Data Preparation & Automation in Excel

WORKSHOP FOR NAVY

Data Preparation & Automation in Excel

Course Synopsis:

This course aims to provide participants with the knowledge and ability to clean, manage and manipulate data using Microsoft Excel and the functions it provides as well as using Visual Basic Applications (VBA) to automate tasks in Excel.

Learning Objectives:

Upon successful completion of this course, the participants will be able to:

1. Perform basic data preparation, data cleaning and mashup using Excel
2. Identify data and manipulate them using functions and formulas in Excel
3. Manipulate across multiple worksheets and multi-dimensional tables
4. Perform data validation and checking using Excel
5. Solve a given problem and design the solution
6. Apply programming disciplines in Excel VBA programming
7. Design macros to automate task in Excel environment
8. Develop macros using Visual Basic Applications (VBA)

Duration:

4 days course

Lesson Outline

Day 1	
Time	**Topics**
9:00 – 10:15am	**Introduction to Problem Solving and Data Preparation** ➢ Problem Solving and Decision Making ➢ Problem Solving Process ➢ Analysis of Problem
10:15-10:30am	**Tea Break**
10:30-12:30pm	**Data Preparation & Processing** ➢ Using Excel to load data from text files ➢ **Using Excel for data transformation and cleansing** Experiential Learning ➢ Data Processing using Excel
12:30-1:30pm	**Lunch**
1:30-3:00	Experiential Learning • **Applying Logic in Decision Making** • Using relational operators • Using IF and Nested IF • Using Logical Functions
3:30-3:45pm	**Tea Break**
3:45-5:00pm	Experiential Learning (cont…) • **Using VLOOKUP** • Using Conditional Formatting

Day 2	
Time	**Topics**
9:00 – 10:30am	**Organizing Data and Effective Analysis** ➢ Fields and String manipulation ➢ Combining multiple workbooks ➢ Performing data validations ➢ **Using PivotTable** ➢ **& PivotChart for summary and Reporting** <u>Experiential Learning</u> ➢ Organizing Data and Effective Analysis • Manipulating fields in a worksheet • Setting data validation rules • Merging multiple workbooks using macro
10:30-10:45am	**Tea Break**
10:45-12:30pm	<u>Experiential Learning</u> (cont…) • **PivotTable & PivotChart** • Creating utilization chart
12:30-1:30pm	**Lunch**
1:30-3:00	<u>Experiential Learning</u> ➢ More Exercises to recap ➢ Case Study Discussion & Breakout rooms
3:00-3:15pm	**Tea Break**
3:15-5:00pm	**Case Study Discussion & Breakout rooms** ➢ Final presentation and sharing ➢ Question and Answer

Day 3	
Time	**Topics**
9:00 – 10:30am	**Understanding problem solving with use of flow charts** **Introduction to Visual Basic for Applications (VBA)** ➢ What is macro in Excel? <u>Experiential Learning</u> • Writing & recording first macro • Testing the macro

	• Editing a macro • Assigning macro
10:30-11:00 am	**Tea Break**

11:00 - 12:30pm	**Working with the Visual Basic Editor (VBE)**
	• VBE components
	• Working with Project window
	• Working with Code windows
	
	VBA Programming Fundamentals
	• Understanding Objects
	• Comments
	• Variables, Data Types & Constants
	• Assignment statements
	• Arrays
	• Object variables
	• User-Defined Types
	• Manipulating Objects and Collections
	• Conditionals and Loops
	
	<u>Experiential Learning</u>
	• Working on simple VBA program focusing on:
	  ○ Variables, data types and constants
	  ○ Assignment statements
	• Working on VBA program with focus on:
	  ○ Arrays
	  ○ User-defined types of data

12:30 - 1:30pm	**Lunch**

1:30 – 3:00pm	<u>Experiential Learning</u> (cont…)
	• Working on VBA program with focus on:
	  ○ Objects and Collections
	  ○ Conditionals & Loops

3:00 - 3:30pm	**Tea Break**
3:30 - 5:00pm	**VBA Procedures** • Sub-procedures • Executing procedure using Run Sub/Userform & Macro dialog • Calling procedures from module • Passing arguments • Error handling • Creating functions

	Experiential Learning • Writing a sorting procedure • Writing a function

Day 4	
Time	**Topics**
9:00 – 10:30am	***Recap of Day 1*** **Understanding Excel's Events** ➢ Event-handler procedures ➢ Types of events ➢ Workbook-level events **External Data and Files** ➢ Data Connection ➢ Power query ➢ How to work with text files & file operations **Working with UserForm** ➢ Introduction to UserForm
10:30 – 11:00am	**Tea Break**
11:00 – 12:30pm	Experiential Learning • Working on File operations • Working on UserForm
12:30 - 1:30pm	**Lunch**
1:30 - 3:00pm	**Capstone Project on using Excel & Macros** Experiential Learning • Capstone Project on "Writing Macros using VBA" for Excel • Team project of 3-4 members in a team
3:00 – 3:30pm	**Tea Break**
3:30 – 5:00pm	**Project Sharing and Discussion**

Day 1 - Data Processing Using Excel

WORKSHOP FOR NAVY

Learning Outcomes:

At the end of this session, you will be able to:

a) Use Excel to load data from text files
b) Use Excel for data transformation and cleaning

Software(s): Microsoft Excel 2019

Instructions:

- Download the datasets for this hands-on.
- Launch Microsoft Office Excel.
- Select a blank workbook and complete the following exercises.

Excel: Get & Transform

Excel Get & Transform is also called **Power Query**. Power Query used to be an add-in that can be downloaded from Microsoft and install in Excel versions 2010 and 2013, only on Windows, not on Mac. It has been fully integrated in Excel 2019, in the Data tab, in the Get & Transform data group.

Get & Transform is a tool designed specifically for cleansing data. It gets data from tables, other workbooks, CSV files, databases, and even web pages and Facebook. Transform messy data into something useful by merging data, splitting columns and reorganizing data.

Exercise 1: Import Data from Text files

Objective: To import the fixed-width text file data (temperature.txt) into Excel using Excel's text import wizard.

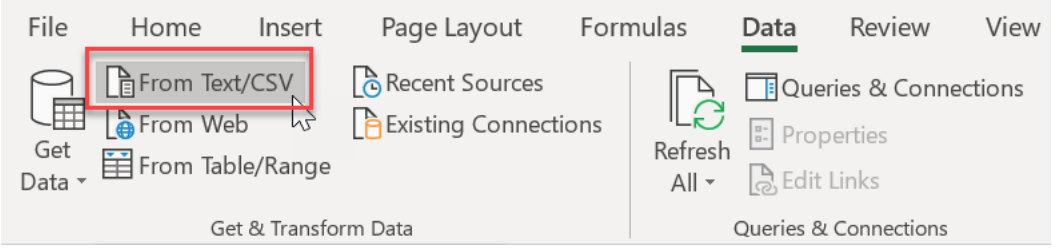
The text file was downloaded from the Web. It contains state, regional, and national (srn) annual data (1895-2005) on temperature. A small portion of the data is shown below.

```
001021895  43.70  37.60  54.50  63.10  69.80  78.10  79.90  79.90  77.90  60.60  53.40  45.60
001021896  44.10  47.90  52.50  67.90  75.70  77.00  80.80  82.00  75.50  63.50  57.50  46.20
001021897  42.60  51.20  60.20  62.00  68.60  80.90  81.00  78.70  75.30  66.90  54.30  47.80
001021898  49.40  45.90  59.00  58.10  73.40  80.30  79.80  78.60  75.10  61.10  49.90  44.10
001021899  44.40  39.90  55.20  61.60  75.80  79.70  80.20  81.10  72.50  66.00  55.50  44.90
001021900  44.00  44.10  52.60  63.50  71.20  76.30  79.70  81.50  77.60  68.70  54.90  46.80
001021901  46.10  43.10  53.00  57.40  70.00  78.40  82.10  78.50  72.00  62.50  48.70  41.70
001021902  43.20  40.60  54.80  61.60  75.30  80.80  82.60  82.00  73.00  63.30  57.60  45.30
001021903  43.60  48.20  59.20  60.70  69.40  73.20  79.90  80.40  73.00  63.20  50.90  40.80
001021904  41.70  49.20  58.20  60.00  69.60  77.90  78.30  78.20  76.60  64.50  52.20  46.70
001021905  39.10  39.50  59.50  63.50  74.40  79.10  79.40  79.10  75.90  64.20  55.80  43.70
001021906  47.20  45.60  51.50  64.80  69.70  79.00  78.80  80.40  78.10  60.90  55.80  50.30
001021907  54.30  49.10  64.50  58.30  68.10  75.60  80.80  80.40  74.60  63.20  51.40  46.20
001021908  44.30  44.30  62.20  66.90  71.40  77.50  79.80  79.30  74.10  60.40  56.90  50.60
001021909  50.00  50.70  56.10  63.00  68.60  77.90  79.80  80.80  74.70  63.60  59.40  40.80
001021910  45.50  45.30  61.00  61.50  68.60  75.40  78.40  79.40  77.10  66.30  51.20  42.50
```

