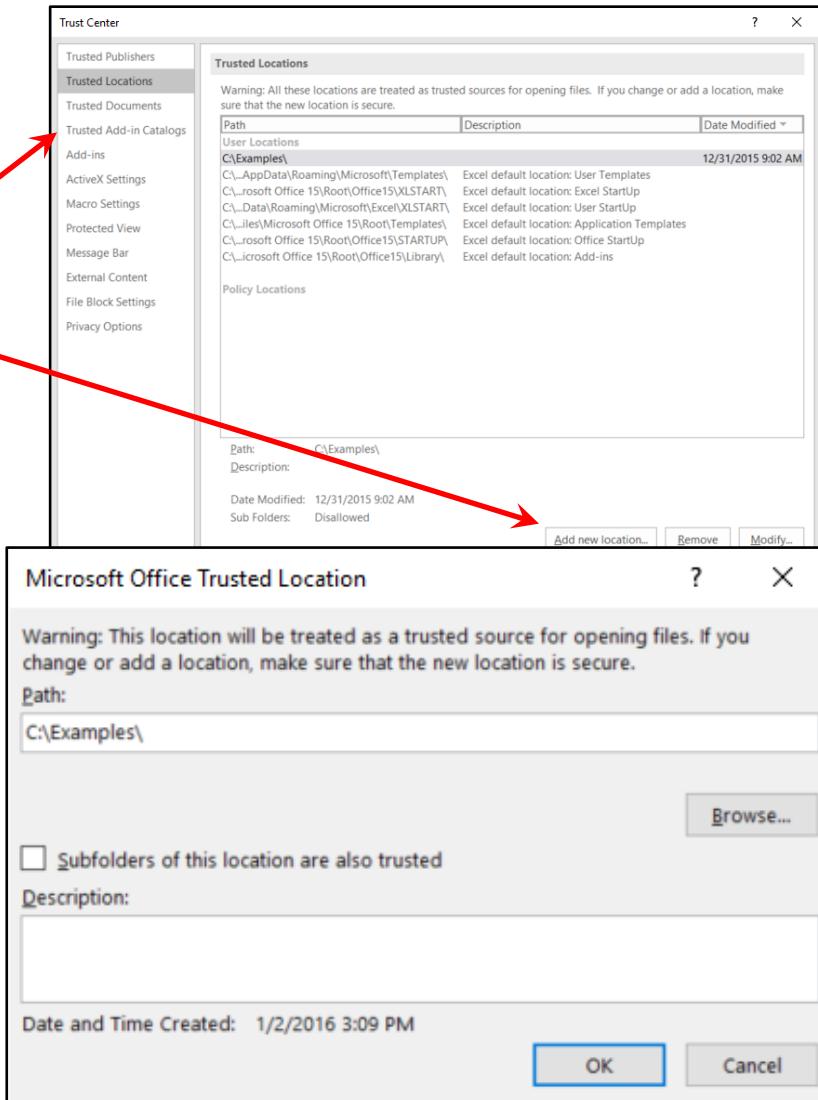
- You can access the Trust Center settings:

- Click the File Tab on the Main Menu
- Select Options
- Select [Trust Center]
- Select [Trust Center Settings]



Trust Center (*cont.*)

- Select [Trusted Locations]
- Click Add New Location
- Enter the full path to your folder
- Select [OK]

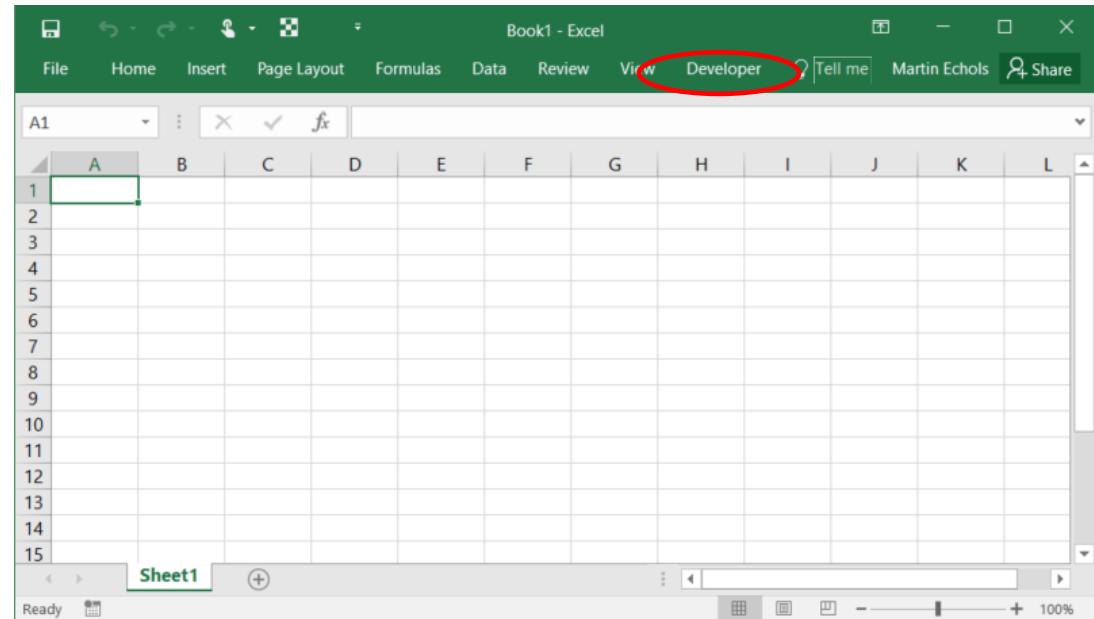


A New File Type

- In addition to the Trust Center concept, another change is the file type extension for Workbooks containing macros.
- When storing these Workbooks you need to select the file type “Excel Macro-Enabled Workbook with the extension “xlsm”.

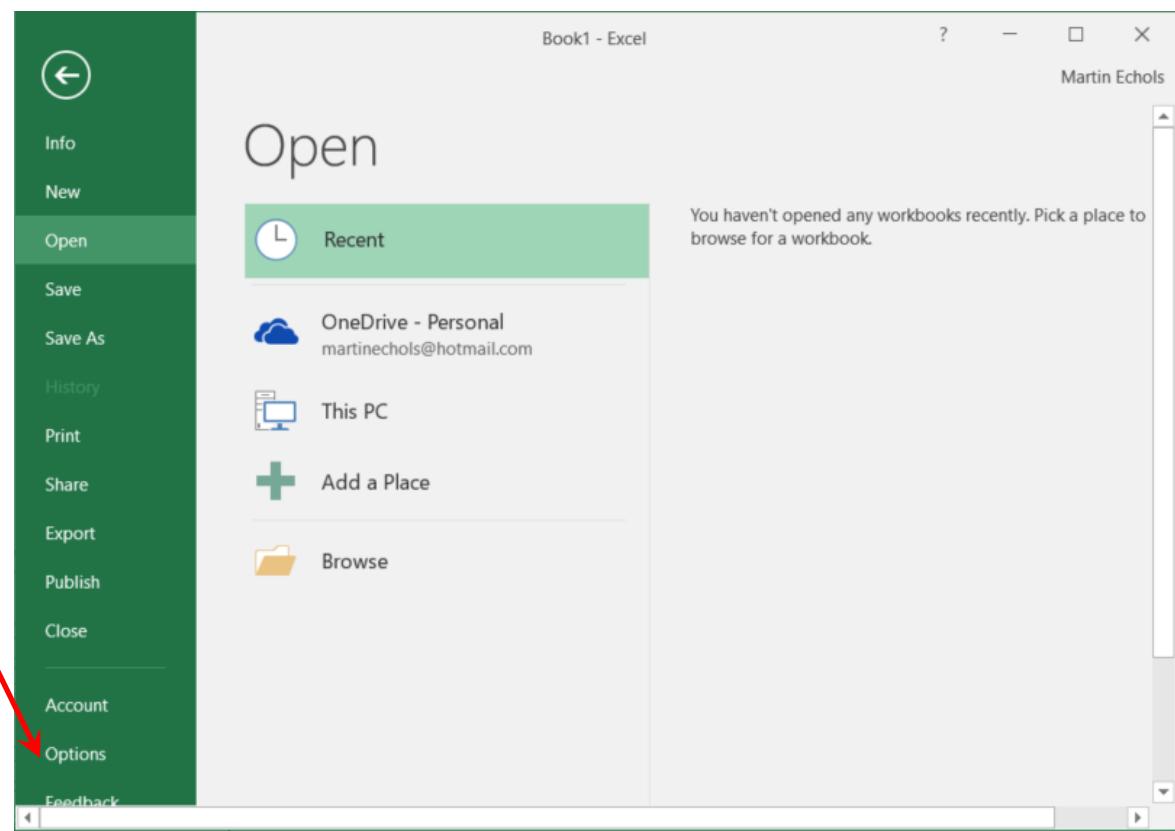
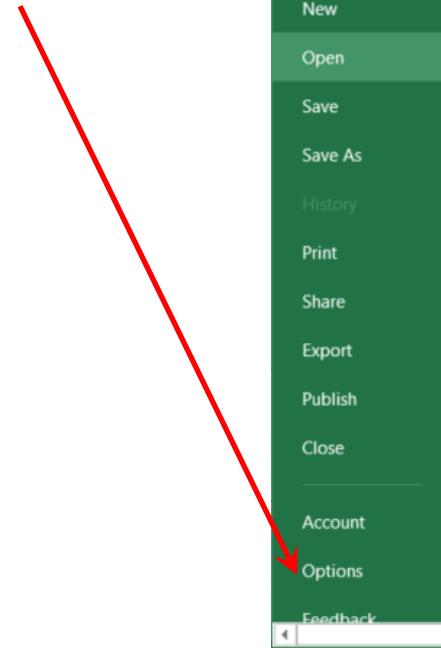
The Developer Tab

- The Developer Tab provides a collection of useful functions for developing applications in Excel.
- The Developer Tab is visible by default, but may be turned on by the following steps



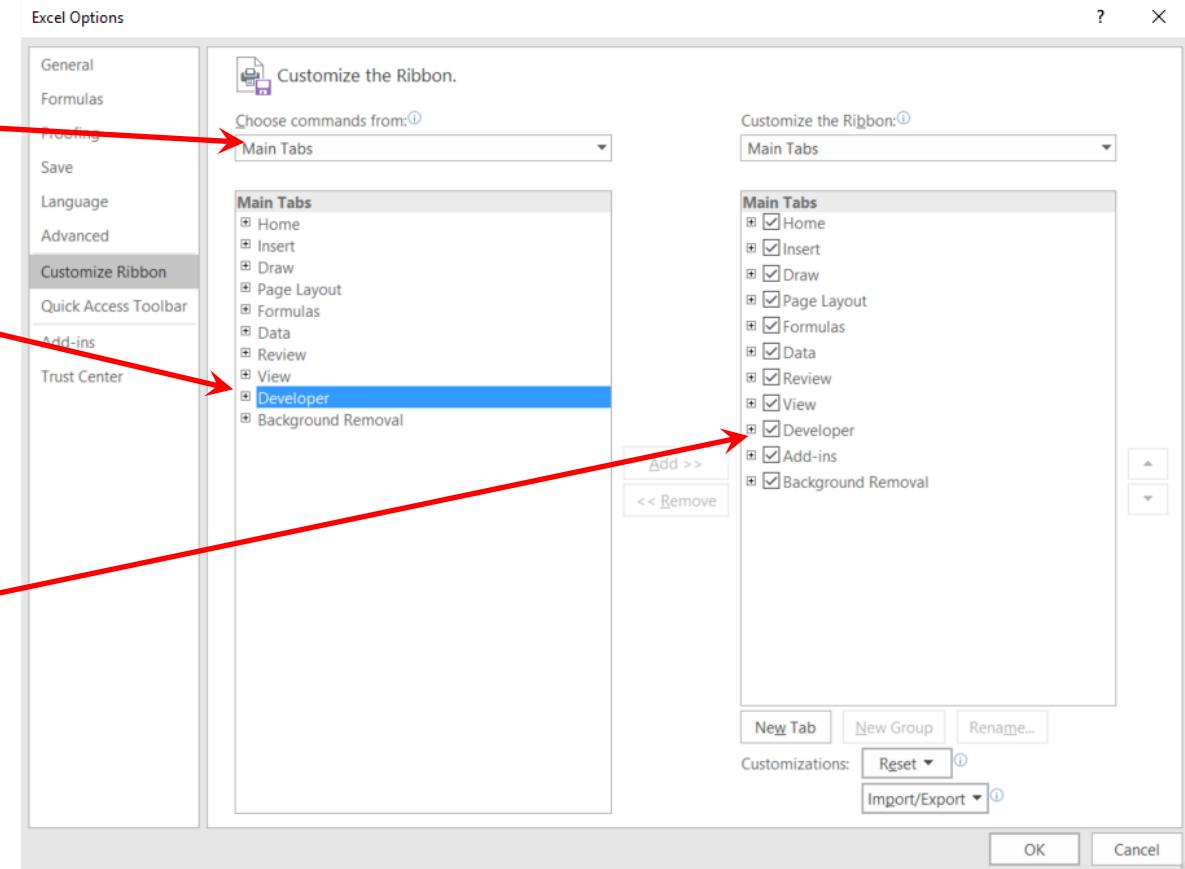
Adding the Developer Tab

- To add the Developer Tab:
 - Click ‘File’ on main menu.
 - Then ‘Options’



Adding the Developer Tab

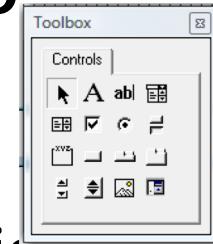
- Select ‘Customize Ribbon’
- In the ‘Choose Commands From’ box select ‘Main Tabs’
- In the ‘Main Tabs’ selection box select ‘Developer’
- Click **Add >>** to add ‘Developer’ to the ‘Customize Ribbon’ list
- Finally, select the checkbox for the Developer in the ‘Customize Ribbon’ list.
- Click OK



What is Visual Basic for Applications

- Visual
 - **Visual** method of creating GUI
- Basic
 - **Beginners All-Purpose Symbolic Instruction Code language**

(VBA)?



Don't need to write code to make buttons, check boxes, etc.

```
Dim I as Integer  
Dim J as Integer  
Dim K as Integer  
I=3  
J=2  
K= I*J  
PRINT "K = " & K
```

- Applications
 - Used by all MS Office **Applications** and hundreds of 3rd-party **Applications**
 - CAD/CAM (AutoCAD), SolidWorks
 - Manufacturing automation (Rockwell Software)
 - ...

What can you do with VBA?

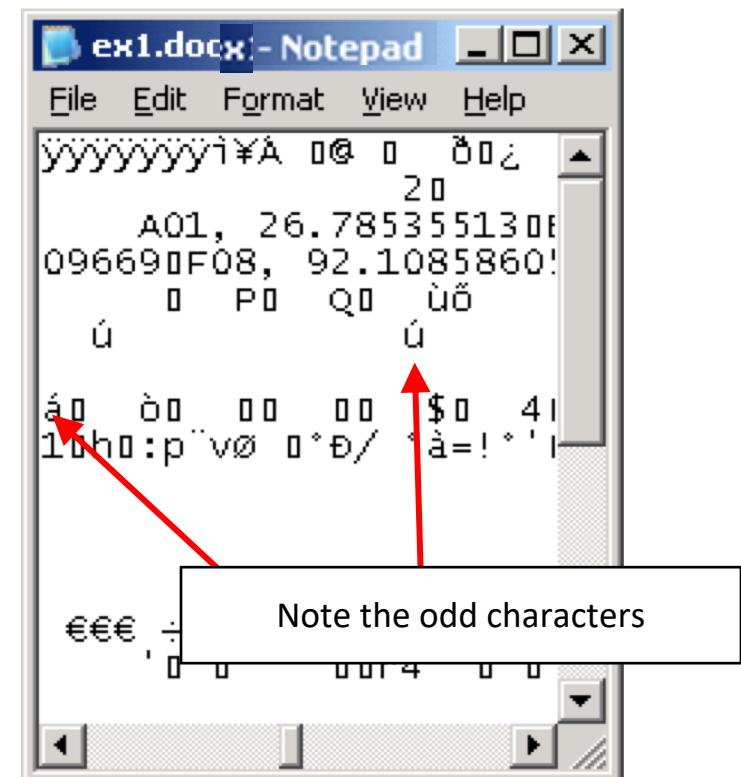
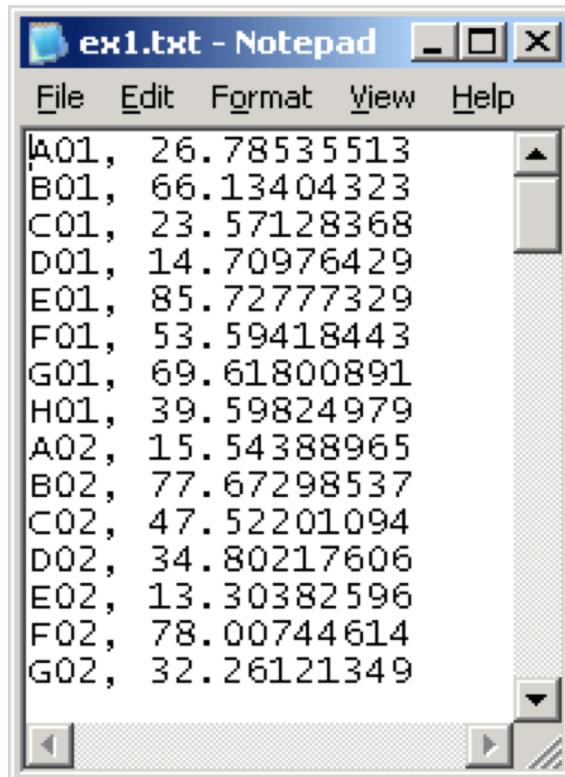
- You can do anything with VBA that you can do interactively through the Excel user interface
 - You can even generate VBA programs automatically
- Modify the interface of Excel to suit your needs
 - Add custom dialogs and message boxes
 - Add custom menu items
 - Add custom toolbars
 - ...
- Customize the functionality of Excel
 - Seamlessly integrate new routines into a familiar interface
 - Limit or prevent access to certain built-in routines
 - ...

Reading Files Into Excel

- Purpose
 - Data processing, storage, display
- Source
 - Data generated from instruments (HPLC, plate reader, balance)
 - Data generated from commercial or in-house applications
 - Many others ...
- Import methods
 - Copy/Paste from another application
 - Text Import Wizard
 - Read directly from a file
- Formats
 - Binary data versus plain text data

Text Versus Binary Files

- Text file opened in NotePad
 - Human readable text, some control characters (tabs, CR, LF)
 - Common format for storing data
 - A Word file opened in NotePad
 - Contains information in non-text formats
 - Often the formats are proprietary



ASCII Encoding

ASCII: American Standard Code for Information Interchange

A table relating bytes 0-127 with single printable and non-printable characters

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	<NUL>	16	<DLE>	32	space	48	0	64	@	80	P	96	`	112	p
1	<SOH>	17	<DC1>	33	!	49	1	65	A	81	Q	97	a	113	q
2	<STX>	18	<DC2>	34	"	50	2	66	B	82	R	98	b	114	r
3	<ETX>	19	<DC3>	35	#	51	3	67	C	83	S	99	c	115	s
4	<EOT>	20	<DC4>	36	\$	52	4	68	D	84	T	100	d	116	t
5	<ENQ>	21	<NAK>	37	%	53	5	69	E	85	U	101	e	117	u
6	<ACK>	22	<SYN>	38	&	54	6	70	F	86	V	102	f	118	v
7	<BEL>	23	<ETB>	39	'	55	7	71	G	87	W	103	g	119	w
8	<BS>	24	<CAN>	40	(56	8	72	H	88	X	104	h	120	x
9	<HT>	25		41)	57	9	73	I	89	Y	105	i	121	y
10	<LF>	26	<SUB>	42	*	58	:	74	J	90	Z	106	j	122	z
11	<VT>	27	<ESC>	43	+	59	;	75	K	91	[107	k	123	{
12	<FF>	28	<FS>	44	,	60	<	76	L	92	\	108	l	124	
13	<CR>	29	<GS>	45	-	61	=	77	M	93]	109	m	125	}
14	<SO>	30	<RS>	46	.	62	>	78	N	94	^	110	n	126	~
15	<SI>	31	<US>	47	/	63	?	79	O	95	_	111	o	127	

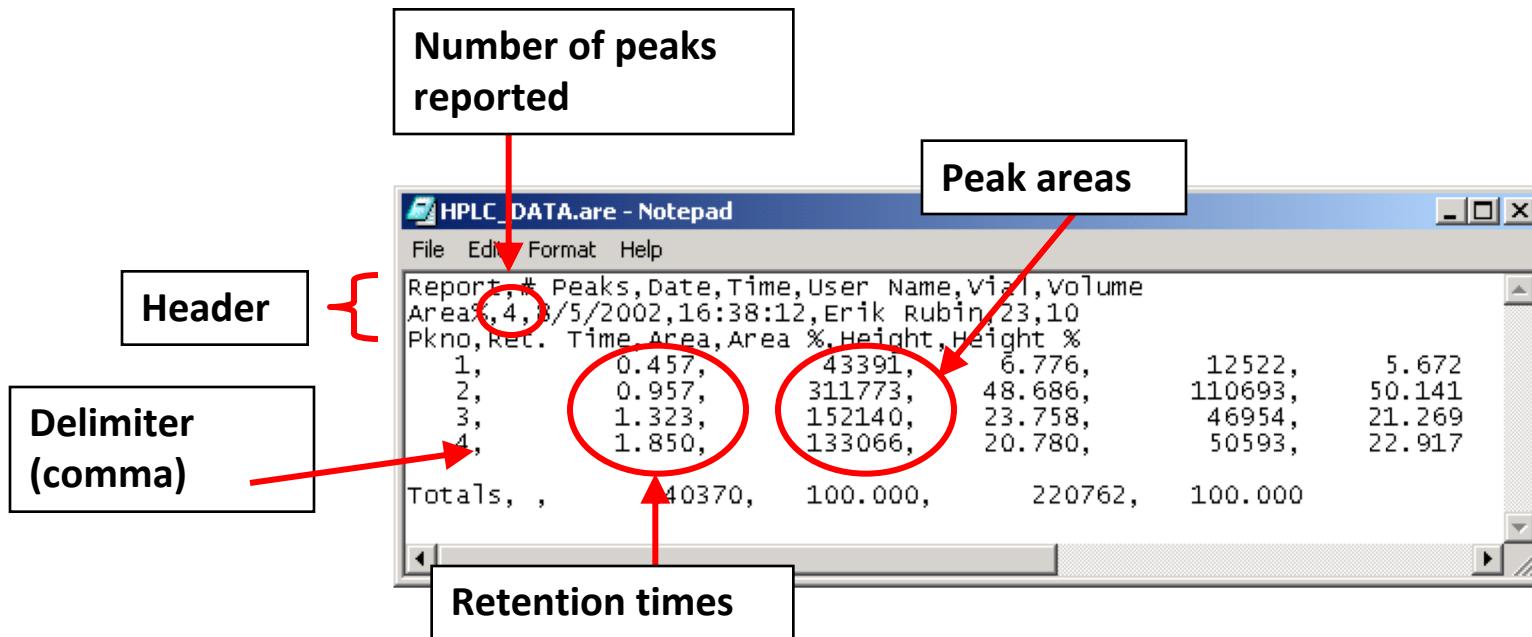
Text File Format Terminology

- Header
 - Section at top of file that applies to all data in the file
 - Usually contains one or more lines of information
- Tabular Data
 - Section(s) in a file containing data organized in rectangular arrays
 - Multiple rows and columns
- Delimiters
 - Punctuation character(s) that mark the beginning and/or end of data items.
 - Some common delimiters

Comma (,)	<Tab>*
Colon (:)	Semicolon (;)
Space ()	

* non-printable character

Example HPLC Text File



Example1: The Text Import Wizard

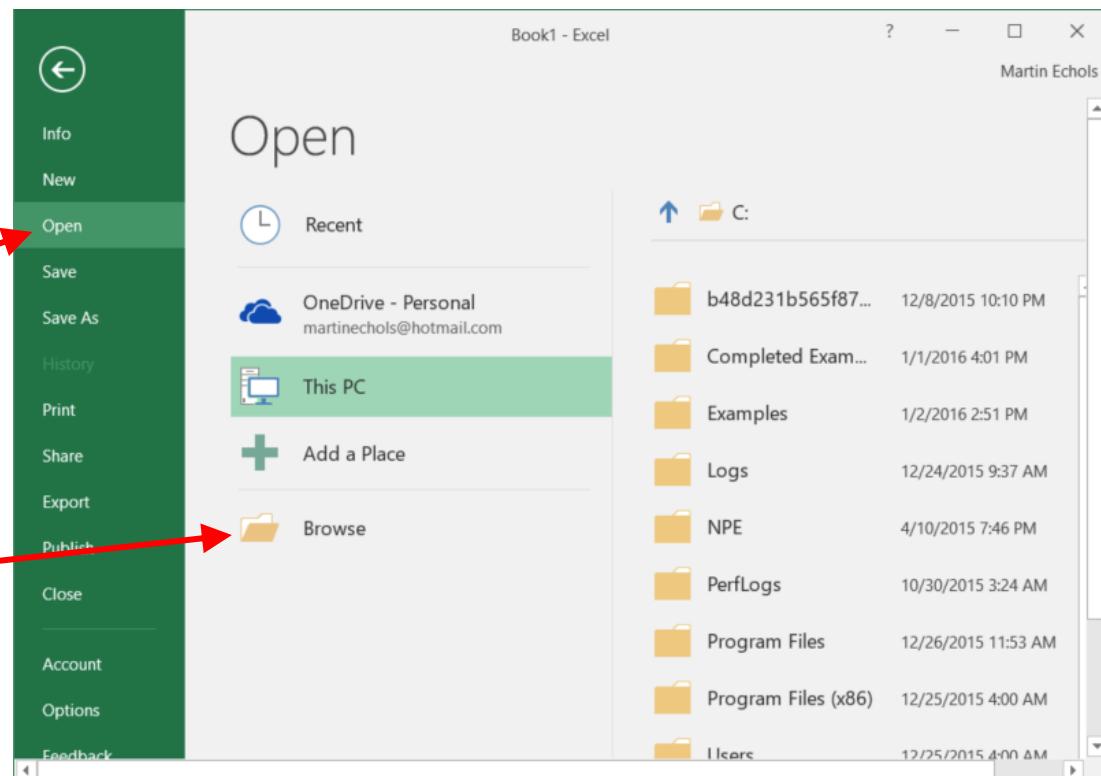
- Assists with transferring the contents of a text file into the cells of a Worksheet
- Accessed when attempting to open text files with the “File | Open” menu option
- The example file ex1.txt has two columns of data separated by a comma: well labels and measurements

A01,	26.78535513
B01,	66.13404323
C01,	23.57128368
D01,	14.70976429
E01,	85.72777329
F01,	53.59418443
G01,	69.61800891
H01,	39.59824979
A02,	15.54388965
B02,	77.67298537
C02,	47.52201094
D02,	34.80217606
E02,	13.30382596
F02,	78.00744614
G02,	32.26121349
...	

ex1.txt

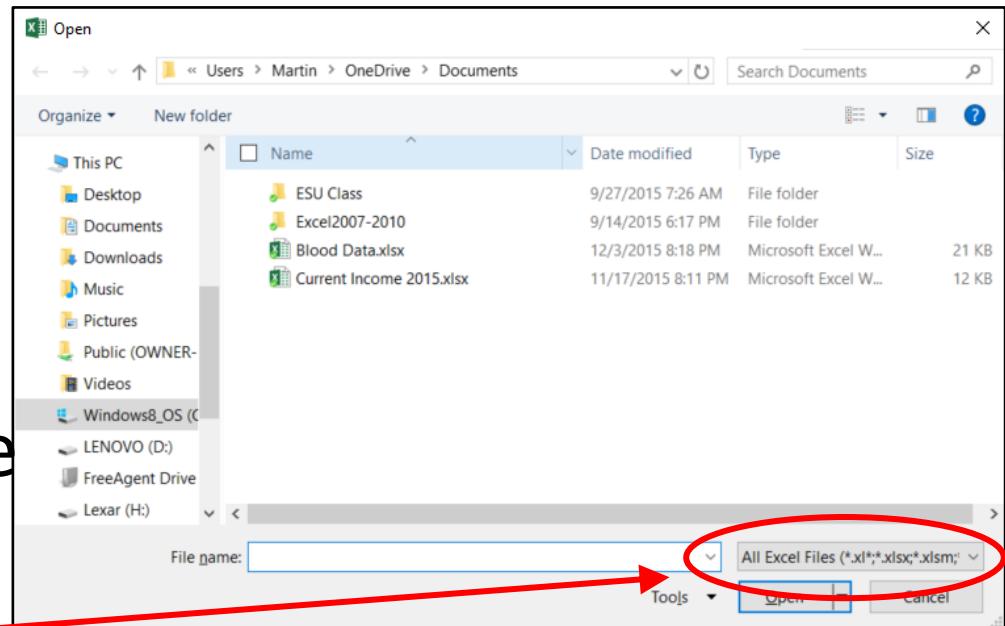
Example 1: Import a Text File

- Open a new Workbook and:
 - Click the File tab on the main menu
 - Select ‘Open’
 - Start navigating to the Examples folder by selecting ‘Browse’.

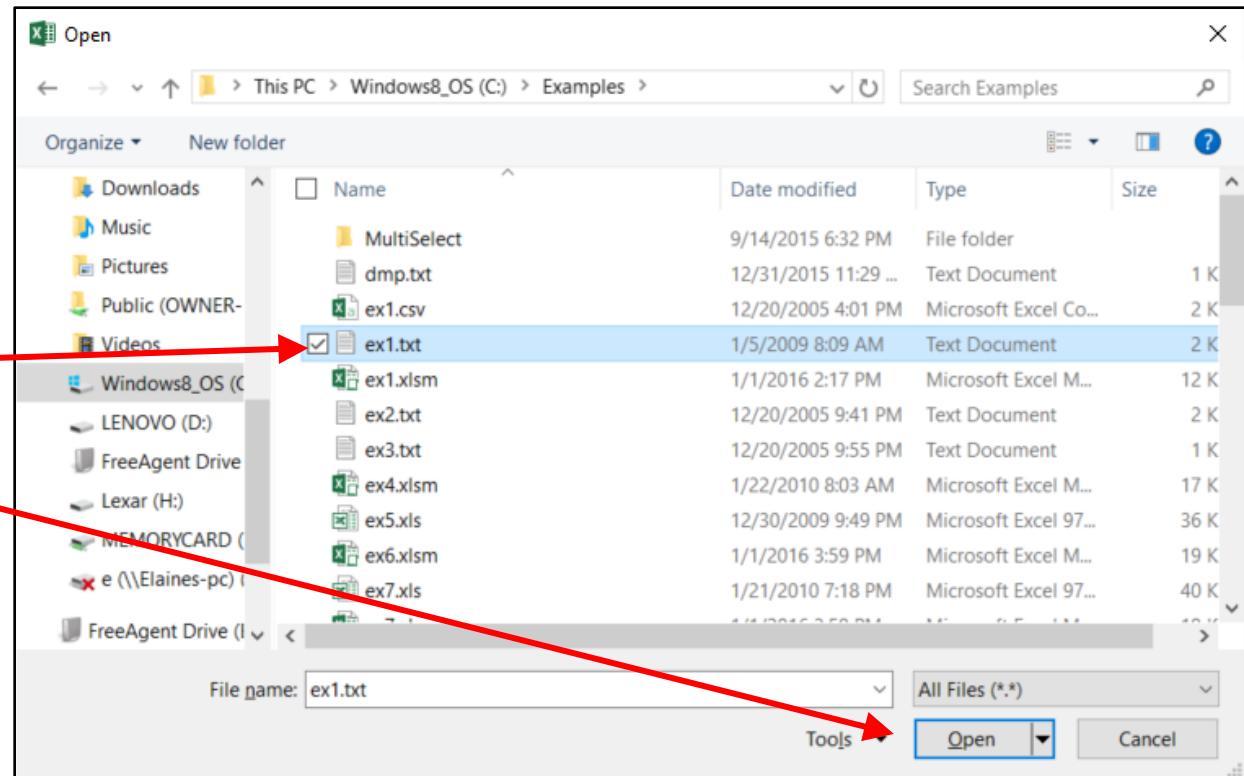


Example 1: Import a Text File

- Navigate to the Examples folder
- Remember to set the file designation to 'All Files'.

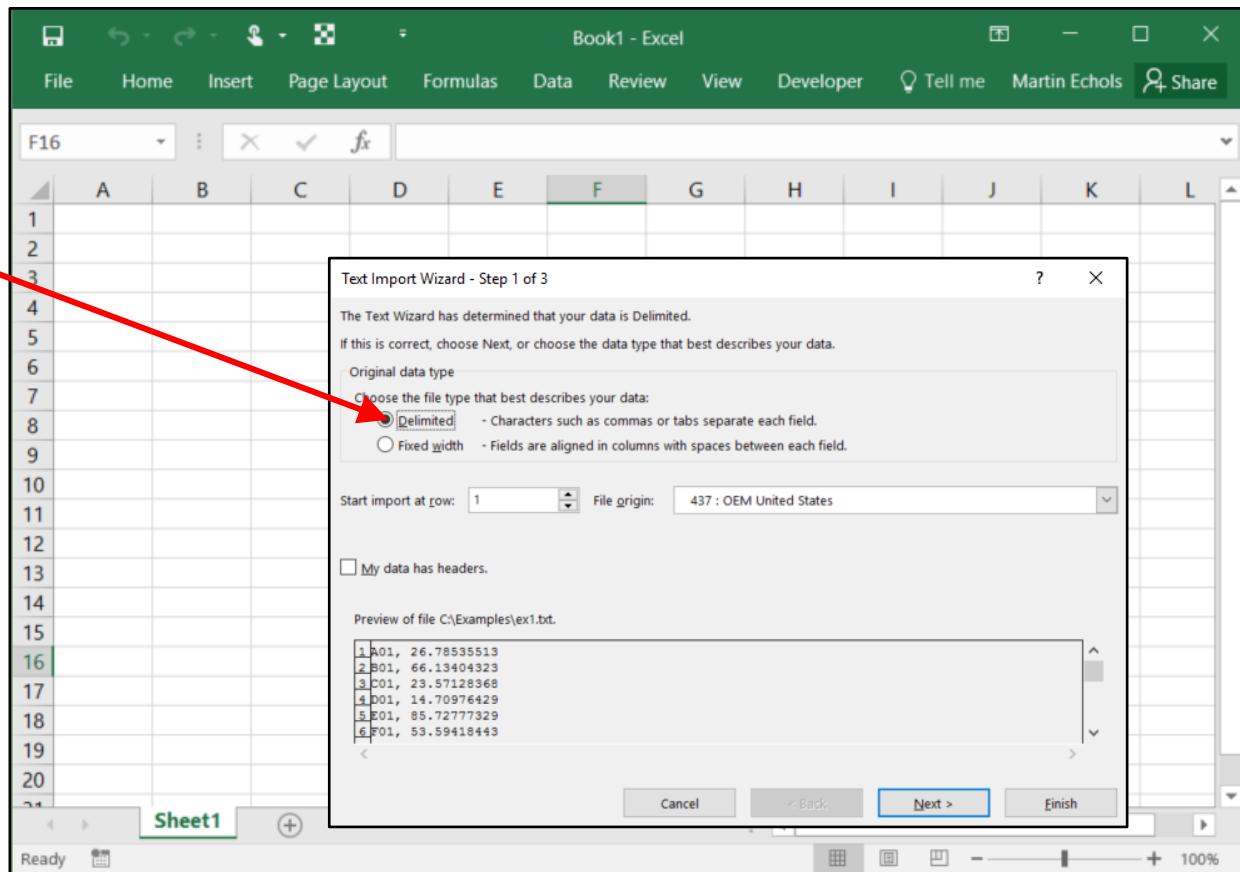


Example 1: Import a Text File (cont.)



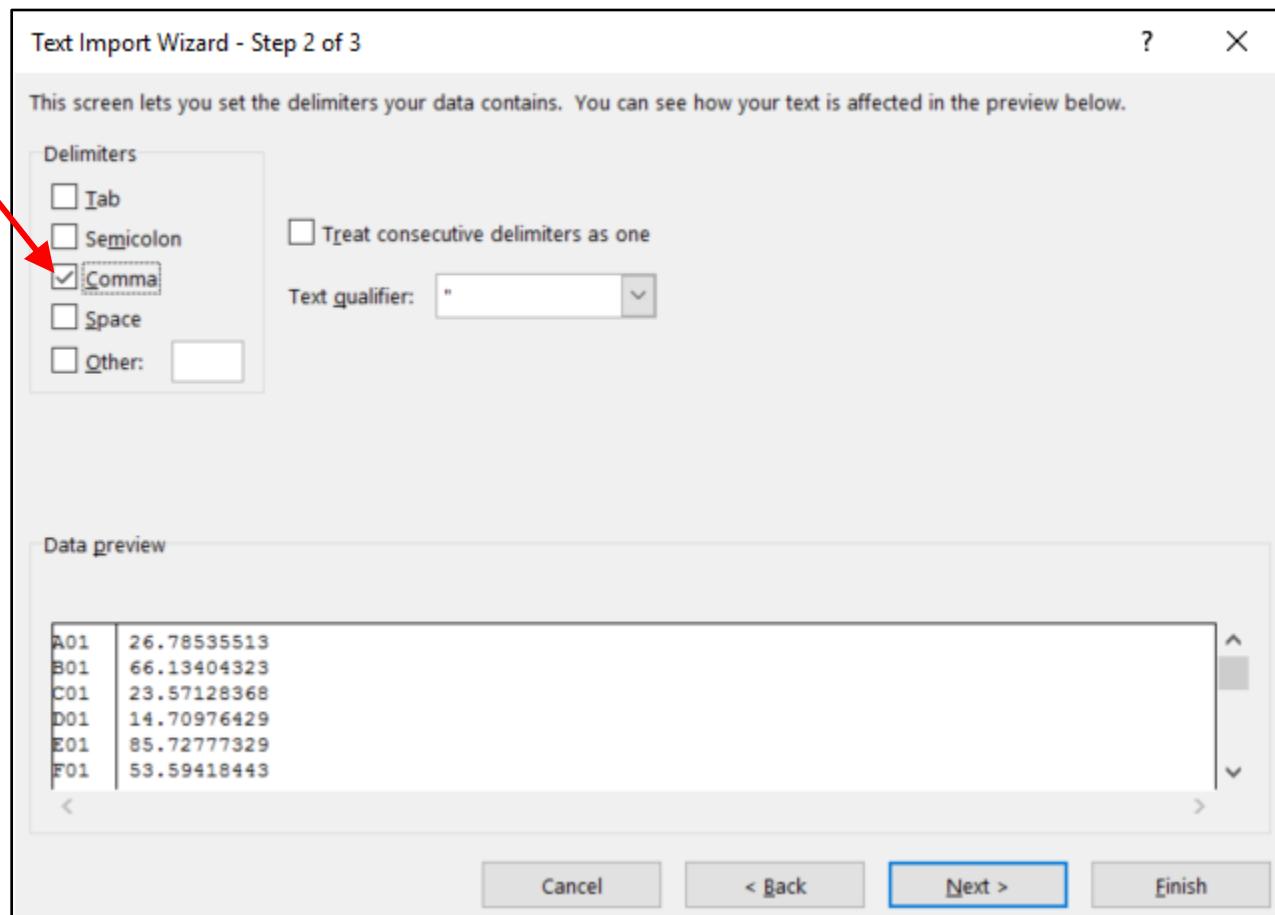
Example 1: Step 1 of 3

- Choose one of the two types of text files: delimited columns or columns of fixed width
- Click [Next >]



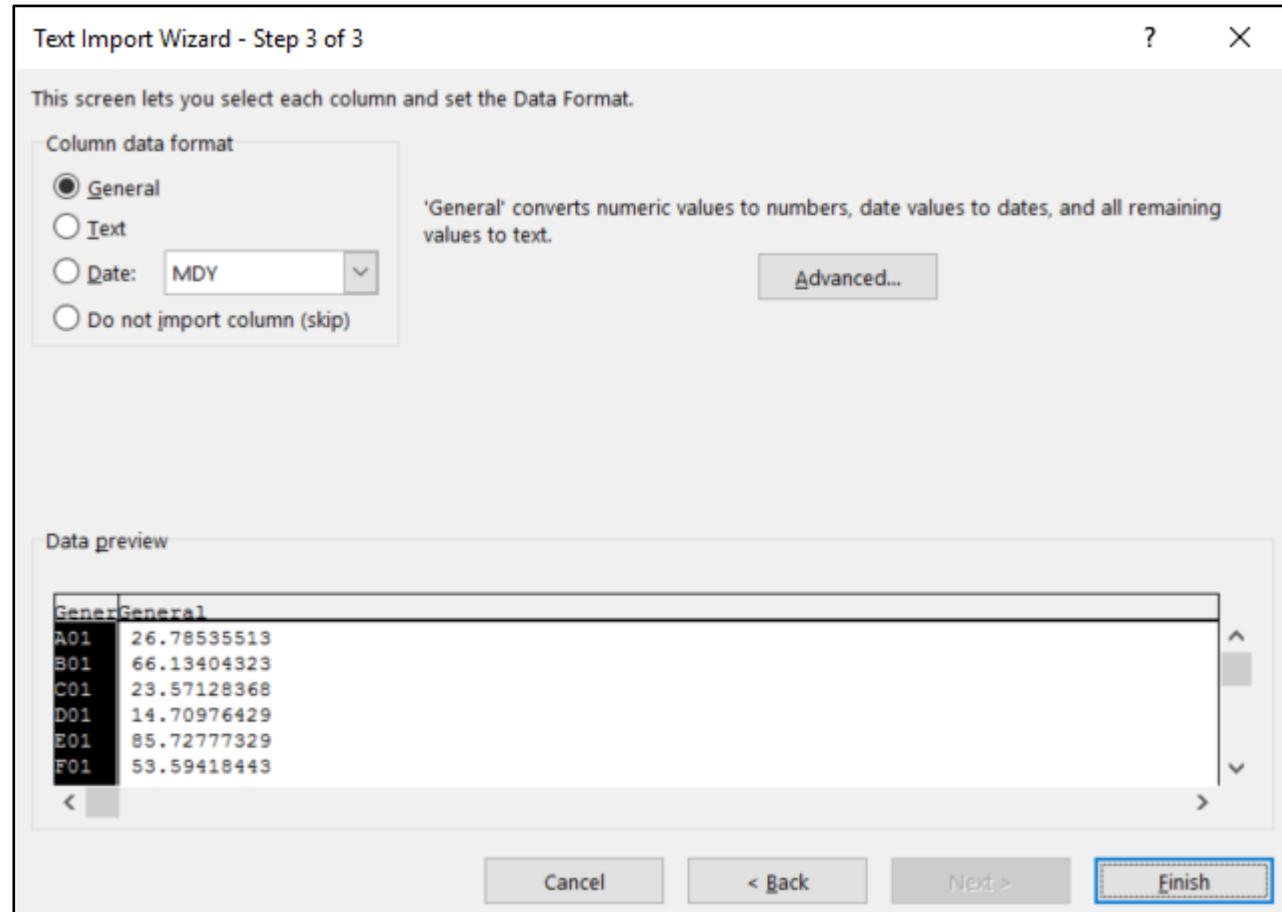
Example 1: Step 2 of 3

- Select the character(s) that separate data in columns of the text file
- A file can have multiple delimiter characters
- Click [Next >]



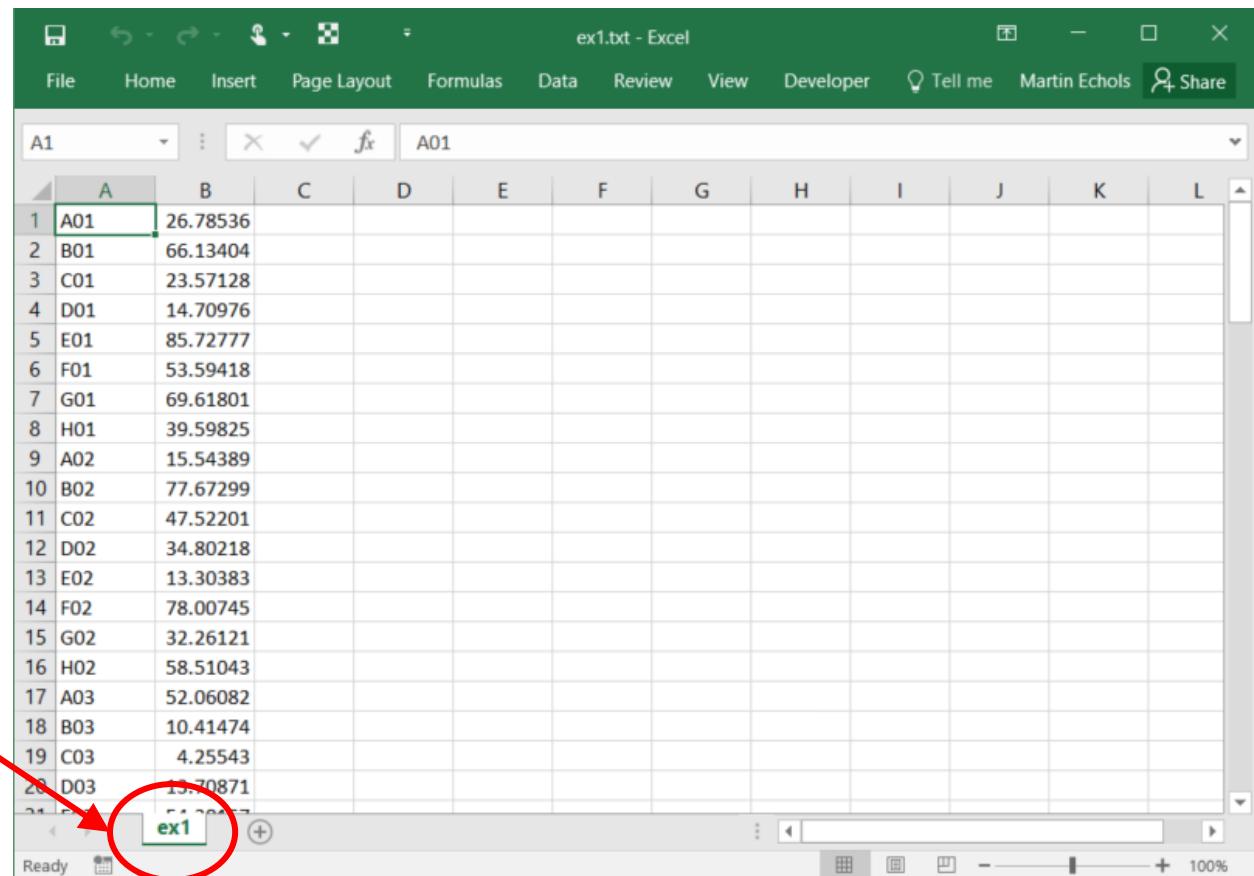
Example 1: Step 3 of 3

- Select a format for the data in each column
- Using General is often the best option
- Click [Finish]



Example 1: The Imported Data

- The two columns of data are loaded into the two columns of the Worksheet
- Also note that the name of the Worksheet is changed to the name of the data file

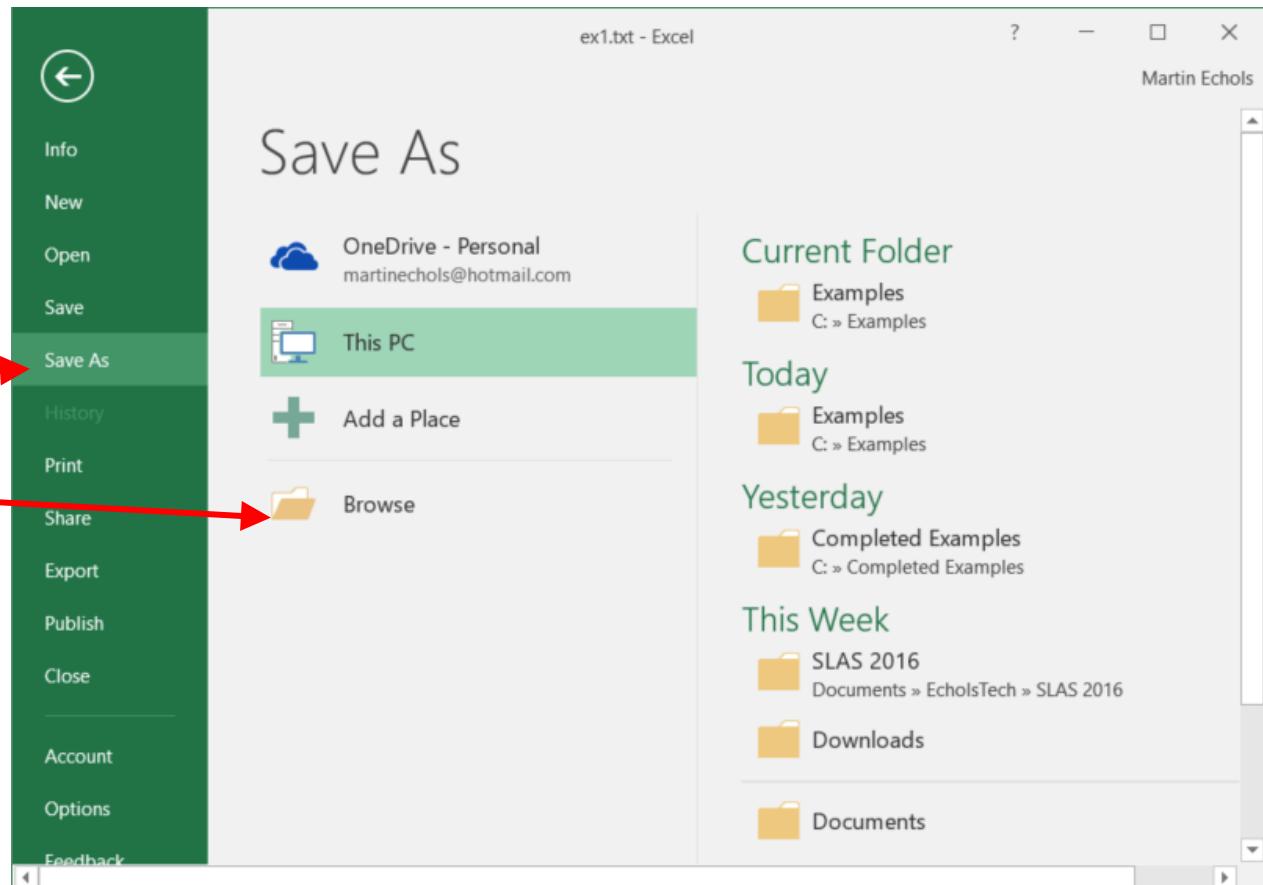


A screenshot of Microsoft Excel showing a single worksheet named "ex1". The worksheet contains data in columns A and B. Column A contains labels such as A01, B01, C01, D01, E01, F01, G01, H01, A02, B02, C02, D02, E02, F02, G02, H02, A03, B03, C03, D03, and E03. Column B contains numerical values corresponding to each entry in column A. The worksheet tab at the bottom is also labeled "ex1".