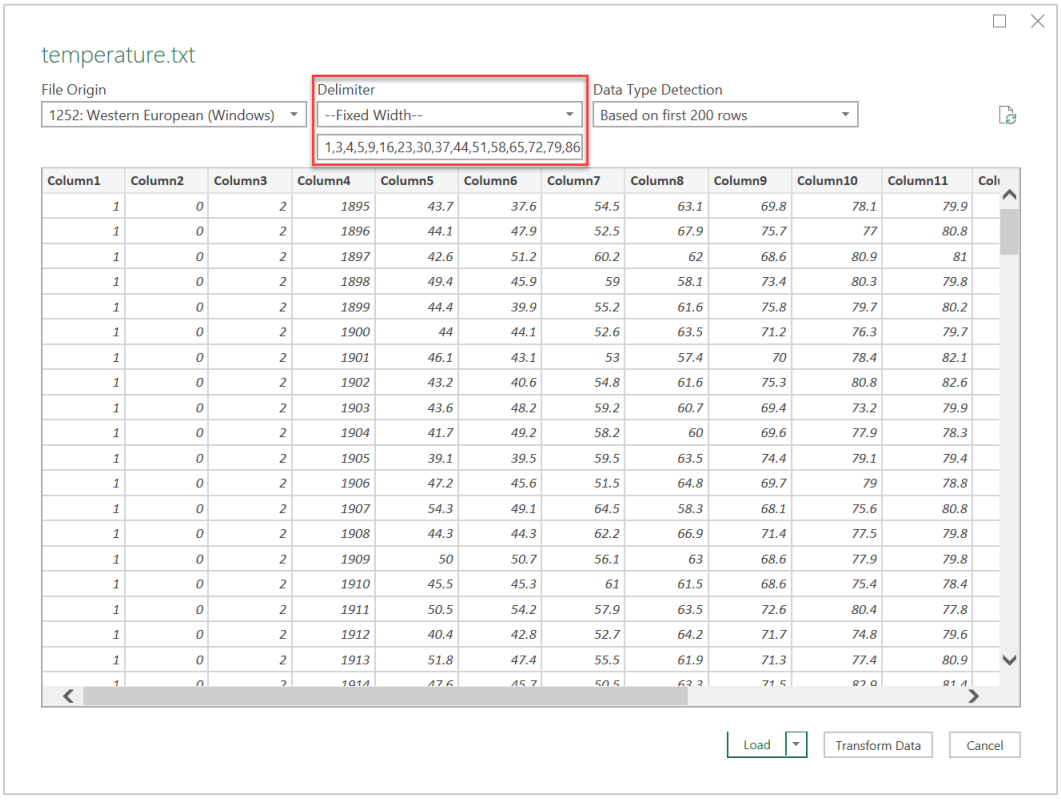
Additionally, the Web site from which the text file was downloaded also has a data dictionary (dictionary.txt) that indicates what the variables are and how they are stored in columns. Part of this data dictionary is shown below.

```
FILE FORMAT:

Element          Record
Name             Position  Element Description

STATE-CODE         1-3     STATE-CODE as indicated in State Code Table above.
                           Range of values is 001-110.

DIVISION-NUMBER     4      DIVISION NUMBER.  Value is 0 which indicates an area-
                           averaged element.

ELEMENT-CODE        5      1 = Precipitation
                           2 = Temperature (adjusted for time of observation
                                 bias)
                           5 = PDSI

YEAR               6-9     This is the year of record.  Range is 1895 to current
                           year processed.

JAN-VALUE         10-16    Monthly Temperature format:  Range of values b-50.00
                           to b140.00 degrees Fahrenheit.  Decimals retain a
                           position in the 7-character field.  Missing values in
                           the latest year are indicated by b-99.90.

                           Monthly Precipitation format:  Range of values 00.00
                           to 99.99.  Decimal point retains a position in the
                           7-character field.  Missing values in the latest year
                           are indicated by bb-9.99.
```

1. Under the **Data** tab, in the **Get & Transform Data** group, click **From Text/CSV**. Select the **temperature.txt** text file.



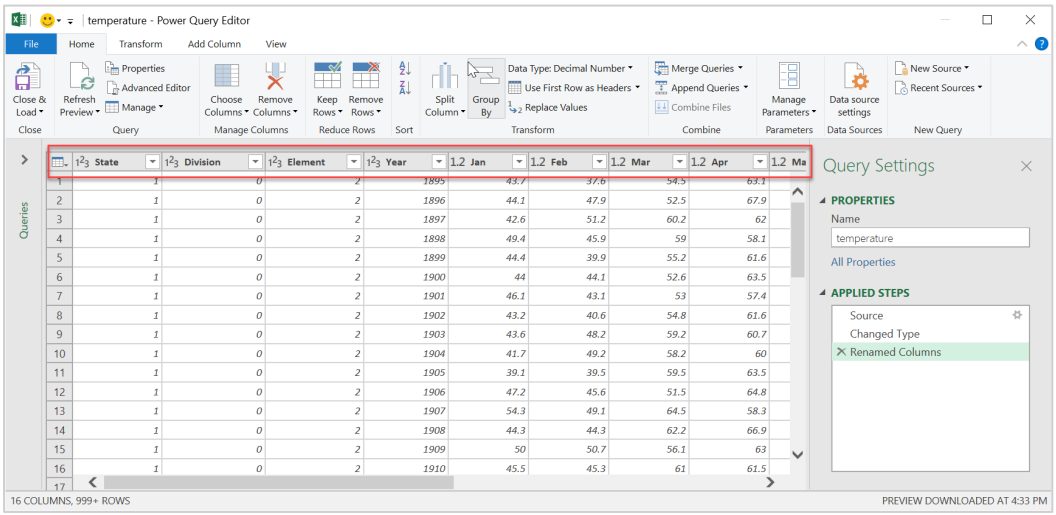
2. Select **--Fixed Width--** for the delimiter and enter the width to separate the columns as follows: **1,3,4,5,9,16,23,30,37,44,51,58,65,72,79,86**. Use the **dictionary.txt** file as a guide.



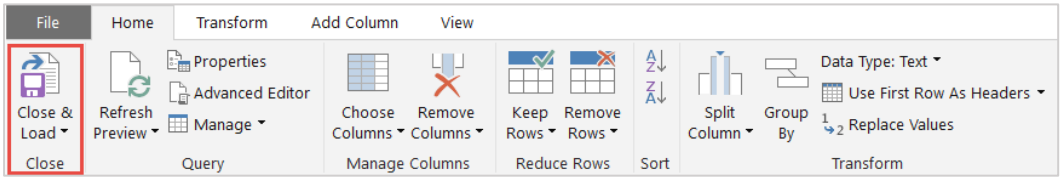
Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11
1	0	2	1895	43.7	37.6	54.5	63.1	69.8	78.1	79.9
1	0	2	1896	44.1	47.9	52.5	67.9	75.7	77	80.8
1	0	2	1897	42.6	51.2	60.2	62	68.6	80.9	81
1	0	2	1898	49.4	45.9	59	58.1	73.4	80.3	79.8
1	0	2	1899	44.4	39.9	55.2	61.6	75.8	79.7	80.2
1	0	2	1900	44	44.1	52.6	63.5	71.2	76.3	79.7
1	0	2	1901	46.1	43.1	53	57.4	70	78.4	82.1
1	0	2	1902	43.2	40.6	54.8	61.6	75.3	80.8	82.6
1	0	2	1903	43.6	48.2	59.2	60.7	69.4	73.2	79.9
1	0	2	1904	41.7	49.2	58.2	60	69.6	77.9	78.3
1	0	2	1905	39.1	39.5	59.5	63.5	74.4	79.1	79.4
1	0	2	1906	47.2	45.6	51.5	64.8	69.7	79	78.8
1	0	2	1907	54.3	49.1	64.5	58.3	68.1	75.6	80.8
1	0	2	1908	44.3	44.3	62.2	66.9	71.4	77.5	79.8
1	0	2	1909	50	50.7	56.1	63	68.6	77.9	79.8
1	0	2	1910	45.5	45.3	61	61.5	68.6	75.4	78.4
1	0	2	1911	50.5	54.2	57.9	63.5	72.6	80.4	77.8
1	0	2	1912	40.4	42.8	52.7	64.2	71.7	74.8	79.6
1	0	2	1913	51.8	47.4	55.5	61.9	71.3	77.4	80.9

3. Click **Transform Data**.

4. In the Power Query Editor, double click the column heading to rename the column using the **dictionary.txt** file as a guide.



5. Click **Close & Load** to load the data into Excel.



The data is imported into Excel, as shown below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	State	Division	Element	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2	1	0	2	1895	43.7	37.6	54.5	63.1	69.8	78.1	79.9	79.9	77.9	60.6	53.4	45.6
3	1	0	2	1896	44.1	47.9	52.5	67.9	75.7	77	80.8	82	75.5	63.5	57.5	46.2
4	1	0	2	1897	42.6	51.2	60.2	62	68.6	80.9	81	78.7	75.3	66.9	54.3	47.8
5	1	0	2	1898	49.4	45.9	59	58.1	73.4	80.3	79.8	78.6	75.1	61.1	49.9	44.1
6	1	0	2	1899	44.4	39.9	55.2	61.6	75.8	79.7	80.2	81.1	72.5	66	55.5	44.9
7	1	0	2	1900	44	44.1	52.6	63.5	71.2	76.3	79.7	81.5	77.6	68.7	54.9	46.8
8	1	0	2	1901	46.1	43.1	53	57.4	70	78.4	82.1	78.5	72	62.5	48.7	41.7
9	1	0	2	1902	43.2	40.6	54.8	61.6	75.3	80.8	82.6	82	73	63.3	57.6	45.3
10	1	0	2	1903	43.6	48.2	59.2	60.7	69.4	73.2	79.9	80.4	73	63.2	50.9	40.8
11	1	0	2	1904	41.7	49.2	58.2	60	69.6	77.9	78.3	78.2	76.6	64.5	52.2	46.7
12	1	0	2	1905	39.1	39.5	59.5	63.5	74.4	79.1	79.4	79.1	75.9	64.2	55.8	43.7
13	1	0	2	1906	47.2	45.6	51.5	64.8	69.7	79	78.8	80.4	78.1	60.9	55.8	50.3
14	1	0	2	1907	54.3	49.1	64.5	58.3	68.1	75.6	80.8	80.4	74.6	63.2	51.4	46.2
15	1	0	2	1908	44.3	44.3	62.2	66.9	71.4	77.5	79.8	79.3	74.1	60.4	56.9	50.6
16	1	0	2	1909	50	50.7	56.1	63	68.6	77.9	79.8	80.8	74.7	63.6	59.4	40.8
17	1	0	2	1910	45.5	45.3	61	61.5	68.6	75.4	78.4	79.4	77.1	66.3	51.2	42.5
18	1	0	2	1911	50.5	54.2	57.9	63.5	72.6	80.4	77.8	78.9	80	68.6	50.1	50.1
19	1	0	2	1912	40.4	42.8	52.7	64.2	71.7	74.8	79.6	79	76.7	65.9	50.1	47.3
20	1	0	2	1913	51.8	47.4	55.5	61.9	71.3	77.4	80.9	80.3	72.8	60.9	56.1	47.5
21	1	0	2	1914	47.6	45.7	50.5	63.3	71.5	82.9	81.4	78.7	72.8	63.7	52.6	43
22	1	0	2	1915	43.3	48.2	46.8	64.5	74.2	78.6	80.2	78.7	76.2	66.7	56.1	46.4
23	1	0	2	1916	52.5	46.8	54.3	61.7	73.2	77.3	78.3	79.7	73.3	64.4	54.3	47.3
24	1	0	2	1917	50.2	48.5	56.9	63.6	65.9	77.5	79.4	77.9	72.7	58	51	40
25	1	0	2	1918	37.8	53.2	61.9	60.7	72.6	79.5	77.9	80.4	69.7	68.6	53	50.2
26	1	0	2	1919	44.1	47.3	56.2	62.2	68.9	78.3	79.4	79	74.9	74	56.8	47.2

6. Rename the worksheet as **Temperature**. Save your work.

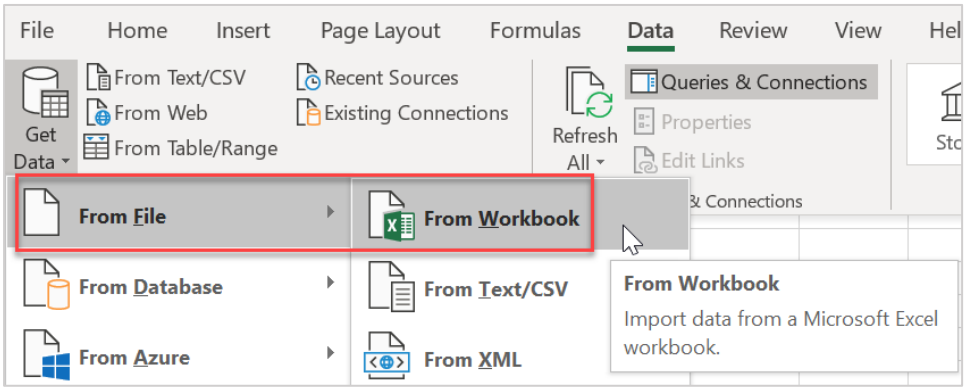
Exercise 2: Combing Data from multiple data source

Objective: To use Power Query's Query Editor to import data from a local Excel file that contains product information, and from an OData feed that contains product order information.

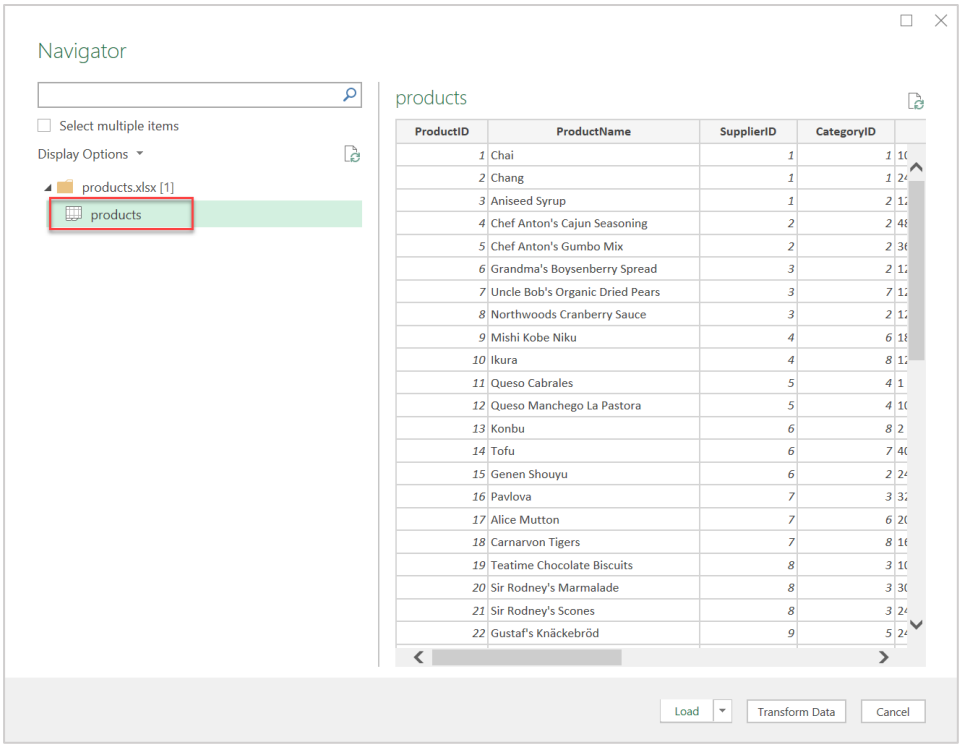
You will perform transformation and aggregation steps and combine data from both sources to produce a Total Sales per Product and Year report.

Importing Products Excel

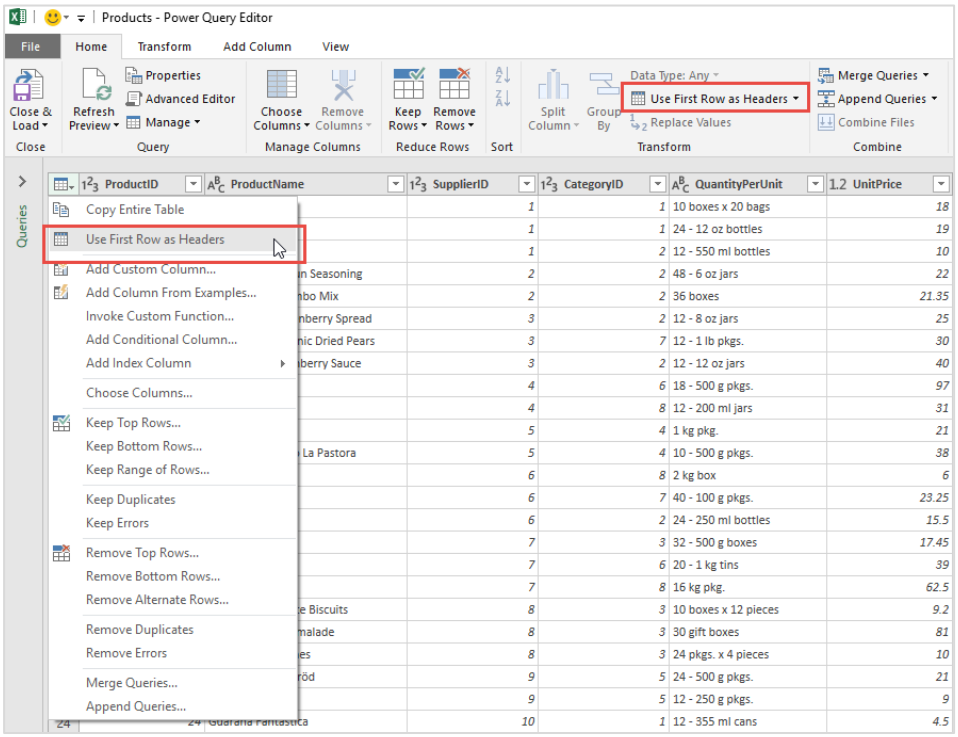
1. Under the **Data** tab, in the **Get & Transform** group, click **Get Data** → **From File** → **From Workbook** and select the **products.xlsx** Excel file.



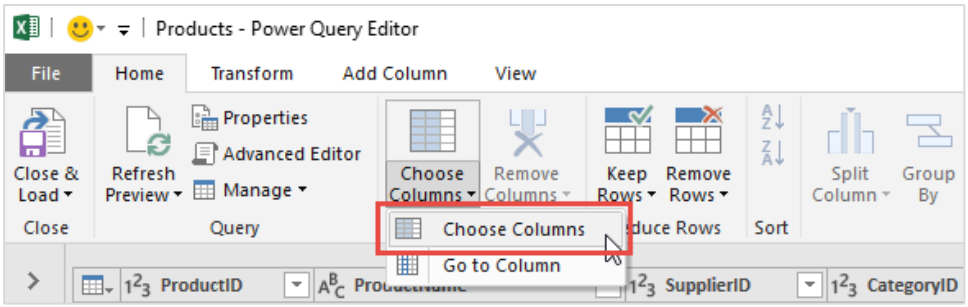
2. In the Query Navigator, select **products** and click on **Transform Data**.



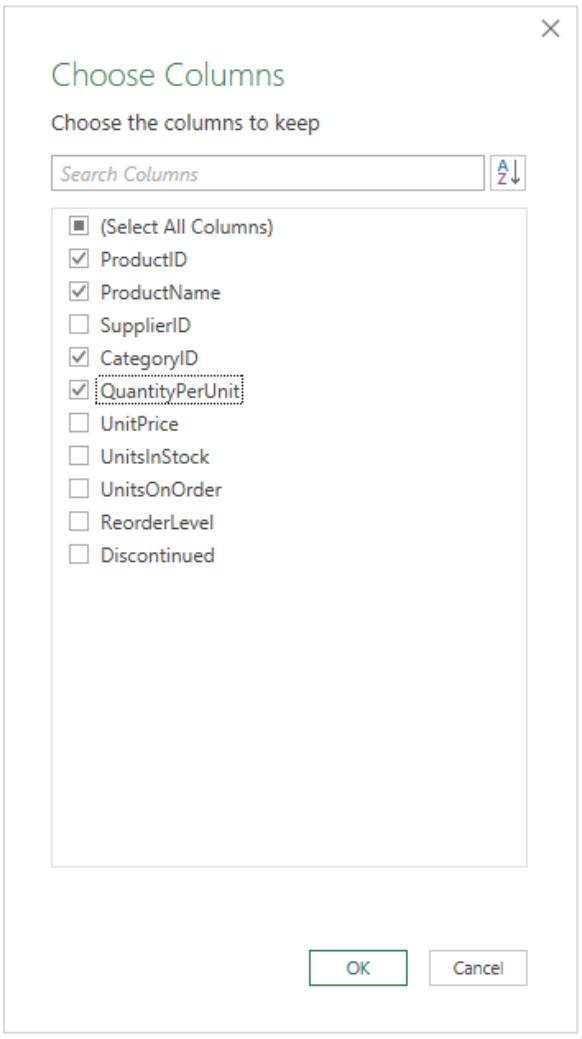
3. If the first row is not use as table column headers, click the table icon in the top-left corner of the data preview, select **Use First Row as Headers** or click **Use First Row as Headers** from the **Transform** group.



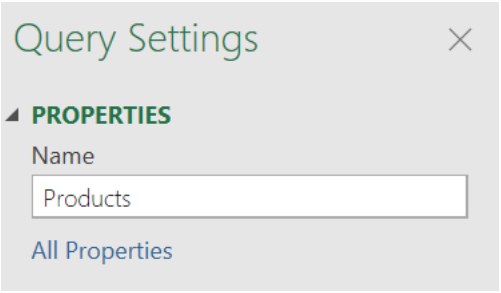
4. Remove all columns except **ProductID**, **ProductName**, **CategoryID**, and **QuantityPerUnit**. In the Power Query Editor, click **Choose Columns** and select **Choose Columns**.