	A	B
1	A01	26.78536
2	B01	66.13404
3	C01	23.57128
4	D01	14.70976
5	E01	85.72777
6	F01	53.59418
7	G01	69.61801
8	H01	39.59825
9	A02	15.54389
10	B02	77.67299
11	C02	47.52201
12	D02	34.80218
13	E02	13.30383
14	F02	78.00745
15	G02	32.26121
16	H02	58.51043
17	A03	52.06082
18	B03	10.41474
19	C03	4.25543
20	D03	15.70871
21	E03	54.22047

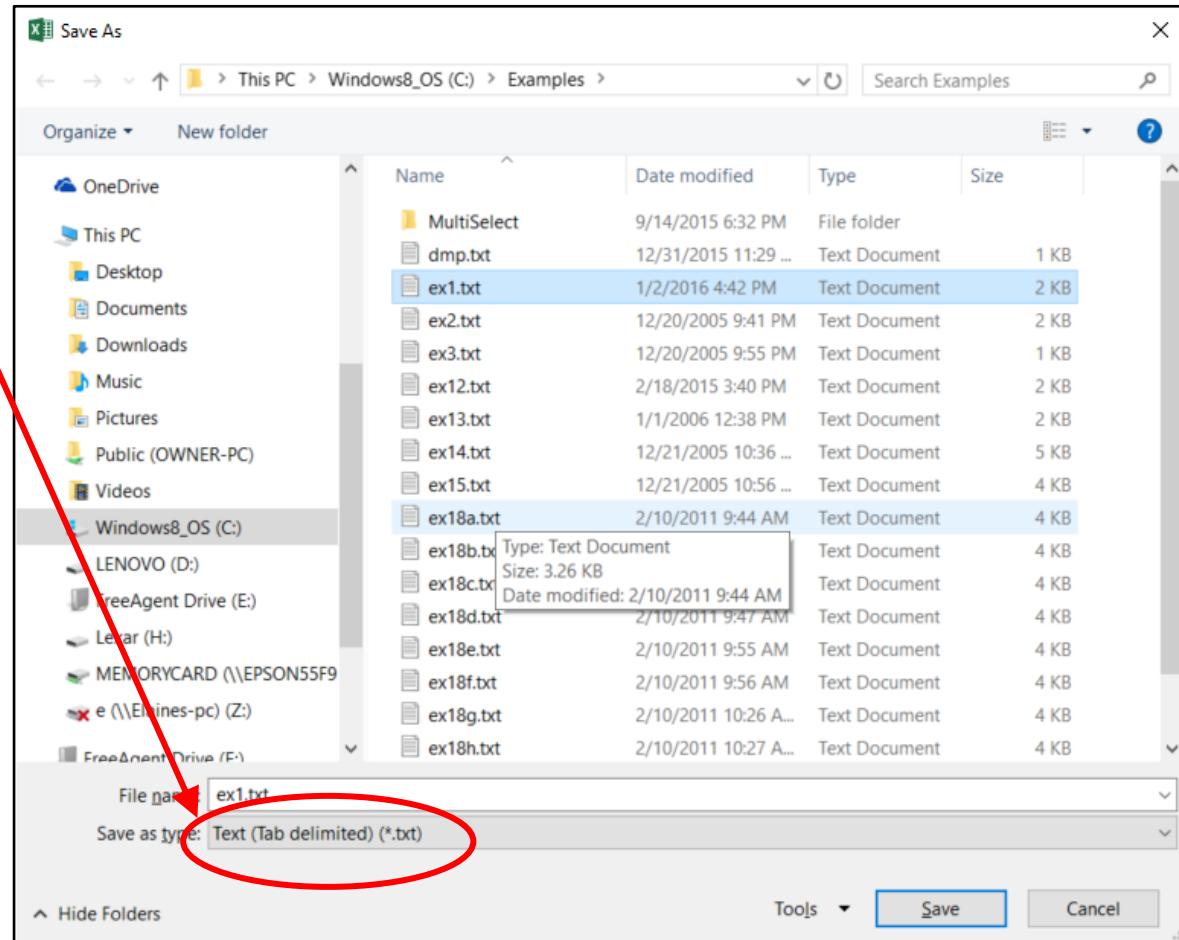
Example 1: Save the Imported File

- Select the “File from the main menu.
- Select Save As” menu option
- Select Browse



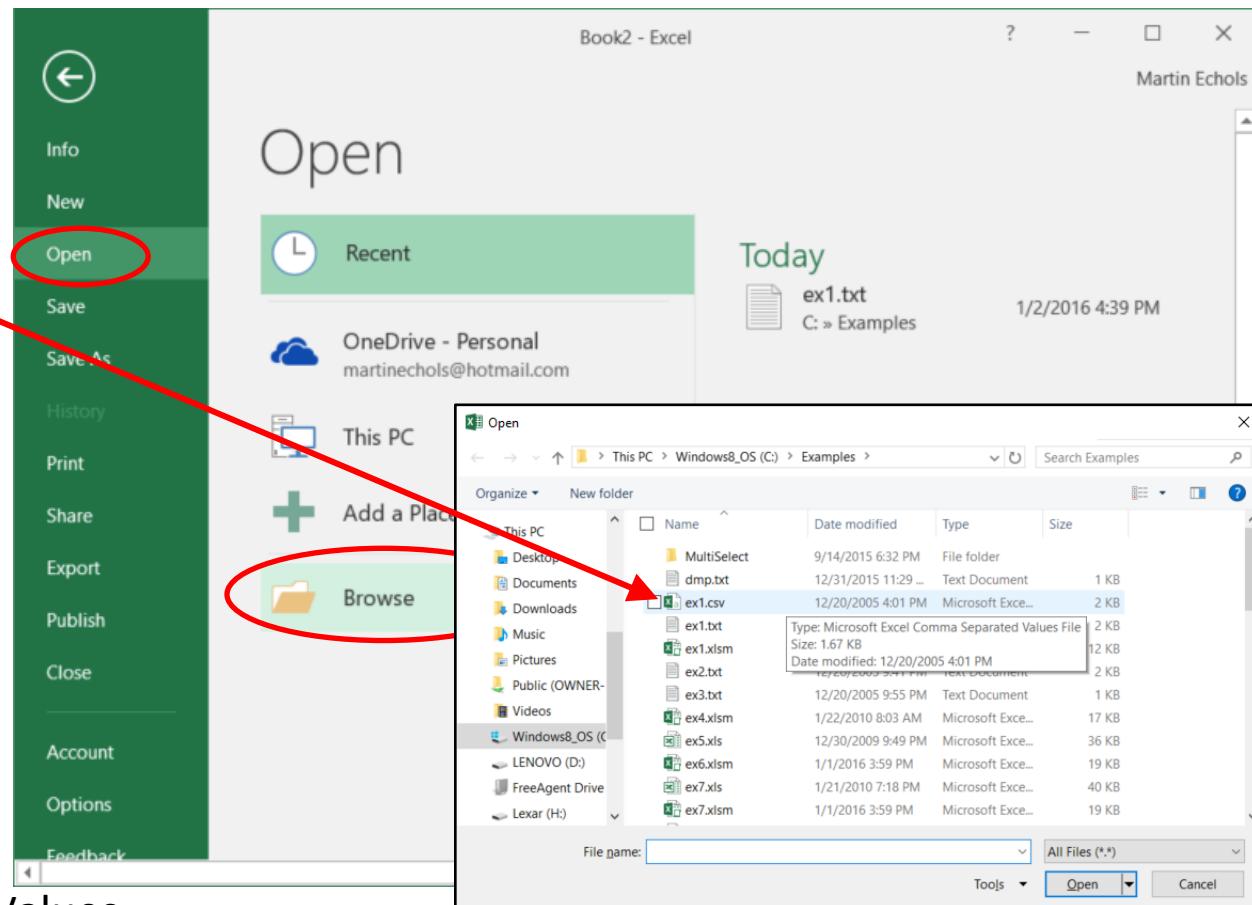
Example 1: Save the Imported File

- Note that the file type is set to “Text (Tab delimited)”
- Change file type to “Microsoft Excel Workbook (*.xlsx)”
- Click [Save]
- File is saved in Excel format, not Text format



Example 1: Import a CSV* File

- Simply changing the file extension to “csv” informs Excel about the file format
- Add a new Workbook
- Open the file “ex1.csv”
- This file is identical to ex1.txt, with a different file extension
- The text is imported directly. The Text Import Wizard is not displayed



*CSV = Comma Separated Values

Example 2: Import a Tab-delimited File

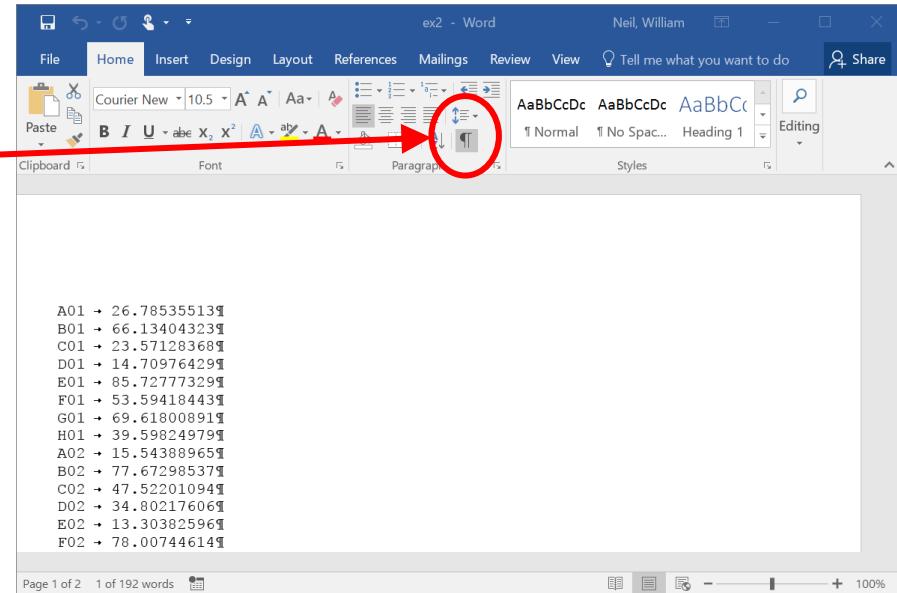
- The delimiter in ex2.txt is the tab character
- Tab does not have a “printable” character
- How do we know tab and similar formatting marks are present in a file?

A01	26.78535513
B01	66.13404323
C01	23.57128368
D01	14.70976429
E01	85.72777329
F01	53.59418443
G01	69.61800891
H01	39.59824979
A02	15.54388965
B02	77.67298537
C02	47.52201094
D02	34.80217606
E02	13.30382596
F02	78.00744614
G02	32.26121349
...	

ex2.txt

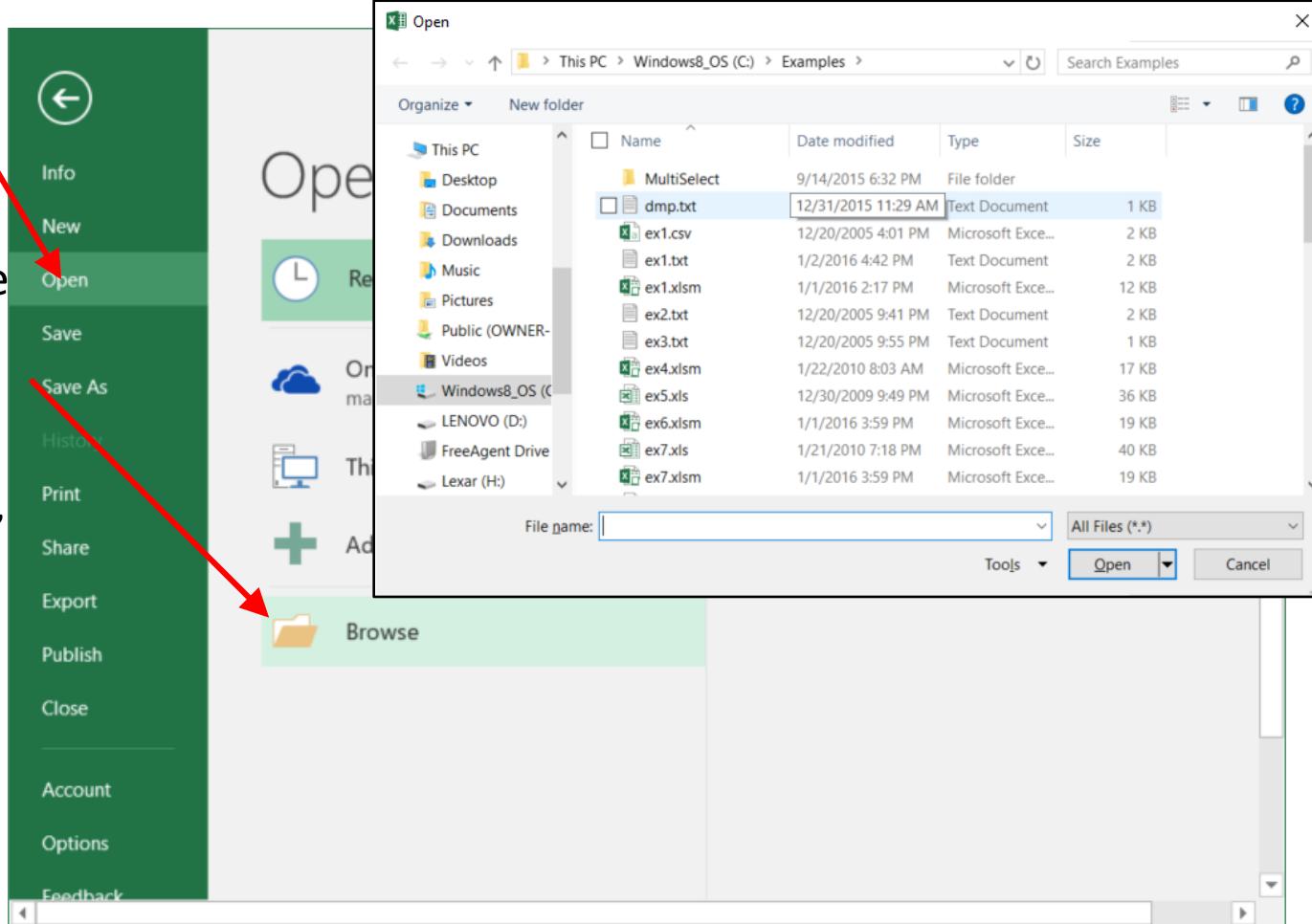
Example 2: View Formatting Marks

- MS Word will display certain formatting marks if the Show/Hide toolbar button is turned on
 - the button with the ¶ symbol
- Formatting marks are displayed with symbols and other designations:
 - <TAB> is displayed as “→”
 - <CR><LF> is displayed as “¶ ”



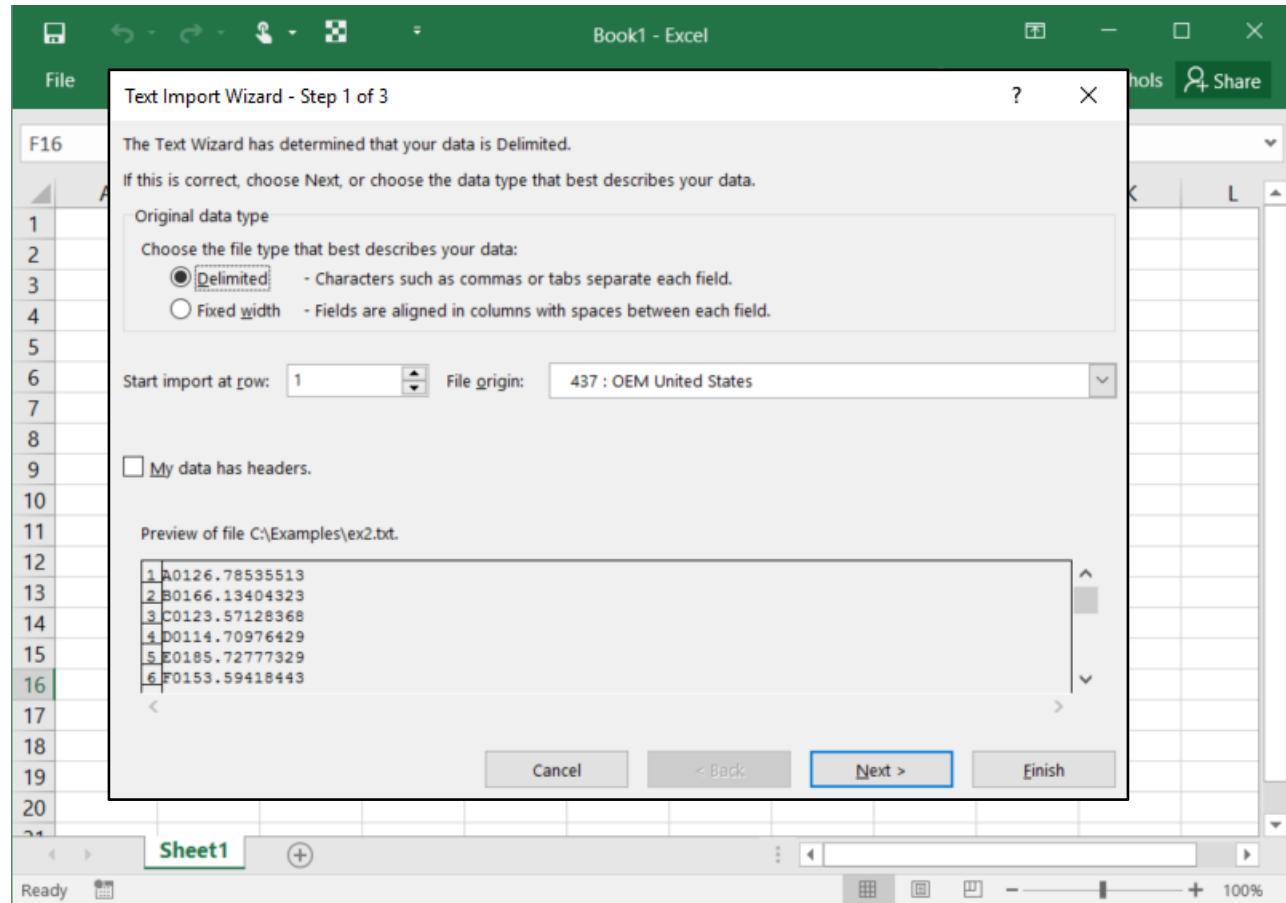
Example 2: Open the Tab-delimited File

- Create a new workbook and open the file ex2.txt
- Ensure that the “All Files (*.*)” file type is selected
- Select ‘Browse’ to navigate to the Examples and select the file



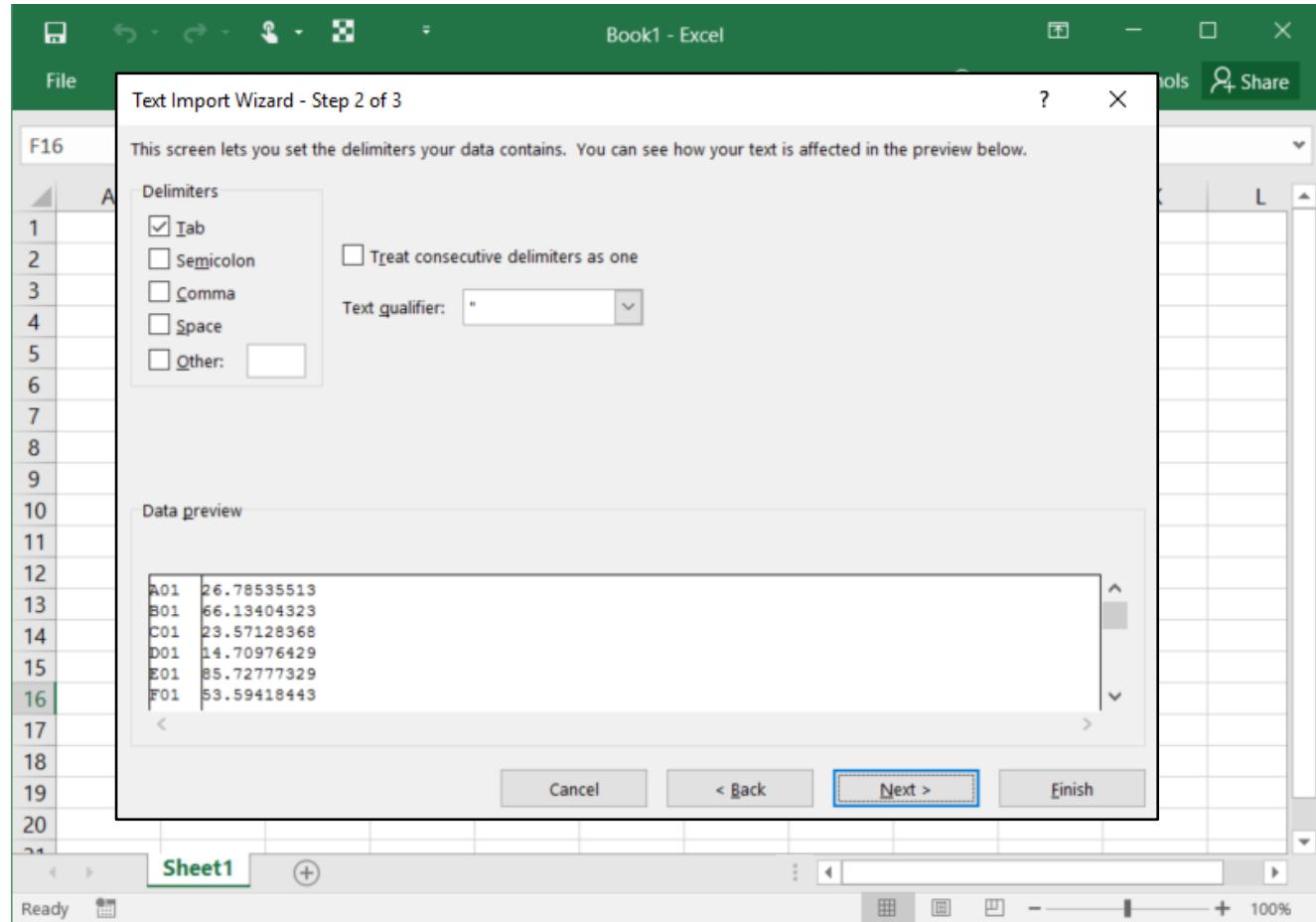
Example 2: Step 1 of 3

- Note that the non printing tab characters do not appear.
- Choose “Delimited”
- Click [Next >]



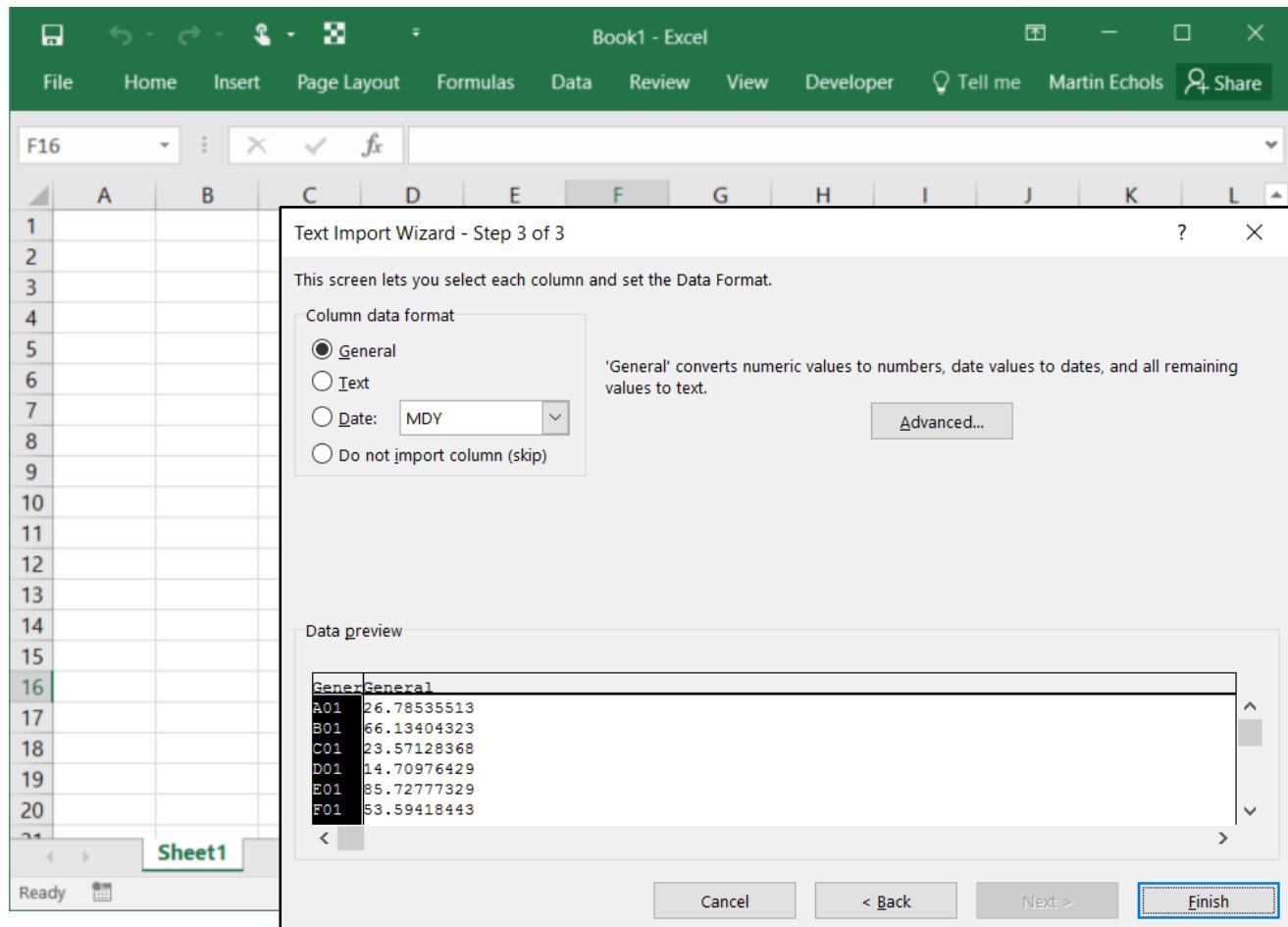
Example 2: Step 2 of 3

- Select “Tab”
- Note how the Preview pane correctly parses the file
- Click [Next >]



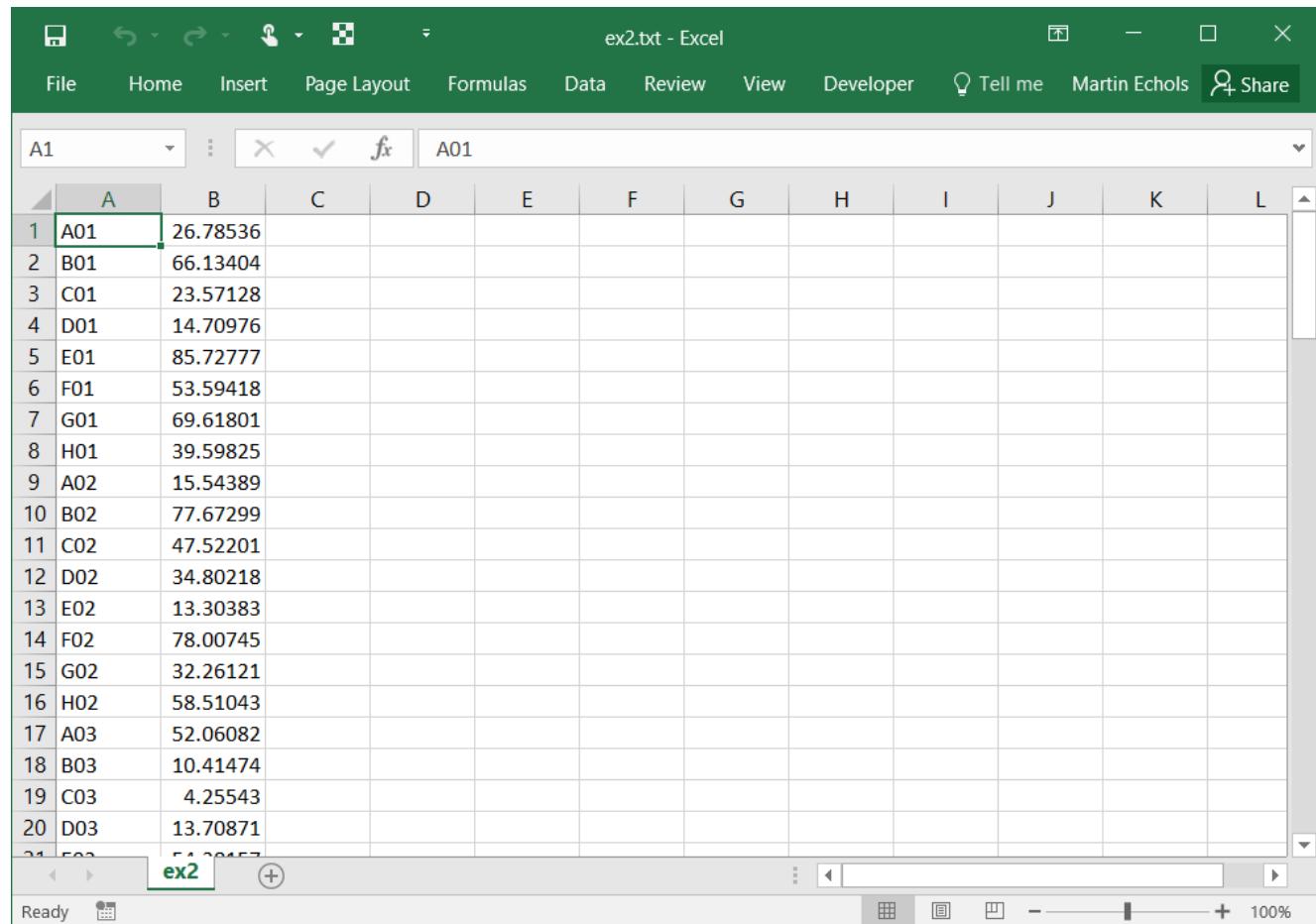
Example 2: Step 3 of 3

- Select a format for the data in each column
- Using General is often the best option
- Click [Finish]



Example 2: The Imported Data

- The end result is identical to Example 1



A screenshot of Microsoft Excel showing a table of data. The window title is "ex2.txt - Excel". The ribbon tabs are File, Home, Insert, Page Layout, Formulas, Data, Review, Developer, Tell me, Martin Echols, and Share. The active cell is A1, which contains "A01". The data starts at row 1 and continues through row 20. Column A contains labels like A01, B01, C01, etc., and column B contains numerical values. The table is bordered by a thin gray line.

	A	B	C	D	E	F	G	H	I	J	K	L
1	A01	26.78536										
2	B01	66.13404										
3	C01	23.57128										
4	D01	14.70976										
5	E01	85.72777										
6	F01	53.59418										
7	G01	69.61801										
8	H01	39.59825										
9	A02	15.54389										
10	B02	77.67299										
11	C02	47.52201										
12	D02	34.80218										
13	E02	13.30383										
14	F02	78.00745										
15	G02	32.26121										
16	H02	58.51043										
17	A03	52.06082										
18	B03	10.41474										
19	C03	4.25543										
20	D03	13.70871										

Example 3: Import Fixed-Width Columns

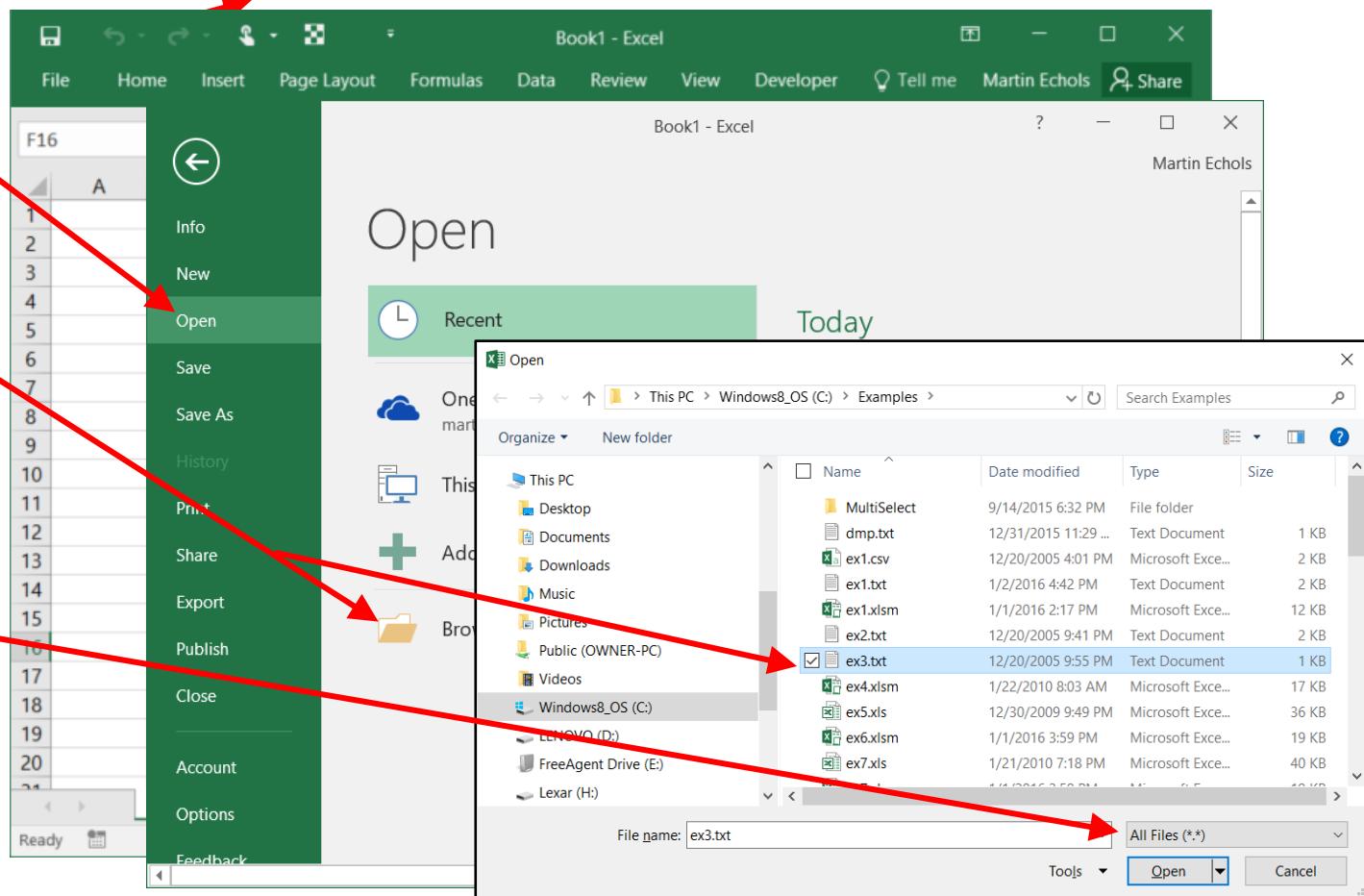
- Another common format is a data file whose columns are left-aligned using multiple tab or whitespace characters
- The Text Import Wizard can be used to import files in this format

0.03	16.73	789
146.57	4.19	144.21
741.36	0.08	15.18
5.42	78.06	126.27
597.68	60.26	208.37
34.78	5.34	134.3
118.93	307.15	133.71
172.41	63.39	0.02
218.45	0.32	11.54
44.7	108.39	2.5

ex3.txt

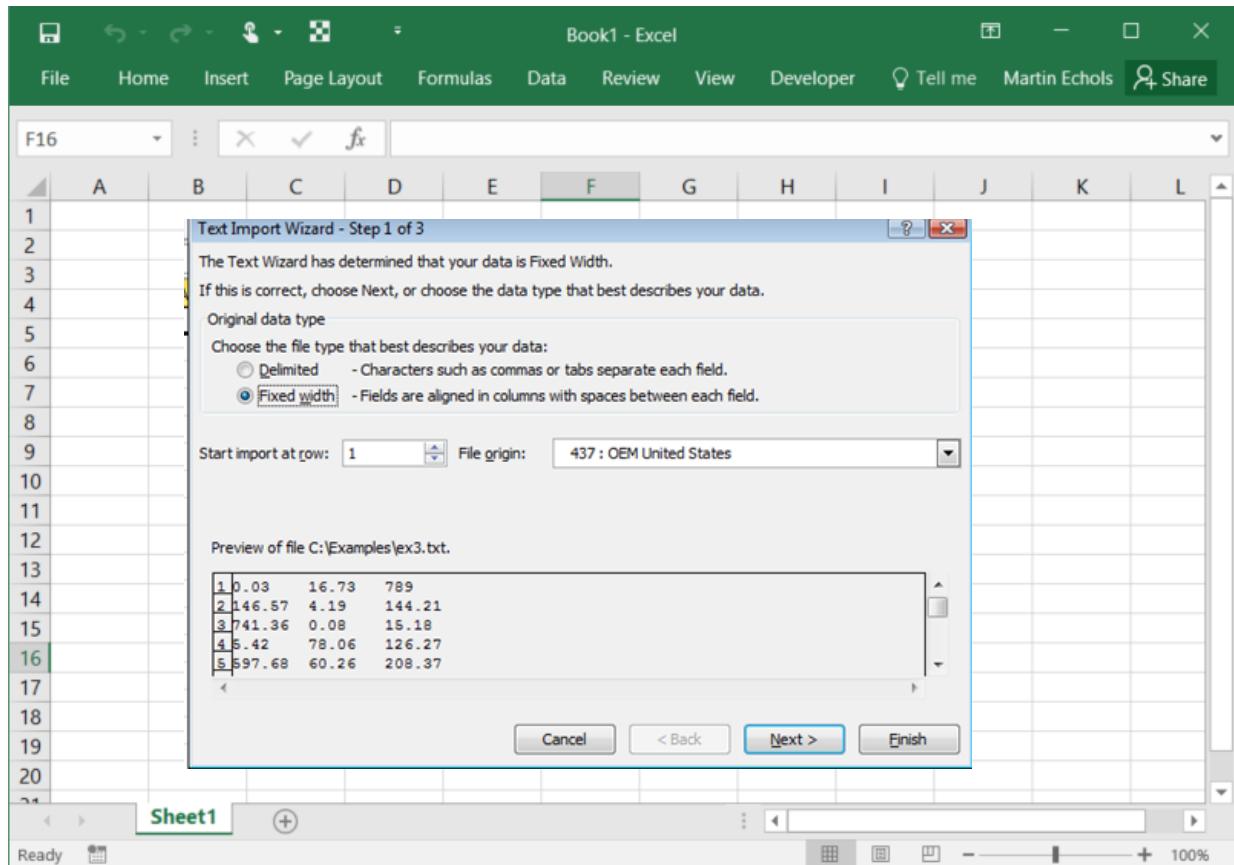
Example 3: Open the Fixed-Width File

- Create a new workbook and open the file ex3.txt
- Navigate to the Examples and select the file
- Ensure that the “All Files (*.*)” file type is selected



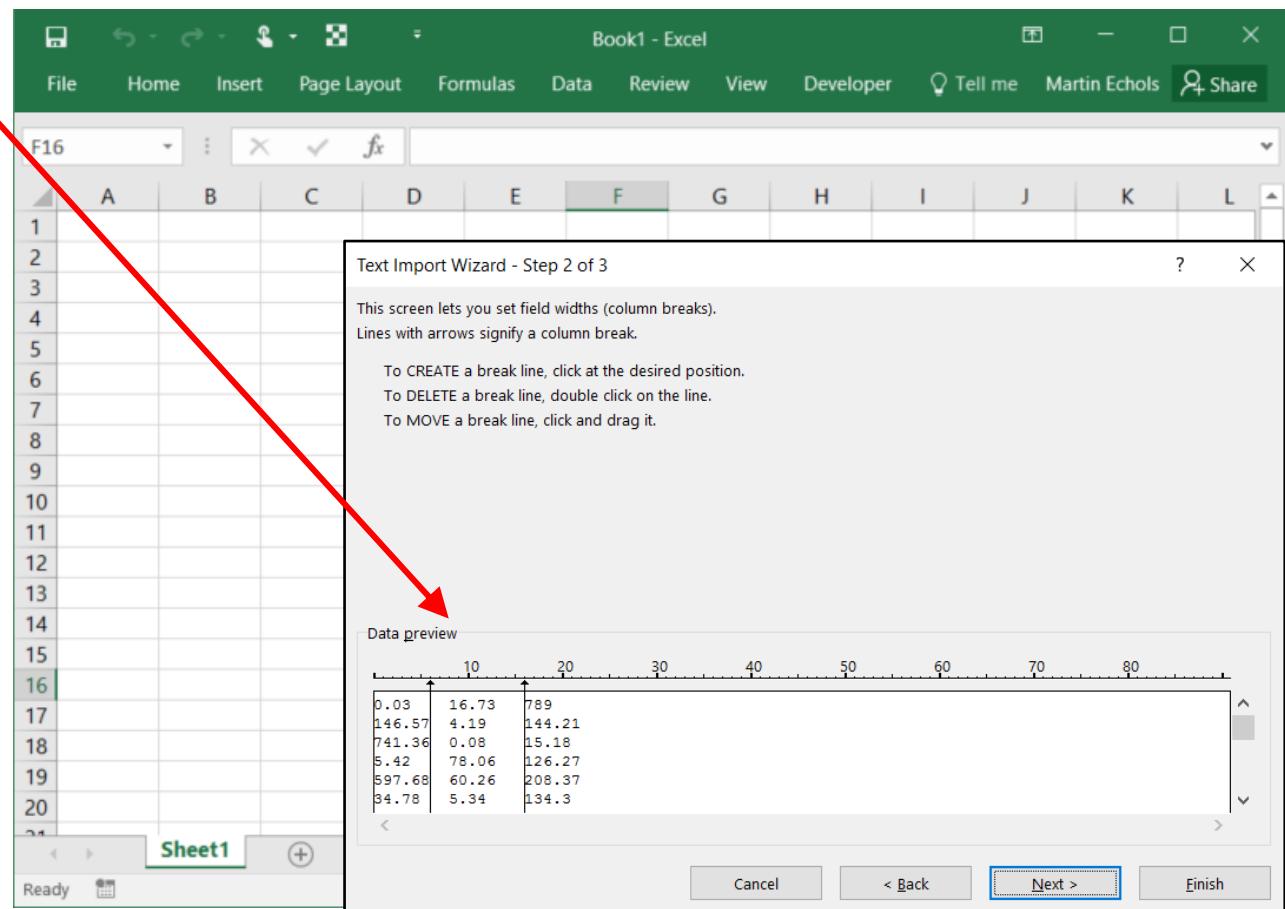
Example 3: Step 1 of 3

- Note that Excel has already suggested using the Fixed width option. If not, select it
- Click [Next >]



Example 3: Step 2 of 3

- Add, delete and drag break lines to mark columns as needed.
- Click [Next >]



Example 3: Step 3 of 3

- Select a format for the data in each column
- Using General is often the best option
- Click [Finish]

The screenshot shows a Microsoft Excel window titled "Book1 - Excel". The ribbon tabs are visible at the top. A dialog box titled "Text Import Wizard - Step 3 of 3" is open over the spreadsheet. The dialog box contains the following information:

- Column data format:** The "General" radio button is selected. A tooltip explains: "'General' converts numeric values to numbers, date values to dates, and all remaining values to text."
- Data preview:** A small table shows the first few rows of data with "General" applied to all columns.
- Buttons:** "Cancel", "Back", "Next >", and "Finish" (highlighted with a blue border).

The spreadsheet area shows a table with columns A through L and rows 1 through 20. The "Sheet1" tab is selected at the bottom left.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

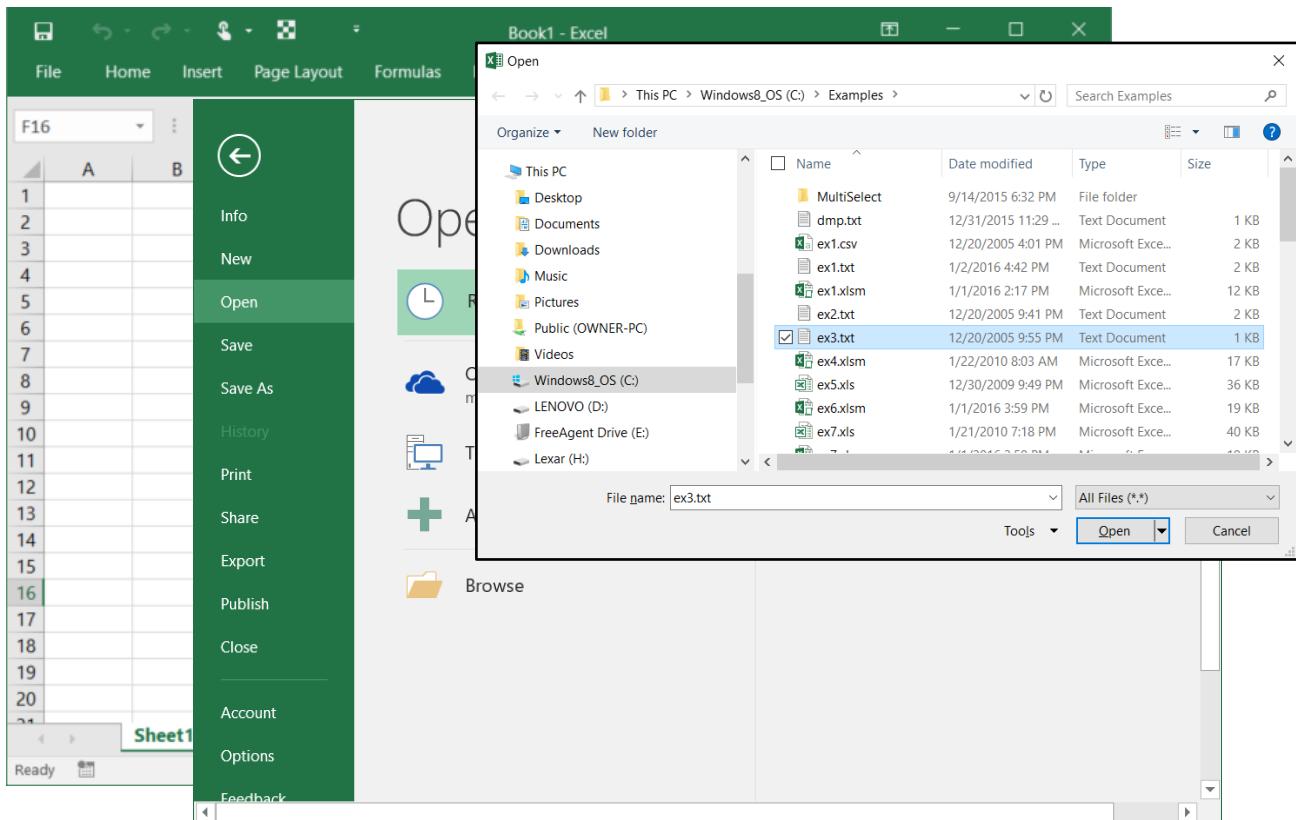
Example 3: The Imported Data

- The data are properly imported into the cells of the Worksheet

	A	B	C	D	E	F	G	H	I	J	K	L
1	0.03	16.73	789									
2	146.57	4.19	144.21									
3	741.36	0.08	15.18									
4	5.42	78.06	126.27									
5	597.68	60.26	208.37									
6	34.78	5.34	134.3									
7	118.93	307.15	133.71									
8	172.41	63.39	0.02									
9	218.45	0.32	11.54									
10	44.7	108.39	2.5									
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												

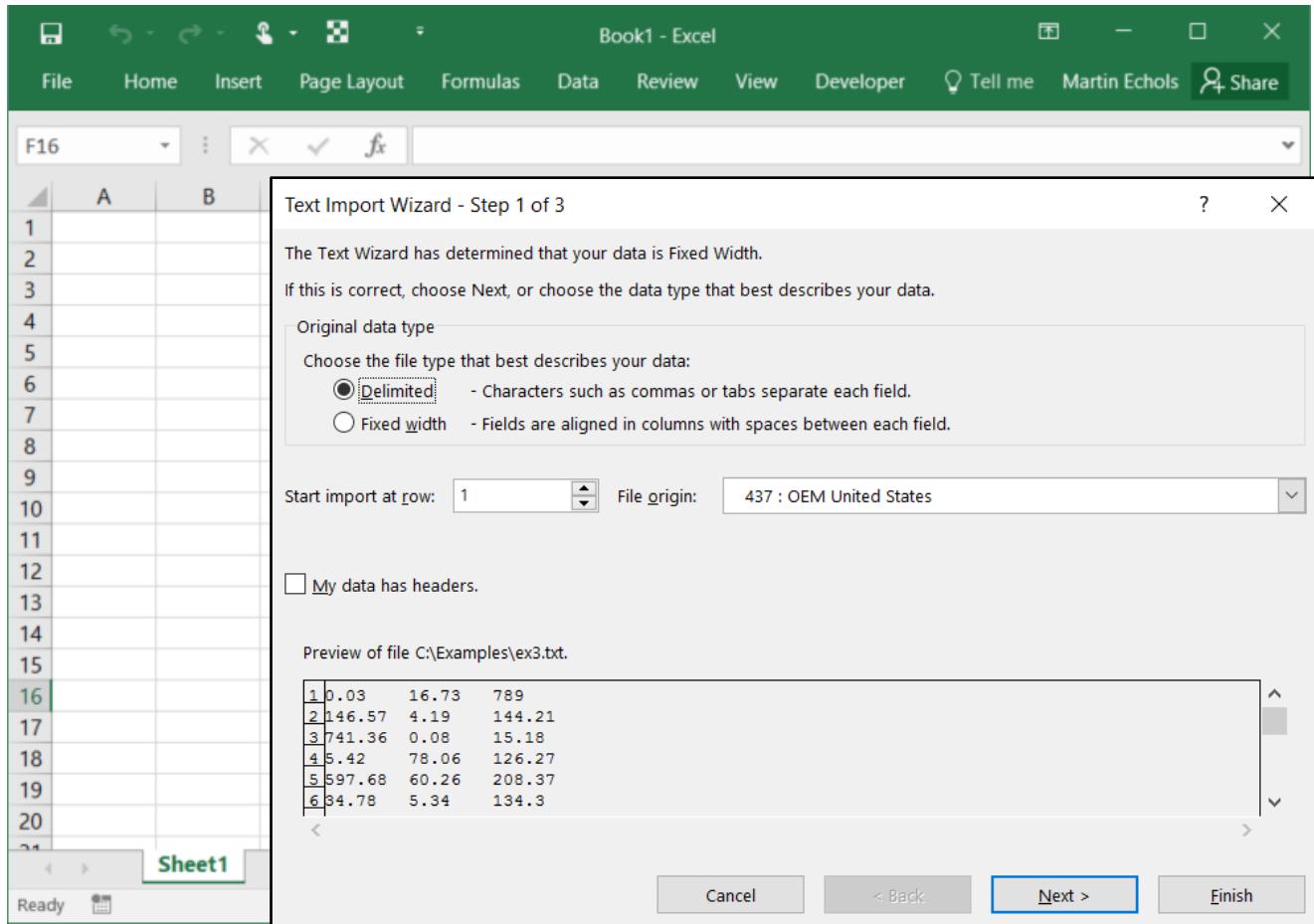
Example 3 (Part 2): Multiple Delimiters

- This file can also be parsed using the “Delimited” option
- Close the workbook, create a new one, and open the ex3.txt file again



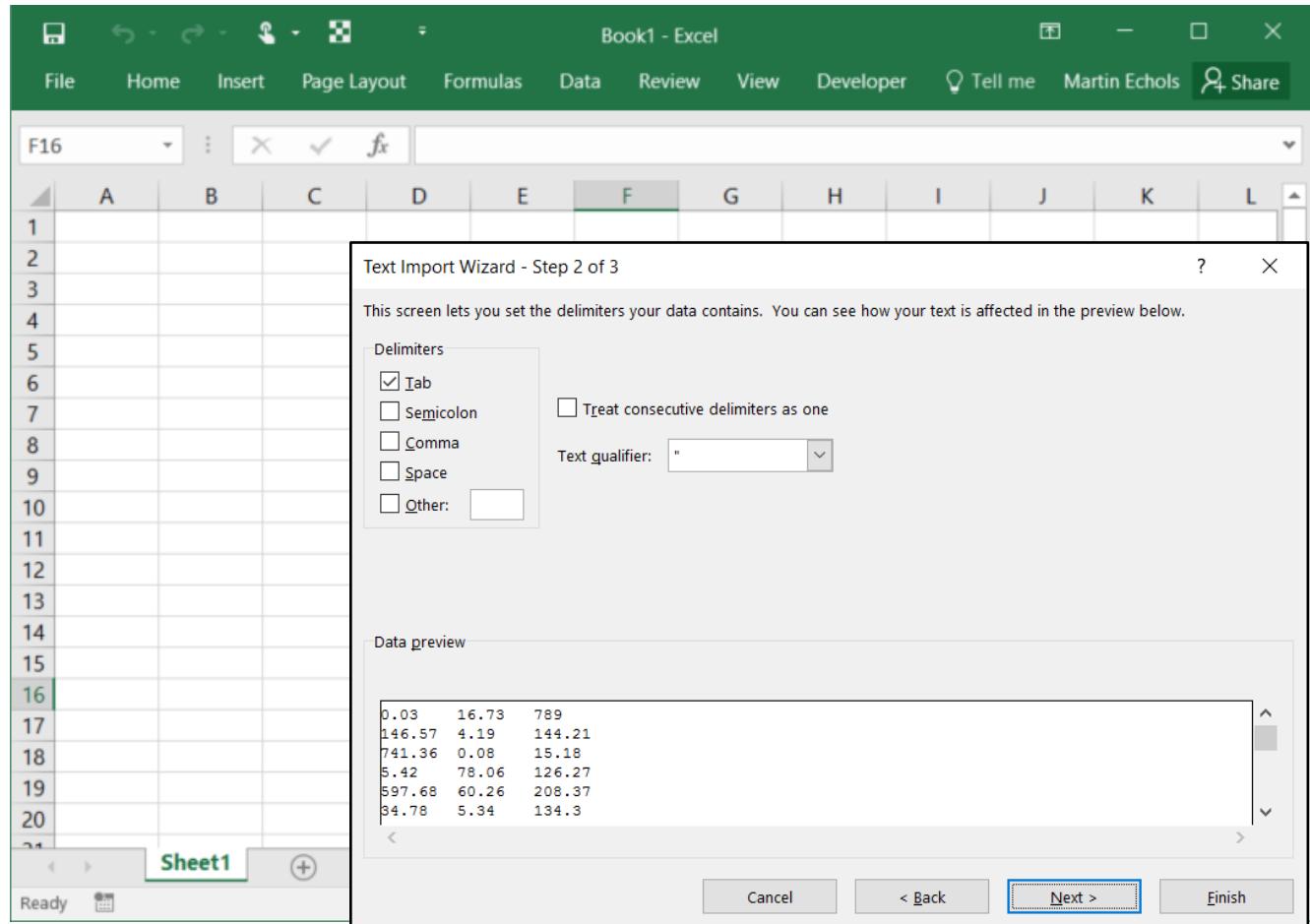
Example 3 (Part 2): Step 1 of 3

- Choose
“Delimited”
- Click [Next >]



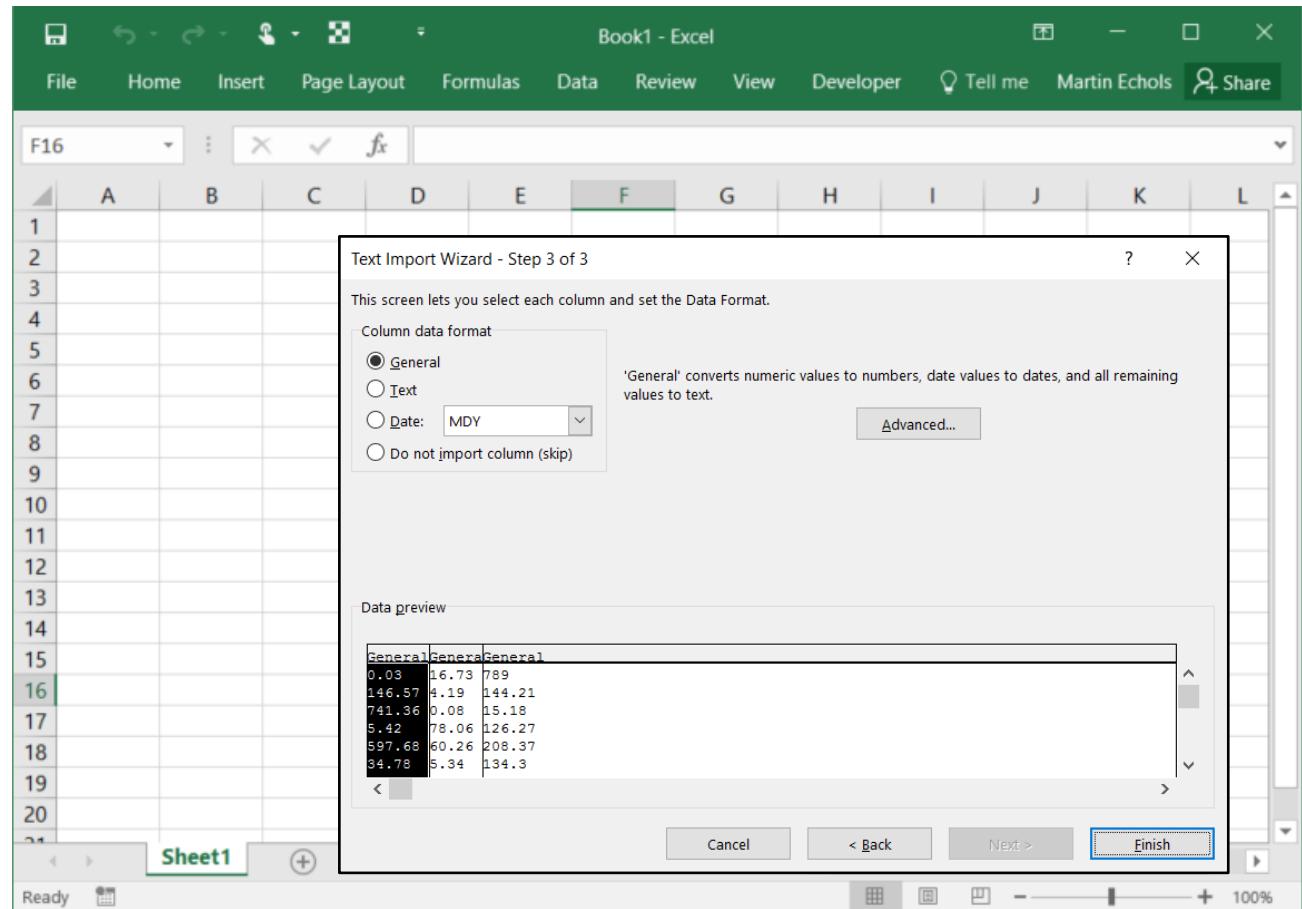
Example 3 (Part 2): Step 2 of 3

- Experiment with the options. Try selecting “Space” only
- Choose both the “Space” and “Treat consecutive delimiters as one” options
- Click [Next >]



Example 3 (Part 2): Step 3 of 3

- Select a format for the data in each column
- Using General is often the best option
- Click [Finish]



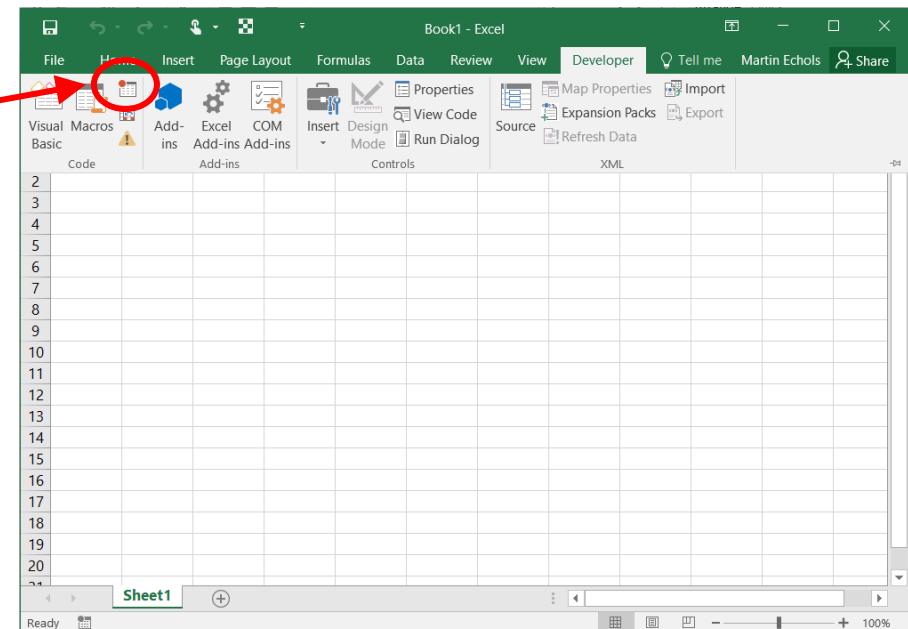
Example 3 (Part 2): The Imported Data

- The three columns of data are loaded into the three columns of the Worksheet.
- Once again, note that the name of the Worksheet is changed to the name of the data file.

A	B	C
1 0.03	16.73	789
2 146.57	4.19	144.21
3 741.36	0.08	15.18
4 5.42	78.06	126.27
5 597.68	60.26	208.37
6 34.78	5.34	134.3
7 118.93	307.15	133.71
8 172.41	63.39	0.02
9 218.45	0.32	11.54
10 44.7	108.39	2.5
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Macros and the Macro Recorder

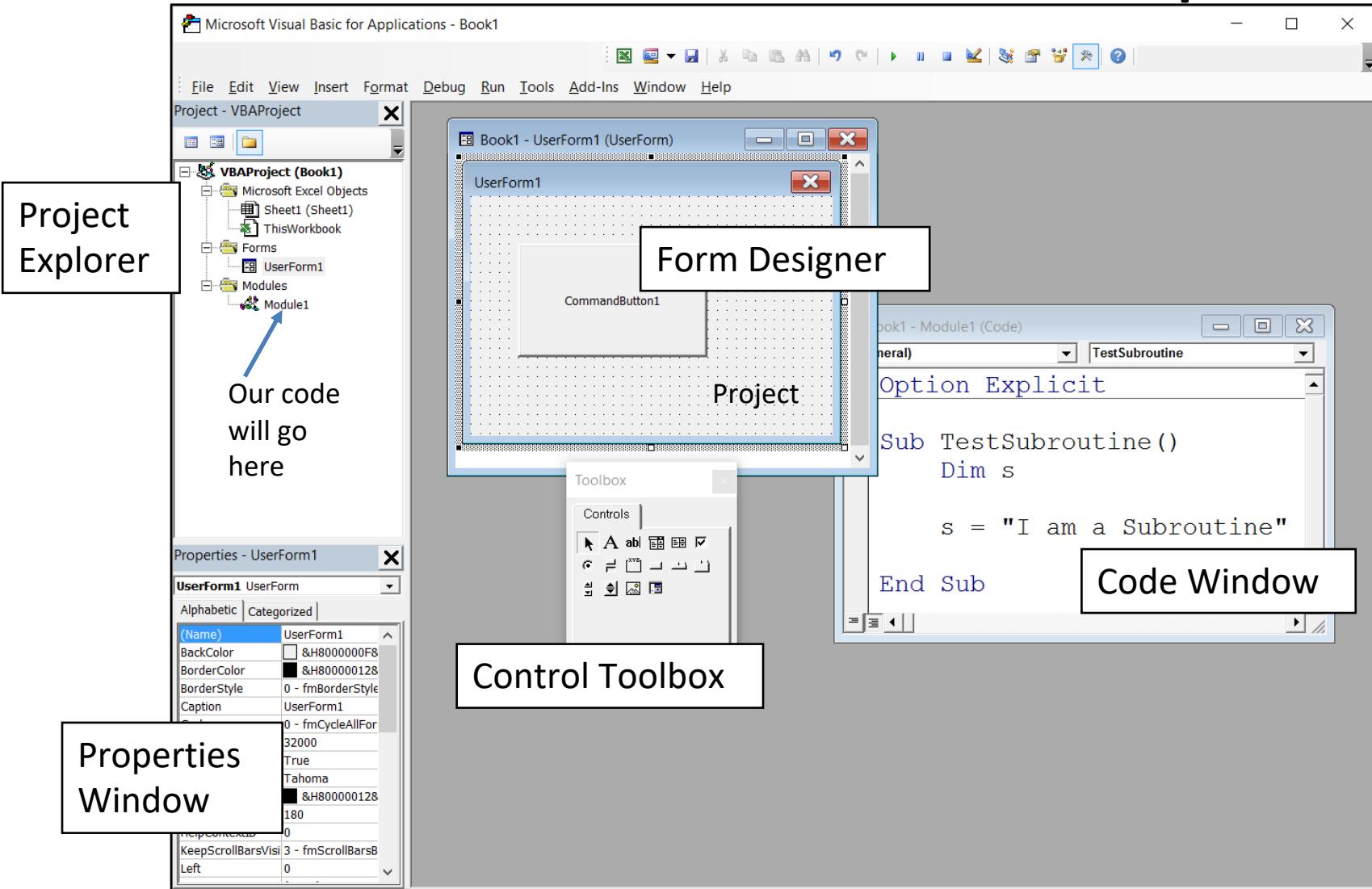
- If you perform a task (sequence of steps) repeatedly in Excel, a macro can automate this task for you.
- The macro recorder stores (*records*) information from each step in the task so that it can be rerun (*played back*) automatically.
- To start the macro recorder:
 - Select the Developer tab
 - Click the record button.
- Excel will record each and every step that you carry out (mistakes and corrections included) until you hit the *stop button*.



The Visual Basic Editor

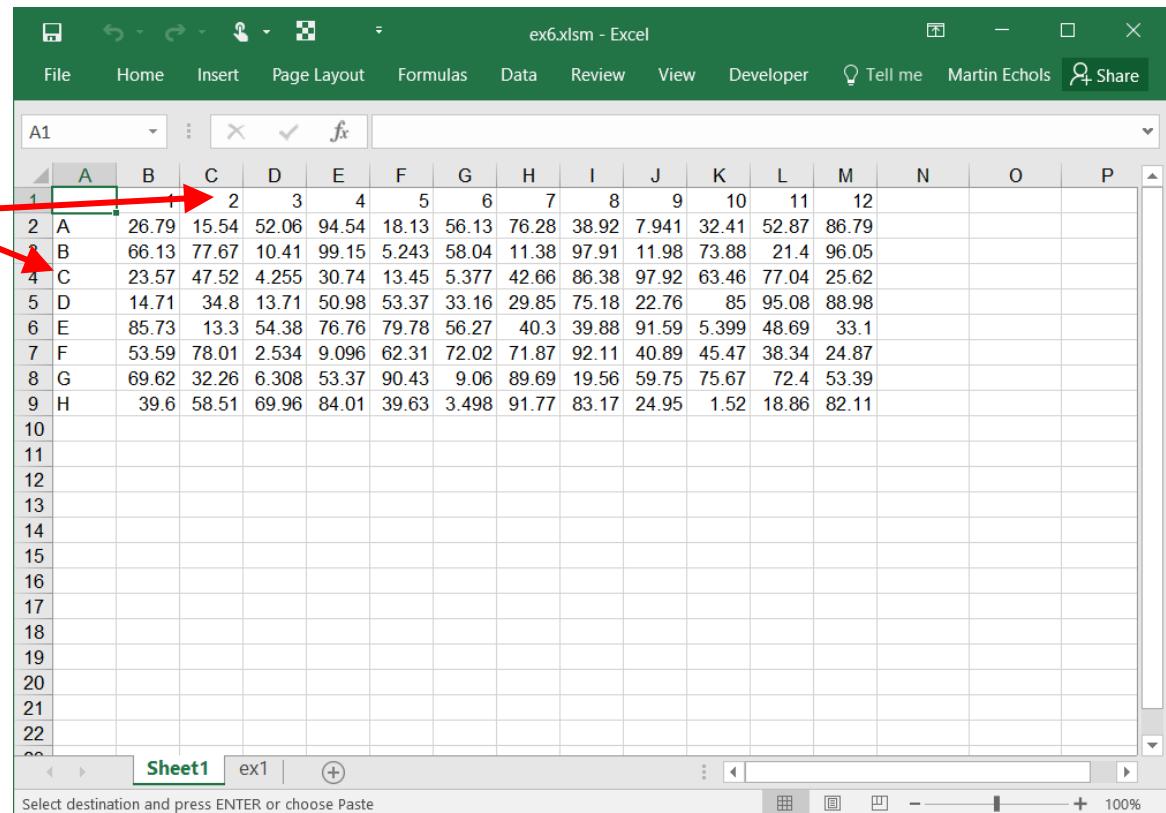
- Excel stores the information for replaying a macro in the form of *Visual Basic for Applications* (VBA) source code
- The VBA code for the macro is stored in the Workbook and can be viewed and edited in the *Visual Basic Editor* (VBE)
- The VBE can be accessed from the main Excel user interface using two methods
 - The hotkey sequence Alt+F11
 - Selecting the menu sequence “Developer| Visual Basic”

The Visual Basic Editor ... Exposed



Example 4: Reformat Imported Data

- Create a macro that reformats the imported data as an 8 x 12 array on a new Worksheet
- Note the presence of row and column labels



A screenshot of Microsoft Excel showing a data grid. The grid has 9 rows and 12 columns. Row 1 contains labels A through L. Row 2 contains data for row 1. Row 3 contains data for row 2. Row 4 contains data for row 3. Row 5 contains data for row 4. Row 6 contains data for row 5. Row 7 contains data for row 6. Row 8 contains data for row 7. Row 9 contains data for row 8. Red arrows point from the text "Note the presence of row and column labels" to the labels in Row 1 and the first few cells in Row 2.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		2	3	4	5	6	7	8	9	10	11	12			
2	A	26.79	15.54	52.06	94.54	18.13	56.13	76.28	38.92	7.941	32.41	52.87	86.79		
3	B	66.13	77.67	10.41	99.15	5.243	58.04	11.38	97.91	11.98	73.88	21.4	96.05		
4	C	23.57	47.52	4.255	30.74	13.45	5.377	42.66	86.38	97.92	63.46	77.04	25.62		
5	D	14.71	34.8	13.71	50.98	53.37	33.16	29.85	75.18	22.76	85	95.08	88.98		
6	E	85.73	13.3	54.38	76.76	79.78	56.27	40.3	39.88	91.59	5.399	48.69	33.1		
7	F	53.59	78.01	2.534	9.096	62.31	72.02	71.87	92.11	40.89	45.47	38.34	24.87		
8	G	69.62	32.26	6.308	53.37	90.43	9.06	89.69	19.56	59.75	75.67	72.4	53.39		
9	H	39.6	58.51	69.96	84.01	39.63	3.498	91.77	83.17	24.95	1.52	18.86	82.11		
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															

Example 4: Reformat Imported Data

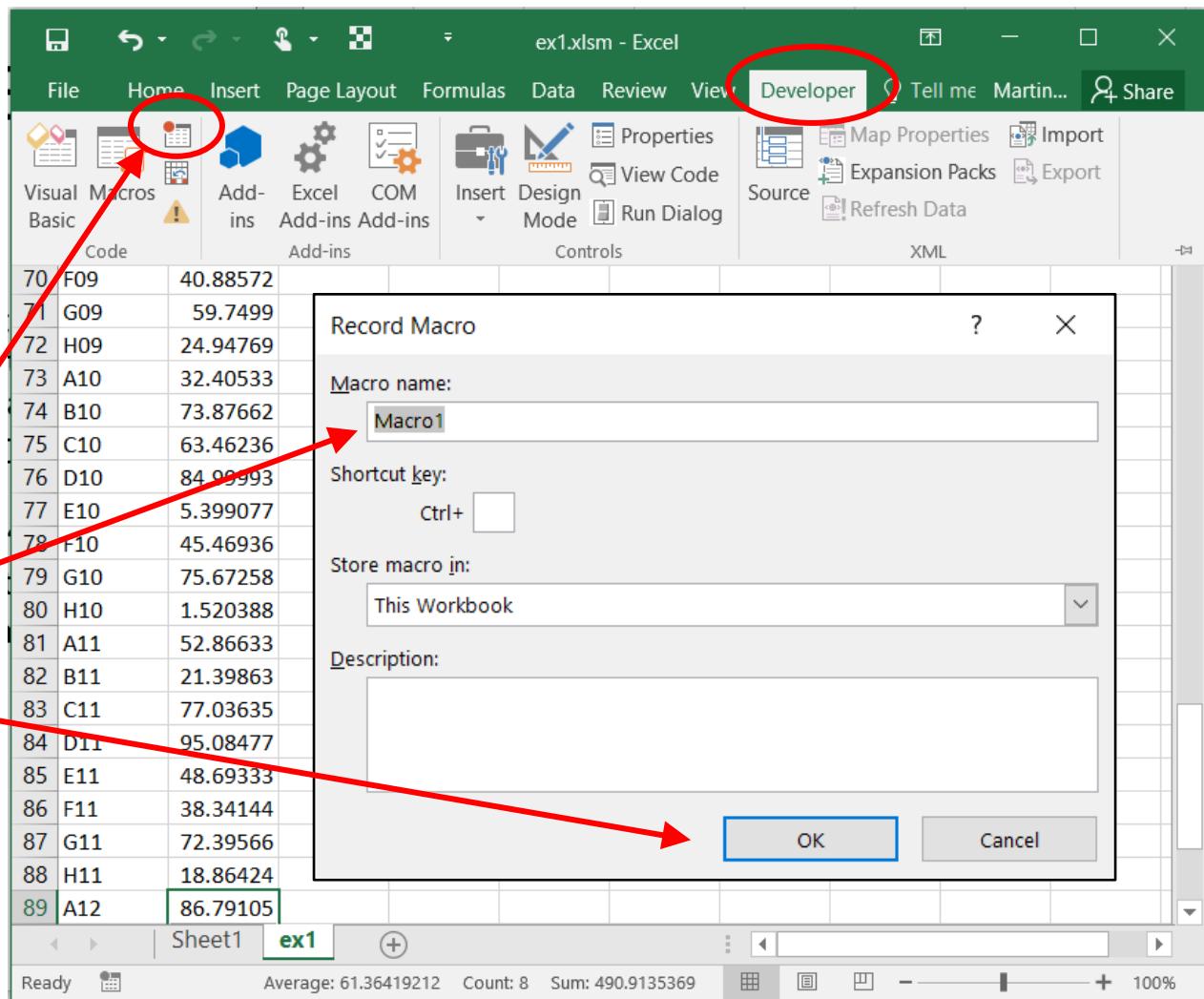
- We will essentially cut the column in file ex1.txt into 12 columns each with 8 rows as shown here.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	A01	26.78536												
2	B01	66.13404												
3	C01	23.57128												
4	D01	14.70976												
5	E01	85.72777												
6	F01	53.59418												
7	G01	69.61801												
8	H01	39.59825												
9	A02	15.54389												
10	B02	77.67299												
11	C02	47.52201												
12	D02	34.80218												
13	E02	13.30383												
14	F02	78.00745												
15	G02	32.26121												
16	H02	58.51043												
17	A03	52.06082												
18	B03	10.41474												
19	C03	4.25543												
20	D03	13.70871												
21	E03	54.38157												
22	F03	2.533874												
23	G03	6.307792												
24	H03	69.95962												
25	A04	94.54007												

Example 4: Record a Macro

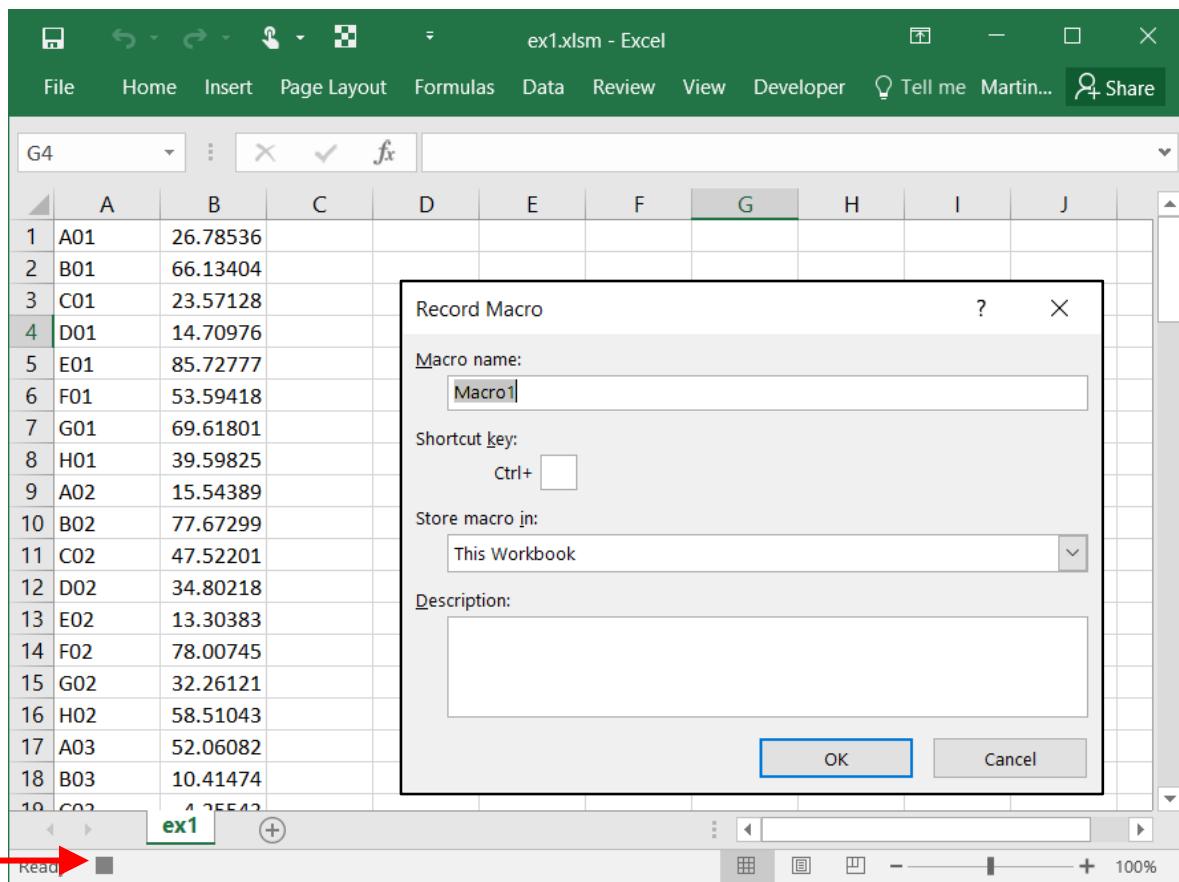
- Start with the Workbook that you saved in Example 1
- Select the Developer Tab.
- Select the “Record Macro” button
- Enter a name for the macro
- press OK

[for this course we use
The default name, macro1]



Example 4: Record a Macro

- The Stop Recording button appears at the bottom of the window.
- Insert a new Worksheet, enter row and column headers, and copy/paste data between sheets in groups of 8 (each column)

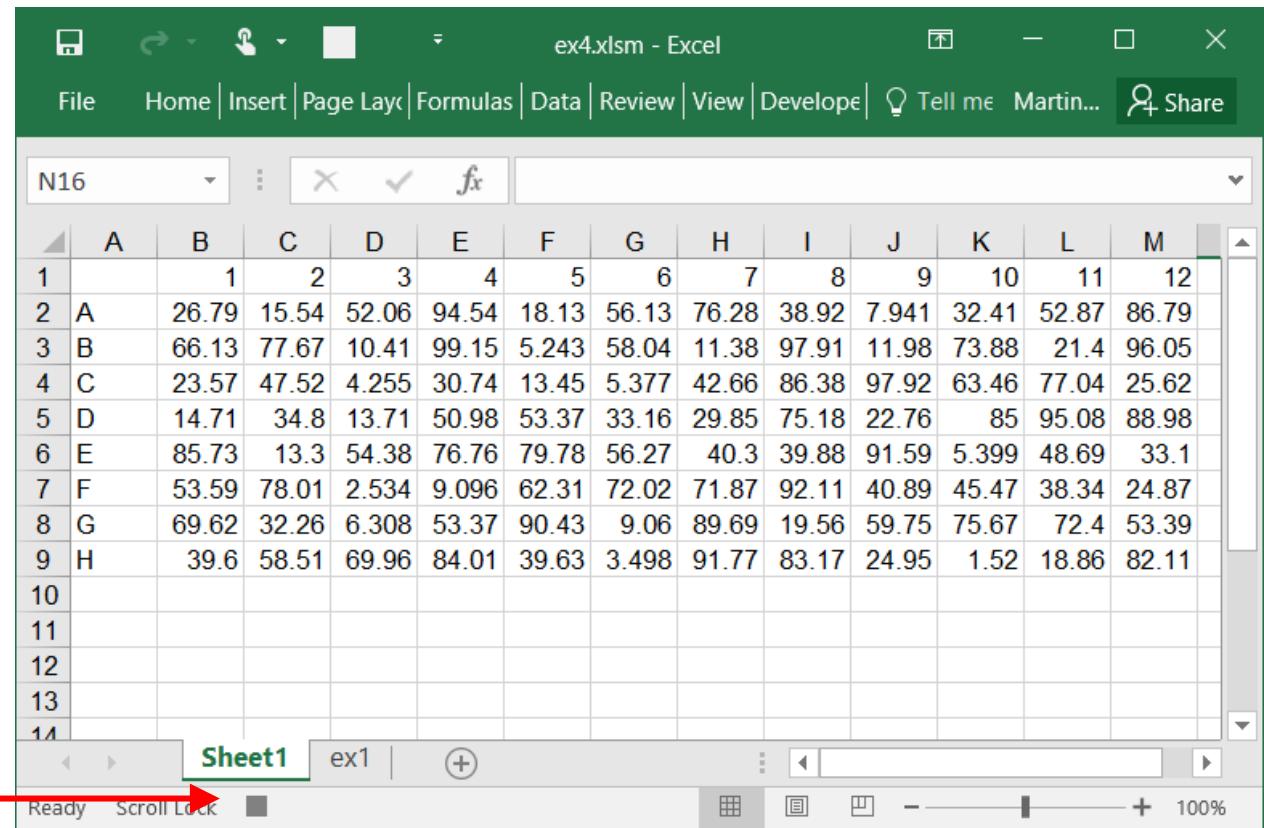


stop recording
here

Example 4: Record a Macro

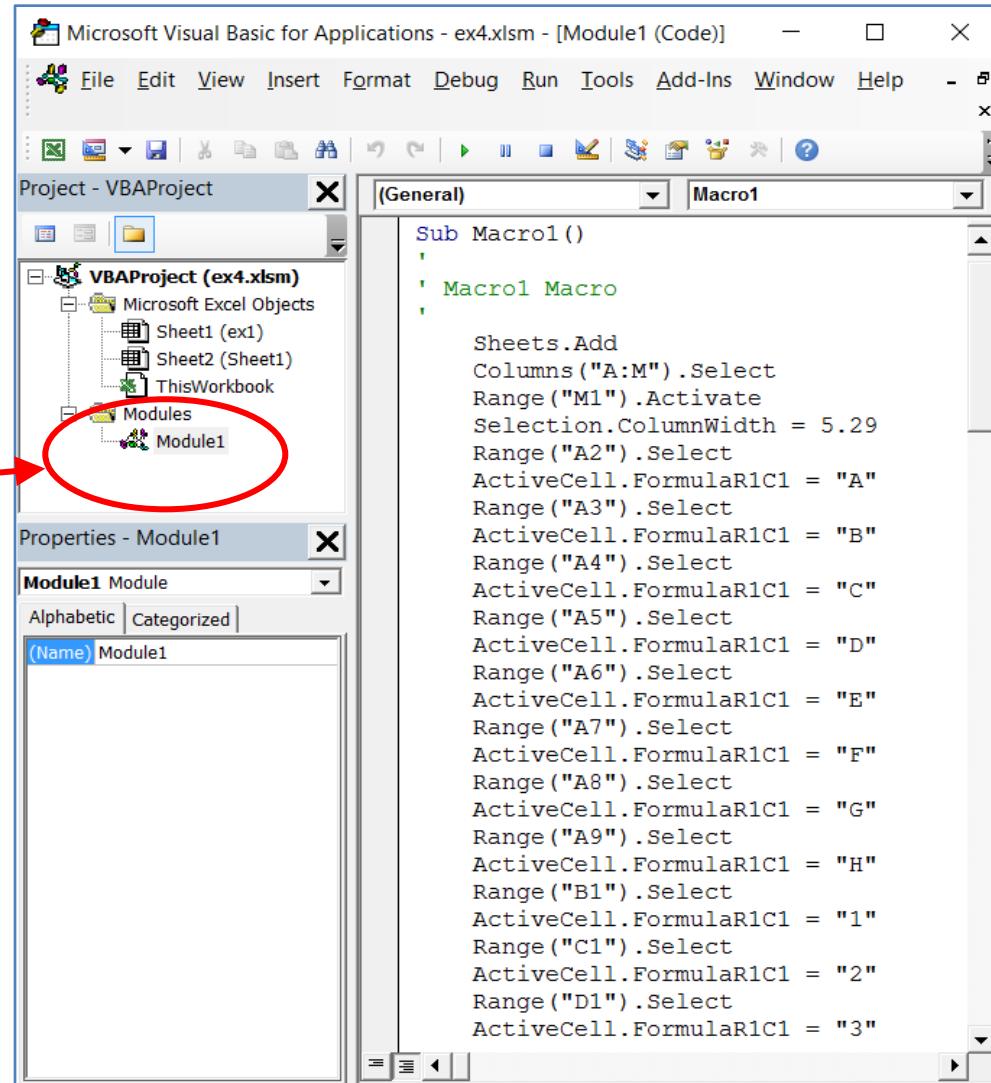
- The ‘Stop Recording’ button appears at the bottom of the window. You can also select ‘Developer -> Stop Recording’
- Click the Stop Recording button.

stop recording
here



Example 4: Study the Recorded Macro

- Launch the Visual Basic Editor (Alt+F11)
- Expand “Modules” and double-click on “Module1”
- Study the generated code.
- Can you determine or guess what each line of code does?



A screenshot of the Microsoft Visual Basic for Applications (VBA) editor. The title bar reads "Microsoft Visual Basic for Applications - ex4.xlsm - [Module1 (Code)]". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, Help. The toolbar has various icons for file operations and code editing. The left pane shows the "Project - VBAProject" window for the workbook "ex4.xlsm", with nodes for Microsoft Excel Objects, Sheet1 (ex1), Sheet2 (Sheet1), ThisWorkbook, Modules, and Module1. A red arrow points from the text "double-click on ‘Module1’" to the "Module1" node in the Project Explorer. The right pane displays the code for "Macro1()".

```
Sub Macro1()
    ' Macro1 Macro
    '
    Sheets.Add
    Columns("A:M").Select
    Range("M1").Activate
    Selection.ColumnWidth = 5.29
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "A"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "B"
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "C"
    Range("A5").Select
    ActiveCell.FormulaR1C1 = "D"
    Range("A6").Select
    ActiveCell.FormulaR1C1 = "E"
    Range("A7").Select
    ActiveCell.FormulaR1C1 = "F"
    Range("A8").Select
    ActiveCell.FormulaR1C1 = "G"
    Range("A9").Select
    ActiveCell.FormulaR1C1 = "H"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "1"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "2"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "3"

```

Visual Basic Syntax

- For the remainder of the course we will be introducing some Visual Basic syntax and concepts.
- We do this within the context of examples to give the syntax and concepts a useful framework and to provide you with reference material.

Macros are VBA Subroutines

- Procedures are parameterized blocks of code (sequence of statements) that can be reused
- VBA offers two basic procedure types:
 - Subroutines
 - Accepts arguments for processing
 - **Does not** return a value
 - Functions
 - Accepts arguments for processing
 - **Does** return a value
- Subroutines and Functions can be defined and stored in any code module

Sub Statement

Declares the name, arguments, and code that form the body of a Subroutine procedure.

Syntax:

```
[Private | Public] [Static] Sub name [(arglist)]
    [statements]
    [Exit Sub]
    [statements]
End Sub
```

Example:

```
Sub AddOne(I As Integer)
    I = I + 1
End Sub
```

```
J = 10
AddOne J
```

Function Statement

Declares the name, arguments, and code that form the body of a Function procedure.