5. In the Choose Columns dialog, select **ProductID**, **ProductName**, **CategoryID**, and **QuantityPerUnit** and click **OK**.



6. Before you import the products data into Excel, name the query **Products** in the **Query Settings** pane.

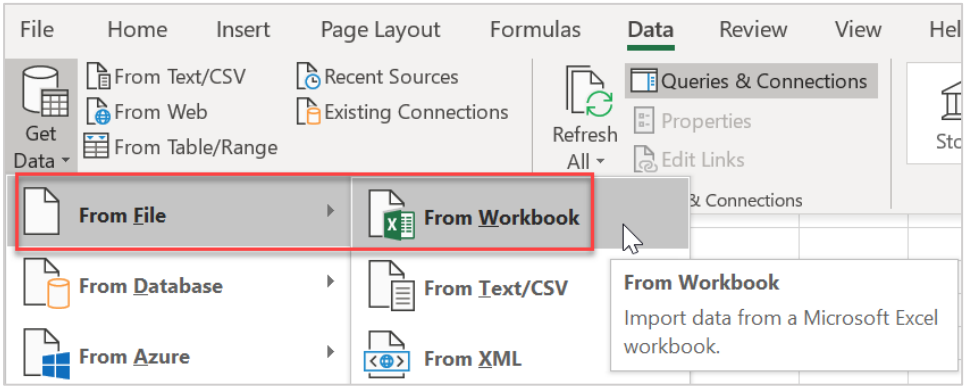


7. Click on **Close & Load** to load the data into Excel.

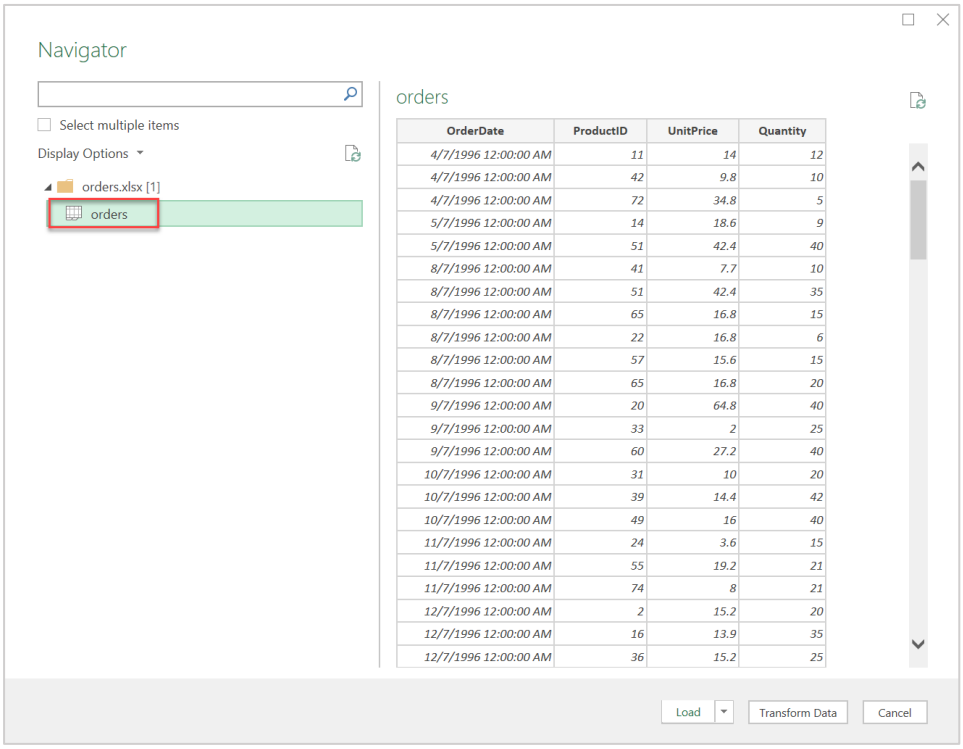
8. Rename the worksheet as **Products**. Save your work.

Importing Orders from another Excel

1. Under the **Data** tab, in the **Get & Transform** group, click **Get Data** → **From File** → **From Workbook** and select the **orders.xlsx** Excel file.

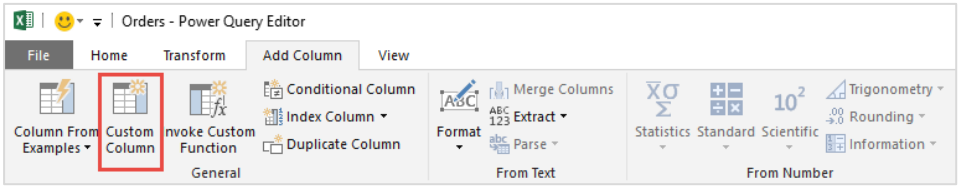


2. In the Query Navigator, select **orders** and click on **Transform Data**.



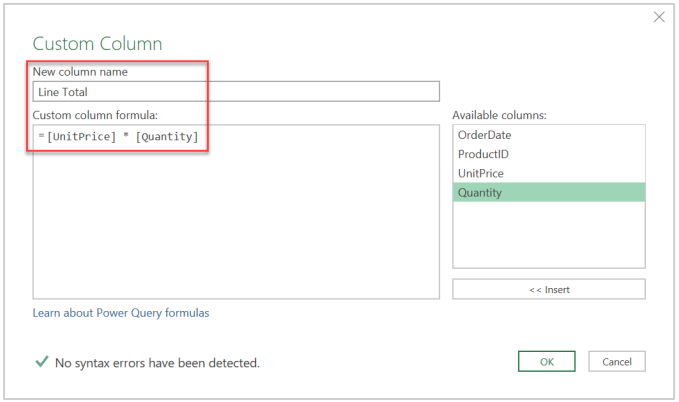
Calculate Line Total for each Order row

1. In the Query Editor ribbon, select **Add Column** and click **Custom column** or click the table icon at the top-left corner of the preview and select **Insert Column → Custom**.



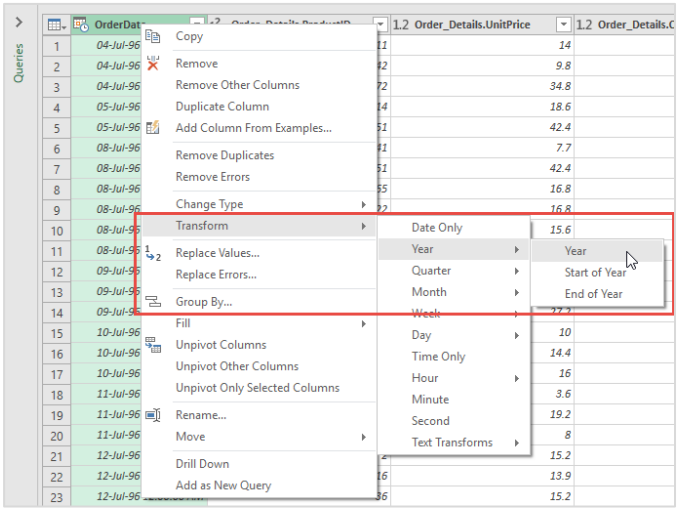
2. In the Insert Custom Column dialog, enter **[UnitPrice] * [Quantity]** in the **Custom Column Formula** textbox.

   In the **New column** name textbox, enter **Line Total**. Click **OK**.



Transform OrderDate Column

1. In the Query Preview grid, right-click the **OrderDate** column, select **Transform → Year → Year**.

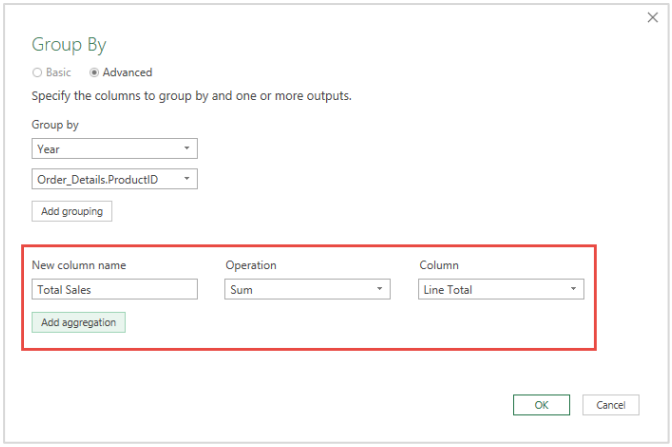


2. Rename the **OrderDate** column to **Year**. Double-click the **OrderDate** column and enter **Year** or right-click the **OrderDate** column, click **Rename** and enter **Year**.

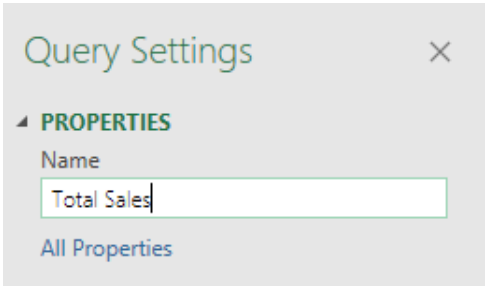
Group rows by ProductID and Year

1. In the Query Preview grid, select **Year** and **Order_Details.ProductID**. Right-click one of the headers and click **Group By**.

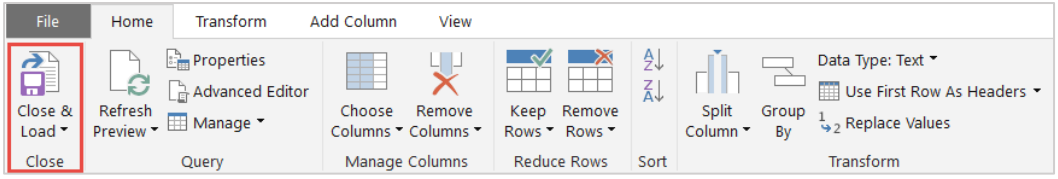
2. In the **Group By** dialog, enter **Total Sales** in the **New column name** textbox. In the **Operation** dropdown, select **Sum**. In the **Column** dropdown, select **Line Total**. Click **OK**.