Syntax:

```
[Public | Private] [Static] Function name [(arglist)] [As type]  
    [statements]  
    [name = expression]  
    [Exit Function]  
    [statements]  
    [name = expression]  
End Function
```

Example:

```
Function IsEven(I As Integer) As Boolean  
    IsEven = False  
    If I Mod 2 = 0 Then IsEven = True  
End Function
```

```
J = 21  
B = IsEven(J)
```

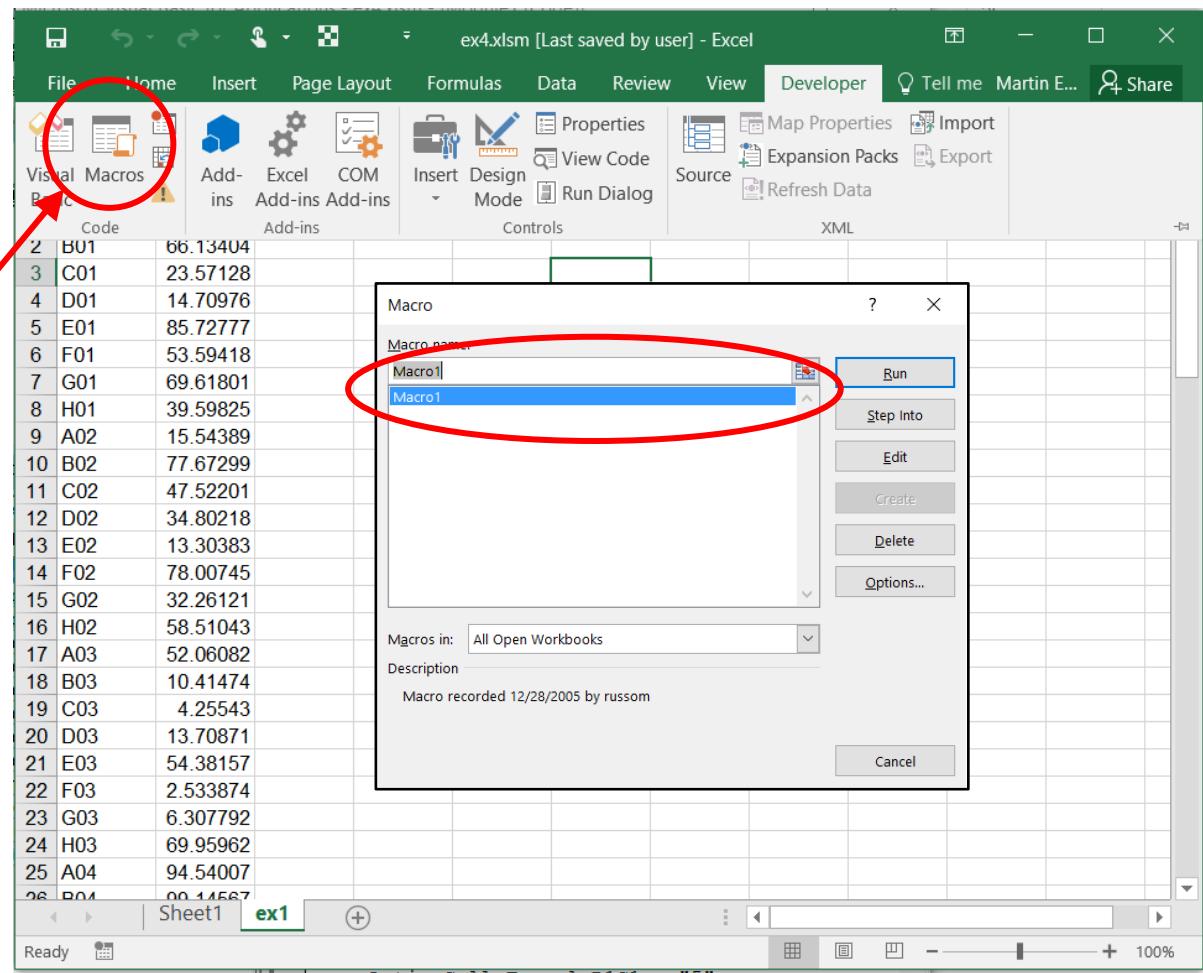
Example 4: Study the Recorded Macro

A few sample lines of code and what they do ...

- `Sheets.Add`
 - Adds a new Worksheet to the Workbook. The name is assigned automatically.
- `ActiveCell.FormulaR1C1 = "A"`
 - Inserts text into the currently active cell.
- `Sheets("ex1").Select`
 - Activates the named Worksheet.
- `Range("B1:B8").Select`
 - Selects a group of cells.
- `Selection.Copy`
 - Copies the selected cells onto the Clipboard.
- `ActiveSheet.Paste`
 - Pastes the Clipboard contents into the Active Worksheet.

Example 4: Run the Macro

- Run the recorded macro to see how well it works
- Select the Developer Tab then Macros
- Choose the macro from the pop up and click [Run]



Example 4: Problems with the Macro

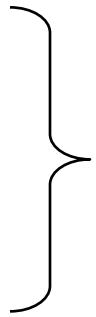
- Depending on how you recorded the macro, many problems may have occurred
 - If you started the macro with a different worksheet activated, the wrong data may have been copied.
 - A new sheet was added to the Workbook and the row and columns entered properly, but the data was probably pasted into the Worksheet created when the first Macro was recorded, not the new one.
 - You may not be able to tell it yet, but data is copied from the Worksheet named “ex1.” The macro will not work if a data file with a name different than “ex1” is imported.
 - Any mistakes that you may have made while recording will be repeated faithfully during playback.
 - While not detrimental to the proper operation of the macro, the action of switching back and forth between sheets captured in the macro is very unattractive.
 - The code generated faithfully reproduces your actions. This is generally not the most efficient way to complete a task.

Variables

- Variables are elements of a programming language that store data during the execution of a program, including references to objects.
- As a program executes, the data that a variable contains can be changed (hence the name *variable*).
- In VBA, variables are assigned a name (for reference in the program) and a data type. This process is called a *declaration*. Thus, one *declares* a variable.
- There are 12 intrinsic data types in VBA and many many object types, depending upon the libraries that are loaded.
- A variable can only hold data of the same data type for which it was declared.

Data Types

Boolean	(bln)	Decimal	(dec)
Byte	(byt)	String	(str)
Integer	(int)	Currency	(crn)
Long	(lng)	Date	(dat)
Single	(sng)	Object	(obj)
Double	(dbl)	Variant	(vnr)

Worksheet	(ws)	 Important Excel-specific objects
Range	(rng)	
Workbook	(wbk)	

Declaring Variables in VBA

- To declare a variable in VBA, the following syntax is used:

```
Dim variable-name [As data-type]
```

For example, to declare a variable named strMyVar as intrinsic data type String, use the following syntax

```
Dim strMyVar As String
```

- If the [As data-type] clause is omitted, the variable will be declared as intrinsic data type *Variant*.
- A Variant is a special data type that can hold data of any type.

Declaring Variables in VBA

- In VBA, all variables are initialized automatically.
 - Integers, Longs, Singles, Doubles, etc. get a value of 0
 - Strings are set to an empty string “”
 - Booleans get the value ‘False’
 - Objects get the value ‘Nothing’
 - Variants get initial values for all types or ‘Empty’

Assigning Data to Variables

- Data are assigned to variables using the assignment operator (=)
For example, to store the string value "A string" in the variable named strMyVar, use the following code ...

```
Dim strMyVar As String  
strMyVar = "A string"
```

- A variable can also be assigned to the result of an expression or to another variable

```
Dim intMyNum As Integer  
Dim intAnotherNum As Integer  
intMyNum = 10  
intAnotherNum = intMyNum / 2
```

Assigning Data to Variables

- There is a caveat if the assignment data type is an Object (this includes intrinsic data type Object and the three key Excel object types).
- The **Set keyword** must be placed before the variable name.

```
Dim wksMyWorkSheet As WorkSheet  
Set wksMyWorkSheet = ActiveSheet
```

A Pitfall to Avoid When Declaring Variables

- By default, you are not required to declare variables in VBA
 - implicitly declared variables automatically get type Variant
 - spelling errors can lead to subtle bugs

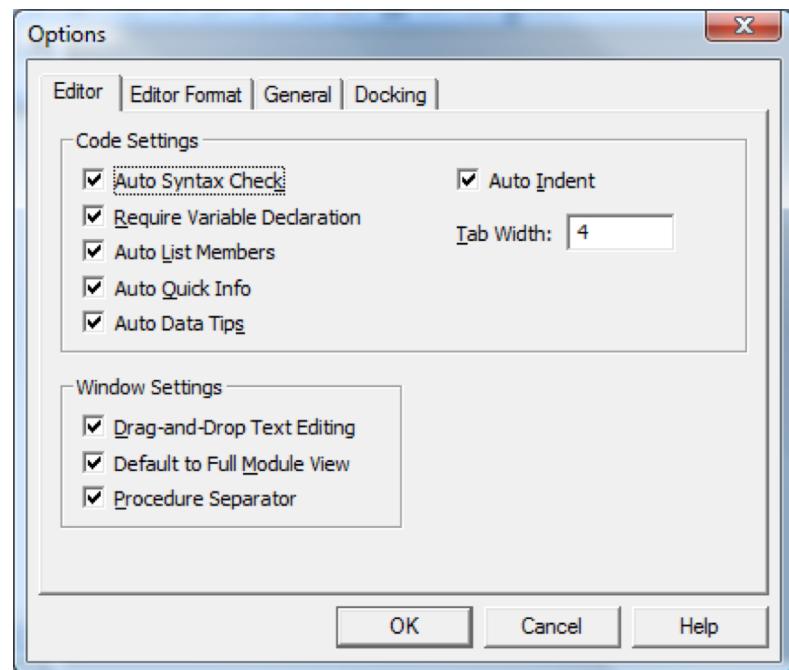
```
intBig = 1234
```

```
intBig = intBg/2
```

- Type the statement `Option Explicit` at the top of any module to enforce variable declaration

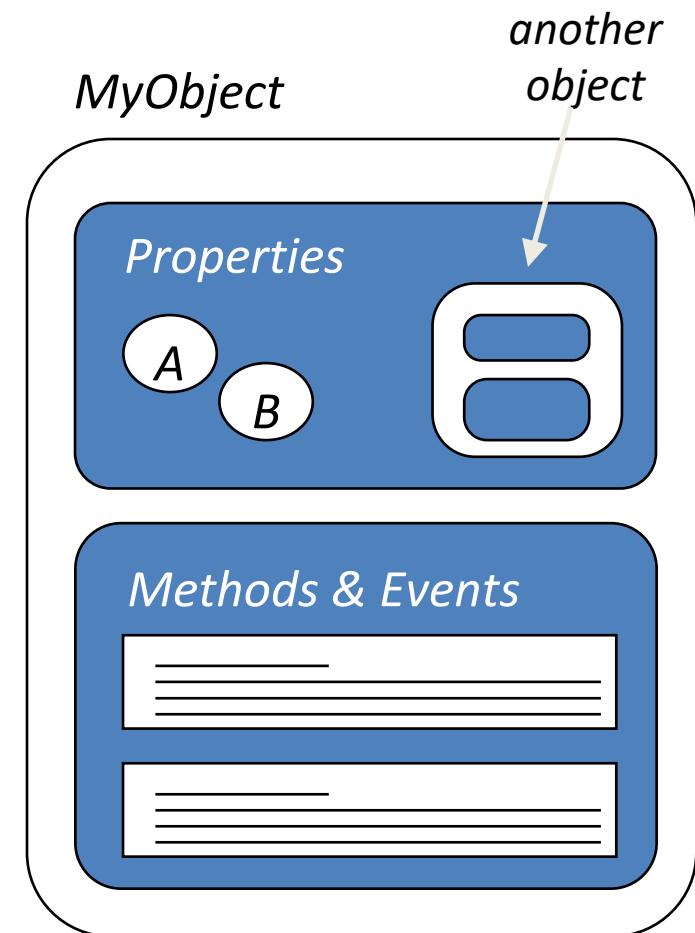
Do these now!

- Automatically set `Option Explicit` for all new modules by selecting the “Tools | Options” menu and checking “Require Variable Declaration” (in the VBE)



Introduction to Objects

- A collection of *properties*, *methods* and *events*, *encapsulated* as a single software unit
- Property
 - object data
- Method
 - object does something
- Event
 - object responds to something that happens



Understanding Objects

-- The Bicycle Object --



Property	Method	Event
Color	Go	OnPedal
Size	Stop	OnBrake
Weight	Jump	OnPullHandleBarsUp
	GoRight	OnSteerRight
	GoLeft	OnSteerLeft

Using an Object's Properties

- An object's properties store data related to the object.
- A property can be read/write (i.e. you can set its value and read its value) or read-only (i.e., you can only retrieve its value).
- An object's properties are accessed using *dot-notation*

```
SomeVariable = Object.Property  
Object.Property = SomeVariable
```
- For example, in Example 4, the macro stored code that looked like:

```
ActiveCell.FormulaR1C1 = "A"
```
- In this case, ActiveCell is the object and FormulaR1C1 is a read/write property of the object of type String

Using an Object's Methods

- An object's methods usually perform an action that modify the object's properties
- A method may take one or more parameter arguments and may even return a value.
- Object methods are like Subroutines or Functions encapsulated by an object.
- Like a property, a method is accessed using the dot notation

```
Object.Method [parameters]  
SomeVariable = Object.Method([parameters])
```

Using an Object's Methods (Cont.)

- In Example 4 we saw code that looked like `ActiveSheet.Paste`
- In this case, `ActiveSheet` is an object that represents the activated worksheet and `Paste` is a method that pastes the contents of the clipboard to the active sheet.
- We also saw code like `Sheets.Add`
- In this case, `Sheets` is an object that represents the collection of worksheets in the Excel workbook. `Add` is a method that adds a new worksheet to the collection.
- The `Add` method returns a value -- a reference to the newly added worksheet object.
- For example:

```
Dim wsDestinationSheet as Worksheet  
Set wsDestinationSheet = Sheets.Add
```

Commenting Out

- In the next sections we will be replacing existing lines of code with new code.
- We do this by *Commenting Out* the previous line(s) of code before adding new lines.
- The expression Comment Out refers to inactivating a line of VBA code by placing an apostrophe at the beginning of the line:
- Example: ‘print a

Example 5: Fix the Destination Worksheet

- In Example 4 data was pasted to the wrong Worksheet.
- The problem occurred because the destination Worksheet was referred to by name in the macro.
- Change the macro to do the following.

- Declare a variable of *type object* to hold a reference to a Worksheet.

```
Dim wsDestination As Worksheet
```

- Save a reference to the newly created Worksheet in the new variable.

```
'Sheets.Add
```

```
Set wsDestination = Sheets.Add()
```

- Use the worksheet reference variable to select the destination Worksheet instead of the collection and worksheet name

```
'Sheets("Sheet1").Select
```

```
wsDestination.Select
```

Example 5: Create a Worksheet Variable

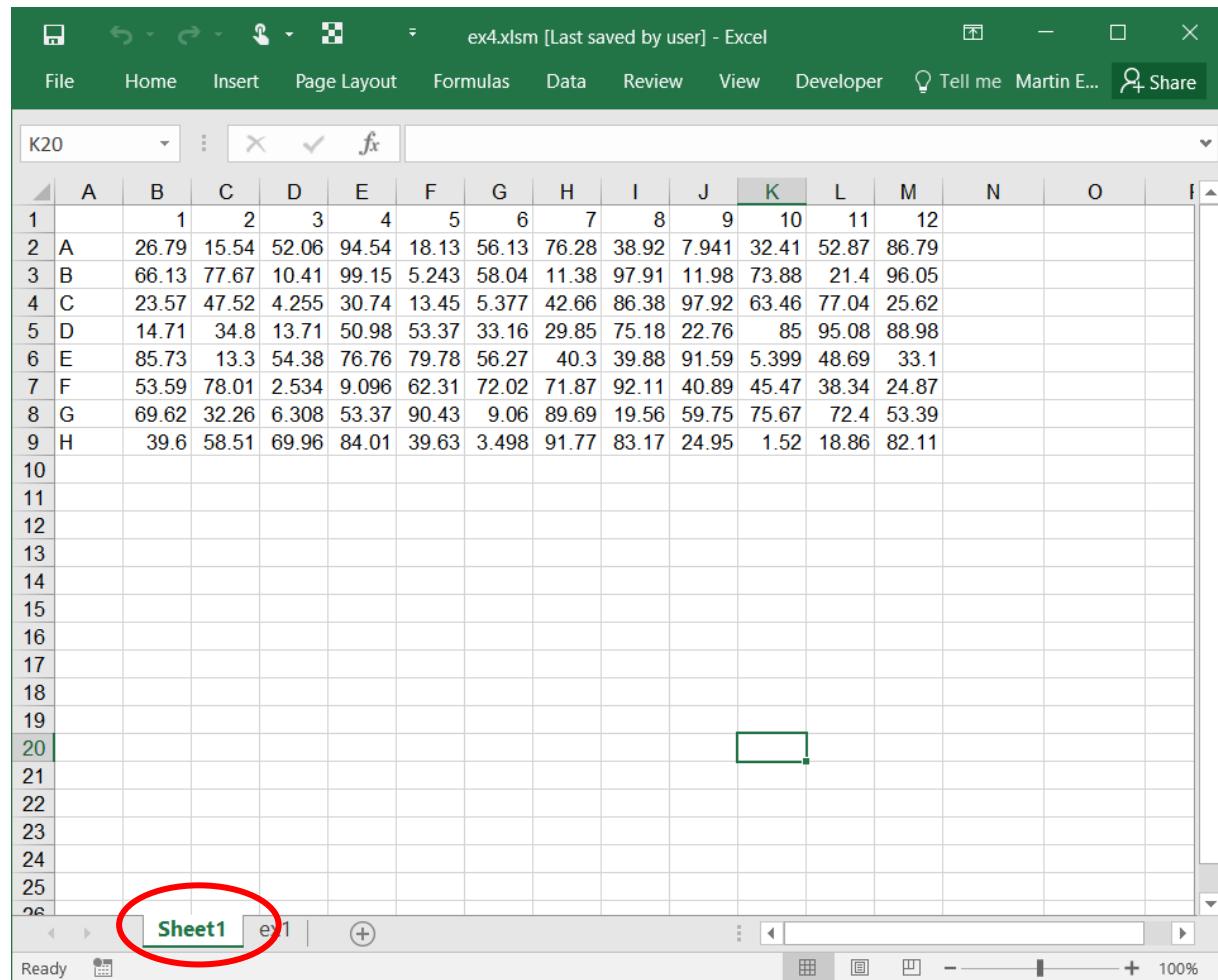
- Comment out the code that creates a new Worksheet and replace with code that declares a variable of type Worksheet (wsDestination), adds a new Worksheet and saves a reference in wsDestination.
- Throughout the macro, find and replace all references to Worksheet “Sheet1” with the new Worksheet variable, wsDestination.

```
'Sheets.Add  
Dim wsDestination As Worksheet  
Set wsDestination = Sheets.Add()  
  
'Sheets("Sheet1").Select  
wsDestination.Select
```

```
'Sheets("Sheet1").Select  
wsDestination.Select
```

Example 5: Rerun the Macro

- Rerun the edited macro.
- The data should now be copied to the proper destination sheet.



The screenshot shows a Microsoft Excel spreadsheet titled "ex4.xlsm [Last saved by user] - Excel". The data is organized into columns A through O and rows 1 through 20. The first few rows contain headers and numerical values. Row 20 is currently selected, indicated by a green background. The bottom navigation bar shows the tabs "Sheet1" (which is circled in red), "ex1", and a new sheet icon. The status bar at the bottom right shows "100%".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		1	2	3	4	5	6	7	8	9	10	11	12		
2	A	26.79	15.54	52.06	94.54	18.13	56.13	76.28	38.92	7.941	32.41	52.87	86.79		
3	B	66.13	77.67	10.41	99.15	5.243	58.04	11.38	97.91	11.98	73.88	21.4	96.05		
4	C	23.57	47.52	4.255	30.74	13.45	5.377	42.66	86.38	97.92	63.46	77.04	25.62		
5	D	14.71	34.8	13.71	50.98	53.37	33.16	29.85	75.18	22.76	85	95.08	88.98		
6	E	85.73	13.3	54.38	76.76	79.78	56.27	40.3	39.88	91.59	5.399	48.69	33.1		
7	F	53.59	78.01	2.534	9.096	62.31	72.02	71.87	92.11	40.89	45.47	38.34	24.87		
8	G	69.62	32.26	6.308	53.37	90.43	9.06	89.69	19.56	59.75	75.67	72.4	53.39		
9	H	39.6	58.51	69.96	84.01	39.63	3.498	91.77	83.17	24.95	1.52	18.86	82.11		
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															

Example 6: Fix the Source Worksheet

- In Example 4 data was copied from a named Worksheet (ex1).
- If a new data file is imported the source Worksheet will have a different name.
- Change the macro to do the following.

- Declare a variable to hold a reference to the source Worksheet.

```
Dim wsSource As Worksheet
```

- Save a reference to the current active Worksheet in the new variable.

```
Set wsSource = ActiveSheet
```

- Use the Worksheet reference to select the source Worksheet instead of the Worksheet name

```
' Sheets("ex1").Select
```

```
wsSource.Select
```

Example 6: Another Worksheet Variable

- Declare a new Worksheet variable (wsSource) and set it to the active Worksheet. The top of the macro now looks as follows.
- Throughout the macro, find and replace all references to Worksheet “ex1” with the new Worksheet variable, wsSource.

```
' Declare variables
Dim wsSource As Worksheet
Dim wsDestination As Worksheet

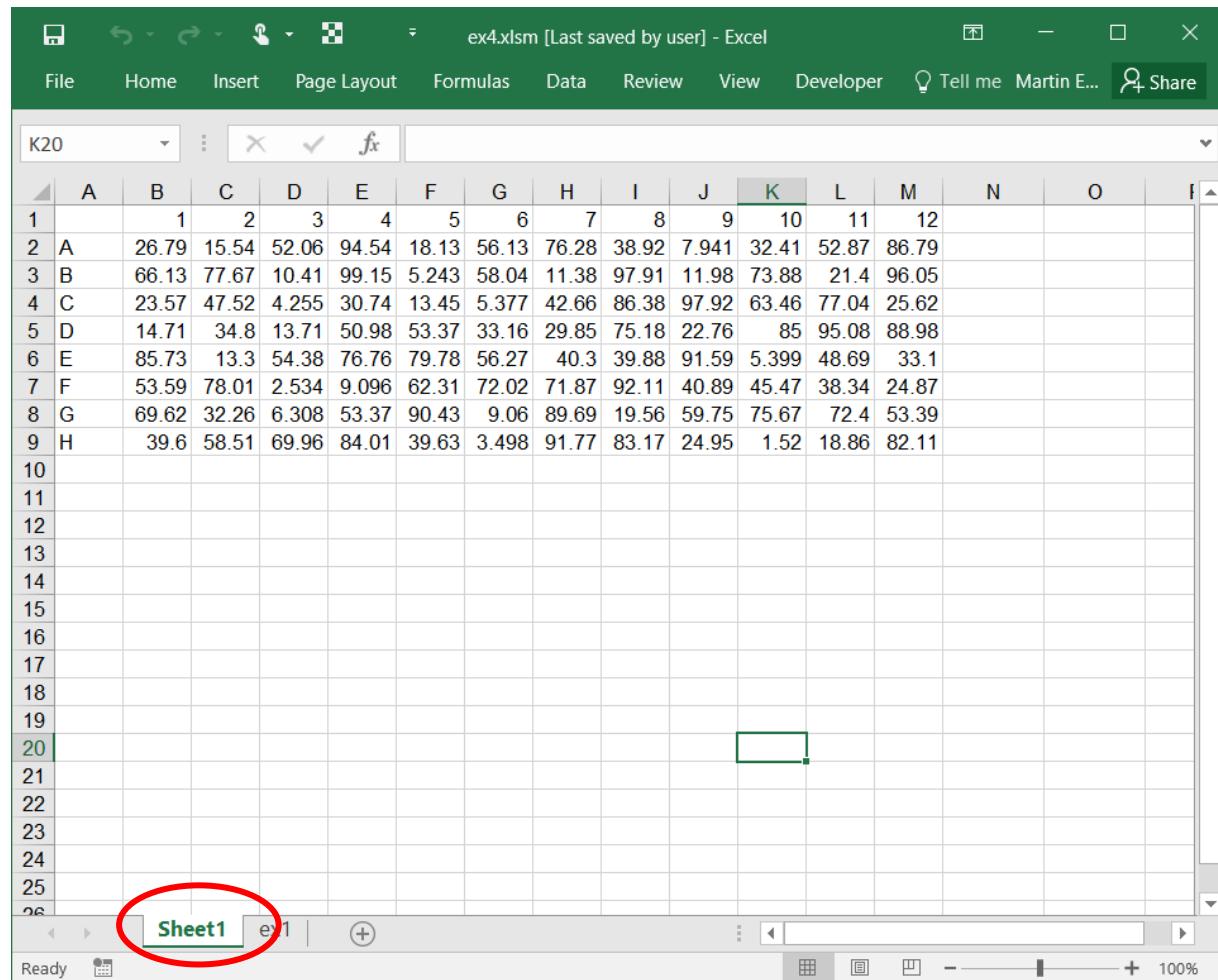
' Save a reference to the active sheet
Set wsSource = ActiveSheet

' Create a new sheet, save a reference
Set wsDestination = Sheets.Add()
wsDestination.Select
```

```
'Sheets("ex1").Select
wsSource.Select
```

Example 6: Rerun the Macro

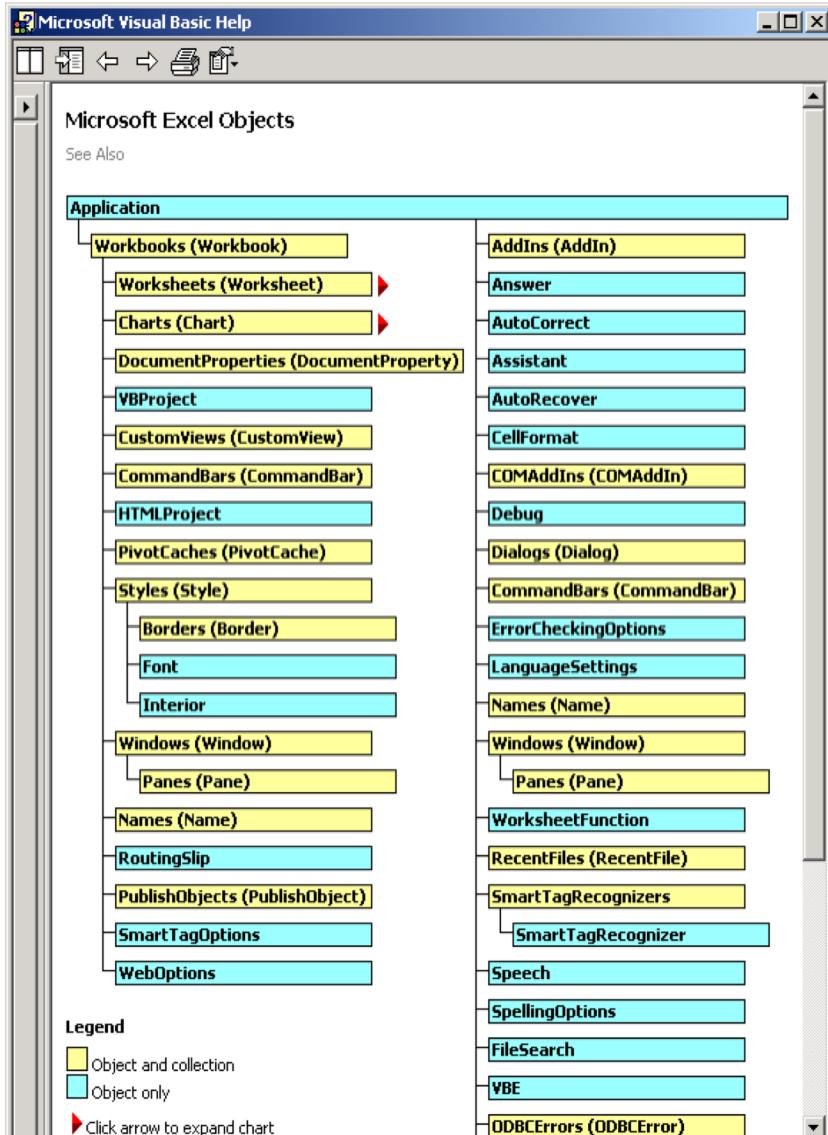
- Rerun the edited macro.
- Ensure that everything still works.
- Note that the macro still does not work correctly if it is run with an active worksheet other than the one with imported data.



The screenshot shows an Excel spreadsheet titled "ex4.xlsm [Last saved by user] - Excel". The ribbon menu is visible at the top. A data table is displayed in the main area, starting with row 1 and column A. The table contains 9 rows of data with columns labeled from A to K. Row 20 is highlighted in green, indicating it is the active sheet. The "Sheet1" tab is circled in red at the bottom left of the screen, while the "ex4" tab is also visible. The status bar at the bottom shows "Ready" and "100%".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		1	2	3	4	5	6	7	8	9	10	11	12		
2	A	26.79	15.54	52.06	94.54	18.13	56.13	76.28	38.92	7.941	32.41	52.87	86.79		
3	B	66.13	77.67	10.41	99.15	5.243	58.04	11.38	97.91	11.98	73.88	21.4	96.05		
4	C	23.57	47.52	4.255	30.74	13.45	5.377	42.66	86.38	97.92	63.46	77.04	25.62		
5	D	14.71	34.8	13.71	50.98	53.37	33.16	29.85	75.18	22.76	85	95.08	88.98		
6	E	85.73	13.3	54.38	76.76	79.78	56.27	40.3	39.88	91.59	5.399	48.69	33.1		
7	F	53.59	78.01	2.534	9.096	62.31	72.02	71.87	92.11	40.89	45.47	38.34	24.87		
8	G	69.62	32.26	6.308	53.37	90.43	9.06	89.69	19.56	59.75	75.67	72.4	53.39		
9	H	39.6	58.51	69.96	84.01	39.63	3.498	91.77	83.17	24.95	1.52	18.86	82.11		
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															

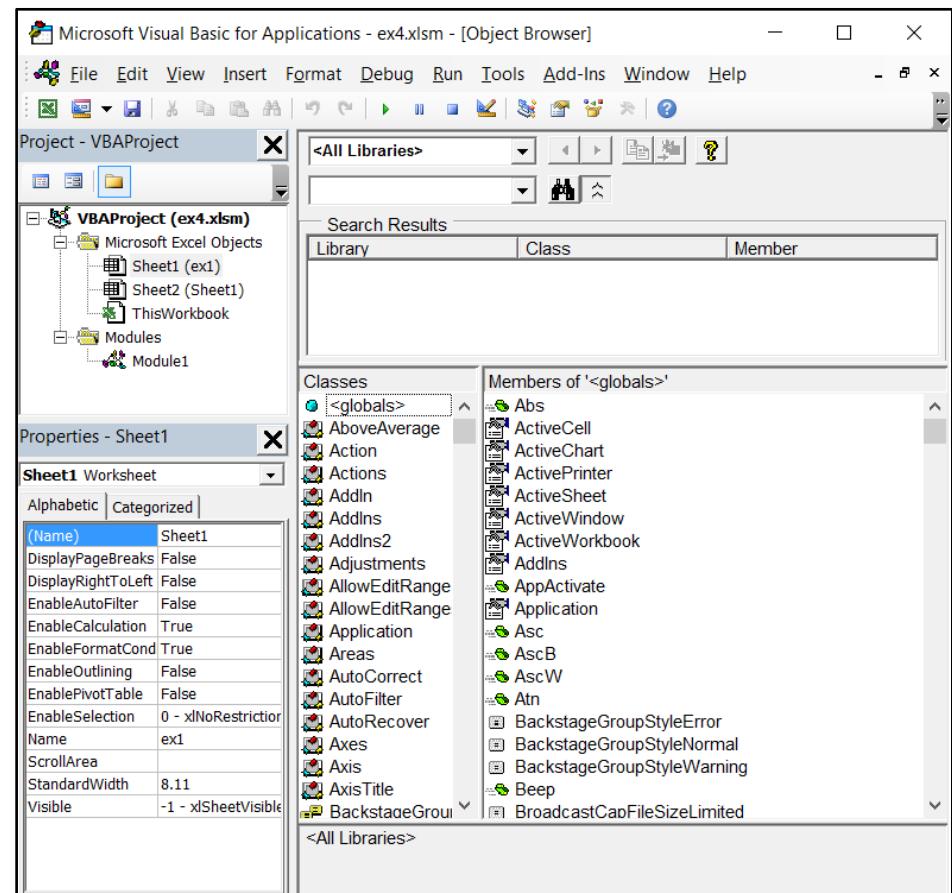
The Excel Object Hierarchy



- The Excel Object Model is a hierarchical representation of all of the objects that are exposed to VBA by the Excel application.
- Interaction with Excel objects is the key to programming Excel.
- Understanding the Excel object model and how to use it is the hardest part of programming Excel.

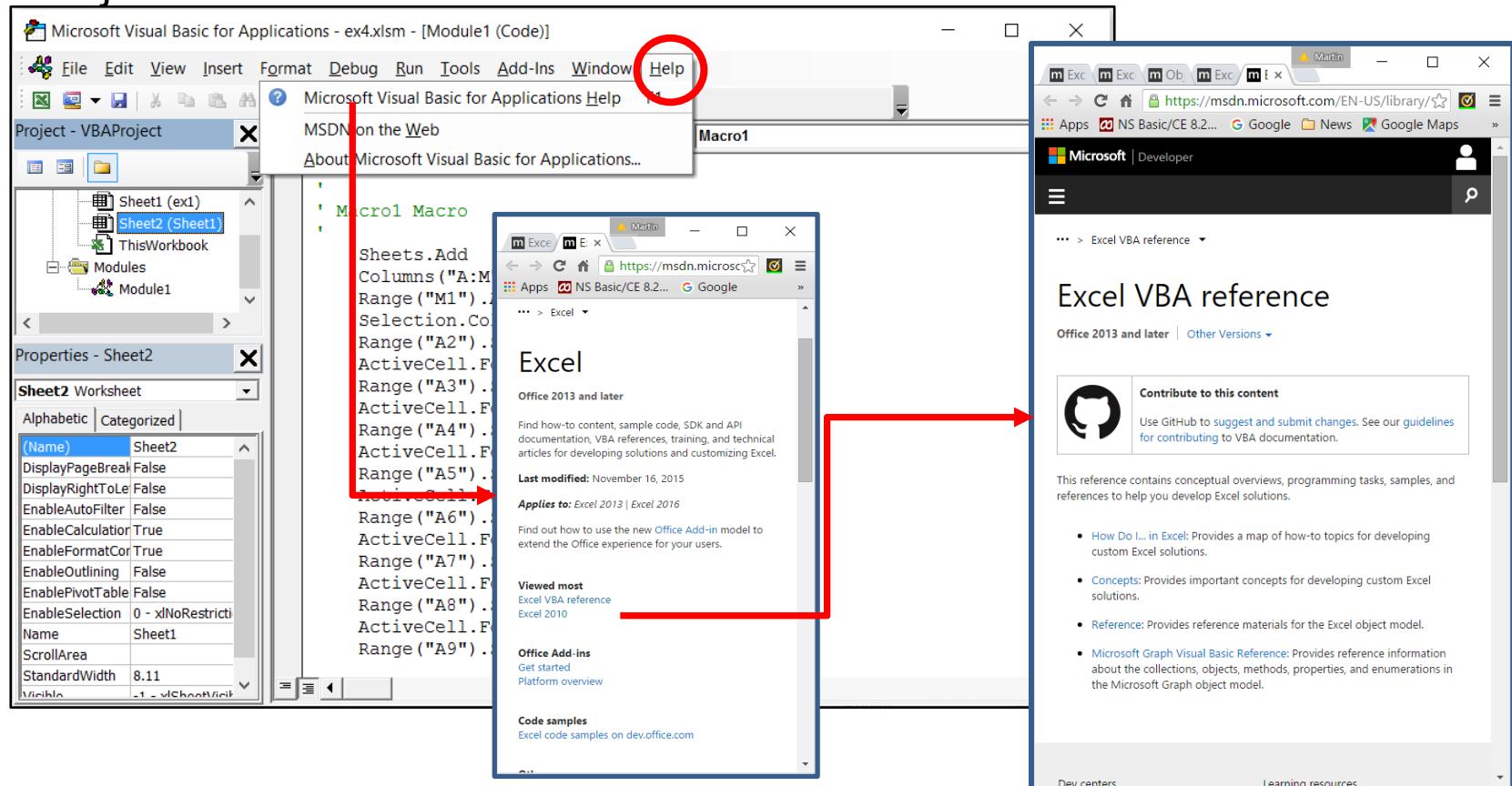
Learning the Excel Object Model

- The **object browser** is a dynamic view of all the objects, modules and constants in all the libraries available to VBA in Excel.
- It is the definitive resource for learning about object properties, methods and events.
- To display the object browser, choose the menu option “View | Object Browser” from the VBE menu.



Learning the Excel Object Model

- The **on-line help** system includes detailed information about the properties, methods, and events associated with each object and the contexts in which the objects are used.



Understanding The Range Object

- There is a Workbook object for each open Workbook in Excel.
- There is a Worksheet object for each Worksheet in an Excel Workbook.
- Unlike these other Excel objects, the Range object has no clearly identifiable interface widget in the Excel user interface.

A Range can represent a single cell...

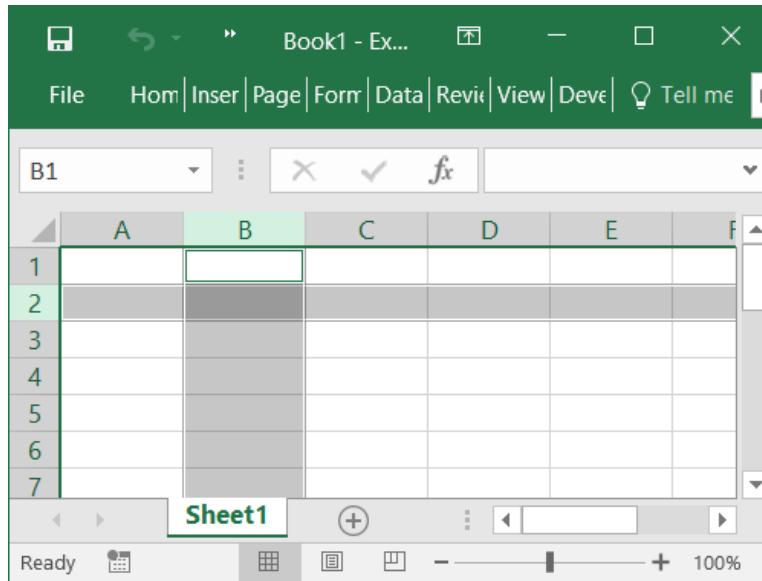
A screenshot of Microsoft Excel showing a single cell, A2, selected. The cell contains the value '1'. The formula bar above shows 'B2'. The ribbon menu is visible at the top, and the status bar at the bottom indicates 'Sheet1'.

A screenshot of Microsoft Excel showing a range of cells selected. The range A1:B2 is highlighted with a green border. Cell A1 contains the value '1', and cell B2 is empty. The formula bar above shows 'B2'. The ribbon menu is visible at the top, and the status bar at the bottom indicates 'Sheet1'.

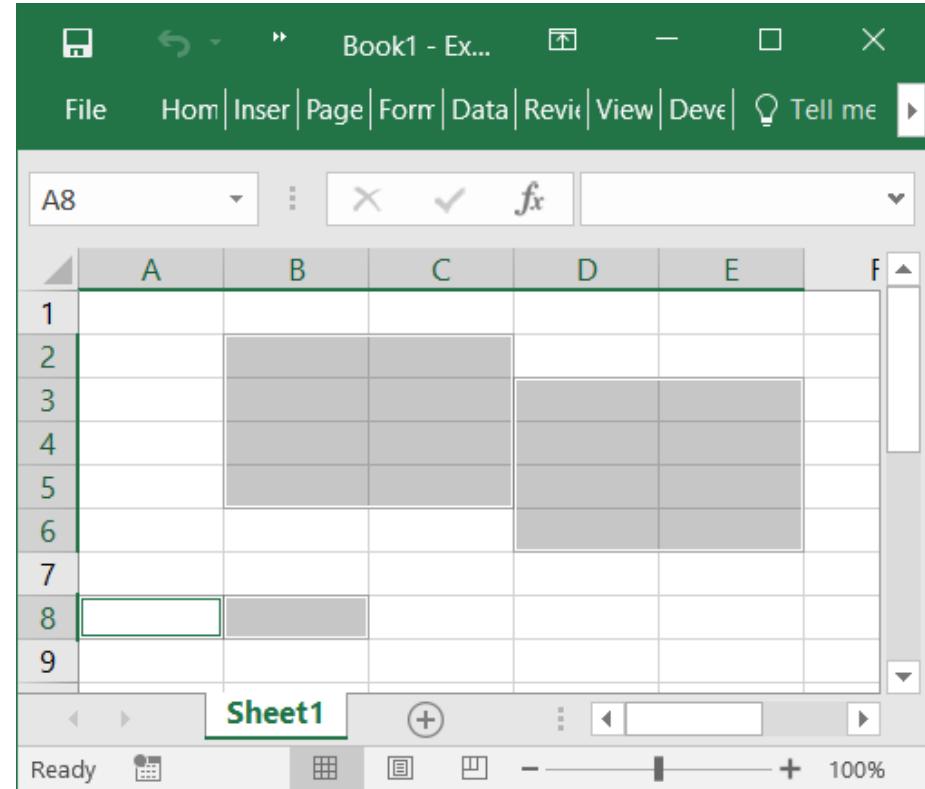
...or a collection of cells

Understanding The Range Object

A Range can be an entire row or column...



This screenshot shows a Microsoft Excel window with the title bar "Book1 - Ex...". The ribbon tabs are visible, and the active cell is B1. The entire column B is highlighted in green, indicating it is selected. The status bar at the bottom shows "Ready".



This screenshot shows a Microsoft Excel window with the title bar "Book1 - Ex...". The ribbon tabs are visible, and the active cell is A8. Three distinct ranges are selected: a single cell A8, a range from B2 to D6, and a range from B8 to C8. The status bar at the bottom shows "Ready".

... or a collection of multiple areas on a worksheet

Important Range Object *Properties*

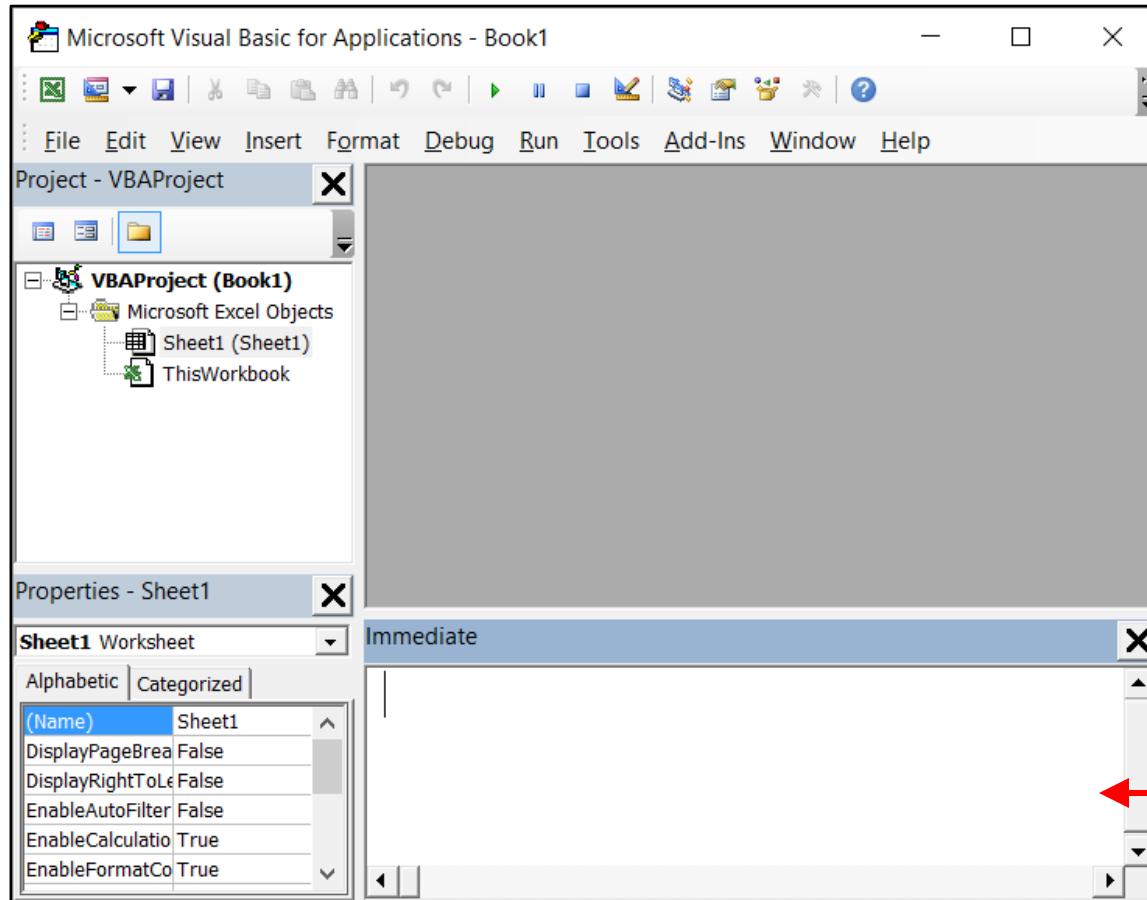
- **Formula**
 - Returns or sets the object's formula in A1-style notation.
- **Value**
 - Sets or gets the value of the specified cell. Returns an array if there is more than one cell in the Range object.
- **Font**
 - Returns a Font object that represents the font of the specified Range object.
- **Address**
 - Returns a string representing the range reference.
- **NumberFormat**
 - Sets or gets the format code for the object.

The Immediate Window

- Displays information resulting from debugging statements in your code or from commands typed directly into the window.
- The Immediate Window is useful for:
 - Testing or fixing (debugging) code
 - Printing variables while the macro is paused
 - Testing subroutines or functions
 - Viewing print statements while the macro is running
 - Placing a Debug.Print Statement in your code sends output to the immediate window

```
Debug.Print "Value of J is " & J
```

Immediate Window



To Access the
Immediate
Window:

1. Press **Alt+F11** to enter the Visual Basic Editor (VBE)
2. Press **Ctrl+G** to show the Immediate Window

Enter queries here

Getting and Putting Data into Cells

- Close Excel and re-open a fresh workbook.
- Enter “**It Gets**” in cell **B1** of Sheet1.
- Press **Alt+F11** to enter the VBE.
- Press **Ctrl+G** to display the Immediate Window.
- Enter the code on the next two slides into the Immediate window followed by the enter key to test the examples.
- After entering the statements you may print the value of the variable.

Getting Values from Cells

```
Print Cells(1,2).Value ' row 1 column 2 - can be variables
```

```
Print Sheet1.Cells(1,2).Value
```

```
Print ActiveSheet.Cells(1,2).Value
```

```
Print Worksheets("Sheet1").Range("B1").Value
```

```
Print ThisWorkbook.ActiveSheet.Range("B1").Value
```

```
Print ActiveCell.Value
```

The ? character can be used as shorthand for the Print statement

Putting Values Into a Cells

```
Cells(2,2).Value = "It Puts!"
```

```
Sheet1.Cells(2,2).Value = "It Puts!"
```

```
ActiveSheet.Cells(1,2).Value = "It Puts!"
```

```
Worksheets("Sheet1").Range("B2").Value = "It Puts!"
```

```
ThisWorkbook.ActiveSheet.Range("B2").Value = "It Puts!"
```

```
ActiveCell.Value = "It Puts!"
```

Example 7

- The recorded macro in example 6 faithfully repeats your actions, many of which can be performed more efficiently with VBA.
- In particular, with VBA you can:
 - set the contents of a cell without the need to select it first.
 - copy data directly from a source to a destination. There is no need to use the clipboard.
- Not only is the result more succinct, it runs faster with less flicker and less impact on the Excel user interface.