3. Before you import the sales data into Excel, name the query **Total Sales** in the **Query Settings** pane.



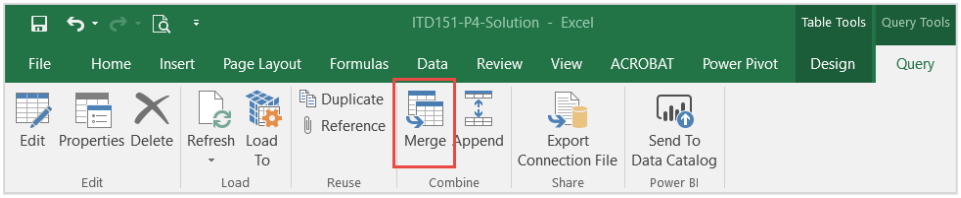
4. Click on **Close & Load** to load the data into Excel.



5. Rename the worksheet as **Total Sales**. Save your work.

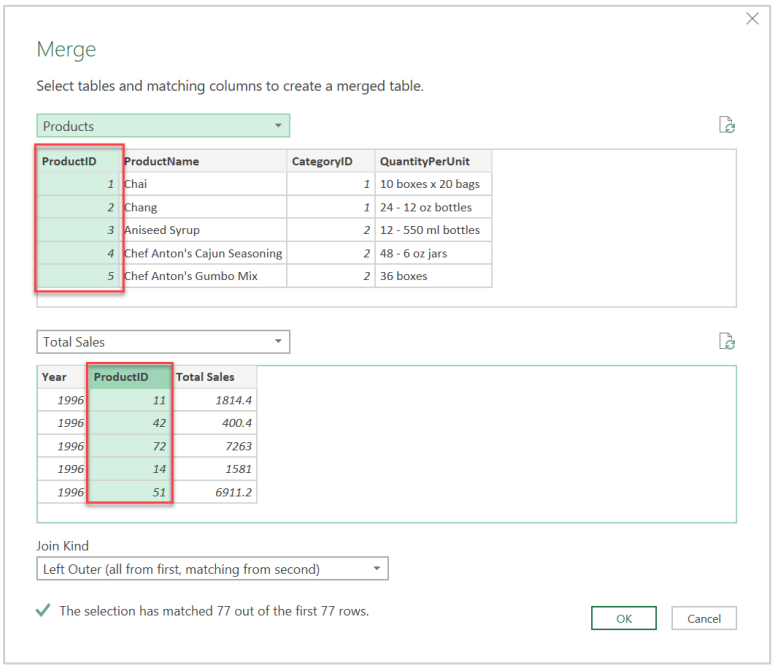
Combine Products and Total Sales

1. In the Excel workbook, select the **Products** worksheet. In the Query ribbon tab, click **Merge**.



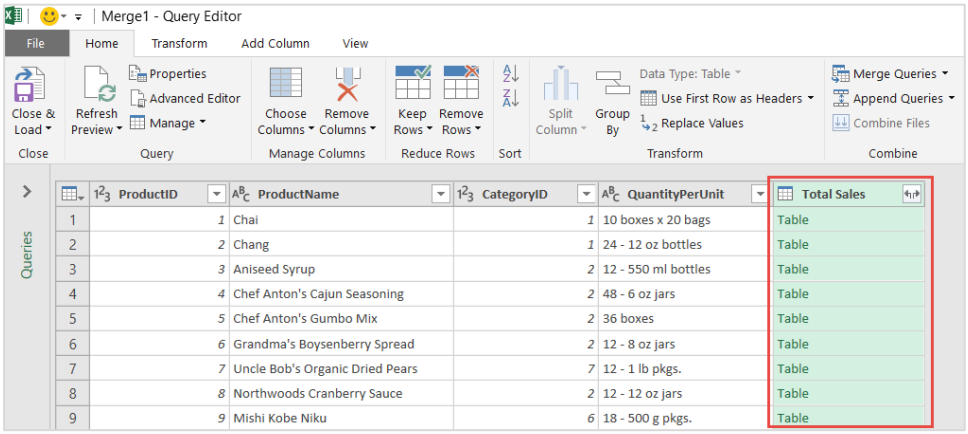
2. In the Merge dialog, select **Products** as the primary table, and select **Total Sales** as the second or related query to merge. **Total Sales** will become a new expandable column.

   To match **Total Sales** to **Products** by **ProductID**, select the **ProductID** column from the **Products** table, and the **ProductID** column from the **Total Sales** table.



   Click **OK**.

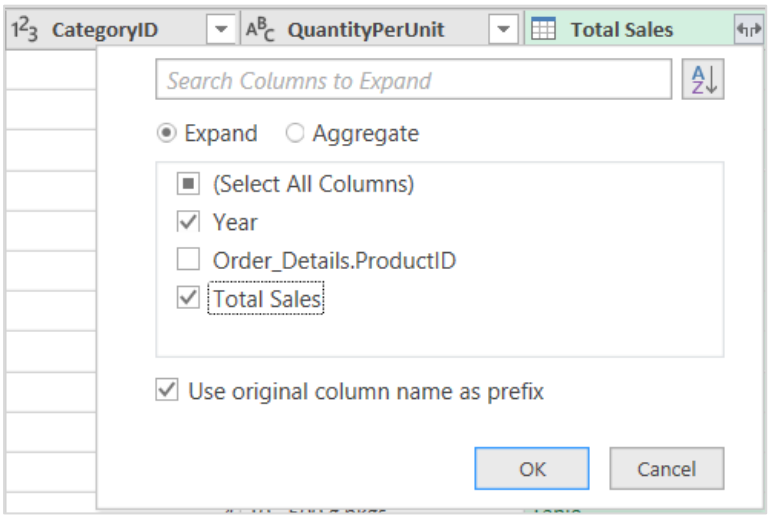
3. After you click **OK**, the **Merge** operation creates a query. The query result contains all columns from the primary table (**Products**), and a single column containing a navigation link to the related table (**Total Sales**). An **Expand** operation adds new columns into the primary or subject table from the related table.



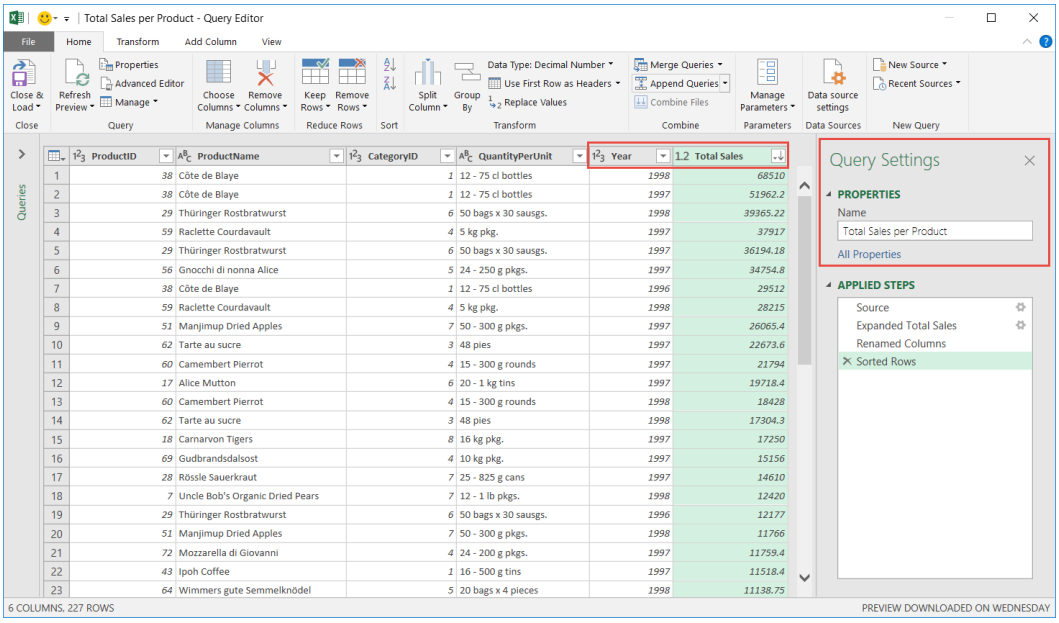
	ProductID	ProductName	CategoryID	QuantityPerUnit	Total Sales
1	1	Chai	1	10 boxes x 20 bags	Table
2	2	Chang	1	24 - 12 oz bottles	Table
3	3	Aniseed Syrup	2	12 - 550 ml bottles	Table
4	4	Chef Anton's Cajun Seasoning	2	48 - 6 oz jars	Table
5	5	Chef Anton's Gumbo Mix	2	36 boxes	Table
6	6	Grandma's Boysenberry Spread	2	12 - 8 oz jars	Table
7	7	Uncle Bob's Organic Dried Pears	7	12 - 1 lb pkgs.	Table
8	8	Northwoods Cranberry Sauce	2	12 - 12 oz jars	Table
9	9	Mishi Kobe Niku	6	18 - 500 g pkgs.	Table

Expand Merge Column

1. In the **Query Preview** grid, click the **Total Sales** expand icon . In the **Expand** dropdown, click **(Select All Columns)** to clear all columns, select **Year** and **Total Sales**. Click **OK**.



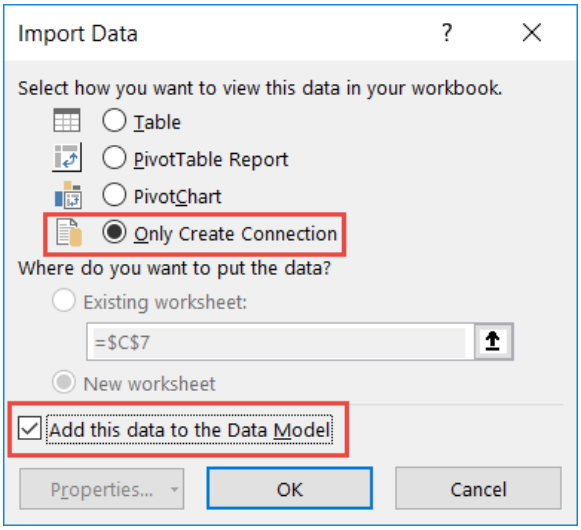
2. Rename these two columns to **Year** and **Total Sales**. **Sort Descending by Total Sales** to find out which products and in which years the products got the highest volume of sales. **Rename** the query to **Total Sales per Product**.