Example 7: Direct Copy

- Replace blocks like:

```
wsSource.Select          ' Select first sheet
Range("B1:B8").Select   ' Select the range
Selection.Copy          ' Copy the data
wsDestination.Select    ' Select second sheet
Range("B2").Select      ' Select target range
ActiveSheet.Paste        ' Paste the data
```

with a direct copy:

```
wsDestination.Range("B2:B9").Value = wsSource.Range("B1:B8").Value
```

- Replace blocks like:

```
Range("A2").Select
ActiveCell.FormulaR1C1 = "A"
```

with a direct assignment:

```
wsDestination.Range("A2").Value = "A"
```

Example 7: Direct Copy

- The new macro will be much easier to understand.
- It should look like the following.

```
Sub Macro1()

    ' Declare variables
    Dim wsSource As Worksheet
    Dim wsDestination As Worksheet

    ' Save a reference to the current sheet
    Set wsSource = ActiveSheet

    ' Create a new sheet and save a reference
    Set wsDestination = Sheets.Add()

    ' Set row a column labels
    wsDestination.Range("A2").Value = "A"
    wsDestination.Range("A3").Value = "B"
    wsDestination.Range("A4").Value = "C"
    wsDestination.Range("A5").Value = "D"
    wsDestination.Range("A6").Value = "E"
    wsDestination.Range("A7").Value = "F"
    wsDestination.Range("A8").Value = "G"
    wsDestination.Range("A9").Value = "H"
    wsDestination.Range("B1").Value = "1"
    wsDestination.Range("C1").Value = "2"
    wsDestination.Range("D1").Value = "3"
    wsDestination.Range("E1").Value = "4"
```

While working on your macro, look for and delete extraneous lines of code that are not useful, like selecting Worksheets, scrolling, etc.

(continued ...)

Example 7: Direct Copy

```
wsDestination.Range("F1").Value = "5"  
wsDestination.Range("G1").Value = "6"  
wsDestination.Range("H1").Value = "7"  
wsDestination.Range("I1").Value = "8"  
wsDestination.Range("J1").Value = "9"  
wsDestination.Range("K1").Value = "10"  
wsDestination.Range("L1").Value = "11"  
wsDestination.Range("M1").Value = "12"
```

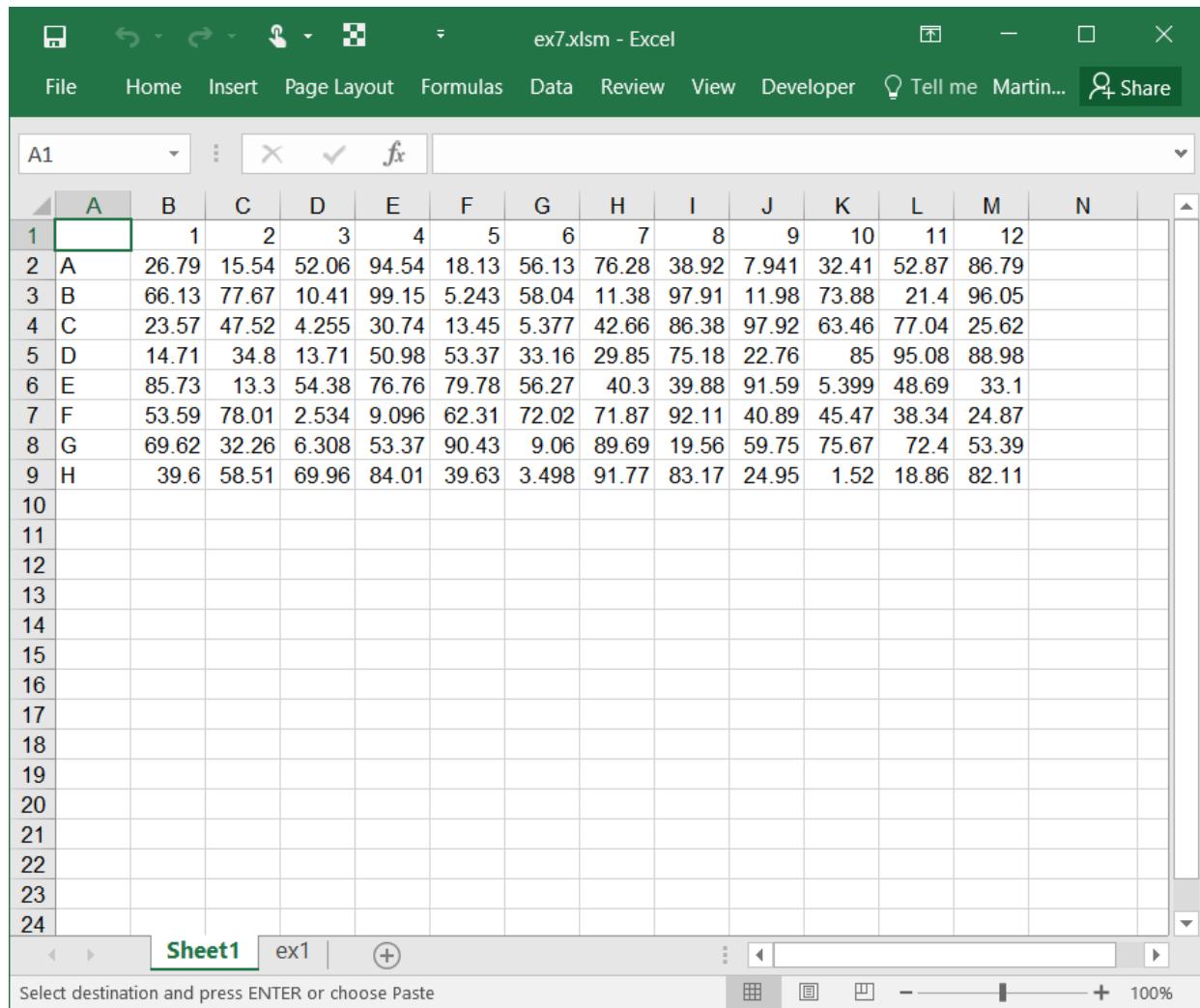
(... continued from previous slide)

```
' Move and reformat all data  
wsDestination.Range("B2:B9").Value = wsSource.Range("B1:B8").Value  
wsDestination.Range("C2:C9").Value = wsSource.Range("B9:B16").Value  
wsDestination.Range("D2:D9").Value = wsSource.Range("B17:B24").Value  
wsDestination.Range("E2:E9").Value = wsSource.Range("B25:B32").Value  
wsDestination.Range("F2:F9").Value = wsSource.Range("B33:B40").Value  
wsDestination.Range("G2:G9").Value = wsSource.Range("B41:B48").Value  
wsDestination.Range("H2:H9").Value = wsSource.Range("B49:B56").Value  
wsDestination.Range("I2:I9").Value = wsSource.Range("B57:B64").Value  
wsDestination.Range("J2:J9").Value = wsSource.Range("B65:B72").Value  
wsDestination.Range("K2:K9").Value = wsSource.Range("B73:B80").Value  
wsDestination.Range("L2:L9").Value = wsSource.Range("B81:B88").Value  
wsDestination.Range("M2:M9").Value = wsSource.Range("B89:B96").Value
```

End Sub

Example 7: Test the Macro

- Rerun the new macro and debug as necessary.
- Note the difference in the way it runs with less flicker.



The screenshot shows a Microsoft Excel window titled "ex7.xlsm - Excel". The ribbon menu includes File, Home, Insert, Page Layout, Formulas, Data, Review, View, Developer, Tell me, and Martin... Share. The active sheet is "Sheet1". A 9x12 grid of data is displayed, starting with row 1 and column A. The first few rows of data are:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		1	2	3	4	5	6	7	8	9	10	11	12	
2	A	26.79	15.54	52.06	94.54	18.13	56.13	76.28	38.92	7.941	32.41	52.87	86.79	
3	B	66.13	77.67	10.41	99.15	5.243	58.04	11.38	97.91	11.98	73.88	21.4	96.05	
4	C	23.57	47.52	4.255	30.74	13.45	5.377	42.66	86.38	97.92	63.46	77.04	25.62	
5	D	14.71	34.8	13.71	50.98	53.37	33.16	29.85	75.18	22.76	85	95.08	88.98	
6	E	85.73	13.3	54.38	76.76	79.78	56.27	40.3	39.88	91.59	5.399	48.69	33.1	
7	F	53.59	78.01	2.534	9.096	62.31	72.02	71.87	92.11	40.89	45.47	38.34	24.87	
8	G	69.62	32.26	6.308	53.37	90.43	9.06	89.69	19.56	59.75	75.67	72.4	53.39	
9	H	39.6	58.51	69.96	84.01	39.63	3.498	91.77	83.17	24.95	1.52	18.86	82.11	

Some VB String Manipulation Functions

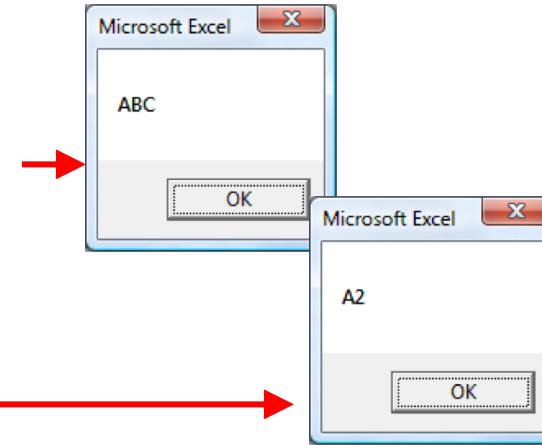
- “&”
 - String concatenation operator.

```
MsgBox "A" & "B" & "C"
```

```
Dim I As Integer  
I = 2
```

```
MsgBox "A" & I
```

' I is automatically converted to a string



String Manipulation Functions

- **Len(<string>)**
 - Returns the number of characters in the given string.
 - Useful for validating input.

1234567
Len ("abcdefg") returns 7

- **Trim(<string>)**
 - Returns a string with leading and trailing spaces removed.
 - Useful for conditioning input.

1234567
Trim(" abcdefg ") returns "abcdefg"

- **UCase(<string>), LCase(<string>)**
 - Converts case of characters in a string to all upper or lower case.
 - Useful for making responses/commands case-insensitive.

If UCase(Trim(strCommand)) = "ANALYZE" Then ...

String Manipulation Functions

- **InStr(<string1>, <string2>)**
 - Returns the position of the first occurrence of string2 in string1.

123456

`InStr("abcabc", "c")` returns 3

`InStr("abcabc", "d")` returns 0

- **InStrRev(<string1>, <string2>)**
 - Same as InStr only starts searching from end of string.

123456789

`InStrRev("173456789", "7")` returns 7

`InStrRev("173456789", "A")` returns 0

- **Replace(<inThisString>, <old>, <new>)**
 - Replace occurrences of old in the string inThisString with new.

`Replace("Data Set 1", " ", "_")` returns "Data_Set_1"

String Manipulation Functions

- `Mid(<string>, <start location>[, <number of characters>])`
 - Returns a specified number of characters beginning from the specified start location in the given string.

1234567

`Mid("abcdefg", 2, 3) returns "bcd"`

`Mid("abcdefg", 5) returns "efg"`

- `Left(<string>, <# characters>), Right (<string>, <# characters>)`
 - Returns a string with left-most (Left) or right-most (Right) no. of chars.

12345678901234567

`strFileName = "C:\Data\12345.dat"`

`Right(strFileName, Len(strFileName) - InStrRev(strFileName, "."))`
`returns "dat"`

`Right(strFileName, Len(strFileName) - InStrRev(strFileName, "\"))`
`returns "12345.dat"`

String Manipulation Functions

- **Asc(<string>)**
 - Returns an ASCII integer code for the first letter in a string.
`Asc("A")` returns 65
`Asc("abc123")` returns 97
- **Chr(<integer>)**
 - Returns a single character string encoded by the ASCII integer code
`Chr(65)` returns "A"
`Chr(97)` returns "a"

Data Type Conversion Functions

- **CStr(<variable>)**
 - Convert most variables to a string.
`CStr(10) returns "10" ' the string "10" not the number 10`
- **CLng(<variable>)**
 - If possible, converts a variable to a long integer.
`CLng("10") ' returns the number 10 as a long`
- **CDbl(<variable>)**
 - If possible, converts a variable to a double.
`CDbl("123") ' returns the number 123.0 as a double`
- **Val(<variable>)**
 - If possible, converts string variable to an integer, otherwise returns a zero
 - `Val("123.456NonNumericAfterNumeric") ' returns 123`
 - `Val("NonNumericBeforeNumeric123.456") ' returns a zero`
- There are conversion functions for all intrinsic data types.

String Manipulation Exercises

Using the functions on the previous pages, provide the answer to the following questions displaying it to the Immediate window using the Print statement. For example, to display the right most character of the string “Automate”, you would type the following and “e” would be displayed.

? Right(“Automate”,1)

- 1) What function(s) would you use to determine the # of characters in the string “ a b c d e ” after the leading and trailing spaces are removed and what is the answer?
- 2) What function(s) would be used to get the first character in the string “I love Excel”
- 3) What function(s) would be used to get the eighth character in the string “I love VBA” and what is the character?
- 4) What function(s) could be used to convert the following string to change “a2853b12” to “A2853B12”?

String Manipulation Exercises (*cont.*)

- 5) What function(s) would be used to change the string “VBA is not fun” to “VBA is fun”?
- 6) What function(s) would be used to get just the file path (folder) of the following string = "C:\Data\12345.dat"
- 7) Determine the word(s) resulting from the following sequential ASCII codes (**without** the ASCII chart) using one function several times in combination with the concatenation operator (Hint - slides 97 and 101)

83 117 112 101 114 32 66 111 119 108 32 76 73 73 73

- 8) Bonus

Write a function that will take any string of ASCII codes formatted as shown above and return the resulting ASCII characters as a string.

Example 8

- Thus far we have ignored the well label and assumed the measurements were in a known location. This may not always be true.
- It is better not to depend on the position of measurements in a file, but instead to use information in the file itself, in this case the well label, to determine the position in the destination Worksheet.
- We will need to convert a well label (in “A01” notation) to a row number and a column number. Let’s write some reusable routines to accomplish this.

Example 8: Convert Well to Row / Column

Plan:

1. Create a Function to do the following
 1. Remove first character from the well label and ensure that it is upper case. Use Left() and UCASE() functions.
 2. Calculate the ASCII character number and subtract 64. Use the Asc() function. This will convert “A” to 1, “B” to 2, ... the row number.
 3. Return the row number
2. Create a second function to do the following
 1. Remove all characters from the well label starting at the second character. Use the Mid() function.
 2. Convert the substring to an Integer or Long using CInt() or CLng() functions.
 3. Return the column number

Example 8: Convert Well to Row / Column

Enter the following two Functions into the macro

```
Function Well2Col(Well As String)

    Dim Col as Long
    ' Extract the Col Number from the Well Label
    Col = CLng(Mid(Well, 2))
    Well2Col = Col

End Function

Function Well2Row(Well As String)

    Dim Row as Long
    ' Extract the Row from the Well label
    Row = Asc(Ucase(Left(Well,1))) - 64
    Well2Row = Row

End Function
```

Example 8: Convert Well to Row / Column

Write a subroutine to test the functions

```
Sub Test()
    Dim Well As String, Row As Long, Col As Long

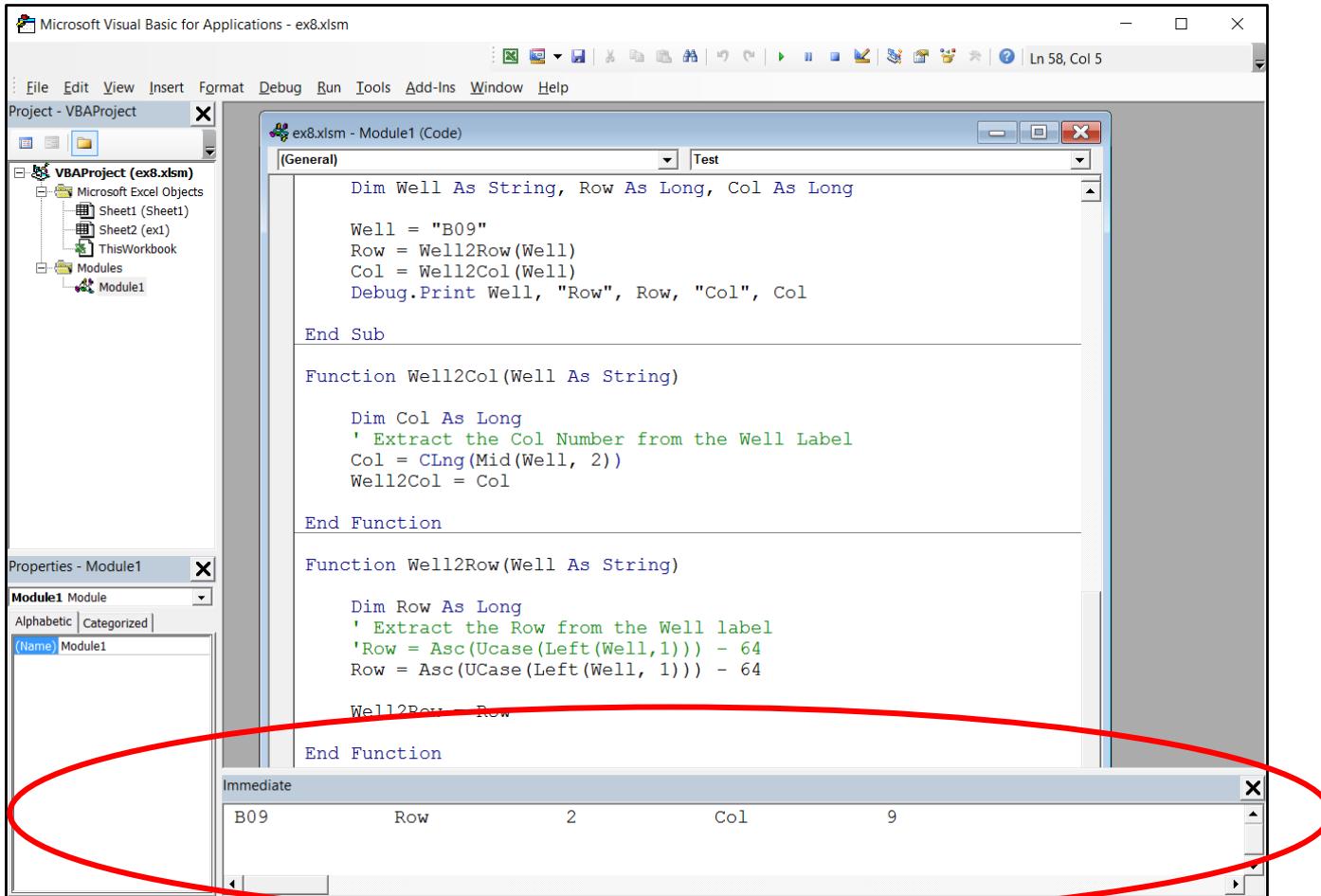
    Well = "B09"
    Row = Well2Row (Well)
    Col = Well2Col(Well)
    Debug.Print Well, "Row", Row, "Col", Col

End Sub
```

Modify Well and call Test from the Immediate Window.

Example 8: Convert Well to Row / Column

- Test your subroutine.



Looping

For-Next Statement

Repeats a group of statements a specified number of times.

Syntax:

```
For counter = start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next [counter]
```

Example:

```
For I = 1 to 10 Step 2
    MsgBox I
Next I
```

For-Next Statement

- The For-Next statement is useful when ...
 - You know the number of iterations ahead of time
 - You want to read data from cells by looping through a fixed number of rows (e.g. 96 rows of a spreadsheet)
- For-Next Statement is not useful when ...
 - You don't know the number of iterations ahead of time
 - Exiting the Loop is based on one or more conditions being met

Conditional Statements

If-Then-Else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax:

```
If condition Then  
    [statements]  
    [ElseIf condition-n Then  
        [elseifstatements]] ...  
    [Else  
        [elsestatements]]  
End If
```

Example:

```
If blnResult = True Then  
    MsgBox "Success!"  
End If
```

Example 9: Use Well Label to Copy Data

Replace the portion of the macro that copies measurements with a loop that uses well labels to determine target cell location.

Plan:

1. For all 96 rows in the data Worksheet ...
2. Copy well label and measurement from the current row
3. Convert the well label to row and column numbers
4. Insert the measurement into the target sheet using the Cells() function and the row and column values.

Example 9: Use Well Label to Copy Data

- Add the following declarations to the previous macro.

```
Dim I As Long, dblMeas As Double  
Dim strWell As String, lngRow As Long, lngCol As Long
```

- Replace the 12 measurement data assignments with the following loop.

```
For I = 1 To 96  
    ' Get well label and measurement value  
    strWell = wsSource.Cells(I, 1).Value  
    dblMeas = wsSource.Cells(I, 2).Value  
  
    ' Convert well label to a row and column  
    lngRow = Well2Row(strWell)  
    lngCol = Well2Col(strWell)  
  
    ' Write measurement to the proper location on the new sheet  
    wsDestination.Cells(lngRow + 1, lngCol + 1).Value = dblMeas  
Next I
```

Example 9: Use Well Label to Copy Data

- The new macro should look like the following:

```
Sub Macro1()
    ' Declare variables
    Dim wsSource As Worksheet
    Dim wsDestination As Worksheet
    Dim I As Long, dblMeas As Double
    Dim strWell As String, lngRow As Long, lngCol As Long

    ' Save a reference to the current sheet
    Set wsSource = ActiveSheet

    ' Create a new sheet and save a reference
    Set wsDestination = Sheets.Add()

    ' Move and reformat all data
    wsDestination.Range("A2").Value = "A"
    wsDestination.Range("A3").Value = "B"
    wsDestination.Range("A4").Value = "C"
    wsDestination.Range("A5").Value = "D"
    wsDestination.Range("A6").Value = "E"
    wsDestination.Range("A7").Value = "F"
    wsDestination.Range("A8").Value = "G"
    wsDestination.Range("A9").Value = "H"
```

(continued ...)

Example 9: Use Well Label to Copy Data

```
wsDestination.Range("B1").Value = "1"          (... continued from previous slide)
```

```
wsDestination.Range("C1").Value = "2"
```

```
wsDestination.Range("D1").Value = "3"
```

```
wsDestination.Range("E1").Value = "4"
```

```
wsDestination.Range("F1").Value = "5"
```

```
wsDestination.Range("G1").Value = "6"
```

```
wsDestination.Range("H1").Value = "7"
```

```
wsDestination.Range("I1").Value = "8"
```

```
wsDestination.Range("J1").Value = "9"
```

```
wsDestination.Range("K1").Value = "10"
```

```
wsDestination.Range("L1").Value = "11"
```

```
wsDestination.Range("M1").Value = "12"
```

```
For I = 1 To 96
```

```
    ' Get well label and measurement value
```

```
    strWell = wsSource.Cells(I, 1).Value
```

```
    dblMeas = wsSource.Cells(I, 2).Value
```

```
    ' Convert well label to a row and column
```

```
    lngRow = Well2Row(strWell)
```

```
    lngCol = Well2Col(strWell)
```

```
    ' Write measurement to the proper location on the new sheet
```

```
    wsDestination.Cells(lngRow + 1, lngCol + 1).Value = dblMeas
```

```
Next I
```

```
End Sub
```

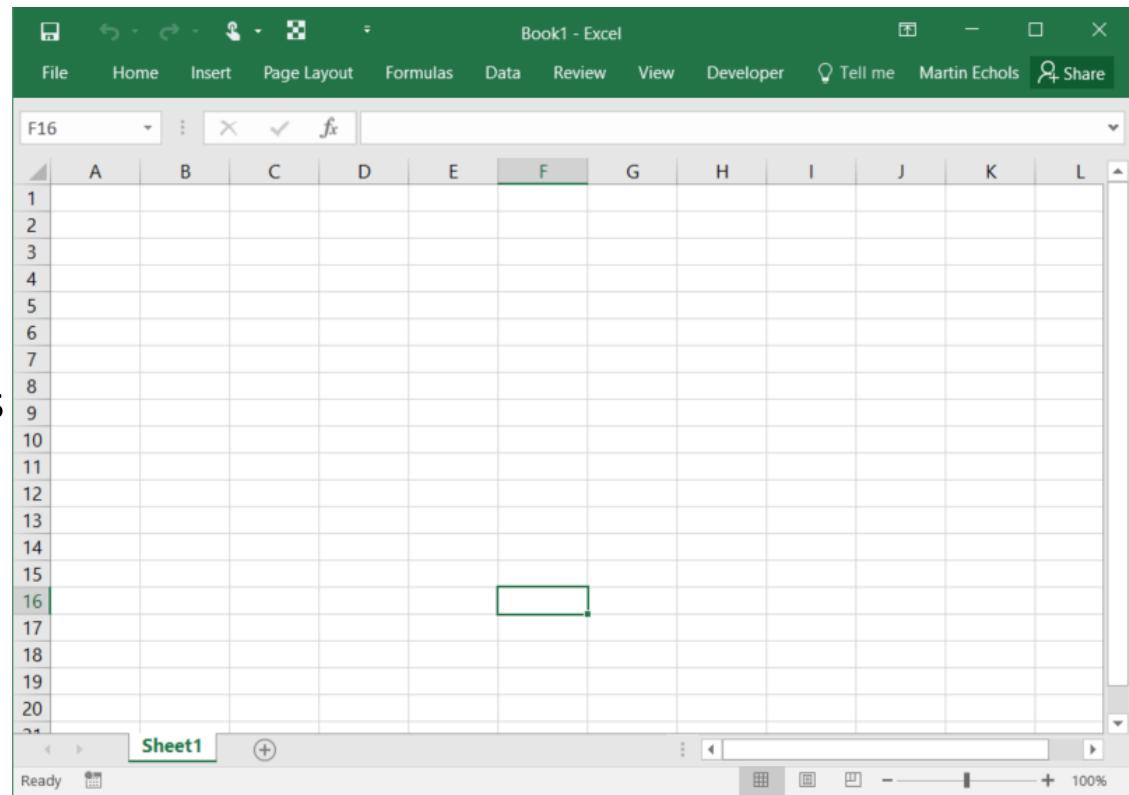
Example 9: Use Well Label to Copy Data

- Ensure that your Well2Row() and Well2Col() functions are still in your module.
- Test and debug your macro.
- Change the well labels in the source sheet and observe how the data are copied to different locations.

The image shows two Excel windows side-by-side. The top window, titled 'ex1.xls...', displays a table of data with columns A through E and rows 1 through 12. Row 1 contains labels: A01, 26.78536, C, D, E. Row 2 contains values: B01, 66.13404. A red arrow points from the value 'B01' to the cell C1 in the bottom window. The bottom window, titled 'ex7.xlsxm - Excel', also has columns A through E and rows 1 through 12. It contains a large grid of numerical values. A blue arrow points from the value 'A02' in the top window to the cell A1 in the bottom window. Both windows have their ribbon menus visible at the top.

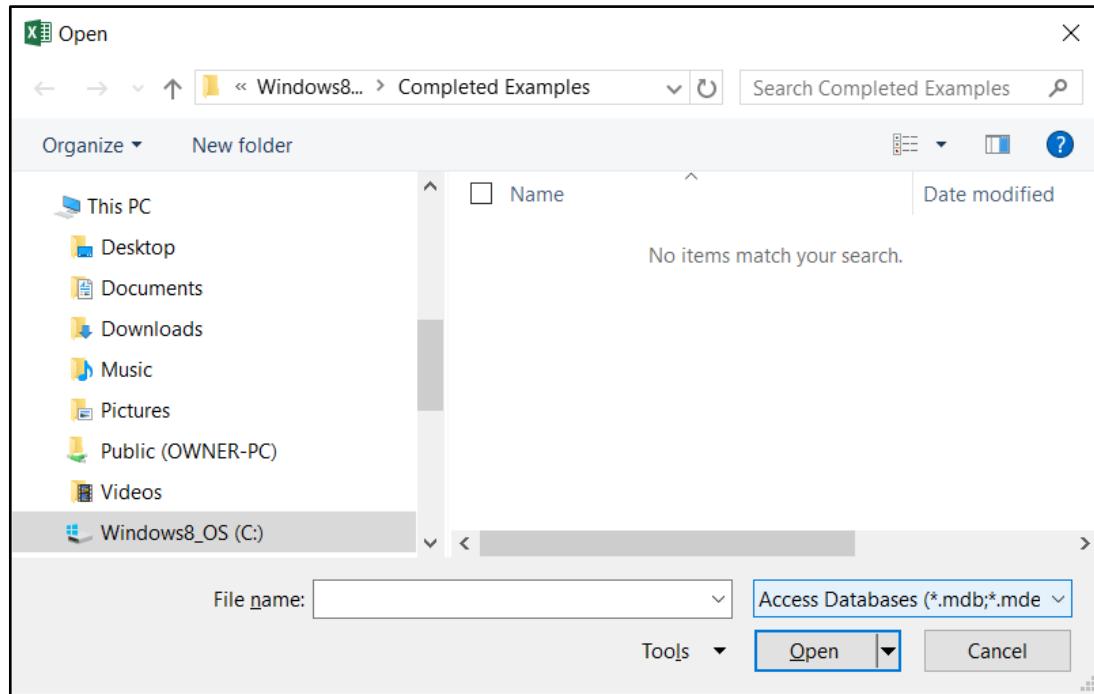
Application Object

- The Application object is the top-level object that represents the entire Excel application
- Application is available everywhere, and provides access to all collections, all application settings, etc.
- See VBA help for a complete listing of all Application properties and methods.



Application.GetOpenFileName

- The Application object has a method called GetOpenFileName that opens a standard File Dialog.
- The method takes multiple optional arguments, the most important being the first one called FileFilter.
- See VBA Help for all arguments.



GetOpenFileName FileFilter

- The file filter argument is a string that is a list of items separated by commas.
- The items come in pairs:
 - a description to display in the File Dialog File Type drop down
 - a wildcard file specification

```
' Get the name of the file to load
strFilter = "Text Files (*.txt), *.txt, All Files (*.*) , *.*"
fileName = Application.GetOpenfilename(strFilter)
```

GetOpenFileName Return Value

- GetOpenFileName returns a special data type called a Variant
- A Variant is a type that can hold *any* of the intrinsic VBA data types (Integer, String, Double, Boolean,)
- GetOpenFileName returns a Variant that is either:
 - *False* (*a Boolean*) -- indicating that the user clicked the Cancel button on the File Dialog.
 - *A String* – holding the full path to the file selected by the user.

Example 10: GetOpenFileName

- Write a subroutine that tests the GetOpenFileName method.
- Select a file and print the name to the Immediate window.

```
Sub LoadFile()
    ' Select a file name
    Dim FileName As Variant
    Dim strFilter As String

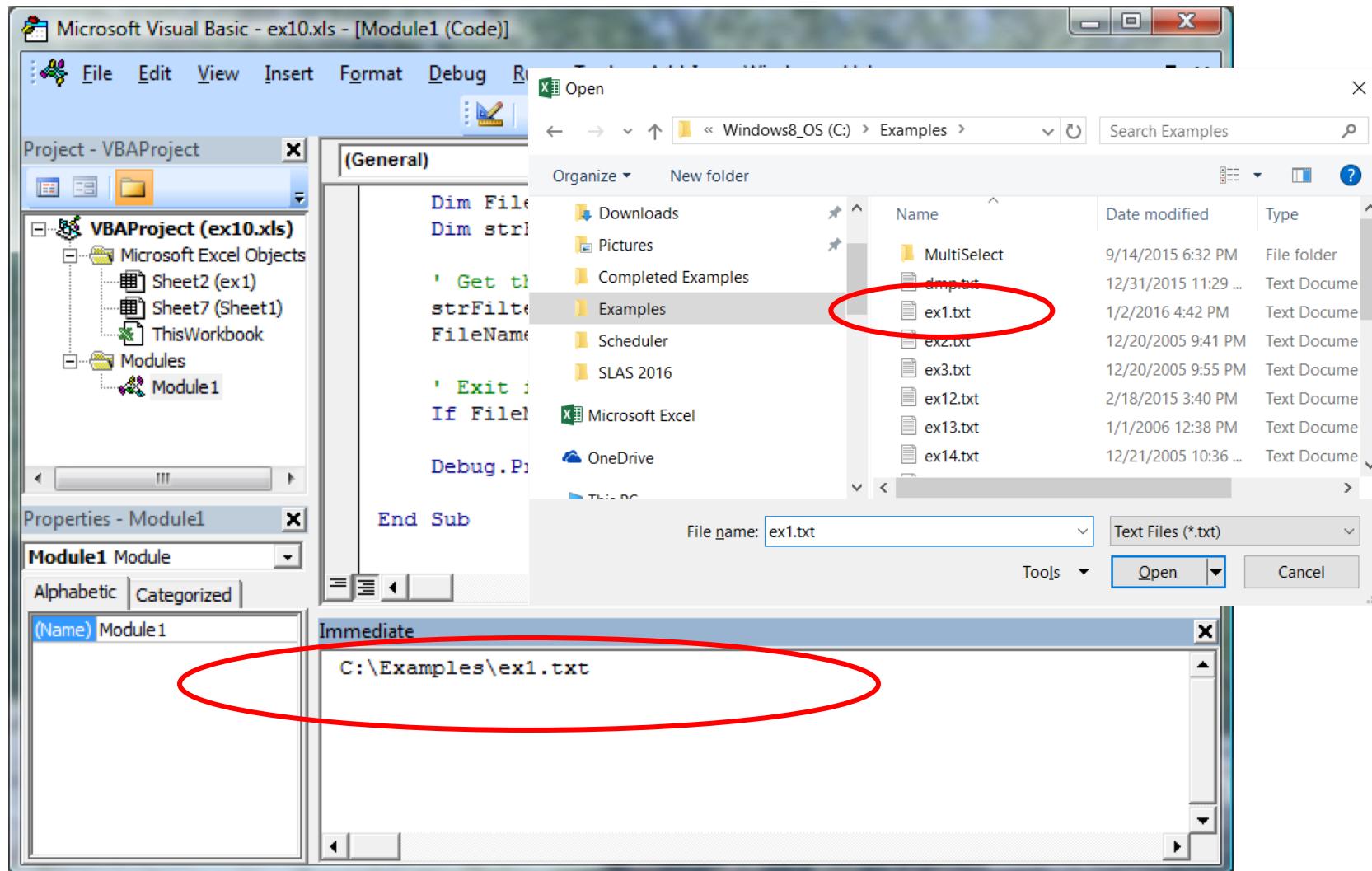
    ' Get the name of the file to load
    strFilter = "Text Files (*.txt), *.txt, All Files (*.*) ,*.*"
    FileName = Application.GetOpenfilename(strFilter)

    ' Exit if cancel selected
    If FileName = False Then Exit Sub

    Debug.Print FileName

End Sub
```

Example 10: GetOpenFileName



Files

VBA offers a full range of file manipulation capabilities ...

- Read and write text files (Sequential files)
- Randomly read and write records of arbitrary size to/from files (Random files)
- Randomly read and write binary data to/from files (Binary files)
- Create, delete, move, rename files
- Create, delete, move, rename folders
- Get file: extension, base name, parent folder, full path, size, ...
- Get drive: volume name, size, available space, state (ready or not), ...
- Get special folders (windows, temp, ...)
- Generate random file names
- ...

We cover Text Files (Sequential Files) only

Opening Files

- All files must be opened before they can be accessed.
- All open files are assigned a unique integer.
- A built-in function called FreeFile will return a unique file number.

```
intFileNum = FreeFile
```

- The Open statement is used to open a file.

```
Open pathname For mode As [#] filenumber
```

- *pathname* is the full path to the file
- *mode* is **Input** (for reading) **Output** (for writing), **Append** (for adding to the end of a file) and others ...
- *filenumber* is a unique number for the file, returned by FileNum

Closing Files

- All files must be closed when finished accessing them, or when their access mode is to be changed.

- The Close statement is used to close an open file.

Close [<#] *filenumber*] [, [<#] *filenumber*] .

• •

- Refer to the VBA help on the Open and Close statements for more detail.

Reading Files

- Reading the contents of a Sequential File can be accomplished using several VBA statements.
- The most common statements to read a file are the Input # and Line Input # statements
- Use Input # to read a **comma-delimited** list of data items from a file and assign them to a list of variables

Input #*filenumber, var1 [, var2 [, var3 ...]]]*

Example. If the statement:

Input #1, A, B, C

was used to read the following line from a file

12, 13, 14

then A=12, B=13 and C=14

Reading Files (cont'd)

- Use Line Input # to read an entire line from a file.
- A line is terminated with the Carriage-Return + Linefeed sequence of characters.
- Example. If the statement:

Line Input #1, A

was used to read the following line from a file

12, 13, 14

then A="12, 13, 14"

Writing Files

- Data can be written to a file with VBA using several different statements.
- The most common statement for writing data to a file is the Print # statements.

```
Print #filenumber, [item1, [item2, [item3,  
...]
```

- Example. The statement:

```
Print #1, 1, "A"
```

would write the following line to a file

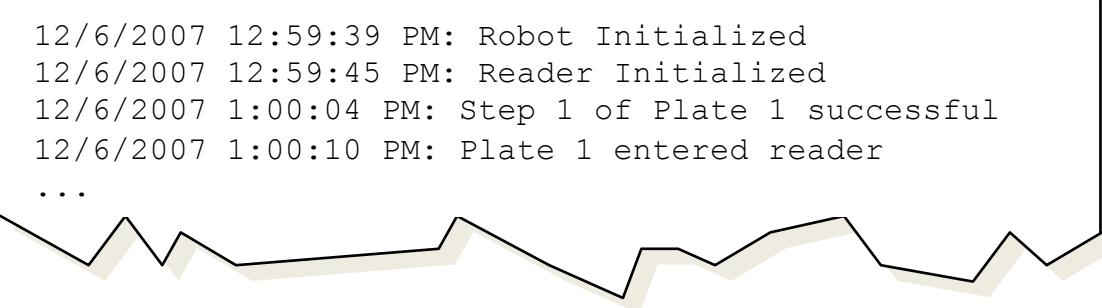
1 A

Example: Creating a Log File

It is frequently necessary to create a file to log the progress of an instrument or save historical data. Messages or data are added to log files intermittently over time.

To create and write to a log file ...

- Open a file for Append
 - Write a message to the file with a date-time stamp
 - Close the file
-
- With Append, the file is automatically created if does not already exist



```
12/6/2007 12:59:39 PM: Robot Initialized
12/6/2007 12:59:45 PM: Reader Initialized
12/6/2007 1:00:04 PM: Step 1 of Plate 1 successful
12/6/2007 1:00:10 PM: Plate 1 entered reader
...
...
```

Example: Creating a Log File

In a new workbook insert a new Code Module (Insert | Module) and enter the following code in the Module's Code Window.

```
Sub LogMessage(strMessage As String, strLogFile As String)
    ' Log a message to the log file.
    Dim intFileNum As Integer

    intFileNum = FreeFile
    Open strLogFile For Append As #intFileNum
    Print #intFileNum, CStr(Now) & ":" & strMessage
    Close #intFileNum

End Sub
```

In the Immediate Window, enter the following line of code and type [Enter]

```
LogMessage "Robot Initialized", "C:\Examples\test.log"
```

Inspect the contents of the file test.log now in the root of the C drive.

Add this function to any VBA program and call it to generate instant log files

Note: "Now" is a built-in VBA function that returns the current date and time.

Example 11: Read Lines from a File

Subroutine Plan:

1. Use GetOpenFileName to select a file name. Exit if cancelled.
2. Get an unused file number using the FreeFile function.
3. Open the file
4. Loop from 1 to 96
5. Read one line from the file
6. Print the line in the Immediate Window
7. Close the file

Example 11: Read Lines from a File

- The complete subroutine should look like the following.

```
Sub LoadFile()
    ' Read all the lines in a text file
    Dim FileName As Variant
    Dim strFilter As String
    Dim intFileNum As Integer
    Dim I As Long
    Dim strLine As String

    ' Get the name of the file to load
    strFilter = "Text Files (*.txt), *.txt, All Files (*.*),*.*"
    FileName = Application.GetOpenFilename(strFilter)

    ' Exit if cancel selected
    If FileName = False Then Exit Sub
```

(continued ...)

Example 11: Read Lines from a File

(... continued from previous slide)

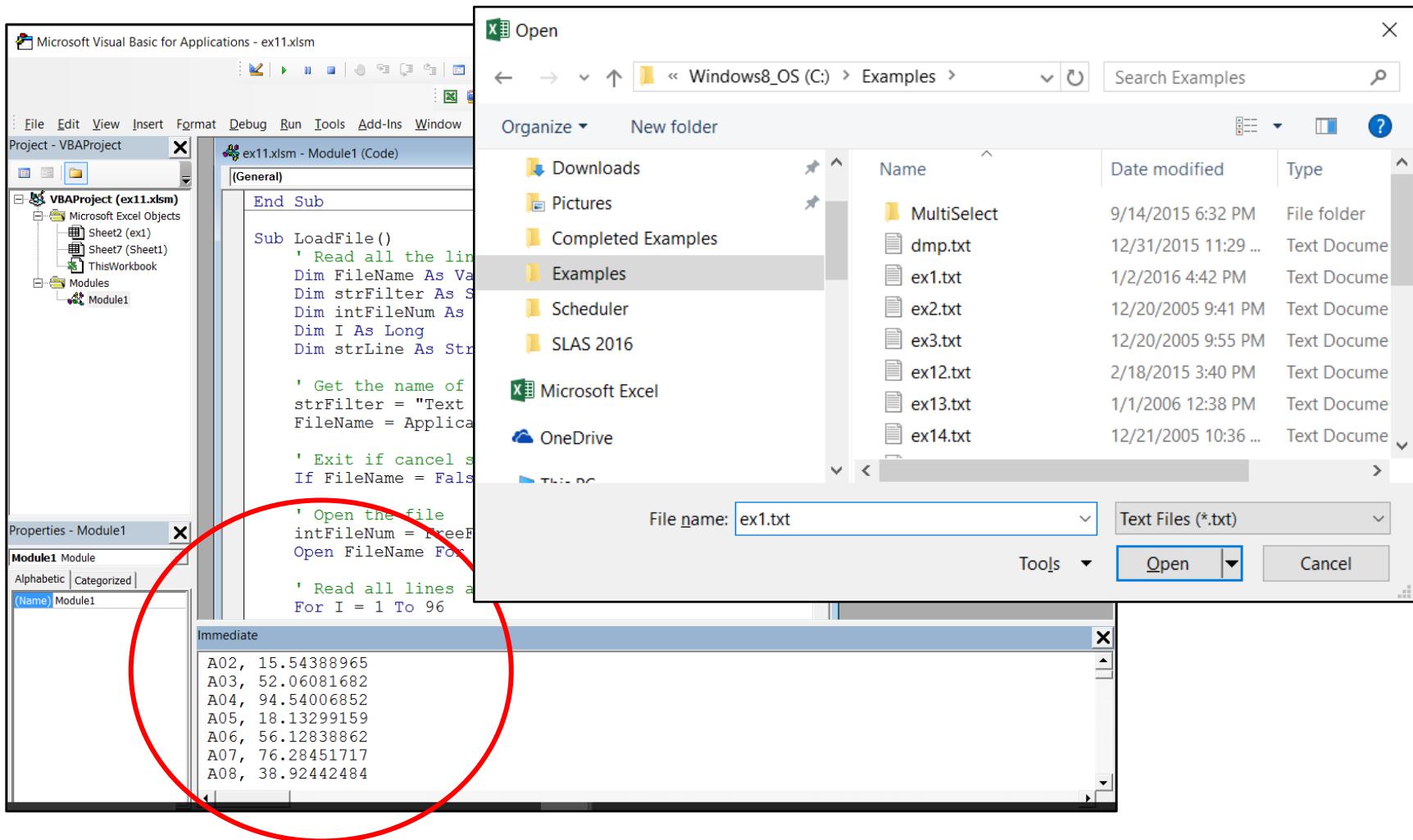
```
' Open the file
intFileNum = FreeFile
Open FileName For Input As #intFileNum

' Read all lines and print to Immediate window
For I = 1 To 96
    Line Input #intFileNum, strLine
    Debug.Print strLine
Next I

' Close the window
Close #intFileNum

End Sub
```

Example 11: Read Lines from a File



VBA Arrays

- A stored series of variables referenced by an index. All elements of a single array are the same data type.

```
Dim arrInts(10) As Integer  
Dim arrMeasure(8,12) As Double  
Dim arrNames(20) As String
```

- Elements of Variant arrays can hold any data types that can be stored in a Variant. They can be mixed.

```
Dim arrStuff(3) As Variant  
arrStuff(0) = 10  
arrStuff(1) = 10.0  
arrStuff(2) = "10"
```

- Can specify initial index in declaration. Default starting index is 0.

```
Dim arrInts(3 to 10) As Integer
```

- Arrays can have multiple dimensions

```
Dim arrMeasure(-4 to 8, 6 to 12) As Double  
Dim arrMatrix(1 to 10, 1 to 10) As Double
```

- Arrays of unknown size can be declared with no dimensions

```
Dim arrParts() As String
```

More String Manipulation Functions

- **Join(<stringArray>[, <delimiter>])**
 - Concatenate an array of strings into one string.
 - Optionally separate each element with a <delimiter>.
 - Use to quickly generate a delimited string.

```
For I = 0 To 4
    strArr(I) = CStr(I * 10)
Next I
strJoin = Join(strArr(), ",")
```

- Generates "0,10,20,30,40"

- **Split(<string>[, <delimiter>])**
 - Split a string into a zero-based array of substrings
 - Substrings are separated by a delimiter string
 - Use to quickly parse a delimited string.

More On Split

Example: Converting a tab-delimited string into a string array

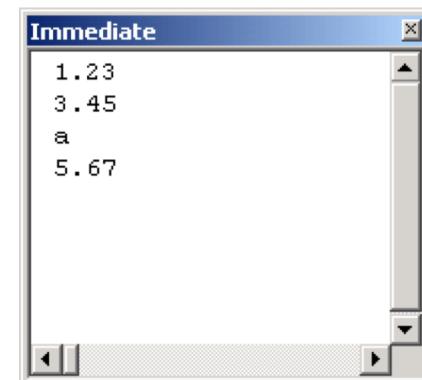
```
Sub TestSplit()
    Dim strInput As String
    Dim arrData() As String      ' An Uninitialized String Array
    Dim I As Long

    ' Build a tab-delimited string
    ' This is exactly like what you would get from Excel
    strInput = "1.23" & vbTab & "3.45" & vbTab & "a" & vbTab & "5.67"

    ' Split the string on the tab character
    arrData() = Split(strInput, vbTab)

    ' Print results
    Debug.Print arrData(0)
    Debug.Print arrData(1)
    Debug.Print arrData(2)
    Debug.Print arrData(3)

End Sub
```



Example 12: Read Data From the File

- In this example we will modify our macro to read data from a file rather than copy imported data from a Worksheet.
- This approach is cleaner, and avoids the need to import the data file.
- To accomplish this we will merge the macros from Examples 9 and 11.
- Note that this will eliminate the need to reference the Worksheet with imported data (wsSource).