	ProductID	ProductName	CategoryID	QuantityPerUnit	Year	Total Sales
1	38	Côte de Blaye	1	12 - 75 cl bottles	1998	68510
2	38	Côte de Blaye	1	12 - 75 cl bottles	1997	51962.2
3	29	Thüringer Rostbratwurst	6	50 bags x 30 sausgs.	1998	39365.22
4	59	Raclette Courdavault	4	5 kg pkg.	1997	37917
5	29	Thüringer Rostbratwurst	6	50 bags x 30 sausgs.	1997	36194.18
6	56	Gnocchi di nonna Alice	5	24 - 250 g pkgs.	1997	34754.8
7	38	Côte de Blaye	1	12 - 75 cl bottles	1996	29512
8	59	Raclette Courdavault	4	5 kg pkg.	1998	28215
9	51	Manjimup Dried Apples	7	50 - 300 g pkgs.	1997	26065.4
10	62	Tarte au sucre	3	48 pies	1997	22673.6
11	60	Camembert Pierrot	4	15 - 300 g rounds	1997	21794
12	17	Alice Mutton	6	20 - 1 kg tins	1997	19718.4
13	60	Camembert Pierrot	4	15 - 300 g rounds	1998	18428
14	62	Tarte au sucre	3	48 pies	1998	17304.3
15	18	Carnarvon Tigers	8	16 kg pkg.	1997	17250
16	69	Gudbrandsdalsost	4	10 kg pkg.	1997	15156
17	28	Rössle Sauerkraut	7	25 - 825 g cans	1997	14610
18	7	Uncle Bob's Organic Dried Pears	7	12 - 1 lb pkgs.	1998	12420
19	29	Thüringer Rostbratwurst	6	50 bags x 30 sausgs.	1996	12177
20	51	Manjimup Dried Apples	7	50 - 300 g pkgs.	1998	11766
21	72	Mozzarella di Giovanni	4	24 - 200 g pkgs.	1997	11759.4
22	43	Ipoh Coffee	1	16 - 500 g tins	1997	11518.4
23	64	Wimmers gute Semmelknödel	5	20 bags x 4 pieces	1998	11138.75

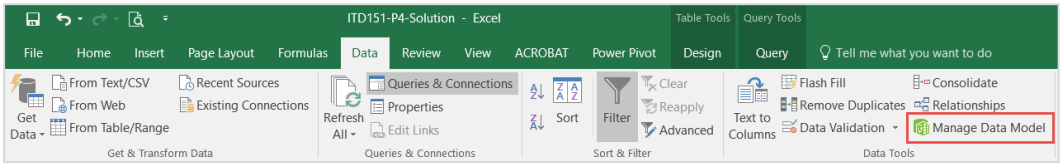
3. In addition to loading query results into an Excel worksheet, Power Query enables you to load a query result into an **Excel Data Model**. After you load data into the **Excel Data Model**, you can use **Power Pivot** and **Power View** to further data analysis.

   In the Query Editor ribbon, select **Close & Load → Close & Load To**. Select **Only Create Connection** and **Add this data to the Data Model**.

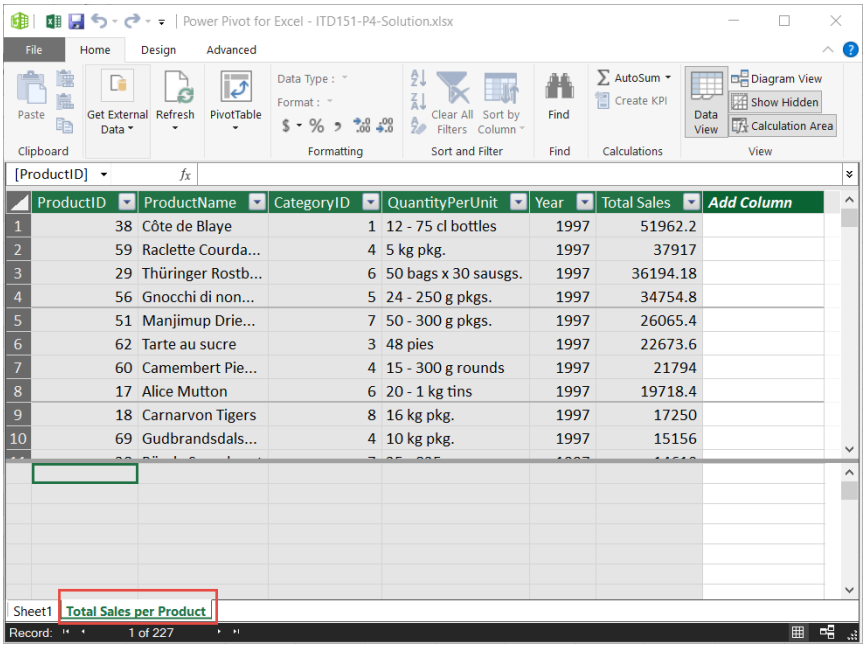


4. Save your work.

5. Under the **Data** tab, select **Manage Data Model** from the Data Tools group.



6. In the Power Pivot window, you will see the merged data loaded in the Data Model.



	ProductID	ProductName	CategoryID	QuantityPerUnit	Year	Total Sales	Add Column
1	38	Côte de Blaye	1	12 - 75 cl bottles	1997	51962.2	
2	59	Raclette Courda...	4	5 kg pkg.	1997	37917	
3	29	Thüringer Rostb...	6	50 bags x 30 sausgs.	1997	36194.18	
4	56	Gnocchi di non...	5	24 - 250 g pkgs.	1997	34754.8	
5	51	Manjimup Drie...	7	50 - 300 g pkgs.	1997	26065.4	
6	62	Tarte au sucre	3	48 pies	1997	22673.6	
7	60	Camembert Pie...	4	15 - 300 g rounds	1997	21794	
8	17	Alice Mutton	6	20 - 1 kg tins	1997	19718.4	
9	18	Carnarvon Tigers	8	16 kg pkg.	1997	17250	
10	69	Gudbrandsdals...	4	10 kg pkg.	1997	15156	

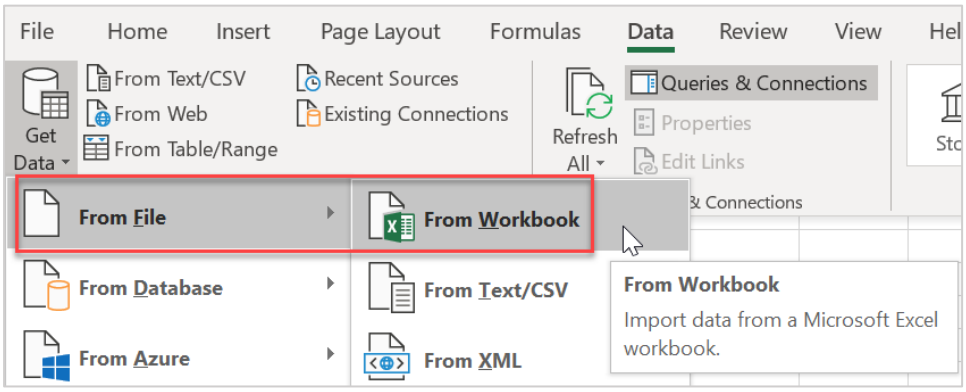
Exercise 3: Use Excel for Data Transformation and Cleaning

Objective: To use Excel Get & Transform to transform data (from Excel file biggest_hawker_centre.xlsx) - for suitable format for viewing.

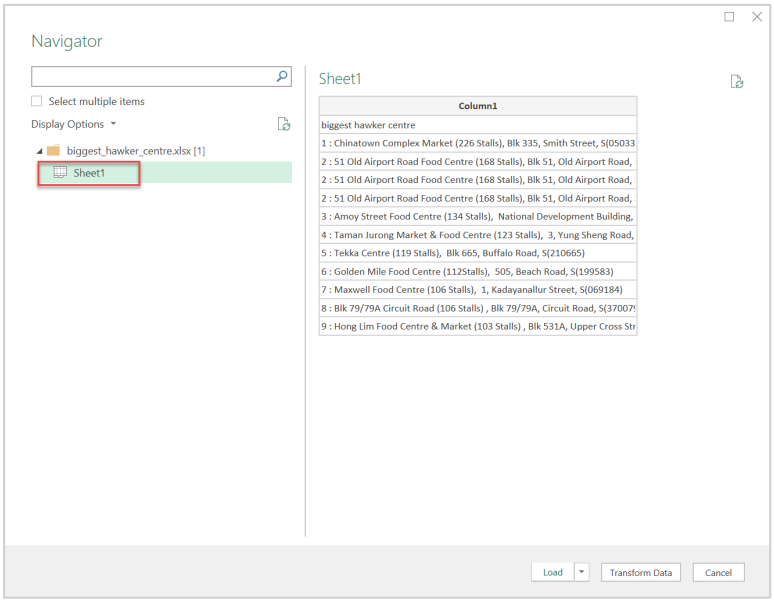
Reference: The dataset was extracted from http://woodlandstoyishun.sg/124-hawker-centres-and-5800-food-stalls-in-singapore/

Hawker centre is a heritage of Singapore. Do you know the distribution of the biggest hawker centres? Let's use Get & Transform for basic transformation tasks to format a list of information on hawker centre. The content in its original format cannot be analysed and hence there is a need to do further processing to extract the relevant fields (i.e., the location information and the number of stalls). The following is the instruction to visualise the distribution:

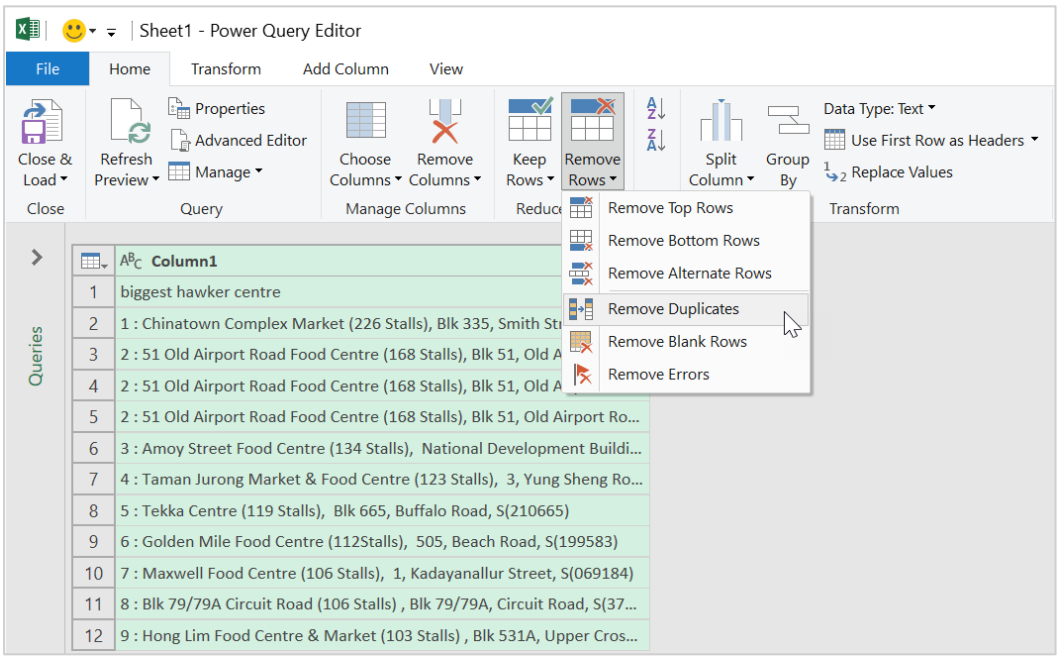
1. Under the **Data** tab, in the **Get & Transform** group, click **Get Data** → **From File** → **From Workbook** and select the **biggest_hawker_centre.xlsx** Excel file.