Example 12: Read Data From the File

Combined Macro Plan:

1. Select a file to load with GetOpenFileName.
2. Add a new Worksheet to the Workbook.
3. Write all row and column headers to the new Worksheet.
4. Open the named file for Input.
5. Loop from 1 to 96.
6. Read one line from the file.
7. Parse the line into well label and measurement value using Split.
8. Convert the well label to row and column values.
9. Write the measurement value to proper cell on the new Worksheet.
10. When the loop completes, close the file.

Example 12: Read Data From the File

- The complete macro should look like the following.

```
Sub Macro1()
```

```
    ' Read all the lines in a text file
    Dim FileName As Variant
    Dim strFilter As String
    Dim intFileNum As Integer
    Dim I As Long, dblMeas As Double
    Dim strLine As String
    Dim strParts() As String
    Dim strWell As String
    Dim lngRow As Long, lngCol As Long
    Dim wsDestination As Worksheet
```

Combined
Declarations

```
    ' Get the name of the file to load
    strFilter = "Text Files (*.txt), *.txt, All Files (*.*) ,*.*"
    FileName = Application.GetOpenFilename(strFilter)
```

Example 11

```
    ' Exit if cancel selected
    If FileName = False Then Exit Sub
```

```
    ' Create a new sheet and save a reference
    Set wsDestination = Sheets.Add()
```

Example 9

(continued ...)

Example 12: Read Data From the File

(... continued from previous slide)

```
' Write row and column headers
wsDestination.Range("A2").Value = "A"
wsDestination.Range("A3").Value = "B"
wsDestination.Range("A4").Value = "C"
wsDestination.Range("A5").Value = "D"
wsDestination.Range("A6").Value = "E"
wsDestination.Range("A7").Value = "F"
wsDestination.Range("A8").Value = "G"
wsDestination.Range("A9").Value = "H"
wsDestination.Range("B1").Value = "1"
wsDestination.Range("C1").Value = "2"
wsDestination.Range("D1").Value = "3"
wsDestination.Range("E1").Value = "4"
wsDestination.Range("F1").Value = "5"
wsDestination.Range("G1").Value = "6"
wsDestination.Range("H1").Value = "7"
wsDestination.Range("I1").Value = "8"
wsDestination.Range("J1").Value = "9"
wsDestination.Range("K1").Value = "10"
wsDestination.Range("L1").Value = "11"
wsDestination.Range("M1").Value = "12"
```

Example 9

(continued ...)

Example 12: Read Data From the File

(... continued from previous slide)

```
' Open the file
intFileNum = FreeFile
Open FileName For Input As #intFileNum

' Read all lines, parse and write to new sheet
For I = 1 To 96
    Line Input #intFileNum, strLine
```

Example 11

```
' Parse the data
strParts = Split(strLine, ",")
```

New

```
strWell = strParts(0)
dblMeas = CDbl(strParts(1))

lngRow = Well2Row(strWell)
lngCol = Well2Col(strWell)
' Write the measurements to the target sheet
wsDestination.Cells(lngRow + 1, lngCol + 1).Value = dblMeas
```

Example 9

```
Next I
```

Example 11

```
' Close the file
Close #intFileNum
```

```
End Sub
```

Example 12: Read Data From the File

- Test and debug your new macro.

The screenshot shows a Microsoft Excel interface with a file import dialog box open over a worksheet containing data.

Excel Worksheet:

	A	B	C	D	E	F	G	H
1		1	2	3	4	5	6	7
2	A	26.78536	15.54389	52.06082	94.54007	18.13299	56.12839	76.28452
3	B	66.13404	77.67299	10.41474	99.14567	5.243083	58.03729	11.37578
4	C	23.57128	47.52201	4.25543	30.74329	13.45405	5.377459	42.66457
5	D	14.70976	34.80218	13.70871	50.97807	53.37121	33.15702	29.84616
6	E	85.72777	13.30383	54.38157	76.76285	79.78222	56.27236	40.29638
7	F	53.59418	78.00745	2.533874	9.096043	62.31232	72.01791	71.87339
8	G	69.61801	32.26121	6.307792	53.37164	90.43056	9.060418	89.69256
9	H	39.59825	58.51043	69.95962	84.01354	39.63293	3.498132	91.77454
10								
11								

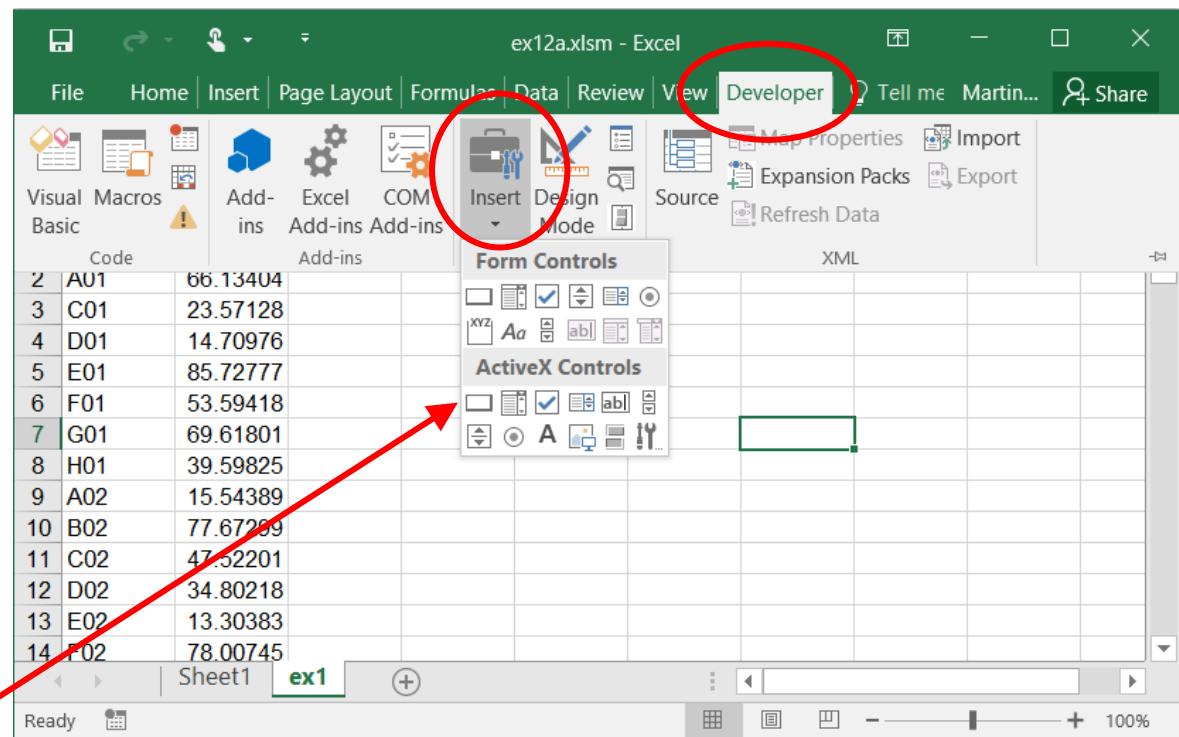
File Import Dialog:

The dialog box shows a list of files in the "Completed Examples" folder, all named "ex1.txt" through "ex18b.txt". The "Text Files (*.txt)" filter is selected, and the "Open" button is highlighted.

Note: Note imported data Worksheet!

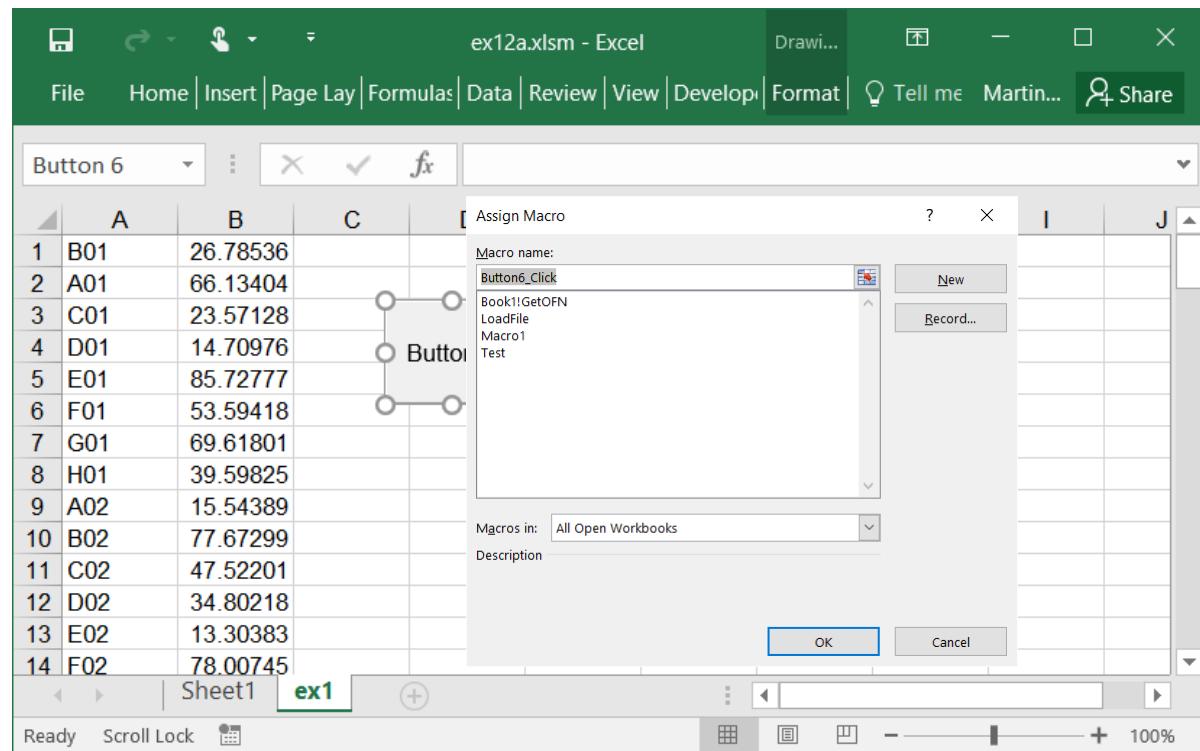
Example 12a: Trigger Macro with a Button

- To make life a little easier, now that your data is being read from a file directly, you can run the macro with a button
- Display the Forms toolbar by selecting the Developer Tab, then selecting the Insert button.
- The Controls bar will appear.



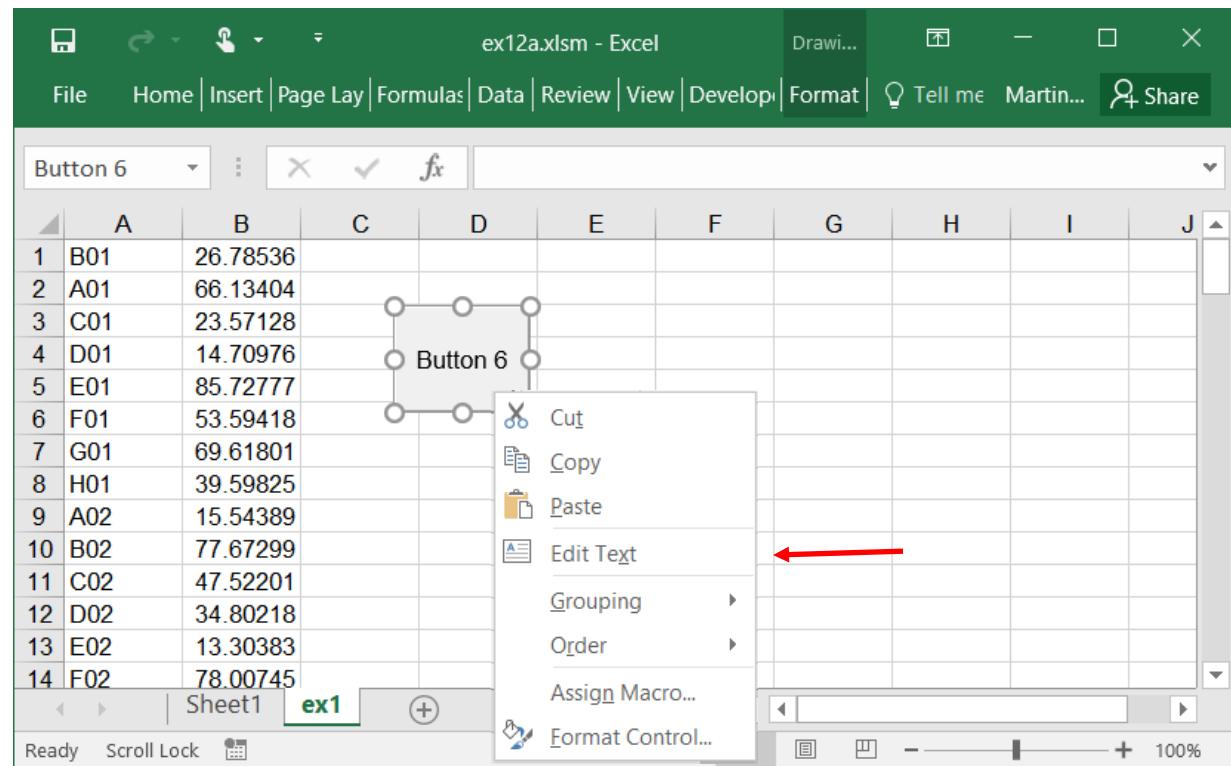
Example 12a: Trigger Macro with a Button

- Click the button tool on the **Form Controls** toolbar and sketch it on your Worksheet
- Select the name of your macro, which will be called when the button is clicked and click [OK]



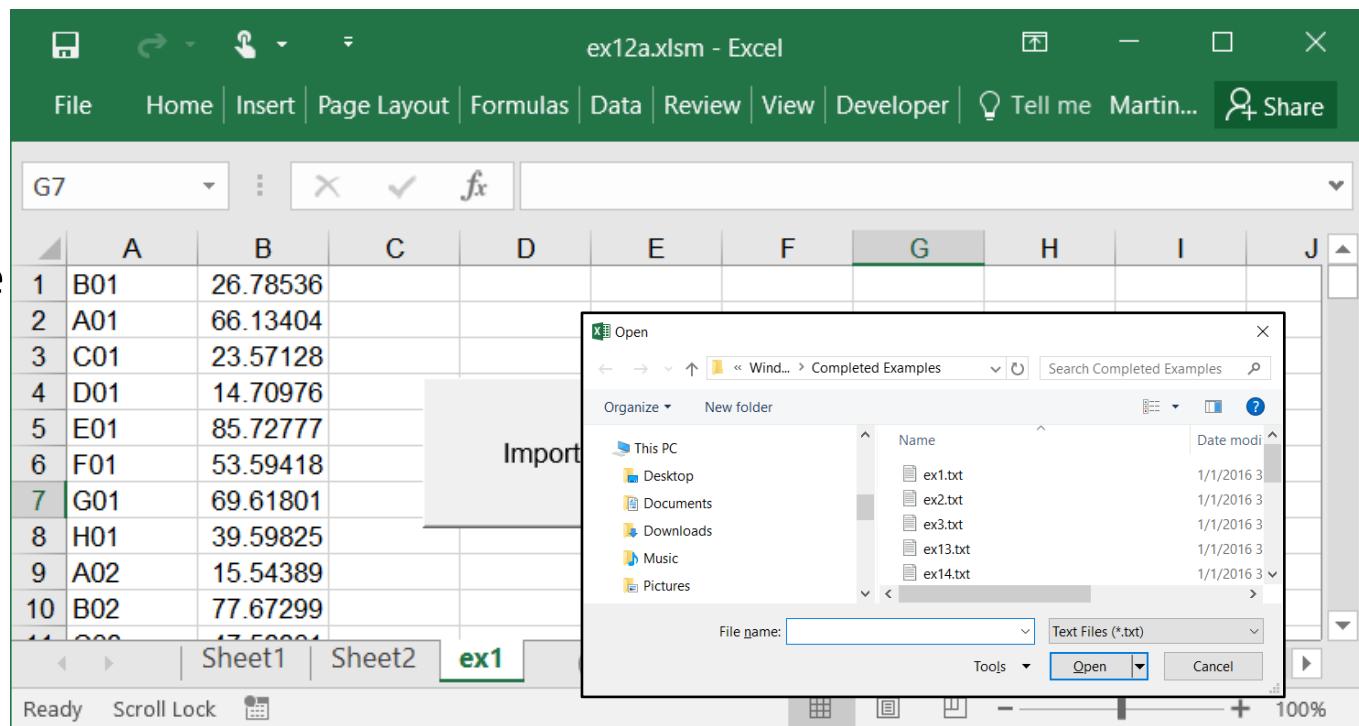
Example 12a: Trigger Macro with a Button

- Dismiss the Forms toolbar
- Right-mouse click the button and change its label to something more meaningful, like “Import”



Example 12a: Trigger Macro with a Button

- Click on the Worksheet to deselect the button
- Then, click the button to test your macro once again



Do ... Loop Statement (Syntax 1)

Repeats a block of statements *while* a condition is True.

Syntax:

```
Do [{While | Until} condition]
    [statements]
    [Exit Do]
    [statements]
Loop
```

Example:

```
I = 1
Do Until I > 3
    MsgBox "I <= 3"
    I = I + 1
Loop
```

Do ... Loop Statement (Syntax 2)

Repeats a block of statements *until* a condition becomes True.

Syntax:

```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

Example:

```
I=1
Do
    MsgBox "Still <= 3"
    I = I + 1
Loop Until I > 3
```

More About Files

- LOF(...) Get length of open file by number
- FileLen(...) Get length of file given its name
- Loc(...) Determine current location in a file
- Seek # ... Go to specified location in a file
- EOF(...) True if at end of a file
- FileDateTime(...) Get date-time of a file
- Dir(...) Return a file or directory name given a pattern
- Kill(...) Delete one of more files by name or pattern
- Name ... Rename or move a file
- FileCopy ... Copy a file

Example 13: Parse a File Header

- Common “real life” measurement files contain more than columns of data
- Often, files include a header section with general information about the table that follows
- Headers contain information such as measurement date, instrument settings, bar codes, user, etc.

```
Date: 1/1/2006
User: A. Einstein
Bar Code: a12345678

A01, 26.78535513
A02, 15.54388965
A03, 52.06081682
A04, 94.54006852
A05, 18.13299159
A06, 56.12838862
A07, 76.28451717
A08, 38.92442484
A09, 7.941233017
A10, 32.4053298
A11, 52.86633397
A12, 86.79105153
B01, 66.13404323
B02, 77.67298537
B03, 10.41473605
B04, 99.14566853
...
```

ex13.txt

Example 13: Parse a File Header

- In Example 12 we used a single loop to parse the data table
- We can use similar techniques to parse the header section
- In the example, the header is composed of name-value pairs, separated with a colon, one pair per line
- The header section is separated from the data table by a blank line

Date:	1/1/2006
User:	A. Einstein
Bar Code:	a12345678
A01,	26.78535513
A02,	15.54388965
A03,	52.06081682
A04,	94.54006852
A05,	18.13299159
A06,	56.12838862
A07,	76.28451717
A08,	38.92442484
A09,	7.941233017
A10,	32.4053298
A11,	52.86633397
A12,	86.79105153
B01,	66.13404323
B02,	77.67298537
B03,	10.41473605
B04,	99.14566853
...	

ex13.txt

Example 13: Parse a File Header

- Use the colon and blank line cues to write a loop that will parse data from the header section of the file
- Because the number of header lines may be unknown, use Do-Loop and Exit Do statements
- Save the bar code in a string variable and use it to rename the new Worksheet

```
' Read and parse all headers.  
Do  
    ' Read a header line and trim it  
    Line Input #intFileNum, strLine  
    strLine = Trim(strLine)  
  
    ' Exit loop if the blank line is found  
    If Len(strLine) = 0 Then Exit Do  
  
    ' Split header line into name and value  
    strParts = Split(strLine, ":")  
  
    ' Save the bar code  
    If strParts(0) = "Bar Code" Then  
        strBarcode = Trim(strParts(1))  
    End If  
Loop  
  
' Rename the new sheet  
wsDestination.Name = strBarcode
```

Example 13: Loop Until End of File

- Modify the data table parsing loop to handle variable length tables
- Use the EOF function to locate the end of the file
- Skip blank lines to make the parsing routine more robust

```
' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)

    Line Input #intFileNum, strLine

        ' Skip blank lines
    If Len(Trim(strLine)) > 0 Then

        ' Parse the data
        strParts = Split(strLine, ",")
        strWell = strParts(0)
        dblMeas = CDbl(strParts(1))

        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

        ' Write measurements to the target sheet
        wsDestination.Cells(lngRow+1, lngCol+1).Value = dblMeas
    End If

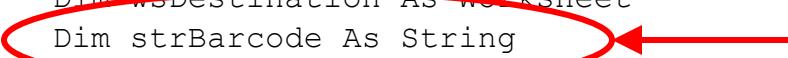
Loop
```

Example 13: The Complete Macro

```
Sub Macro1()
    ' Read all the lines in a text file
    Dim FileName As Variant
    Dim strFilter As String
    Dim intFileNum As Integer
    Dim strLine As String
    Dim strParts() As String
    Dim strWell As String, dblMeas As Double
    Dim lngRow As Long, lngCol As Long
    Dim wsDestination As Worksheet
    Dim strBarcode As String
    ' Get the name of the file to load
    strFilter = "Text Files (*.txt), *.txt, All Files (*.*),*.*"
    FileName = Application.GetOpenFilename(strFilter)

    ' Exit if cancel selected
    If FileName = False Then Exit Sub

    ' Create a new sheet and save a reference
    Set wsDestination = Sheets.Add()
```



A new variable to hold bar code

(continued ...)

Example 13: The Complete Macro

```
' Write row and column headers
wsDestination.Range("A2").Value = "A"
wsDestination.Range("A3").Value = "B"
wsDestination.Range("A4").Value = "C"
wsDestination.Range("A5").Value = "D"
wsDestination.Range("A6").Value = "E"
wsDestination.Range("A7").Value = "F"
wsDestination.Range("A8").Value = "G"
wsDestination.Range("A9").Value = "H"
wsDestination.Range("B1").Value = "1"
wsDestination.Range("C1").Value = "2"
wsDestination.Range("D1").Value = "3"
wsDestination.Range("E1").Value = "4"
wsDestination.Range("F1").Value = "5"
wsDestination.Range("G1").Value = "6"
wsDestination.Range("H1").Value = "7"
wsDestination.Range("I1").Value = "8"
wsDestination.Range("J1").Value = "9"
wsDestination.Range("K1").Value = "10"
wsDestination.Range("L1").Value = "11"
wsDestination.Range("M1").Value = "12"
```

```
' Open the file
intFileNum = FreeFile
Open FileName For Input As #intFileNum
```

(... continued from previous slide)

(continued ...)

Example 13: The Complete Macro

(... continued from previous slide)

```
' Read and parse all headers.  Save only the bar code
Do
    ' Read a header line and trim it
    Line Input #intFileNum, strLine
    strLine = Trim(strLine)

    ' Exit out of the Do-Loop if the blank line is found
    If Len(strLine) = 0 Then Exit Do

    ' Split the header line into name and value
    strParts = Split(strLine, ":")

    ' Save the bar code
    If strParts(0) = "Bar Code" Then
        strBarcode = Trim(strParts(1))
    End If
Loop

' Rename the new sheet
wsDestination.Name = strBarcode
```

(continued ...)

Example 13: The Complete Macro

(... continued from previous slide)

```
' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine

    ' Skip blank lines
    If Len(Trim(strLine)) > 0 Then

        ' Parse the data
        strParts = Split(strLine, ",")
        strWell = strParts(0)
        dblMeas = CDbl(strParts(1))
        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

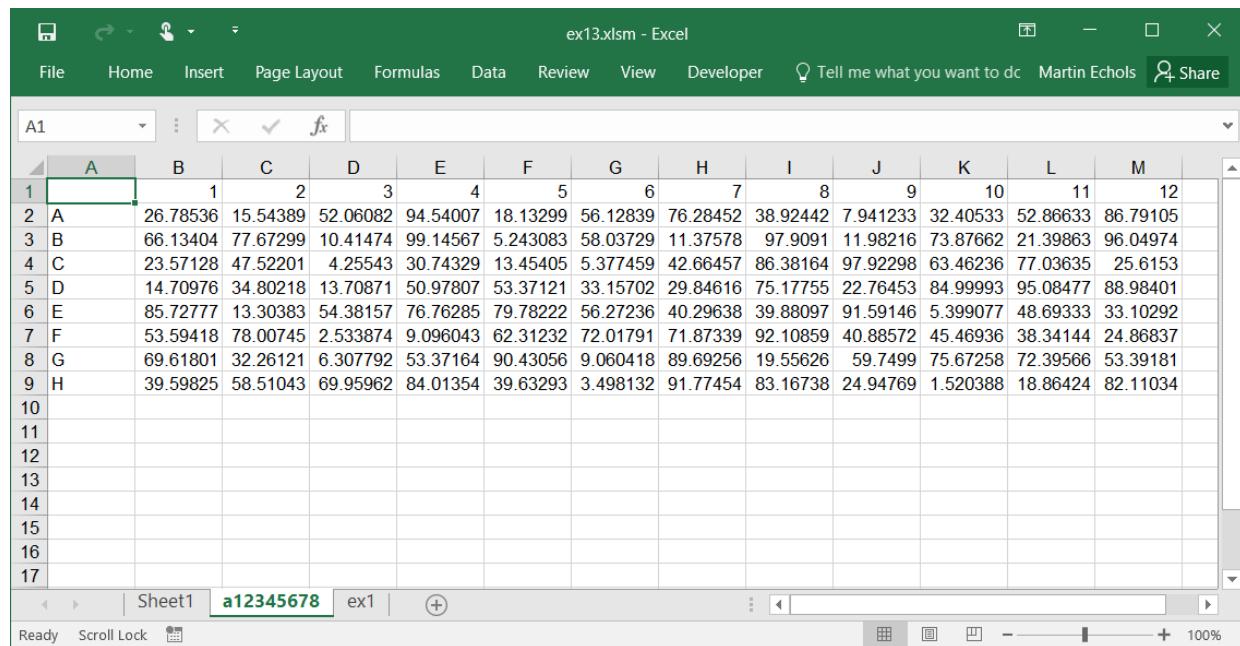
        ' Write measurements to the target sheet
        wsDestination.Cells(lngRow + 1, lngCol + 1).Value = dblMeas
    End If
Loop

' Close the file
Close #intFileNum

End Sub
```

Example 13: The Complete Macro

- Test and debug your new macro
- Import ex13.txt
- Insert blank lines in the file's data table and test again
- Add extra header lines to the file and test again
- What happens when you try to import the file twice in a row?



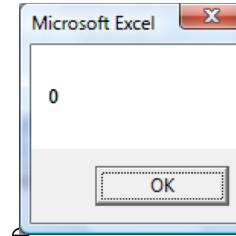
The screenshot shows an Excel spreadsheet titled "ex13.xlsx - Excel". The data is contained in a single sheet named "Sheet1". The data consists of 17 rows of numerical values. Row 1 contains column headers labeled 1 through 12. Rows 2 through 9 contain data points, each starting with a letter (A-H) followed by a series of numbers. Rows 10 through 17 are empty. The formula bar at the top shows the address "a12345678". The status bar at the bottom right indicates "100%".

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		1	2	3	4	5	6	7	8	9	10	11	12
2	A	26.78536	15.54389	52.06082	94.54007	18.13299	56.12839	76.28452	38.92442	7.941233	32.40533	52.86633	86.79105
3	B	66.13404	77.67299	10.41474	99.14567	5.243083	58.03729	11.37578	97.9091	11.98216	73.87662	21.39863	96.04974
4	C	23.57128	47.52201	4.25543	30.74329	13.45405	5.377459	42.66457	86.38164	97.92298	63.46236	77.03635	25.6153
5	D	14.70976	34.80218	13.70871	50.97807	53.37121	33.15702	29.84616	75.17755	22.76453	84.99993	95.08477	88.98401
6	E	85.72777	13.30383	54.38157	76.76285	79.78222	56.27236	40.29638	39.88097	91.59146	5.399077	48.69333	33.10292
7	F	53.59418	78.00745	2.533874	9.096043	62.31232	72.01791	71.87339	92.10859	40.88572	45.46936	38.34144	24.86837
8	G	69.61801	32.26121	6.307792	53.37164	90.43056	9.060418	89.69256	19.55626	59.7499	75.67258	72.39566	53.39181
9	H	39.59825	58.51043	69.95962	84.01354	39.63293	3.498132	91.77454	83.16738	24.94769	1.520388	18.86424	82.11034
10													
11													
12													
13													
14													
15													
16													
17													

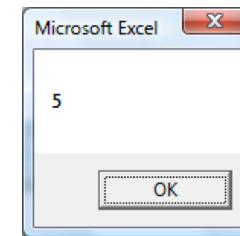
Determining Array Bounds

- Use the UBound and LBound functions to determine array bounds.

```
Sub Test_LBound()
    Dim arrTest(3) As Integer
    arrTest(2) = 10
    MsgBox LBound(arrTest)
End Sub
```



```
Sub Test_UBound()
    Dim arrTest(1 To 10, 2 To 5) As Single
    arrTest(2, 3) = 1.23
    MsgBox UBound(arrTest, 2)
End Sub
```



Rounding Numbers

- Sometimes, it is helpful to round a number to a specific number of decimal places
- VBA provides a Round Function that rounds to a specific number of decimal points.

```
RoundedNum = Round(expression [,numdecimalplaces])
```

For example,

```
Dim dblNum As Double, dblOtherNum As Double  
dblNum = 1.73699  
dblOtherNum = Round(dblNum, 3) 'returns 1.737  
dblOtherNum = Round(dblNum)      'returns 2
```

Example 14: Multiple Measurements

- What if each row in the file's data table contains multiple measurements, the number for each well being variable?
- We want to calculate the average of all measurements and display this value with only two decimal places.

Date: 1/1/2006

User: A. Einstein

Bar Code: a12345678

A01, 26.78535513, 25.65846092, 22.65699257, 29.46282743

A02, 15.54388965, 14.34177274

A03, 52.06081682, 49.26655443, 51.04218928, 53.77858993

A04, 94.54006852, 92.95734924, 93.29783084, 101.3941916

A05, 18.13299159, 16.71899874, 14.12176218

A06, 56.12838862, 61.56463086, 55.12616969

A07, 76.28451717, 68.97690609, 65.15506171

A08, 38.92442484, 40.96943841

A09, 7.941233017, 4.185950722

...

ex14.txt

- **Modify the data table parsing loop:**

- Determine the number of measurements using Ubound
- Sum all values in each row
- Calculate the average
- Round to two decimal places using Round

```

' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine

    ' Skip blank lines
    If Len(Trim(strLine)) > 0 Then
        ' Parse the data
        strParts = Split(strLine, ", ")
        strWell = strParts(0)

        ' Sum and compute the average
        lngNVals = UBound(strParts)
        dblMeas = 0
        For I = 1 To lngNVals
            dblMeas = dblMeas + CDbl(strParts(I))
        Next I

        ' Calculate average and round
        dblMeas = Round((dblMeas/CDbl(lngNVals)),2)

        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

        ' Write the measurements to the target sheet
        wsDestination.Cells(lngRow+1, lngCol+1).Value = dblMeas
    End If
Loop

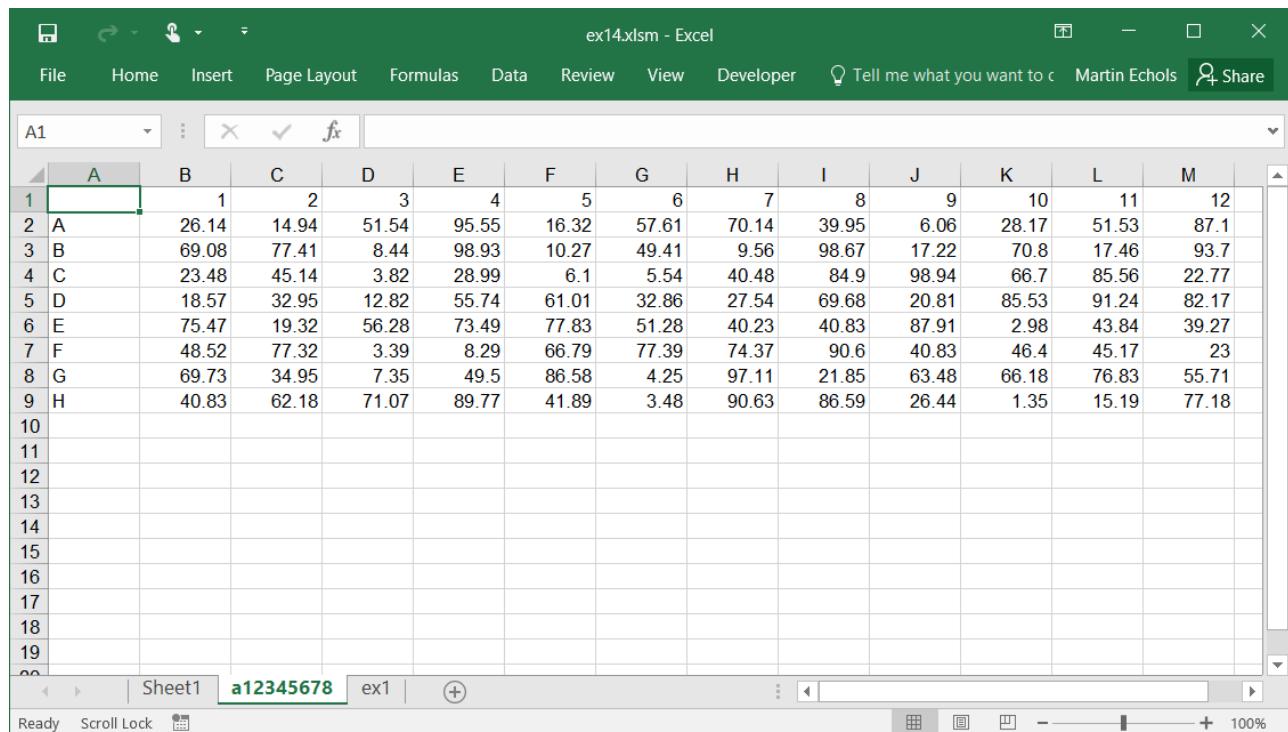
```

Example 14: Multiple Measurements

- Declare your new variables:

```
Dim lngNVals As Long  
Dim I As Long
```

- Test and debug your macro.
- Import ex14.txt.



The screenshot shows an Excel spreadsheet titled "ex14.xlsxm - Excel". The data is organized into 10 rows and 12 columns. Row 1 contains column headers A through M. Rows 2 through 11 contain data points. Row 12 is blank. Row 13 contains the formula "a12345678". The formula bar also displays "a12345678". The status bar at the bottom shows "Ready" and "100%".

A	B	C	D	E	F	G	H	I	J	K	L	M
1	26.14	14.94	51.54	95.55	16.32	57.61	70.14	39.95	6.06	28.17	51.53	87.1
A	69.08	77.41	8.44	98.93	10.27	49.41	9.56	98.67	17.22	70.8	17.46	93.7
B	23.48	45.14	3.82	28.99	6.1	5.54	40.48	84.9	98.94	66.7	85.56	22.77
C	18.57	32.95	12.82	55.74	61.01	32.86	27.54	69.68	20.81	85.53	91.24	82.17
D	75.47	19.32	56.28	73.49	77.83	51.28	40.23	40.83	87.91	2.98	43.84	39.27
E	48.52	77.32	3.39	8.29	66.79	77.39	74.37	90.6	40.83	46.4	45.17	23
F	69.73	34.95	7.35	49.5	86.58	4.25	97.11	21.85	63.48	66.18	76.83	55.71
G	40.83	62.18	71.07	89.77	41.89	3.48	90.63	86.59	26.44	1.35	15.19	77.18
H												
I												
J												
K												
L												
M												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

Example 14b: Using Worksheet Functions

- We can achieve the same operations we performed in example 14 using what are called Worksheet Functions.
- Worksheet functions are the same functions that are normally inserted directly into a spreadsheet cell.
- Example: by typing the function below into a worksheet cell the formula is stored in the cell and the cell displays the value of the summed cells.

=SUM(F1:F10)

	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									

ex14.txt

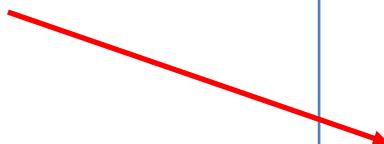
Using Worksheet Functions With VBA

- Every worksheet object contains a WorksheetFunction object. The WorksheetFunction object contains methods (or functions) for most of the functions you can use directly with a worksheet.
- In this example we will use the 'Average' function to get the average of our measurements.
- To do this we will:
 1. Add a new Function routine for our Macro. This function will convert the individual measurement values to type Double as per the Average function's requirement.
 2. Add new Double variable dblParts to hold the measurements.
 3. Modify the data loop in macro1 to call the new function.
 4. Modify the data loop to now use the WorkSheetfunction to average the data.

Using Worksheet Functions

Insert the new variable definition for dblParts at the top of the Macro with the other declarations.

```
Sub Macro1()
    ' Read all the lines in a text file
    Dim FileName As Variant
    Dim strFilter As String
    Dim intFileNum As Integer
    Dim strLine As String
    Dim strParts() As String
    Dim strWell As String, dblMeas As Double
    Dim lngRow As Long, lngCol As Long
    Dim wsDestination As Worksheet
    Dim strBarcode As String
    Dim dblParts() As Double
```



Using Worksheet Functions

- Add this 'StringToDouble' function after Macro1
 - Accepts the string array containing values
 - Derives the number of values in the array
 - Transfers them from the String array to the Double array
 - Returns the Double array

```
Function StringToDouble(strArray() As String) As Double()
    Dim i As Long
    Dim dblArray() As Double
    Dim IngForCounterMax As Long

    ' Get the number of variables in the array - 1 because
    ' we don't want to include the well location

    IngForCounterMax = UBound(strArray) - 1
    ' Resize the double array

    ReDim dblArray(IngForCounterMax)

    For i = 0 To IngForCounterMax
        dblArray(i) = CDbl(strArray(i + 1)) ' skip well
        Debug.Print dblArray(i)
    Next i
    Debug.Print ""

    StringToDouble = dblArray()
End Function
```

- # Modify the value parsing loop:

- Pass the measurement values to `StringToDouble`
- Get the average of all values
- Round to two decimal places using `Round`
- Try running Example 14b, the results should be the same as Example 14.

```
' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine

    ' Skip blank lines
    If Len(Trim(strLine)) > 0 Then
        ' Parse the data
        strParts = Split(strLine, ",")
        strWell = strParts(0)

        ' Sum and compute the average

        ' Call new function to convert Strings to Double
        dblParts = StringToDouble(strParts)

        ' Use the Worksheet Function Average
        dblMeas = WorksheetFunction.Average(dblParts)
        dblMeas = Round(dblMeas, 2)

        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

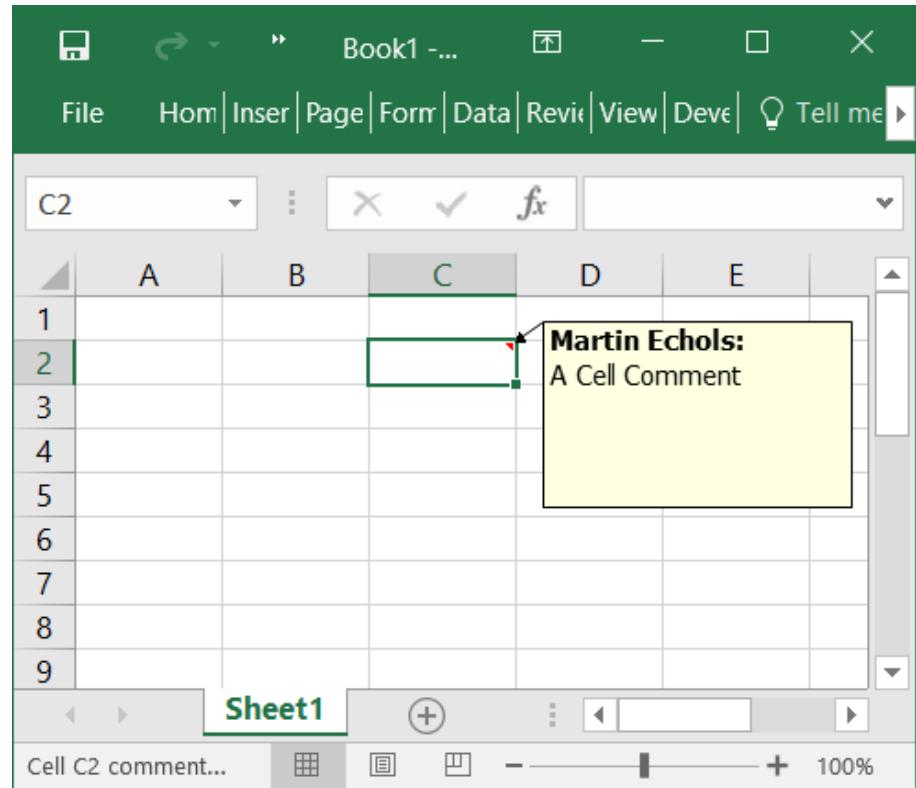
        ' Write the measurement to the target sheet
        wsDestination.Cells(lngRow + 1, lngCol + 1).Value = dblMeas
    End If
Loop
```

Formatting A Worksheet Cell

- Sometimes it isn't possible to represent all the information that you want to in a worksheet cell
- We will explore two useful techniques for adding additional context programmatically to a cell in Excel
 - Adding Cell comments
 - Changing a cell's color

Cell Comments

- Cell comments are simply additional text that can be associated with a particular cell
- When you hover over a cell with a cell comment, a popup box appears that displays the comment text
- You can add a comment to a cell by choosing menu option **Insert | Comment** with the appropriate cell selected



Adding a Cell Comment Through VBA

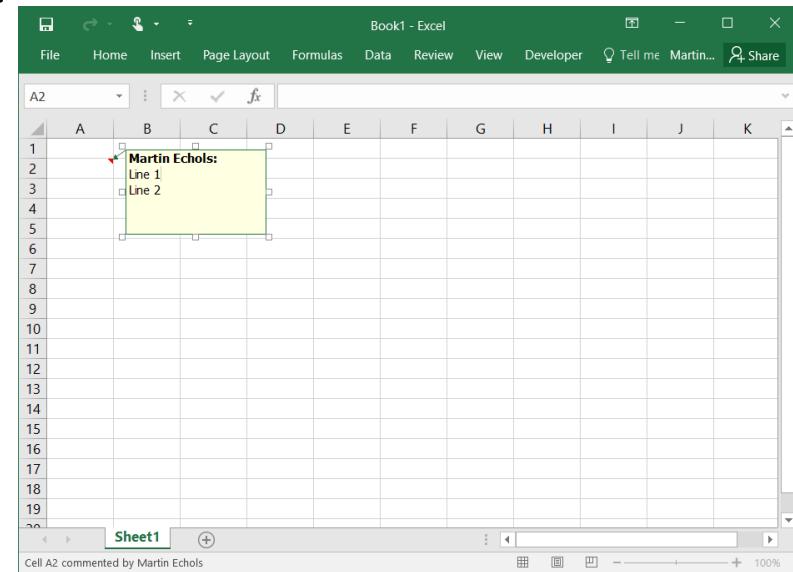
- A cell comment can also be added to a cell programmatically
- An Excel Range object is the VBA representation of a cell and the Range object has a `Comment` property that represents the cell comment
- To create a cell comment programmatically:

```
Dim rngCell As Range  
Set rngCell = ActiveSheet.Range("A1")  
rngCell.AddComment  
rngCell.Comment.Text "Comment Text"
```

- Note that the `Comment` property is actually an object which has a method called `Text` that takes a parameter of type `String`. This is the reason for the ‘double dot notation’ in the code above

Adding a Cell Comment Through VBA

- It might also be of interest to separate portions of a cell comment into multiple lines
- Separate lines in a comment are delimited by a linefeed character. VBA has a built-in constant for the linefeed character: `vblf`.



```
ActiveCell.AddComment
```

```
ActiveCell.Comment.Text "Line 1" & vblf & "Line 2"
```

Colors in Excel

- Every Excel Workbook holds its own array of 56 colors.
- To view the color array:
 - Open a new workbook and display the VBE
 - Insert a new Code Module (Insert | Module)
 - Open the code window for Module
 - Enter the following subroutine
 - Move the cursor to some point within the subroutine
 - Hit F5 to run it and look at Sheet1

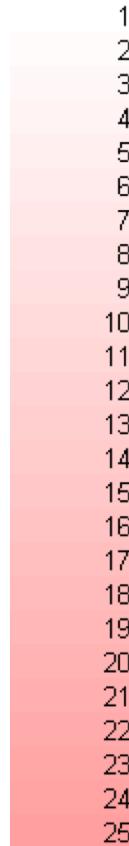
```
Sub Colors()
    Dim I As Integer

    For I = 1 To 56
        Sheet1.Cells(I, 1).Value = I
        Sheet1.Cells(I, 1).Interior.ColorIndex = I
    Next I
End Sub
```

	A
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33

Colors in Excel

- The default color array is somewhat random
- We want a more regular color continuum to indicate magnitude
- Fortunately, a Workbook's color array can be changed



Colors in Excel

- A cell's color is set programmatically through the Range's Interior object, which has a property called Color
- Colors in VBA are represented by RGB (red, green, blue) color values.
- An RGB color value is generated in VBA using the RGB function:
`lngColor = RGB(intRed, intGreen, intBlue)`
- `intRed, intGreen, intBlue` are integers from 0-255 inclusively

To get black, use

RGB (0, 0, 0)

To get green use

RGB (0, 255, 0)

To get cyan, use

RGB (0, 255, 255)

To get magenta, use

RGB (255, 0, 255)

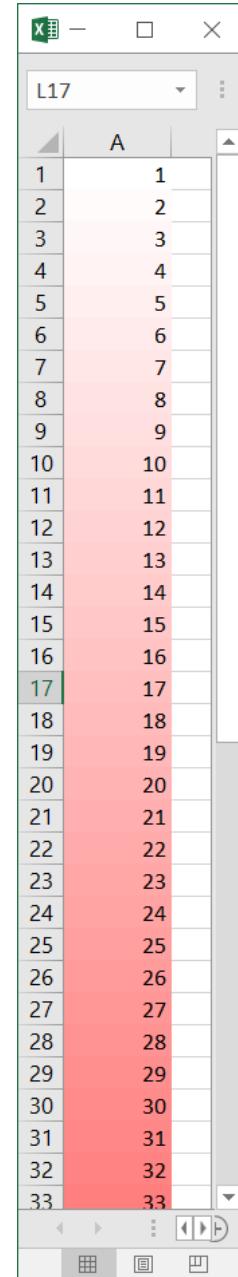
Colors in Excel

- Reset a Workbook's color array and display
- Enter the following code into a Code Module and run it
- To reset the color array to default values, run ActiveWorkbook.ResetColors from the Immediate Window

```
Sub SetColors()
    ' Modify the current workbook's color array
    ' to be a range of Red
    Dim I As Integer
    Dim R As Integer, G As Integer, B As Integer

    For I = 1 To 56
        R = 255
        G = 255 - (4 * (I - 1))
        B = G
        ActiveWorkBook.Colors(I) = RGB(R, G, B)
    Next I

End Sub
```



A screenshot of Microsoft Excel showing a color gradient from white to red across cells A1 to A33. The cells are arranged in a single column labeled 'A'. The colors transition from white at row 1 to a deep red at row 33. Row 17 is highlighted in green, indicating it is the active cell.

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33

Example 15: Cell Formatting

- Expand the macro in Example 14b.
- We want to be able to view the raw data for each well.
 - Insert a comment into each cell that lists the raw data values that went into calculating the average.
- We want to get a sense for the measurement value immediately, without needing to think about the numeric value.
 - Set the Workbook color array to be a range of red values
 - Change the color of each cell to give a graphical indication of the average value – a heat map.

Example 15: Cell Formatting

- Write a function that remaps the color index to a range of red values
- Write another function that takes a measurement value between 0.0 and 100.0 and returns a heat map color.
- Add these new functions to your Code Module

```
Sub SetColors()
    ' Modify the current workbook's color array
    ' to be a range of Red
    Dim I As Integer
    Dim R As Integer, G As Integer, B As Integer

    For I = 1 To 56
        R = 255
        G = 255 - (4 * (I - 1))
        B = G
        ActiveWorkBook.Colors(I) = RGB(R, G, B)
    Next I

End Sub
```

```
Function HeatMapColorIndex(dbIColorIndex As Double) As Long
    ' Convert the given value to a heat map color
    ' By taking the input value from 1 to 100 and
    ' and normalizing to a number from 1 to 56
    HeatMapColorIndex = CLng((dbIColorIndex / 100) * 56)
End Function
```

Example 15: Cell Formatting

Macro Modification Plan:

- Call SetColors once to ensure the workbook has the right color map
- Modify the data table parsing loop.
- Accumulate a string to be used as the cell comment while the sum is being calculated.
 - For each iteration of the inner loop, concatenate the next measurement value to the comment string.
- Get a reference to the target cell before writing the average value.
- After the average value is written
 - Insert a cell comment and then set it to the concatenated string.
 - Change the cell's Interior.ColorIndex property to the color index calculated from the average measurement value.

- The new data table parsing loop should look like the following.

```

SetColors          ' Reset color map

' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine
    strComment = "Raw Data:" & vbLf
    If Len(Trim(strLine)) > 0 Then

        ' Parse the data
        strParts = Split(strLine, ",")
        strWell = strParts(0)

        ' Sum and compute the average

        ' Call new function to convert Strings to Double
        dblParts = StringtoDouble(strParts)

        ' Use the Worksheet Function Average
        dblMeas = WorksheetFunction.Average(dblParts)
        dblMeas = Round(dblMeas, 2)
        ' Use the Join function to populate the comment
        strComment = strComment + Join(strParts, vbLf)

        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

        ' Get cell reference and write the measurement
        Set rngCell = wsDestination.Cells(lngRow + 1, lngCol + 1)
        rngCell.Value = dblMeas

        ' Add a comment to the target containing the raw data
        rngCell.AddComment
        rngCell.Comment.Text strComment

        ' Color the cell appropriately
        rngCell.Interior.ColorIndex = HeatMapColorIndex(dblMeas)
    End If
Loop

```

Example 15: Cell Formatting

- Declare your new variables:

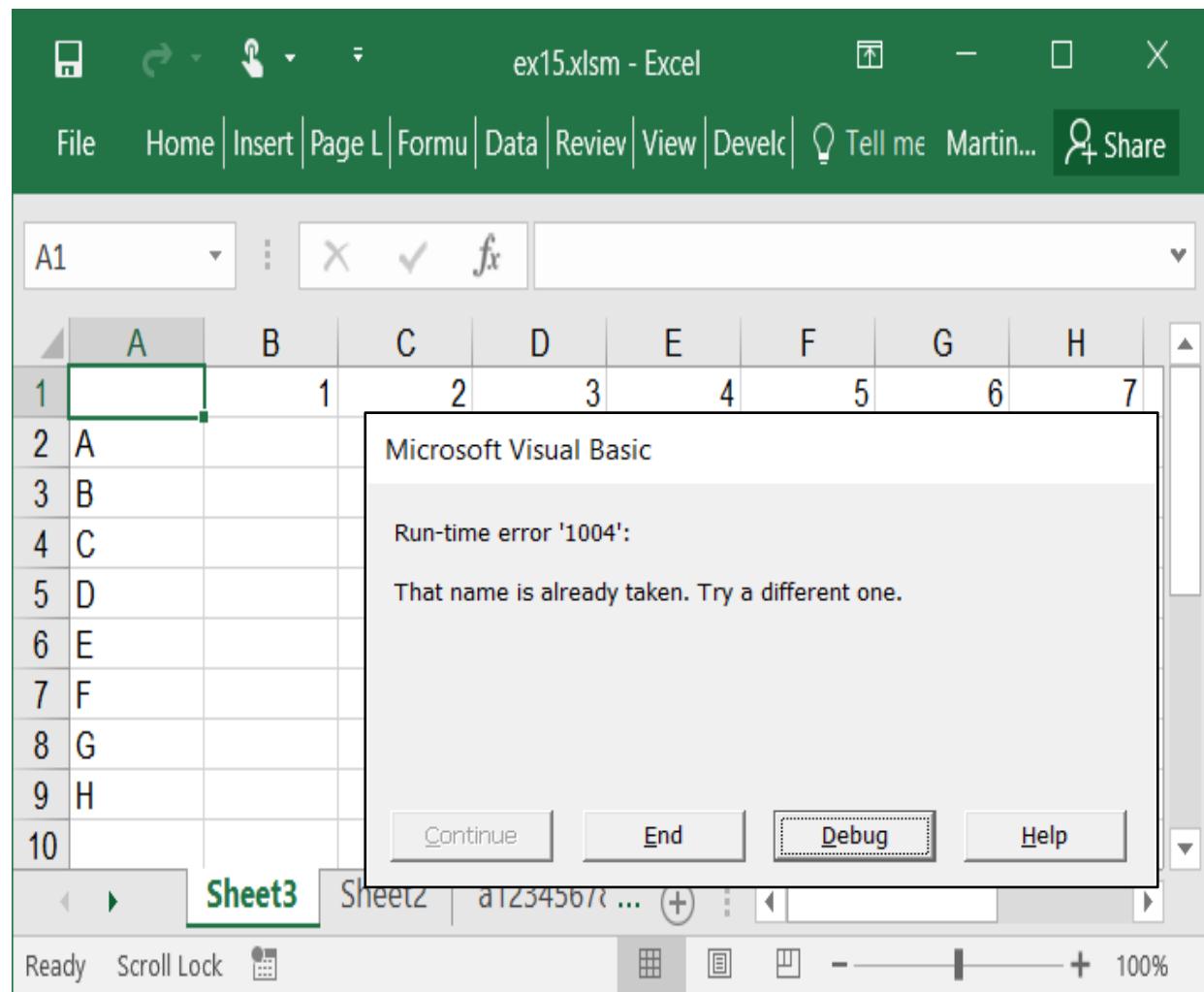
```
Dim rngCell As Range  
Dim strComment As String
```

- Run the SetColors subroutine at least once
- Test and debug your macro.
- Use the file ex15.txt.

A	B	C	D	E	F	G	H	I	J	K	
1		1	2	3	4	5	6	7	8	9	10
2	A	26.14	14.94	51.54	95.55	16.32	57.61	70.14	39.95	6.06	28.17
3	B	69.08	77.41	8.44	98.93	10.27	49.41	9.56	98.67	17.22	70.8
4	C	23.48	45.14	3.82	28.99	6.1	5.54	40.48	84.9	98.94	66.7
5	D	18.57	32.95	12.82	55.74	61.01	32.86	27.54	69.68	20.81	85.53
6	E	75.47	19.32	56.28	73.49	77.83	51.28	40.23	40.83	87.91	2.98
7	F	48.52	77.32	3.39	8.29	66.79	77.39	74.37	90.6	40.83	46.4
8	G	69.73	34.95	7.35	49.5	86.58	4.25	97.11	21.85	63.48	66.18
9	H	40.83	62.18	71.07	89.77	41.89	3.48	90.63	86.59	26.44	1.35
10											

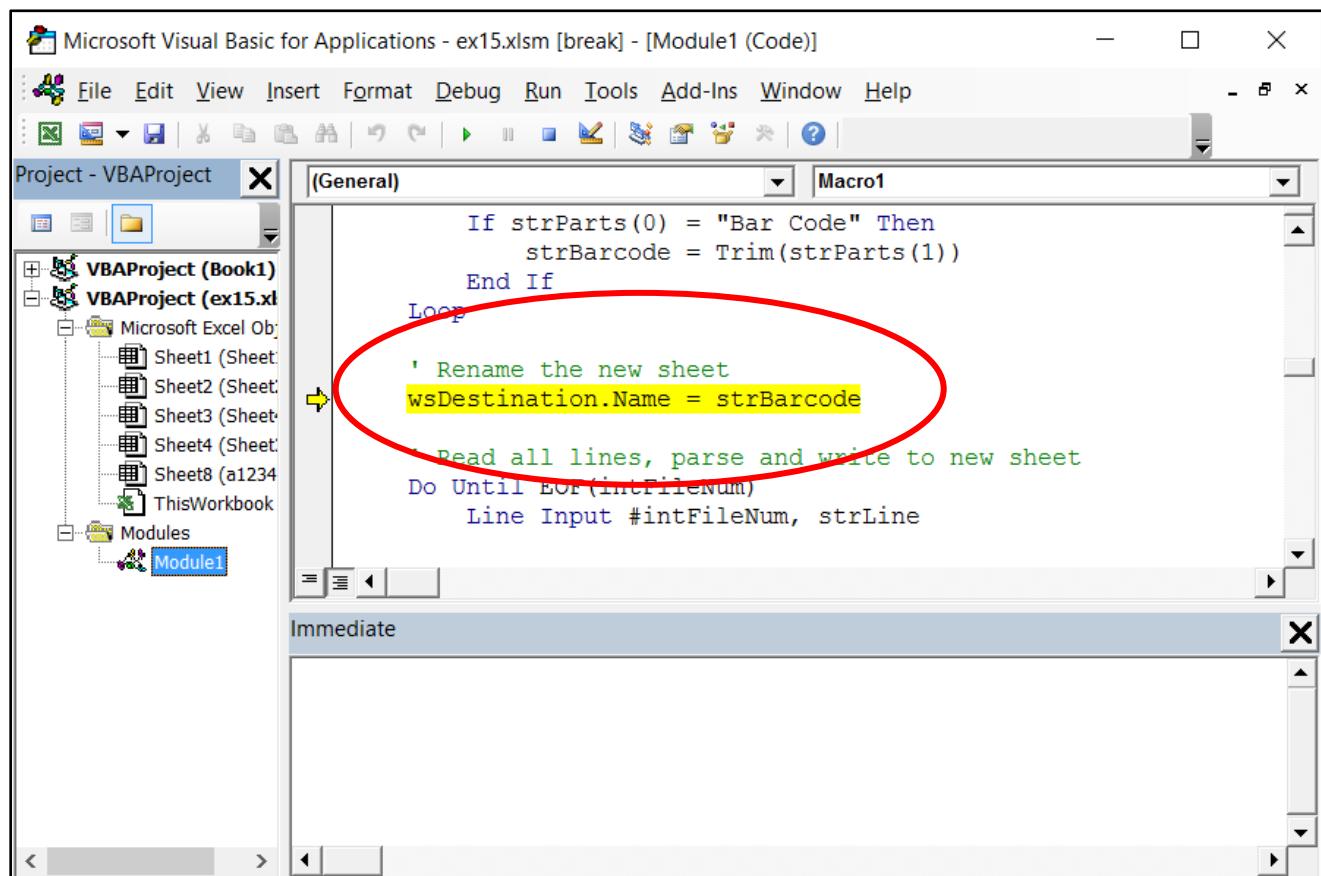
Handling Runtime Errors

- When you attempt to import the same data file twice, we get a run-time error.
- Try it, and click [Debug] to view the offending line in the VBA debugger.
- Note that this is error number 1004.



Handling Runtime Errors

- As expected, the error occurs when the macro attempts to assign an existing name for the new Worksheet.
- Click the blue Stop button on the toolbar to stop the debugger (the blue box).



```
If strParts(0) = "Bar Code" Then
    strBarcode = Trim(strParts(1))
End If
Loop

' Rename the new sheet
wsDestination.Name = strBarcode

' Read all lines, parse and write to new sheet
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine
```

Handling Runtime Errors

- Runtime errors occur during the execution of your code
- Runtime errors can be trapped and handled in your program, making it more robust and usable
- Error Handlers can be installed and removed in a VBA procedure using the ‘On Error’ statement
- Error Handlers can:
 - Tell your program to jump to a different line in your code
 - Tell your program to skip the statement on which there was an error and continue to run

Handling Runtime Errors

- To set up an error handler add an On Error statement to a procedure at the point you want the error handling behavior to start.

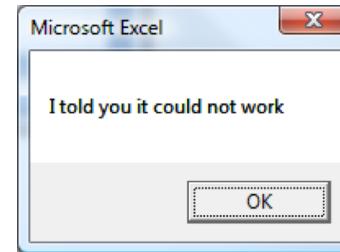
```
On Error GoTo ErrorHandler
```

- Add an appropriate Exit statement, the line label and the error handling code to the end of the macro

```
Sub ThisCannotWork()
    Dim I As Integer
    On Error GoTo ErrorHandler
    I = 1 / 0
    Exit Sub
'
```

ErrorHandler:

```
    MsgBox "I told you it could not work"
End Sub
```



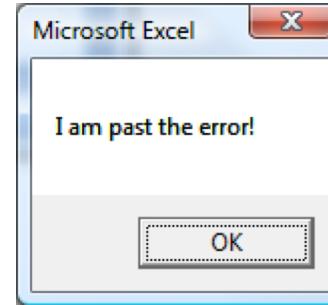
What happens if you forget the Exit Sub?

Handling Runtime Errors

- To ignore an error and just keep going, use the On Error Resume Next statement

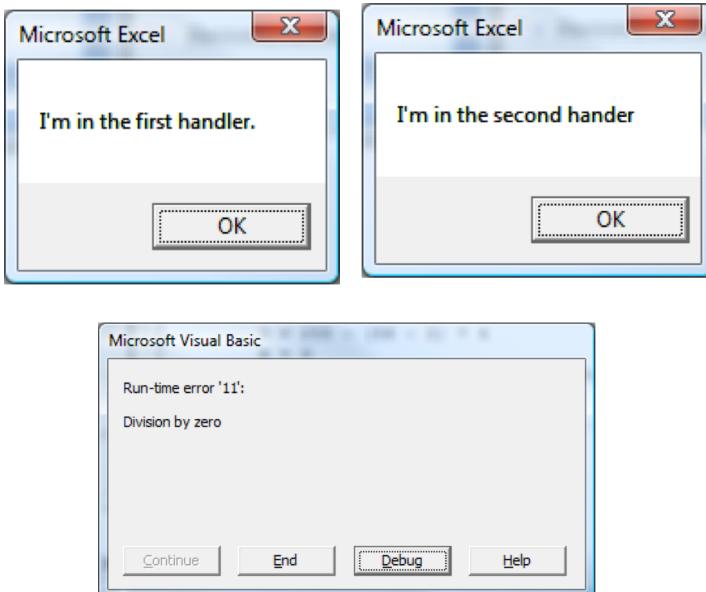
```
Sub IgnoreTheError()
    Dim I As Integer
    On Error Resume Next

    I = 1 / 0
    MsgBox "I am past the error!"
End Sub
```



Handling Runtime Errors

- To change an error handler, execute a second On Error GoTo ... statement.
- To cancel an error handler, use the On Error GoTo 0 statement.
- To resume execution after an error use the Resume statement.



```

Sub ThisCannotWork()
    Dim I As Integer, J As Integer
    On Error GoTo FirstHandler
    J = 0
    I = 1 / J
    On Error GoTo SecondHandler
    J = 0
    I = 1 / J
    On Error GoTo 0
    J = 0
    I = 1 / 0
    Exit Sub

FirstHandler:
    MsgBox "I'm in the first handler."
    J = 1
    Resume ' Try again

SecondHandler:
    MsgBox "I'm in the second hander"
    J = 1
    Resume ' Try again
End Sub

```

Handling Runtime Errors

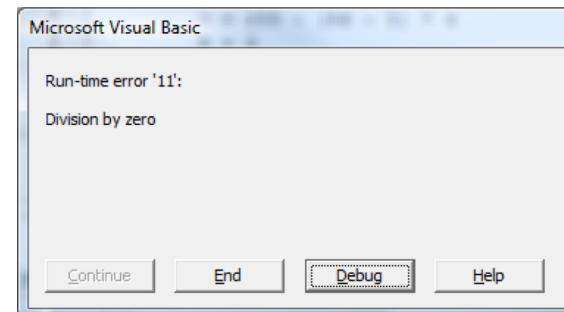
- Use the built-in Err object to learn about the runtime error.
- Err's properties are populated automatically when a runtime error occurs.
 - Err.Number holds the current error number
 - Err.Description holds a description of the current error

```
Sub ThisCannotWork()
    Dim I As Integer
    On Error GoTo ErrorHandler
    I = 1 / 0
    Exit Sub
```

ErrorHandler:

```
    MsgBox "Error " & CStr(Err.Number) & ": " & Err.Description
```

```
End Sub
```



Example 16: Handling Errors

Error Handling Plan

- When error 1004 occurs, modify the new worksheet name and try again.
- Change the name by appending a sequential number in parentheses: (1), (2), ...
- Display a message box with the error number and description on any other error.

Example 16: Handling Errors

```
' Handle the case of a duplicate sheet name
If Err.Number = 1004 Then
    strBarcode = ChangeWorkSheetName(strBarcode)
    Resume

' Display any other error message
Else
    MsgBox "Error " & CStr(Err.Number) & ":" & Err.Description
    Exit Sub
End If
```

Error Handling Plan

Write a function to change the Worksheet name

1. Look for a left parenthesis at the end of the current Worksheet name.
1. If found, extract the copy number from the parentheses and add 1 to it and also create the new barcode
1. If not found, just use a copy number of 1 and create the new barcode
1. Return the new Worksheet name using the original bar code and the new copy number.

```
Function ChangeWorkSheetName(ByVal strBarcode) As String
    Dim intLeftParen As Long
    Dim intCopyNum As Long
    Dim strNewBarcode As String

    ' Look for a left parenthesis
    intLeftParen = InStrRev(strBarcode, "(")

    ' Calculate the next copy number and
    ' the new sheet name
    If intLeftParen > 0 Then
        intCopyNum = Val(Mid(strBarcode, intLeftParen + 1)) + 1
        strNewBarcode = Left(strBarcode, intLeftParen - 2) & _
                      " (" & CStr(intCopyNum) & ")"
    Else
        strNewBarcode = strBarcode & " (1)"
    End If

    ' return the new sheet name to the calling program
    ChangeWorkSheetName = strNewBarcode
End Function
```

What does a function do?

Example 16: Handling Errors

- Set up the error handler by adding an On Error line to the top of the macro

```
On Error GoTo Macro1_ErrorHandler
```

- Add the Exit Sub, line label and error handling routine to the end of the macro as shown below and on the next several slides.

```
...
Exit Sub
'
Macro1_ErrorHandler:
...
```

- Write a function called ChangeWorksheetName and test it in the immediate window giving it different bar code scenarios.

For example

```
? ChangeWorksheetName("12345678") should return 12345678 (1)
? ChangeWorksheetName("12345678 (1)") should return 12345678 (2)
? ChangeWorksheetName("12345678 (2)") should return 12345678 (3)
```

Example 16: The Complete Macro

```
Sub Macro1()
    ' Read all the lines in a text file
    Dim FileName As Variant
    Dim strFilter As String
    Dim intFileNum As Integer
    Dim strLine As String
    Dim strParts() As String
    Dim strWell As String, dblMeas As Double
    Dim lngRow As Long, lngCol As Long
    Dim wsDestination As Worksheet
    Dim strBarcode As String
    Dim lngNVals As Long, I As Long
    Dim rngCell As Range, strComment As String
    Dim dblParts() as Double

    On Error GoTo Macro1_ErrorHandler

    ' Get the name of the file to load
    strFilter = "Text Files (*.txt), *.txt, All Files (*.*),*.*"
    FileName = Application.GetOpenFilename(strFilter)

    ' Exit if cancel selected
    If FileName = False Then Exit Sub

    ' Create a new sheet and save a reference
    Set wsDestination = Sheets.Add()
```

(continued ...)

Example 16: The Complete Macro

(... continued from previous slide)

```
' Write row and column headers
wsDestination.Range("A2").Value = "A"
wsDestination.Range("A3").Value = "B"
wsDestination.Range("A4").Value = "C"
wsDestination.Range("A5").Value = "D"
wsDestination.Range("A6").Value = "E"
wsDestination.Range("A7").Value = "F"
wsDestination.Range("A8").Value = "G"
wsDestination.Range("A9").Value = "H"
wsDestination.Range("B1").Value = "1"
wsDestination.Range("C1").Value = "2"
wsDestination.Range("D1").Value = "3"
wsDestination.Range("E1").Value = "4"
wsDestination.Range("F1").Value = "5"
wsDestination.Range("G1").Value = "6"
wsDestination.Range("H1").Value = "7"
wsDestination.Range("I1").Value = "8"
wsDestination.Range("J1").Value = "9"
wsDestination.Range("K1").Value = "10"
wsDestination.Range("L1").Value = "11"
wsDestination.Range("M1").Value = "12"
```

(continued ...)

Example 16: The Complete Macro

```
' Open the file
intFileNum = FreeFile
Open FileName For Input As #intFileNum

' Read and parse all headers.  Save only the bar code
Do
    ' Read a header line and trim it
    Line Input #intFileNum, strLine
    strLine = Trim(strLine)

    ' Exit out of the Do-Loop if the blank line is found
    If Len(strLine) = 0 Then Exit Do

    ' Split the header line into name and value
    strParts = Split(strLine, ":")

    ' Save the bar code
    If strParts(0) = "Bar Code" Then
        strBarcode = Trim(strParts(1))
    End If
Loop

' Rename the new sheet
wsDestination.Name = strBarcode

' Set workbook color map
SetColors
```

(... continued from previous slide)

(continued ...)

```

' Read all lines, parse and write to new sheet      (... continued from previous slide)
Do Until EOF(intFileNum)
    Line Input #intFileNum, strLine
    strComment = "Raw Data:" & vbCrLf
    If Len(Trim(strLine)) > 0 Then

        ' Parse the data
        strParts = Split(strLine, ",")
        strWell = strParts(0)
        ' Sum and compute the average

        ' Call new function to convert Strings to Double
        dblParts = StringtoDouble(strParts)

        ' Use the Worksheet Function Average
        dblMeas = WorksheetFunction.Average(dblParts)
        dblMeas = Round(dblMeas, 2)
        strComment = strComment + Join(strParts, vbCrLf)
        lngRow = Well2Row(strWell)
        lngCol = Well2Col(strWell)

        ' Write the measurements to the target sheet
        Set rngCell = wsDestination.Cells(lngRow + 1, lngCol + 1)
        rngCell.Value = dblMeas

        ' Add a comment to the target containing the raw data
        rngCell.AddComment
        rngCell.Comment.Text strComment

        ' Color the cell appropriately
        rngCell.Interior.Color = HeatMapColor(dblMeas)
    End If
Loop

```

(continued ...)

(... continued from previous slide)

```

' Close the file
Close #intFileNum

Exit Sub

'

---


Macro1_ErrorHandler:

' Handle the case of a duplicate sheet name
If Err.Number = 1004 Then
    strBarcode= ChangeWorkSheetName(strBarcode)

    Resume

' Display any other error message
Else
    MsgBox "Error " & CStr(Err.Number) & ": " & Err.Description
    Exit Sub
End If

End Sub

Function ChangeWorkSheetName(ByVal strBarcode) As String
    Dim intLeftParen As Long
    Dim intCopyNum As Long
    Dim strNewBarcode As String

    ' Look for a left parenthesis
    intLeftParen = InStrRev(strBarcode, "(")

    ' Calculate the next copy number and
    ' the new sheet name
    If intLeftParen > 0 Then
        intCopyNum = Val(Mid(strBarcode, intLeftParen + 1)) + 1
        strNewBarcode = Left(strBarcode, intLeftParen - 2) & _
                      " (" & CStr(intCopyNum) & ")"
    End If
End Function

```

(... continued from previous slide)

```
Else
    strNewBarcode = strBarcode & " (1)"
End If
' return the new sheet name to the calling program

ChangeWorkSheetName = strNewBarcode
End Function
```

Example 16: Handling Errors

- Open up example 15.
- Modify the code adding the code that is bolded as shown on the previous slide.
- Run and debug your new macro.
- Open the same file several times to see how the error handler automatically modifies the names of the duplicate Worksheets.

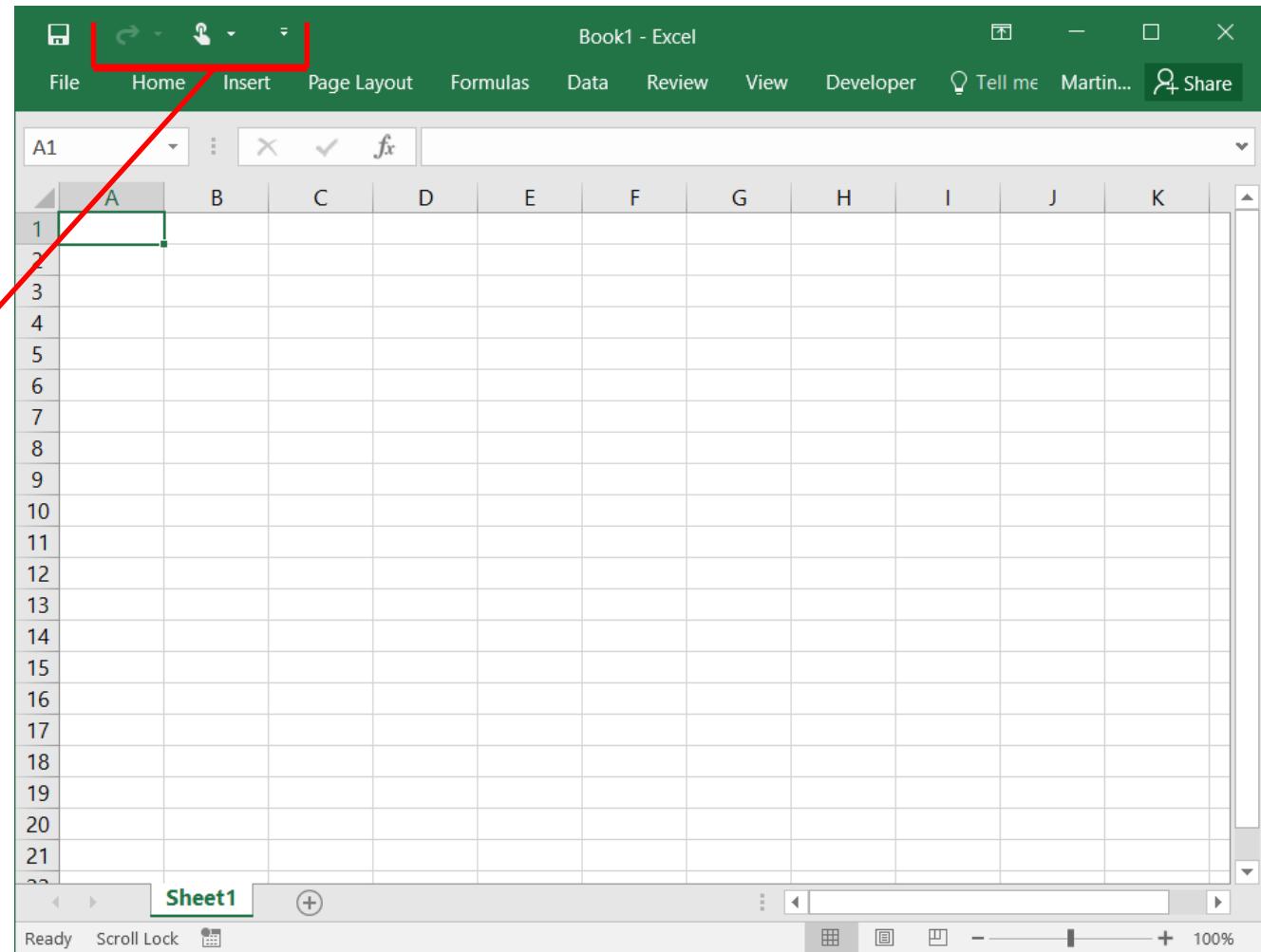
	A	B	C	D	E	F	G	H	I	J	K	L
1	1	2	3	4	5	6	7	8	9	10		11
2	A	26.14	14.94	51.54	95.55	16.32	57.61	70.14	39.95	6.06	28.17	
3	B	69.08	77.41	8.44	98.93	10.27	49.41	9.56	98.67	17.22	70.8	
4	C	23.48	45.14	3.82	28.99	6.1	5.54	40.48	84.9	98.94	66.7	
5	D	18.57	32.95	12.82	55.74	61.01	32.86	27.54	69.68	20.81	85.53	
6	E	75.47	19.32	56.28	73.49	77.83	51.28	40.23	40.83	87.91	2.98	
7	F	48.52	77.32	3.39	8.29	66.79	77.39	74.37	90.6	40.83	46.4	
8	G	69.73	34.95	7.35	49.5	86.58	4.25	97.11	21.85	63.48	66.18	
9	H	40.83	62.18	71.07	89.77	41.89	3.48	90.63	86.59	26.44	1.35	
10												
11												

The Quick Access Toolbar

- Excel provides a means to easily access macros with a single click.
- The mechanism is called the Quick Access Toolbar (QAT).
- Icons can be added to the QAT that will directly launch the desired macro.

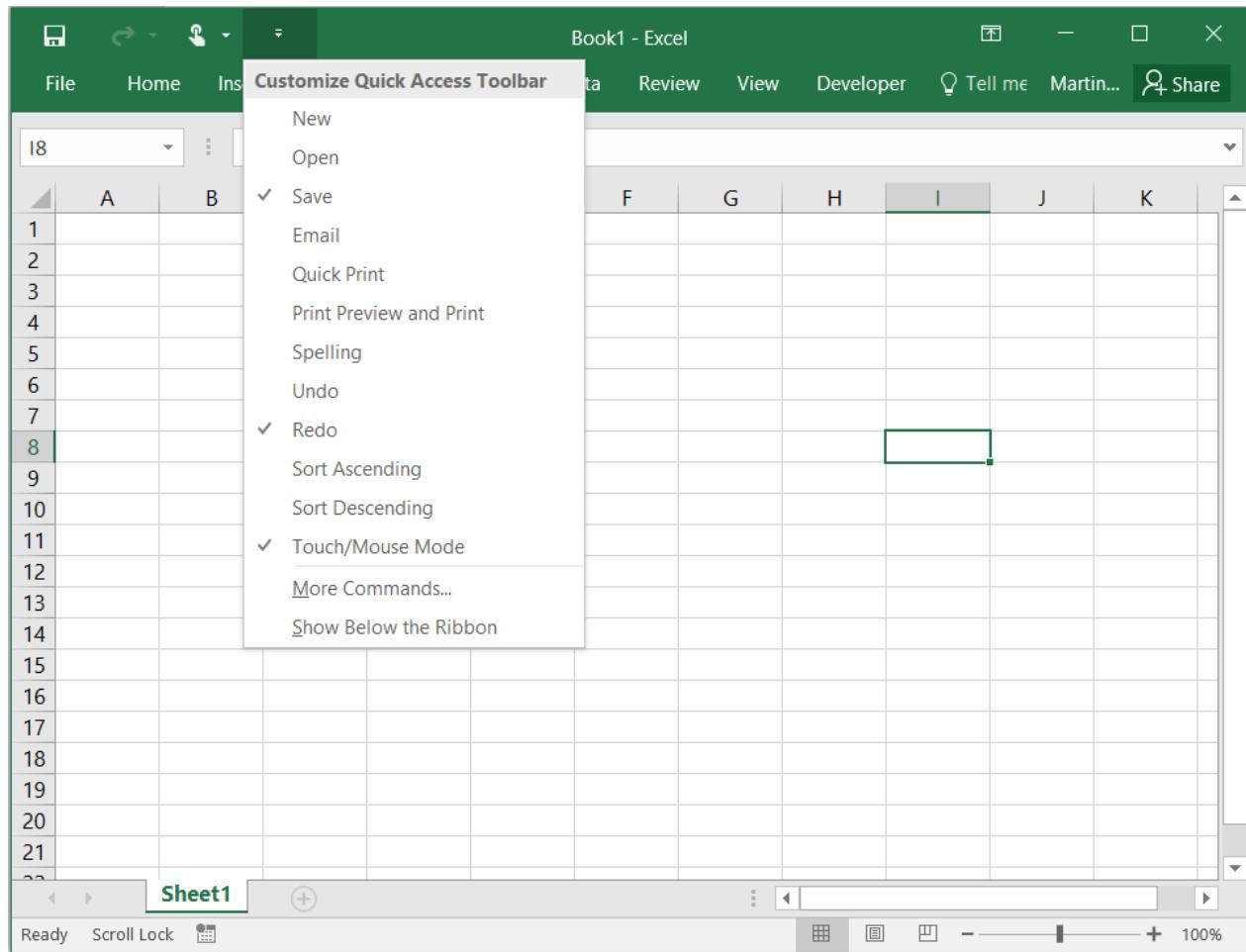
Quick Access Toolbar

- The QAT is located on the left end of the main menu, just to the right of the File Tab.
- At the right end of the Toolbar is the configuration button; .



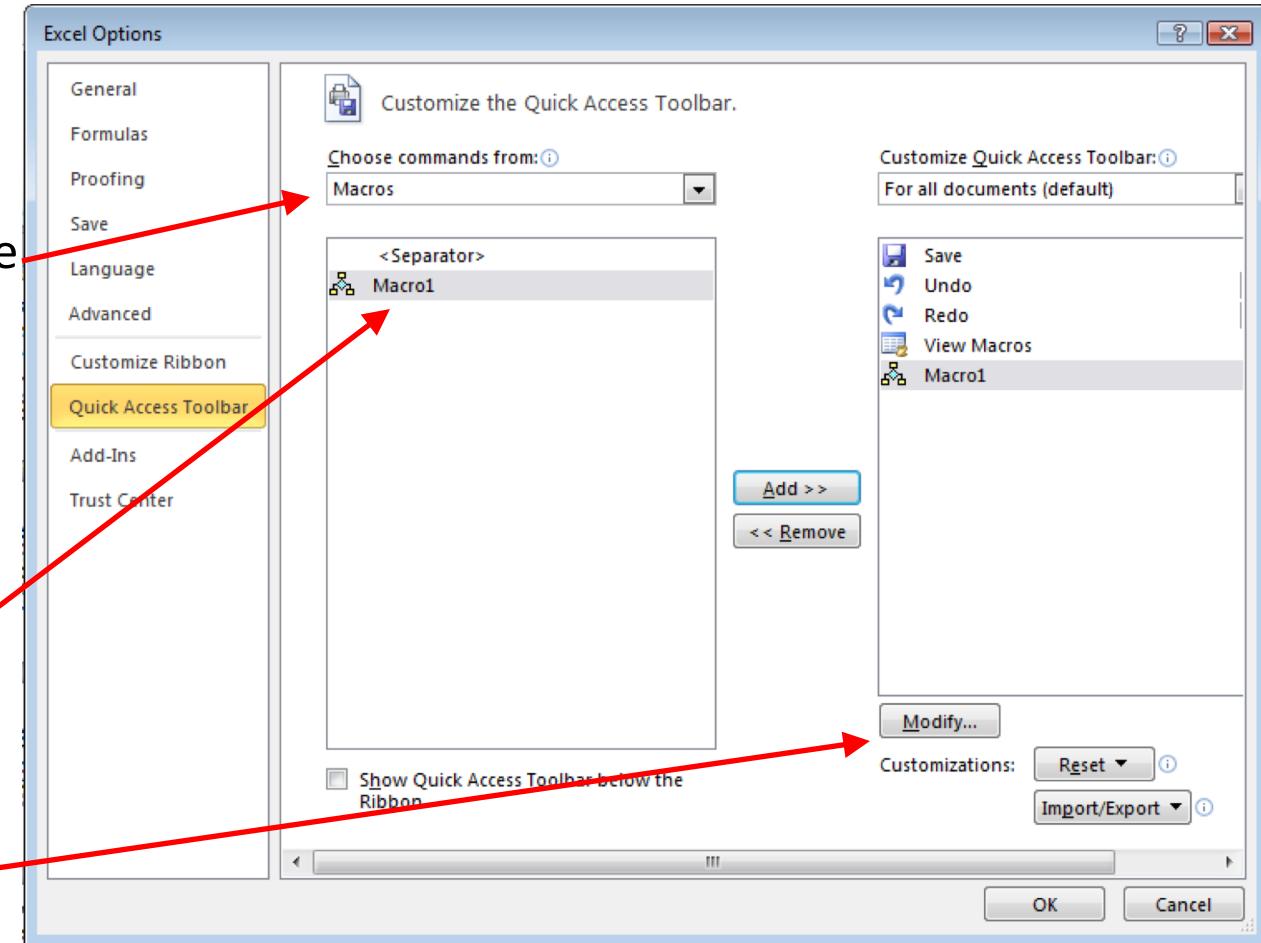
Adding a Macro to the QAT

- Click the button to reveal the Customization drop down.
- Select the More Commands option.



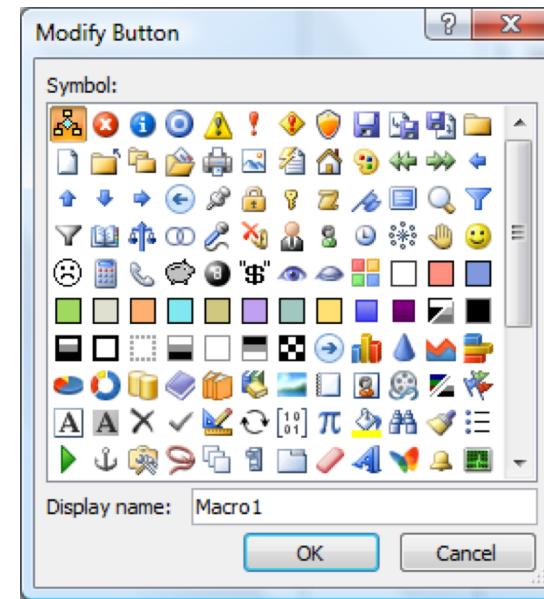
Adding a Macro to the QAT (cont.)

- The customize dialog opens.
- Change the Commands source to Macros.
- The macros that are in the Workbook will be listed. Select Macro1 and click the [Add] button.
- Select Modify to change the associated Icon



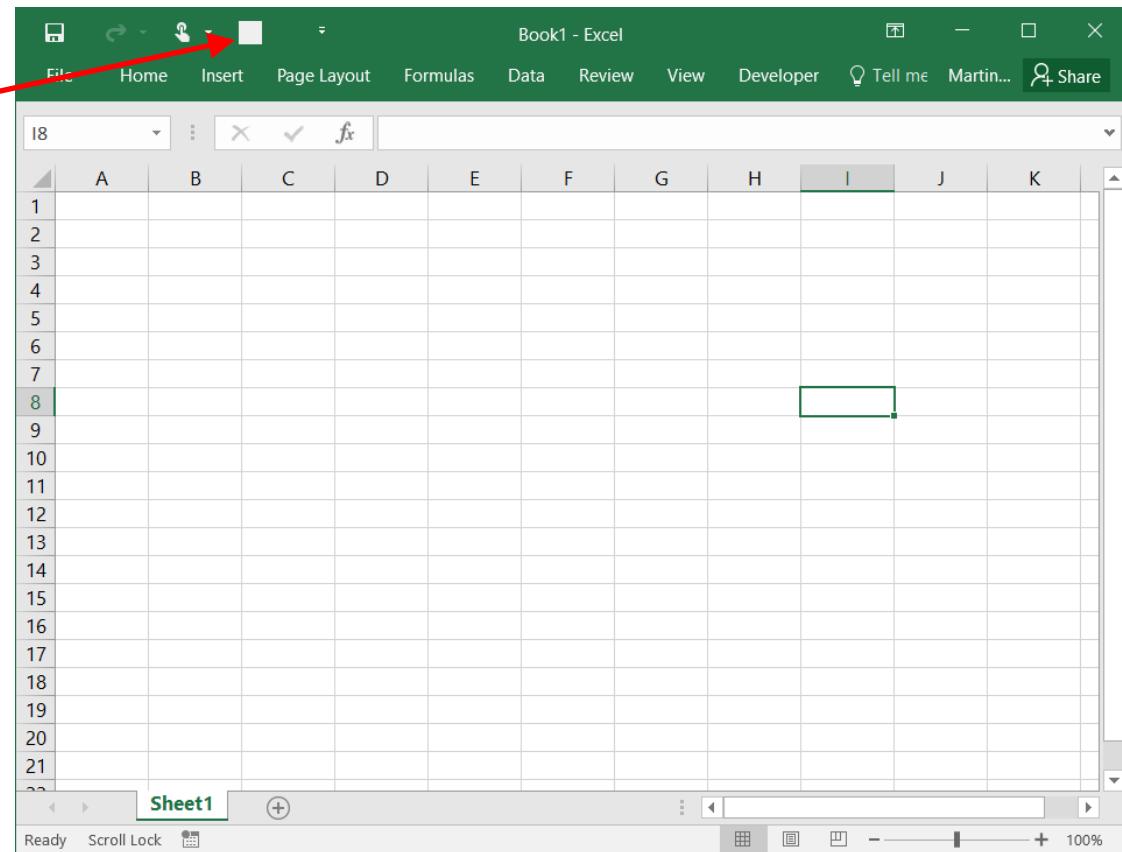
Modify Properties

- The Modify Button dialogue allows you to select an icon which will be displayed in the QAT and enter a display name.
- When you have made selections, click [OK] to complete the process.



Completed QAT

- The selected icon will appear as the left most item in the QAT.
- Hovering the mouse over the icon brings up the display name.
- Clicking the icon will cause Macro1 to execute.



Explicitly Assigning Variables in a Subroutine Call

- So far we have been passing parameters to functions and subroutines by passing **all** the parameters. For these examples we will use a subroutine with 3 parameters v1, v2 ,v3.
 - Example: DoSomething(5, 10, 20) v1= 5, v2=10 and v3=20
- To assign only specific parameters use the assignment operator ‘:=’ for the parameter(s) you want to pass. All other parameters will be empty.
 - Example: DoSomething(v2:= 5) v1 = 0, v2=5 and v3=0
- If a single parameter is being passed and it is the very first parameter then an assignment is not required.
 - Example: DoSomething(10) v1=10, v2=0 and v3=0

Example 17: A Custom Menu

- Delete the custom menu, if it exists
- Use an error handler that ignores any errors in case the menu does not exist
- Get the main menu bar and the Custom menu item.
- Delete the Custom menu item

```
Public Sub DeleteMenu()
    ' Delete the custom menu item, if it exists
    Dim mbMain As CommandBar
    On Error Resume Next

    Set mbMain = Application.CommandBars.ActiveMenuBar
    mbMain.Controls.Item("Custom").Delete

End Sub
```

Scope applies to Subroutines and Functions too.

Example 17: A Custom Menu

- Add a new menu item to the main menu bar.