2. Query Navigator will be open (see screenshot below). Select **Sheet1** and click on **Transform Data**.

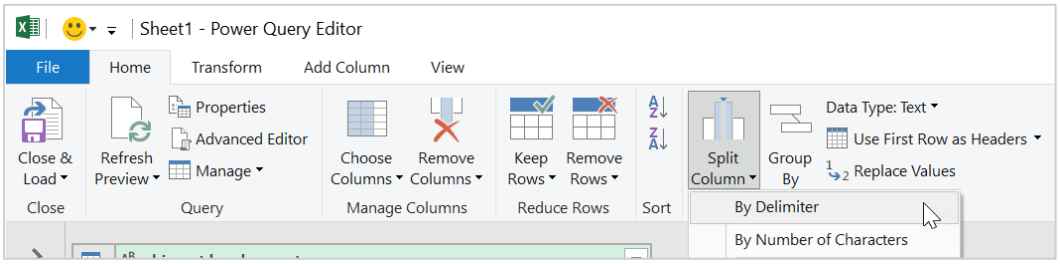


3. Check the data and it is obvious that there are duplicated data. Click on **Remove Rows** → **Remove Duplicates** and 2 rows of duplicated data will be removed.

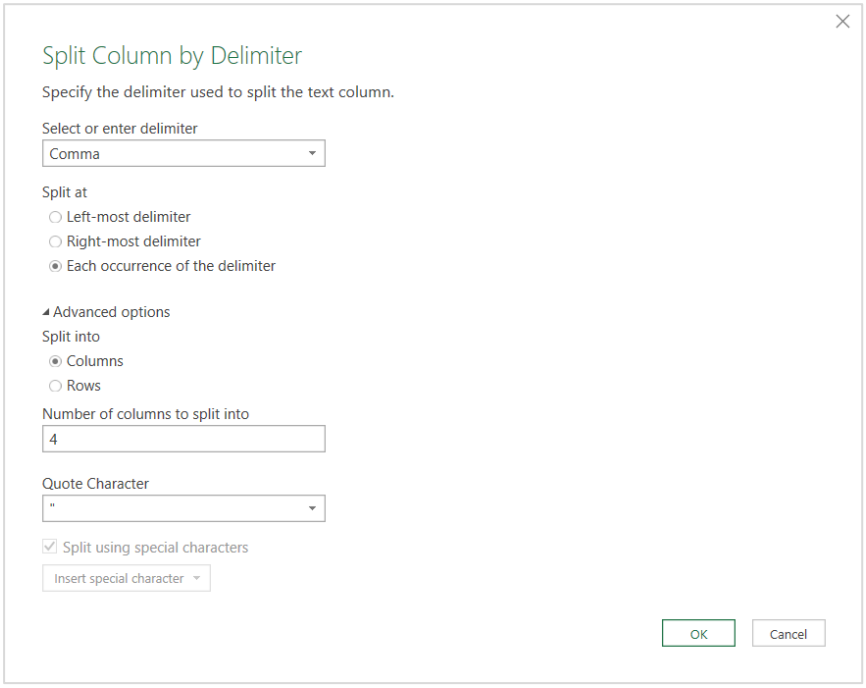


4. Click on **Use First Row as Headers** to make sure the first row is not being used in data transformation.

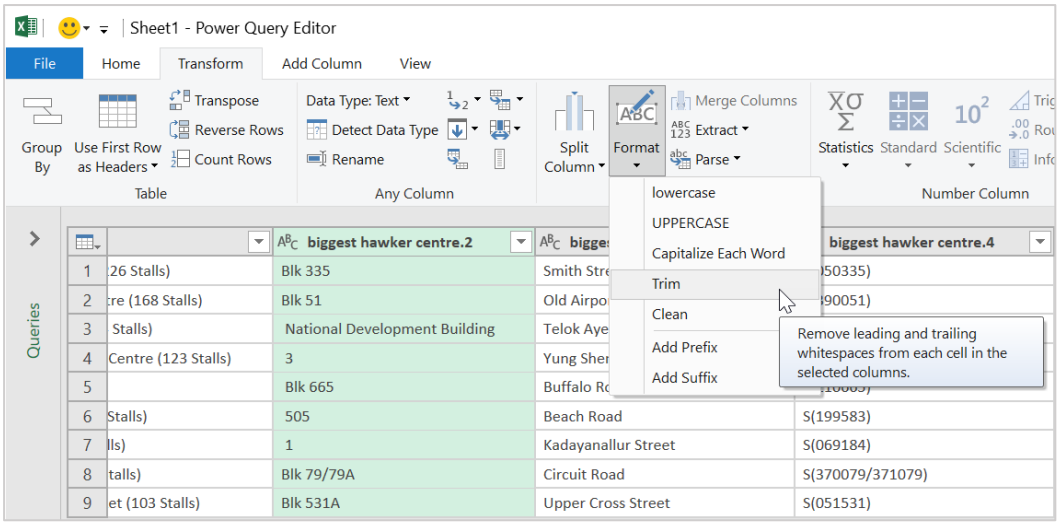
5. Click on **Split Column** and select **By Delimiter**.



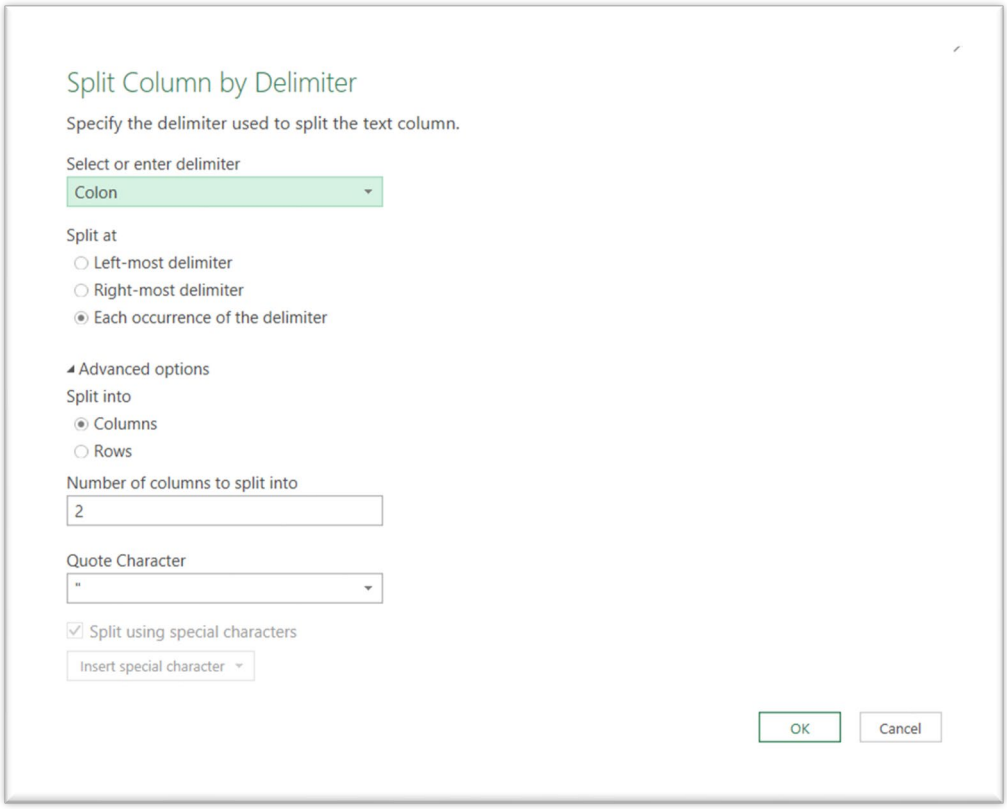
6. Use the default – Comma and the column will be split to 4 columns.

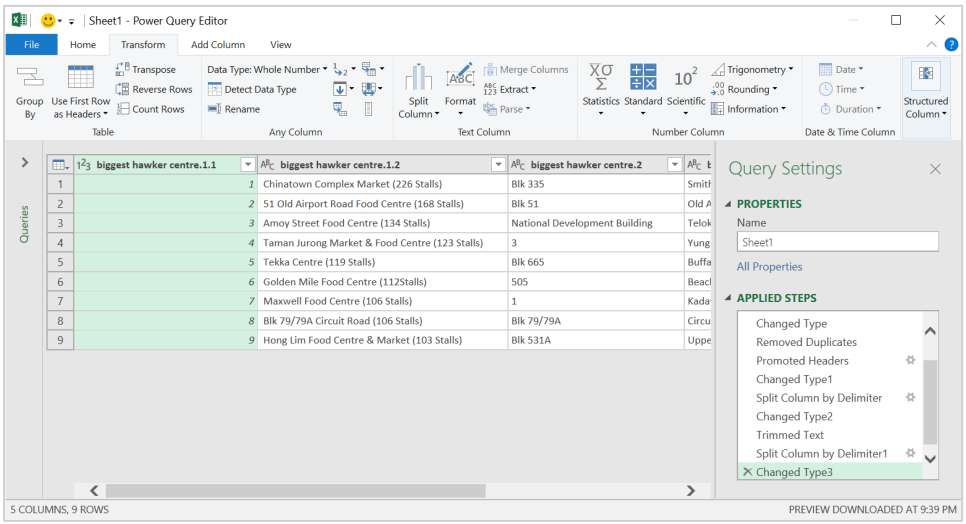


7. Take note that each column has a preceding space due to the original format. There is a need to remove these spaces to make sure that data in each column are in consistent format. In order to do that, click on **Transform** tab and select the column to be "cleaned". Click on **Format** and select **Trim** (as shown below). Do the same for the rest of columns.