```
Public Sub AddMenu()
    ' Add the custom menu to Excel's menu bar

    Dim mbMain As CommandBar
    Dim mnuCustom As CommandBarPopup
    Dim mnuItem As CommandBarButton

    ' First delete the menu if it exists
    DeleteMenu

    ' Get the main menu bar
    Set mbMain = Application.CommandBars.ActiveMenuBar

    ' Add a new menu popup to the menu bar and name it
    Set mnuCustom = mbMain.Controls.Add(Type:=msoControlPopup, Temporary:=True)
    mnuCustom.Caption = "Custom"

    ' Add a new menu item to the menu popup and set its properties
    Set mnuItem = mnuCustom.Controls.Add(Type:=msoControlButton, ID:=1)
    mnuItem.Caption = "Import"
    mnuItem.TooltipText = "Import"
    mnuItem.Style = msoButtonCaption

    ' Attach the macro to the menu option
    mnuItem.OnAction = "Macro1"

End Sub
```



Example 17: A Custom Menu

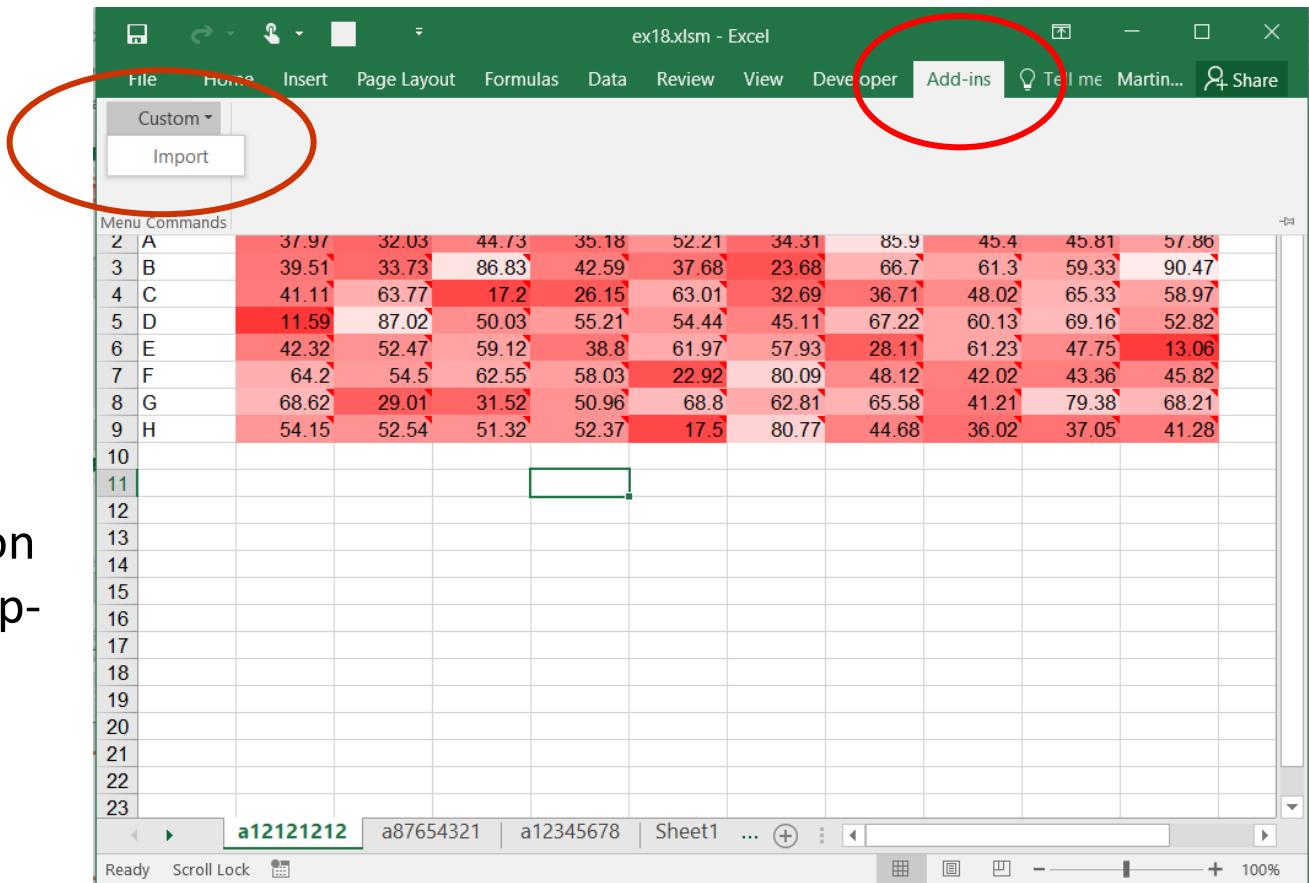
- Delete the custom menu item when the Workbook closes
- Add the custom menu item when the Workbook opens
- Add the following calls to the ThisWorkbook module

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    DeleteMenu
End Sub

Private Sub Workbook_Open()
    AddMenu
End Sub
```

Example 17: A Custom Menu

- Save and close your Workbook
- Open it again
- Click the Add-In menu button.
- Then click the Custom button
- The Import option appears as a drop-down
- Test and debug your final macro



Example 18: Multiple Files at Once

- Extend Example 17 to load and process multiple files at once.
- The Application.GetOpenFilename method has an argument named MultiSelect that, if True, allows multiple file names to be selected in the File Dialog
- The method will return a Boolean, or an array of Variants with selected file names

Example 18: Multiple Files at Once

- Macro Modification Plan:
 - Change `FileName` variable to `Filenames` and dimension two new variables:
`FileName As String, J As Long`
 - Add a `MultiSelect:=True` parameter to `GetOpenFilename`
 - Change the way to check for a Cancel to account for the possibility of an array being returned
 - Wrap the entire import routine in a loop

Example 18: Multiple Files at Once

- Open file from Example 17 and save with new name for Example 18
- Delete the FileName declaration from Macro1 and replace it with the following declaration

```
Dim FileNames As Variant, FileName As String, J As Long
```

- Add the MultiSelect parameter to GetOpenFilename

```
FileNames = Application.GetOpenFilename(strFilter, _  
    MultiSelect:=True)
```

Example 18: Multiple Files at Once

- Change the way that a Cancel is detected. Remove:

```
If FileNames = False Then Exit Sub
```

and replace with:

```
If TypeName(FileNames) = "Boolean" Then  
    If FileNames = False Then Exit Sub  
End If
```

- Wrap the remainder of Macro1 in a loop. Add:

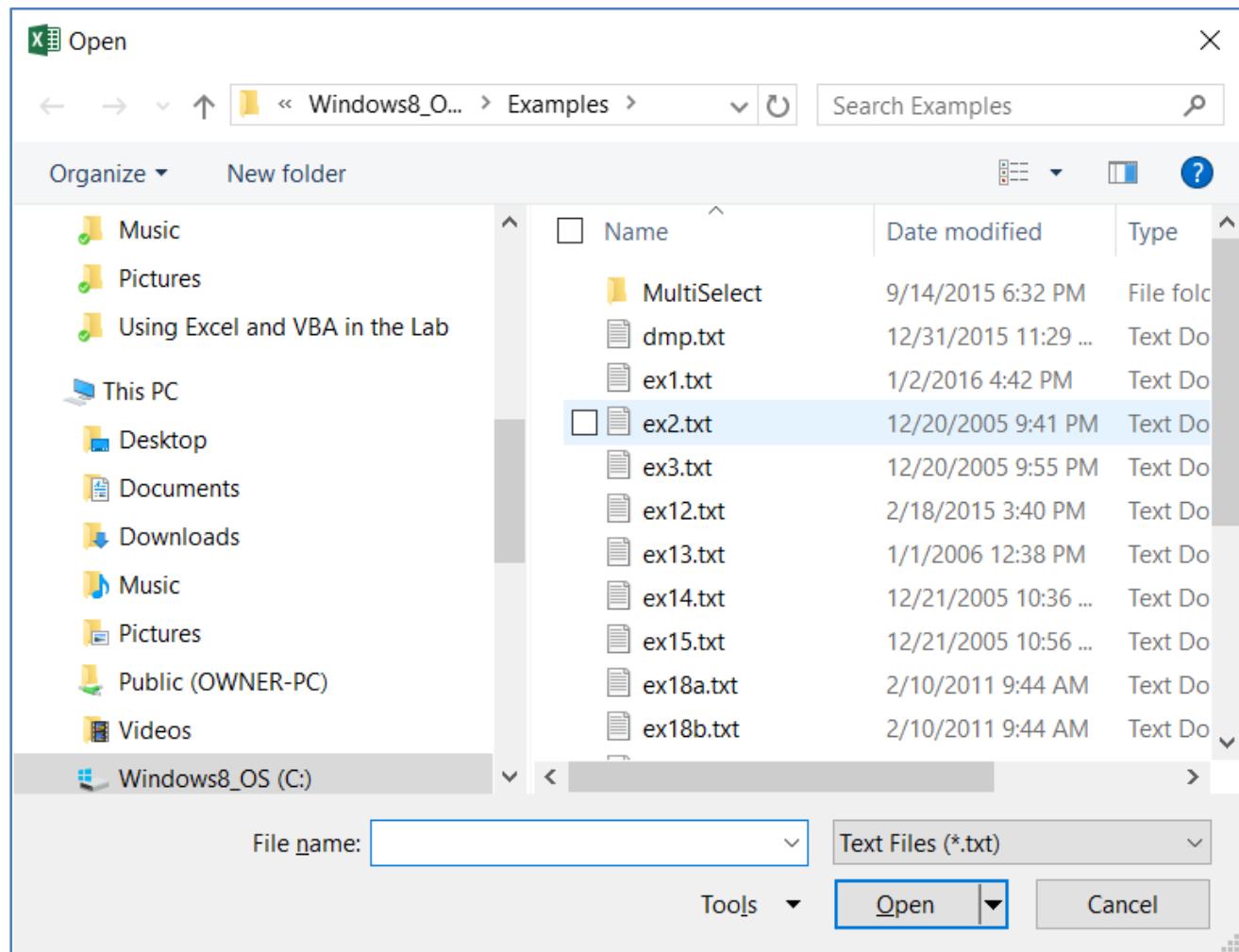
```
' Loop over each file selected  
For J = 1 To UBound(FileNames)  
    FileName = FileNames(J)
```

(all previous code here)

Next J

Example 18: Multiple Files at Once

- Give it a try
- The new File Dialog should allow multiple files to be selected



Example 18: Multiple Files at Once

- All files will be processed and loaded into the Workbook in once shot

	A	B	C	D	E	F	G	H	I	J	K
1		1	2	3	4	5	6	7	8	9	10
2	A	37.97	32.03	44.73	35.18	52.21	34.31	85.9	45.4	45.81	57.86
3	B	39.51	33.73	86.83	42.59	37.68	23.68	66.7	61.3	59.33	90.47
4	C	41.11	63.77	17.2	26.15	63.01	32.69	36.71	48.02	65.33	58.97
5	D	11.59	87.02	50.03	55.21	54.44	45.11	67.22	60.13	69.16	52.82
6	E	42.32	52.47	59.12	38.8	61.97	57.93	28.11	61.23	47.75	13.06
7	F	64.2	54.5	62.55	58.03	22.92	80.09	48.12	42.02	43.36	45.82
8	G	68.62	29.01	31.52	50.96	68.8	62.81	65.58	41.21	79.38	68.21
9	H	54.15	52.54	51.32	52.37	17.5	80.77	44.68	36.02	37.05	41.28
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											

Add-Ins

- What is an Add-In?
 - Program that adds optional commands and features to MS Office
 - Example – Excel analysis tool pack for statistics/engineering
 - Three types
 - Excel Add-in
 - Com (**C**omponent **O**bject **M**odel) add-in *
 - Automation Add-In *

* *Not covered in this course*

Add-Ins (*cont.*)

- Why use Add-Ins?
 - Excel functions and procedures created by others can be shared.
 - VBA Code is stored in a separate Excel workbook (*.xlam file).
 - No need to create a “template workbook”
 - Code updates are made to add-in (.xlam file) not the workbook (.xlsx)
 - Save time by not “recreating the wheel”

Add-Ins (*cont.*)

- Where to get Add-Ins?
 - Excel has many built-in Add-Ins
 - Microsoft web site <http://office.microsoft.com/downloads>
 - Your Company/Institution
 - Third party sites (Google search)*

* Warning – as with any macro be aware of the risk!

Add-Ins (cont.)

- How to use Add-Ins?
 - Install the Add-In on your computer
 - For custom written add-ins, save the document as an Add-in (*.xlam).
 - Location for add-ins
 - The Library folder or one of its subfolders in the C:/Program files/Microsoft Office/OfficeXX/XLStart folder*.
 - Documents and Settings/<user name>/Application Data/Microsoft/AddIns folder.
 - Load the Add-In into Excel – 2 Methods
 - Manual load - From the Developer Tab select Add-Ins – then select the add-in or browse to navigate to it.
 - Automatic load - Place the Add-In into the XLStart folder

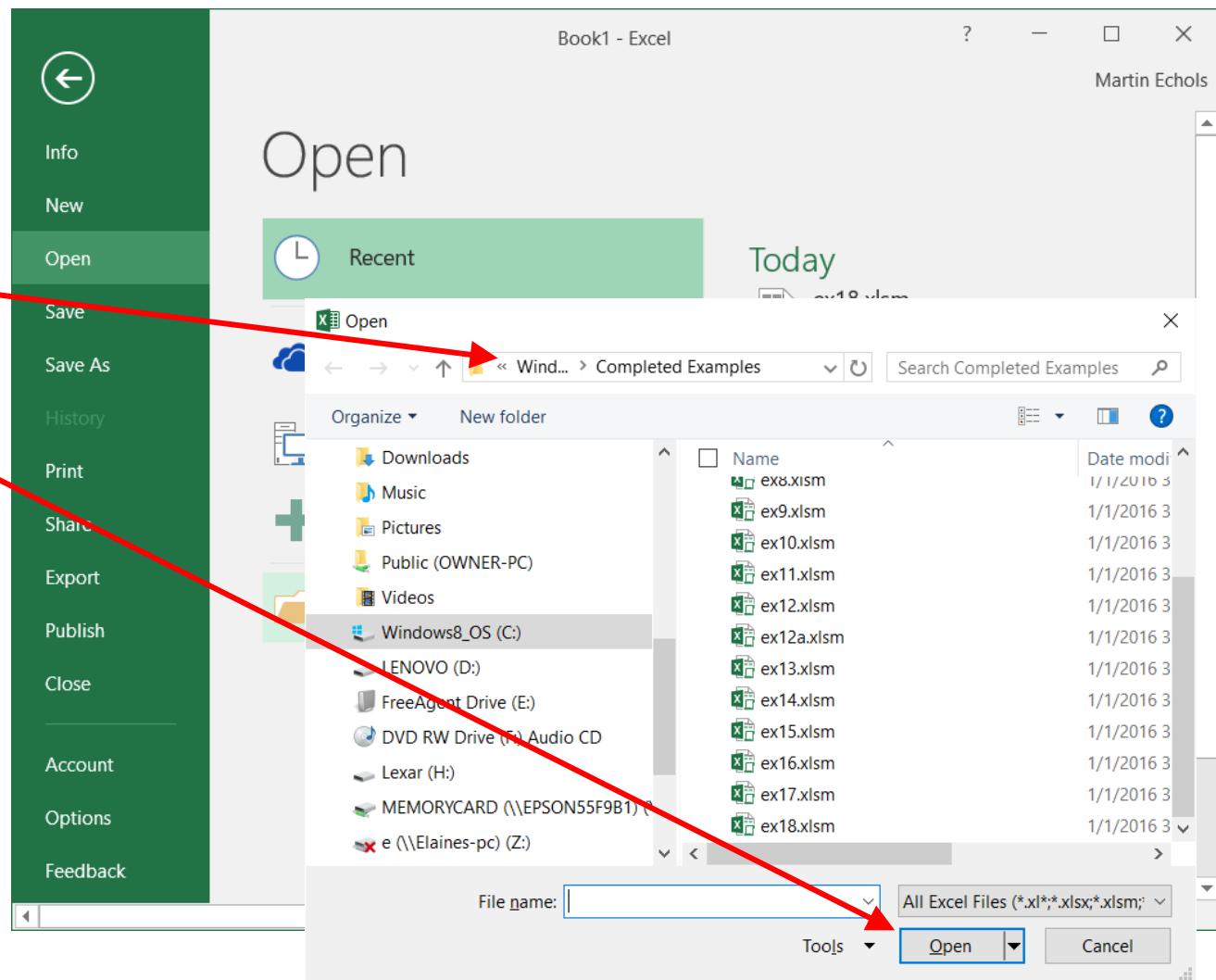
* Suggested location, XX = version # of Microsoft Office (e.g. 13= Excel 2007, 14= Excel 2010, 15= Excel 2013, 16= Excel 2016)

Converting Example 18 to an Add-In

- We can take example 18 and convert to an Add-In. Our goal is to create the Add-In and also prevent users from viewing or editing
 - 1) In the VB editor Project Explorer set the name and description of the VBA project and protect the file from viewing.
 - 2) Give the example a meaningful file name related to its function (e.g., HeatMap)
 - 3) Save the file to the XLSTART directory
 - 4) Close Excel
 - 5) Open up a new workbook.
 - 6) Launch the Add-In from the Custom Menu.

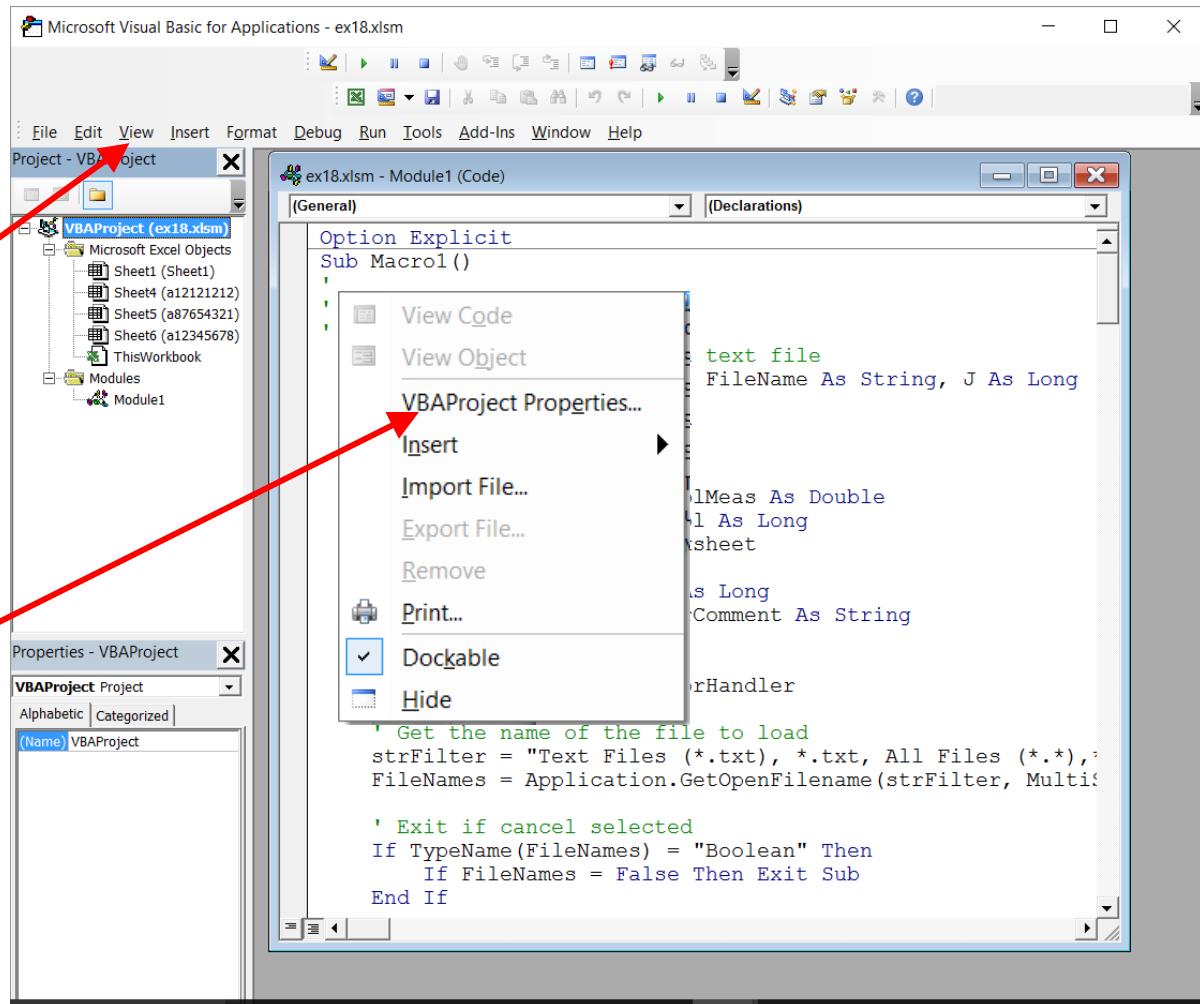
Creating an Add-In - Load Example 18

- Launch Excel and select File / Open
- Navigate to the completed folder and select the file
- Click Open



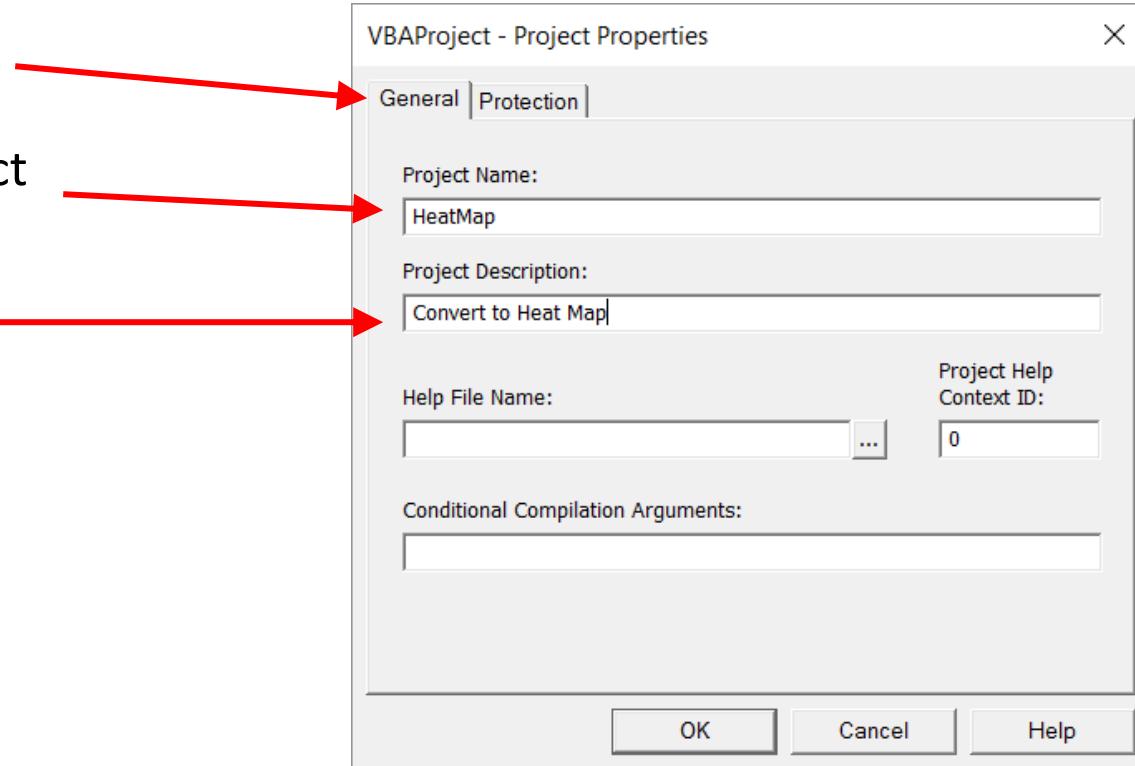
Creating an Add-In – Setting Project Properties

- Select ALT-F11 to toggle to the VBE
- Select the View | Project Explorer menu option
- Select ex18.xlsx
- Right click and select VBAProject Properties



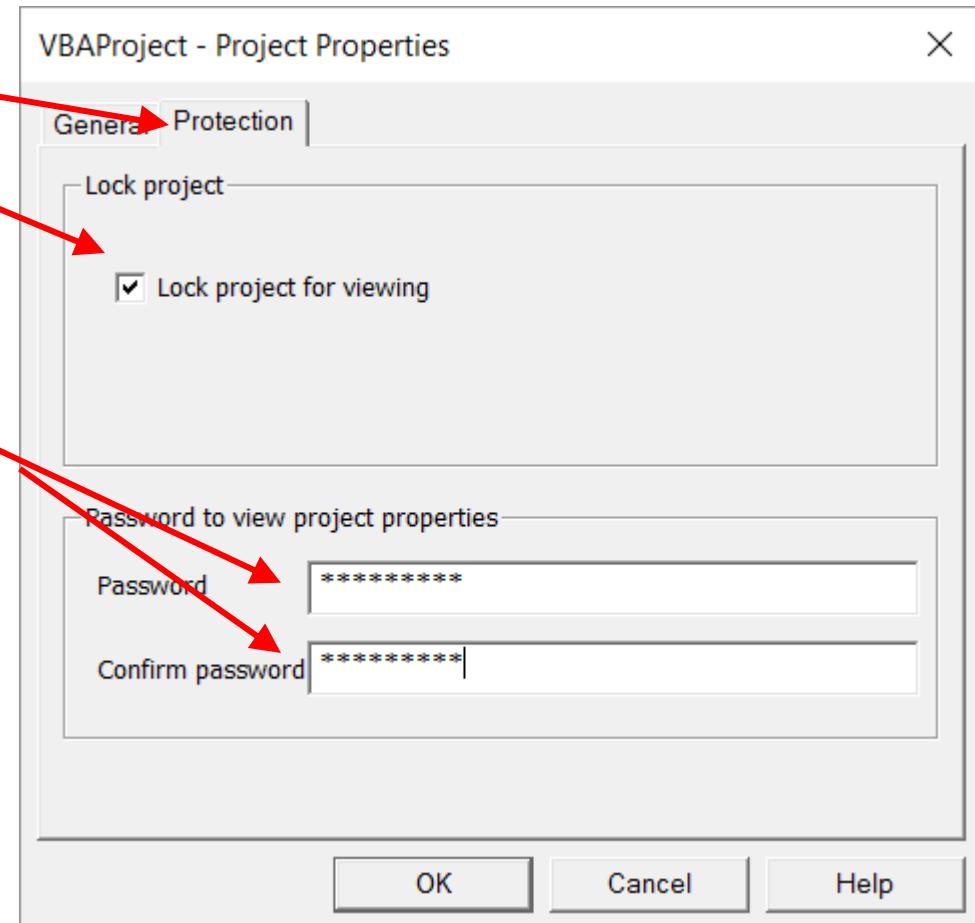
Creating an Add-In – Setting Project Properties (*cont.*)

- Select the General Tab
- Enter a Project Name and Description



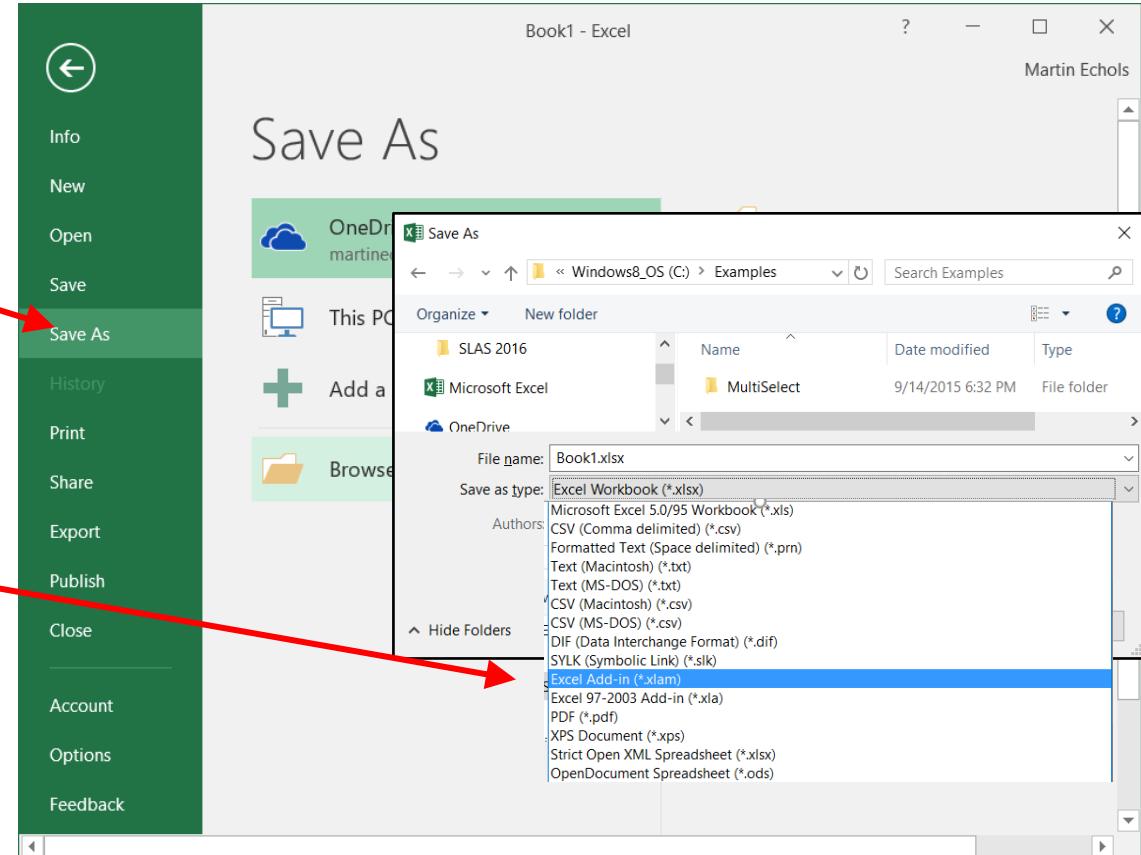
Creating an Add-In – Protecting the Add-In

- Select the Protection Tab
- Check Lock project for viewing
- Enter password and confirmation password of your choosing
- Select OK



Creating an Add-In – Saving the Add-In

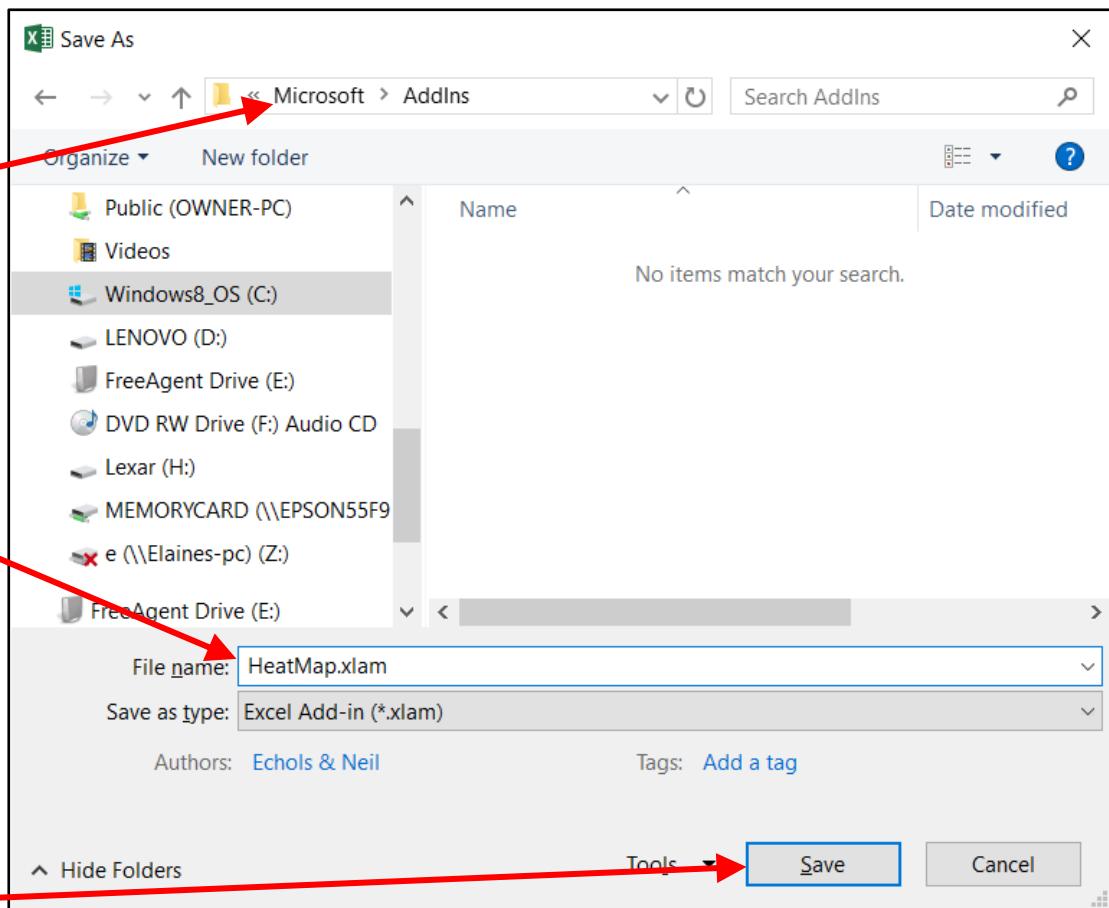
- Press ALT-F11 to toggle back to the Excel Workbook
- Click the File Tab
- Select the Save As menu option
- Ensure that the “Excel Add-In” file type is selected



Creating an Add-In – Saving the Add-In (cont.)

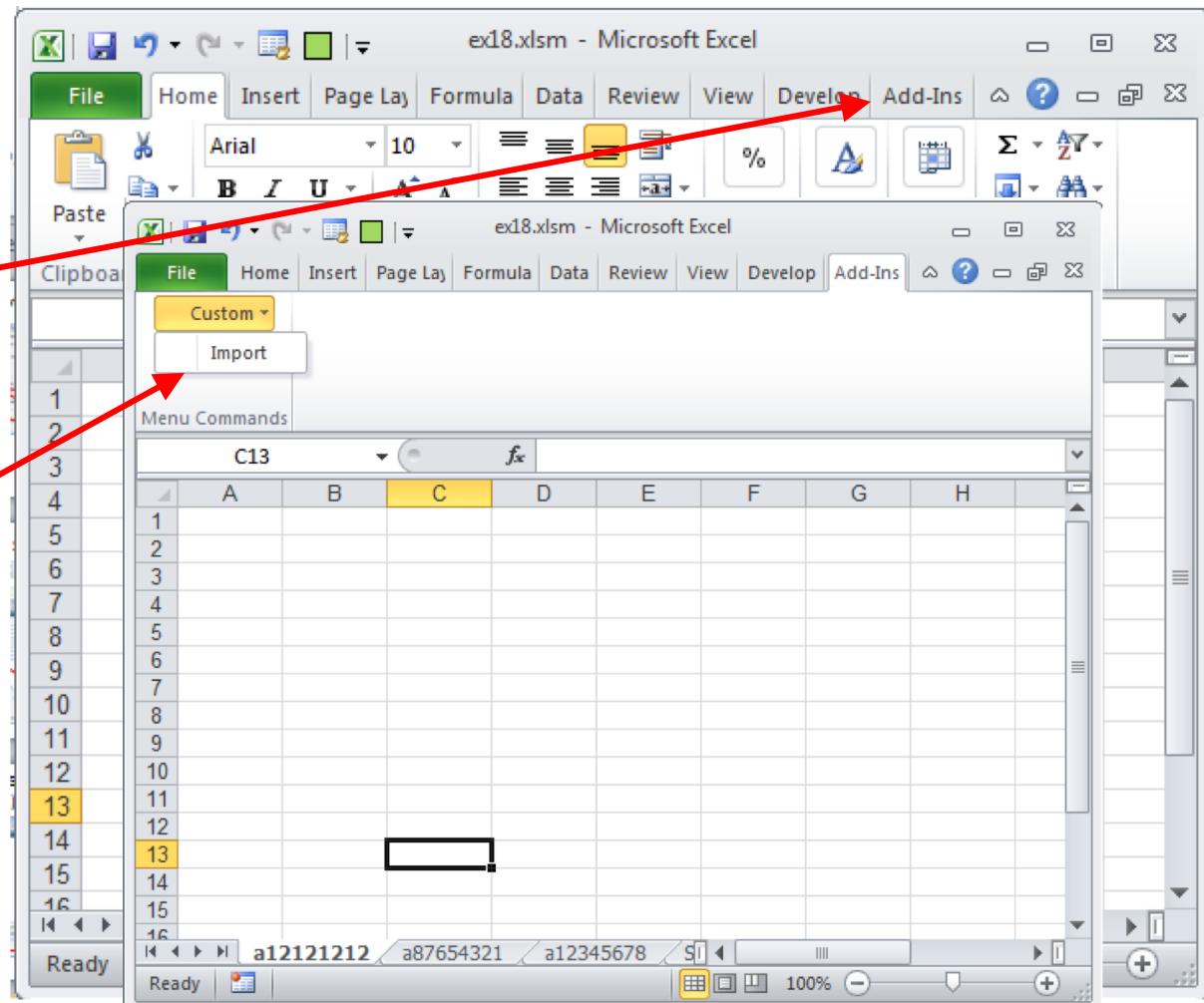
- Navigate to the ...\\Addins folder, this is dependent on your installation of 2016
- Enter *HeatMap.xlam* for the File Name

Click [Save]



Creating an Add In – Testing the Add-In

- Close Excel
- Create a New Workbook
- The Add-In Tab should appear after the Developer Tab
- Click the Add-In Tab to display the Custom Tab.
- Give it a try by clicking Import



Brief Introduction to Web Services

- Definition W3C
 - A software system designed to support machine-to-machine interaction over a *network*.
- Uses open standards and protocols (XML, SOAP, HTTP, etc.)
- Independent of hardware, operating system or programming language
- Makes accessing Oracle and other RDB's easier
- Can often be tested in a Web Browser

Brief Introduction to Web Services (Cont'd)

- **REpresentational State Transfer ful**
 - most commonly used type
 - provides scalability, performance and modifiability
 - simple, lightweight and fast for Web use
- Common HTTP Methods
 - GET Retrieve data
 - POST **Create data, Update data**
 - PUT Create data, **Update data**
 - DELETE Delete data

Web Services and Excel

- Two methods
 - Use Microsoft Library reference*
 - VBE tools references
 - good for performing more complex HTTP methods
 - PUT, POST, UPDATE, DELETE
 - requires knowledge of the object
 - requires reference Microsoft XML, v6.0
C:\windows\system32\msxml6.dll
 - Use the Two Intrinsic Functions
 - WEBSERVICE and XMLFILTER
 - Available in Excel 2013 & 2016
 - GET HTTP Method only

* not covered in this class

Web Services and Excel (cont'd)

- **WEB SERVICE (url)**
 - where url is the **uniform resource locator** (address of web page)
 - It is a GET request
 - Data is returned in a cell
 - #VALUE! Error occurs when
 - the webservice cannot return data based on the url
 - the url is more than 2048 characters
 - if the result is >32676 characters (max allowed in a cell)
 - if the result contains ftp:// or file://

Web Services and Excel (cont'd)

- FILTERXML (xml, path)
 - where xml is a reference to the xml string returned from the WebService function
 - path is the XPath command that will return the requested data
- XPath – XMLPath Language
 - W3School Recommendation
 - uses “path” like language (like file paths) to navigate and parse the data
 - over 200 functions, commands in language
 - all functions are not available in Excel

Xpath Commands ([W3Schools](#))

- Sample XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

- Xpath Commands

Expression	Description
<i>nodename</i>	Selects all nodes with the name " <i>nodename</i> "
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Xpath Example

- Sample XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

Element Name

Element Value

Attribute Name

Attribute Value

- Xpath Commands Examples

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore Note: If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
/bookstore/book[1]	Selects the first book element that is the child of the bookstore
/bookstore/book[1]/price	Selects the price of the first book
/bookstore/book[2]/@title	Selects the title of the second book

Example 1 – Chemistry Web Service From a Browser

- PubChem Web Service
 - Does not require an activation code to use
 - Excellent documentation
 - Can specify the return Data format (text, JSON, XML)
- Example – Click the following from the short course pdf to get the octanol-water partition coefficient (log P) of ibuprofen

[https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC\(C\)CC1=CC=C\(C=C1\)C\(C\)C\(=O\)O/property/XLogP/XML](https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC(C)CC1=CC=C(C=C1)C(C)C(=O)O/property/XLogP/XML)

Example 1 Browser Display Results

```
<?xml version="1.0"?>
- <PropertyTable xmlns="http://pubchem.ncbi.nlm.nih.gov/pug_rest" xs:schemaLocation="http://pubchem.ncbi.nlm.nih.gov/pug_rest.xsd">
  - <Properties>
    <CID>3672</CID>
    <XLogP>3.5</XLogP>
  </Properties>
</PropertyTable>
```

Example 1 – Chemistry Web Service From Excel

Close and save your current workbook, close Excel and Reopen Excel and type the following into Cell A1 of Sheet 1

```
=WEBSERVICE("https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC(C)CC1=CC=C(C=C1)C(C)C(=O)O/property/XLogP/XML")
```

In cell A2 type

```
= FilterXML(A1, "//XLogP")
```

Also try the following

```
=FilterXML(A1, "/XLogP")
```

```
=FilterXML(A1, "XLogP")
```

```
=FilterXML(A1, "//Properties/XLogP")
```

```
=FilterXML(A1, "//Properties//XLogP")
```

Warning.
FilterXML Xpaths
are case sensitive

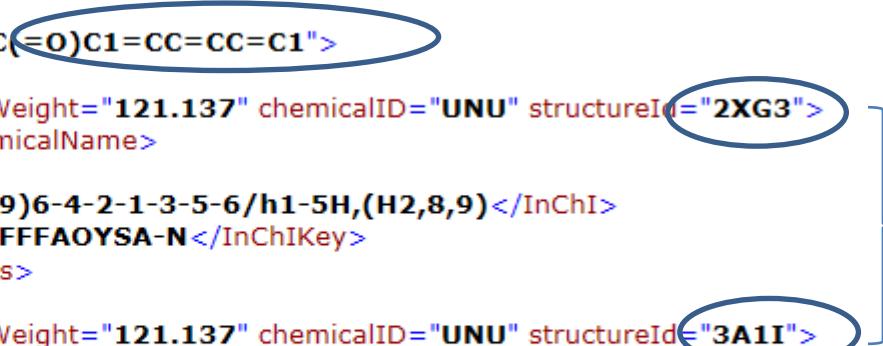
What worked and
what did not?
Give reasons for
each

Example 2 –Biology Web Service from a Browser

- RSCB Protein Data Bank
 - Does not require an activation code to use
 - Good documentation
 - Can return only XML data
- Example – Click the following link from the short course pdf to get a list of all proteins that bind to the ligand Benzamide based on its SMILES input.
- [https://rcsb.org/pdb/rest/smilesQuery?smiles=pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smilesNC\(=O\)C1=CC=CC=C1&search_type=exact](https://rcsb.org/pdb/rest/smilesQuery?smiles=pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smilesNC(=O)C1=CC=CC=C1&search_type=exact)

Example 2 Browser Display Results

```
<?xml version="1.0"?>
- <smilesQueryResult search_type="4" smiles="NC(=O)C1=CC=CC=C1">
  - <ligandInfo>
    - <ligand type="non-polymer" molecularWeight="121.137" chemicalID="UNU" structureId="2XG3">
      <chemicalName>BENZAMIDE</chemicalName>
      <formula>C7 H7 N O</formula>
      <InChI>InChI=1S/C7H7NO/c8-7(9)6-4-2-1-3-5-6/h1-5H,(H2,8,9)</InChI>
      <InChIKey>KXDAEFPNCMNJSK-UHFFFAOYSA-N</InChIKey>
      <smiles>c1ccc(cc1)C(=O)N</smiles>
    </ligand>
    - <ligand type="non-polymer" molecularWeight="121.137" chemicalID="UNU" structureId="3A1I">
      <chemicalName>BENZAMIDE</chemicalName>
      <formula>C7 H7 N O</formula>
      <InChI>InChI=1S/C7H7NO/c8-7(9)6-4-2-1-3-5-6/h1-5H,(H2,8,9)</InChI>
      <InChIKey>KXDAEFPNCMNJSK-UHFFFAOYSA-N</InChIKey>
      <smiles>c1ccc(cc1)C(=O)N</smiles>
    </ligand>
  ... this list continues
```



4-character PDB identification code

Example 2 –Biology Web Service Excel

Close and save your current workbook, close Excel and Reopen Excel and type the following into Cell A1 of Sheet 1

```
=WEBSERVICE("https://www.rcsb.org/pdb/rest/smilesQuery?smiles=NC(=O)C1=CC=CC=C1&search_type=exact")
```

In cell A2 type

```
= FILTERXML(A1, "//ligand/@structureId")
```

Based on the Xpath documentation , what is the purpose of the @ sign?

Which is the four letter code returned? Which item # is it returning?

To get a list of Ids you can enter an array formula by

- 1) highlighting A2 and drag down 4 rows
- 2) placing the cursor in the formula box and pressing Shift-CTRL-Enter
- 3) To clear an array, delete the selected range

To get the 2nd structure ID you can enter

```
= FilterXML(A1, "//ligand[2]/@structureId")
```

Example 19 Web Service

Plan for incorporating Web services into Macro1

- 1) Create a function to get the LogP from the Web Service.
- 2) Declare 2 new variables; one for the SMILES string and for the LogP.
- 3) Modify the **StringToDouble** function to skip over the LogP value so it won't be included in the average.
- 4) Modify the main Macro to parse the smiles string and pass the value to the LogP function.
- 5) Store the LogP in the comments field.

Example 19 Web Service

```
Function String.ToDouble(strArray() As String) As Double()
```

```
    Dim i As Long
```

```
    Dim dblArray() As Double
```

```
    Dim IngForCounterMax As Long
```

```
' Get the number of variables in the array - 2 because  
' we don't want to include the well location or the log P value in the array
```

```
IngForCounterMax = UBound(strArray) - 2
```

```
'Resize the double array
```

```
ReDim dblArray(IngForCounterMax)
```

```
For i = 0 To IngForCounterMax
```

```
    dblArray(i) = CDbl(strArray(i + 2)) ' add 2 to skip over well location and Log P
```

```
    Debug.Print dblArray(i)
```

```
Next i
```

```
Debug.Print ""
```

```
String.ToDouble = dblArray()
```

```
End Function
```

Make these changes to the existing function

Example 19 Web Service (cont'd)

'Declare your new variables at the top of Macro1

```
Dim strSmiles As String  
Dim strLogP As String
```

*Modify the main macro and add the parts **bolded** below*

```
Do Until EOF(intFileNum)  
    Line Input #intFileNum, strLine  
    strComment = "Raw Data:" & vbLf  
  
    If Len(Trim(strLine)) > 0 Then  
  
        ' Parse the data  
        strParts = Split(strLine, ",")  
        strWell = strParts(0)  
strSmiles = strParts(1)  
  
        ' Call new function to convert Strings to Double  
        dblParts = StringtoDouble(strParts)  
  
        ' Use the Worksheet Function Average  
        dblMeas = WorksheetFunction.Average(dblParts)  
        dblMeas = Round(dblMeas, 2)  
strLogP = LogP(strSmiles)  
  
        ' Replace the smiles string with the LogP value  
strParts(1) = "LogP : " & strLogP
```

Example 19 Web Service (cont'd)

There are two files for Example 19

Ex19-3.txt contains 3 samples.

Ex19-96.txt contains 96 samples.

Import Ex19-3.txt first since our network speed might be slow.

Hover over a cell – What do you see?

Extra Examples

Name Of File	Purpose
InvDataBase.xls	A datafile source in the form of an Excel workbook for the DBAccessExample.xlsxm file. It has no macros, just data.
InvDataBase.mdb	A datafile source in the form of an Access Database for the DBAccessExample.xlsxm file. It has no macros, just data. It is for Microsoft Access 2003 and earlier.
InvDataBase.accdb	A datafile source in the form of an Access Database for the DBAccessExample.xlsxm file. It has no macros, just data. It is for Acess2007 and greater.
DBAccessExample.xlsxm	An example of how to use read info from DataBases whether it is Excel, Access or Oracle. You can retype the barcode into the first column and watch it retrieve the data. The barcodes that are used for testing are in the InvDataBase.xls or InvDatabase.mdb. It uses connection strings to access the various databases. Look up “connection strings” in google for more details.
Example18AndOn.xlsxm	Takes the examples that we did an adds many more features, including forms, different colors for the heatmap etc. See the readme in the workbook.
SigFigures.xlsxm	This example shows data according to selected significant figures. It first uses a combination of built in Excel functions, then it “wraps” that function into a custom user defined function (udf). Look at the function called SigFigures in Module 1
PlateUtilityFunction.xlsxm	This example has a list of several functions that do convert from various plate indexing formats. Similar to the subroutine that we did in class called Well2RowCol but since it is a function it only returns one value. So to get the row similar to Well2RowCol use the WellOf96ToRow and to get the column use the WellOf96ToCol. See sheet1 for other functions that you can use. Read the material about add-ins because these functions are just like the built in Excel functions and if you make an add-in, they will be available to anyone that has the add-in in their startup folder.
GettingFileList.xlsxm	Shows an Example of How to get a list of files in a function automatically without using GetOpenFileName

Other Topics?