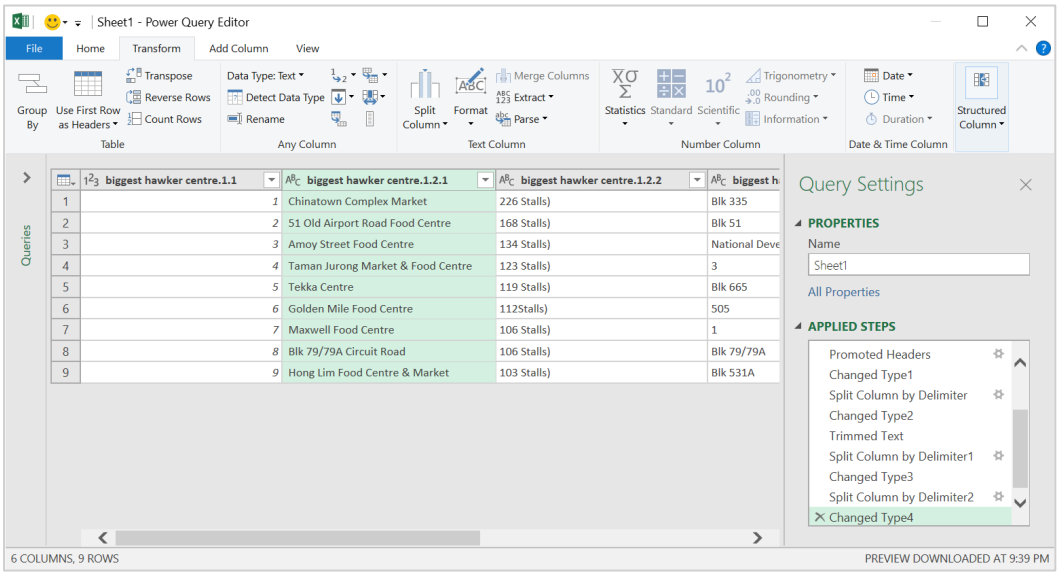


8. In the first column, there is a need to do a further extraction into index number, hawker center name and the number of stores. This can be done using the **Split Column** option in **Transform**. For the first split, choose delimiter as **colon**. This action will extract the index number.

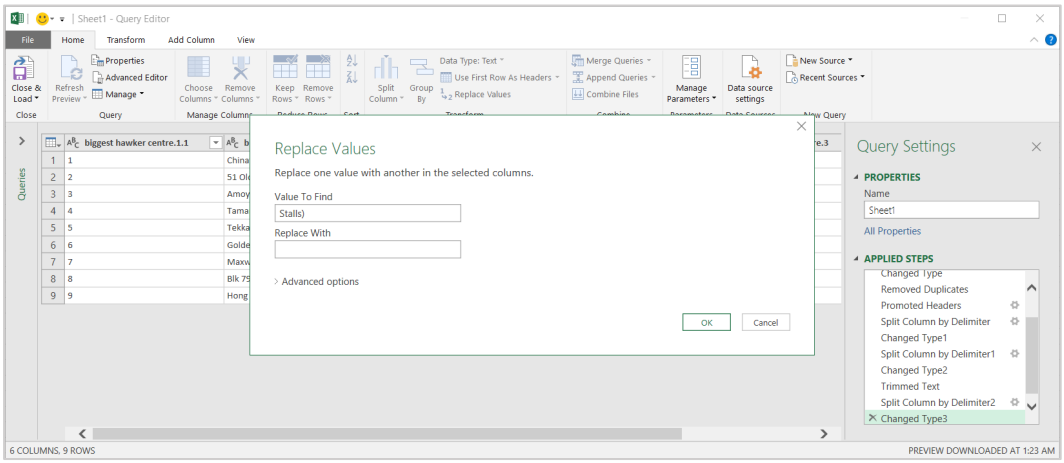


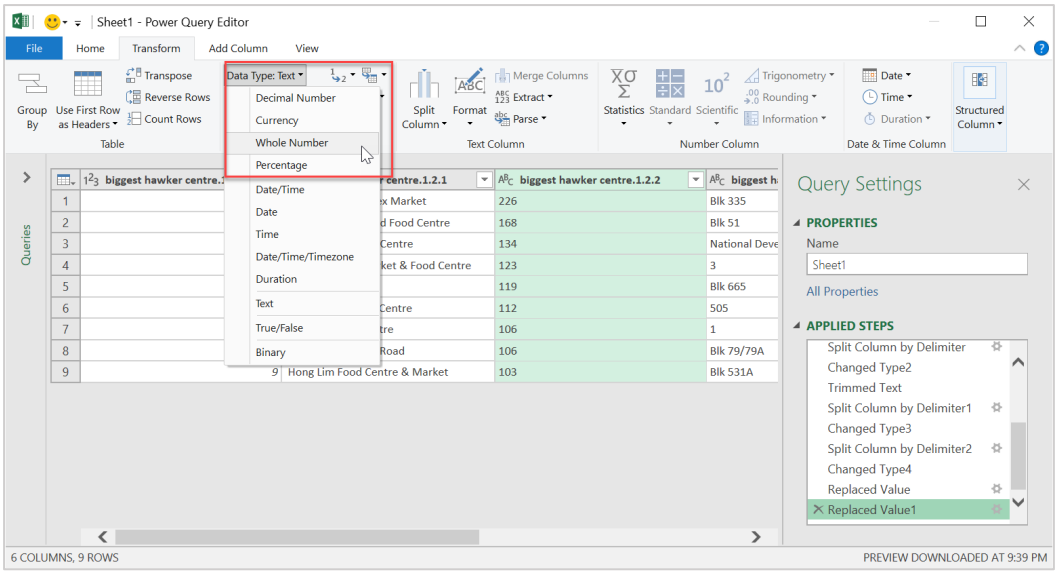


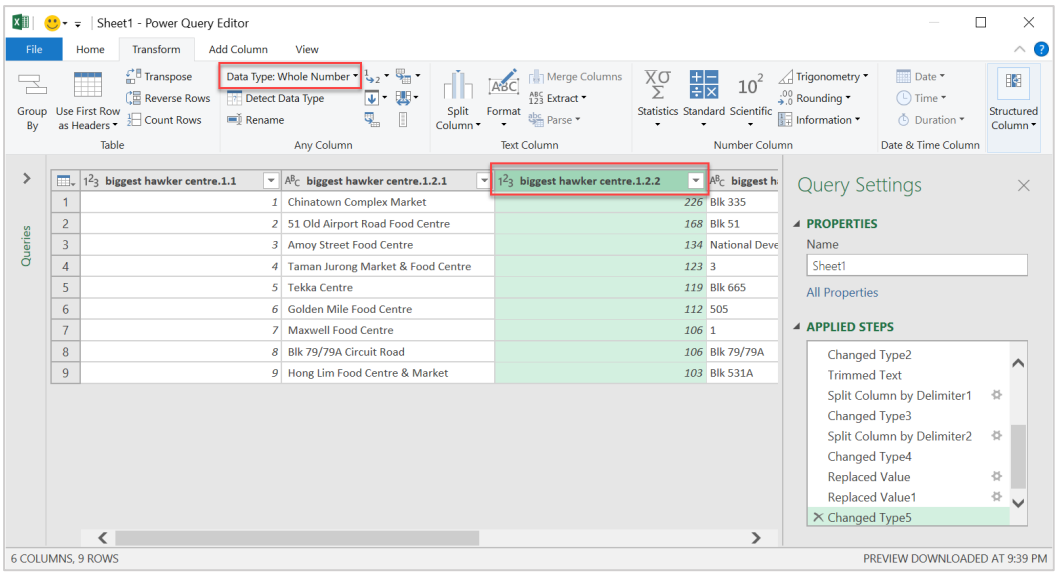
9. The second column now contains the hawker center name and the number of stores. Do another column split and use a custom delimiter – "**(**". The action will result in 2 new columns as shown below:



10. Even though most of the fields are now in its column but we are interested in analysing the number of stores and the field should be in number format and not a combination format (i.e., 106 Stalls) as shown above. There is a need to remove the text. This can be done by choosing **Replace Values**. Value To Find is "**Stalls)**" and it should be replace with "" [or nothing]. Make sure the field is converted to **Whole Number** by changing the Data Type.

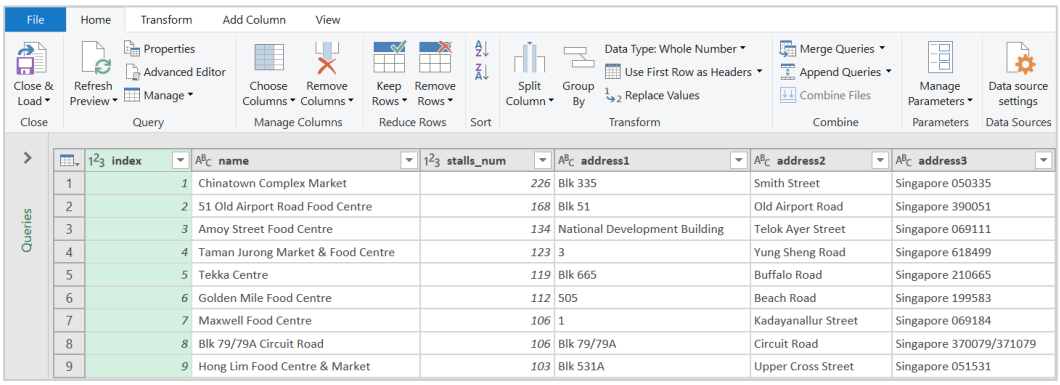






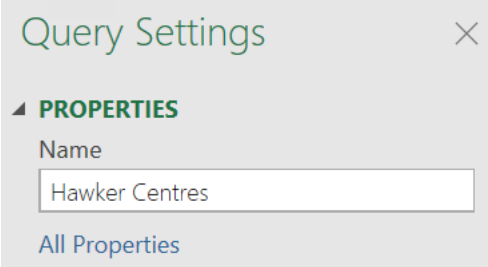
11. The last transformation that needs to be done is converting data inside the last column into Singapore and corresponding postcode. Similarly, this can be done using **Replace Values**. In other word, convert S(050335) into Singapore 050335. Try this on your own. Make sure your final dataset is the same as below.

12. Remember to change the column header into meaningful name. You can refer to the following for example:

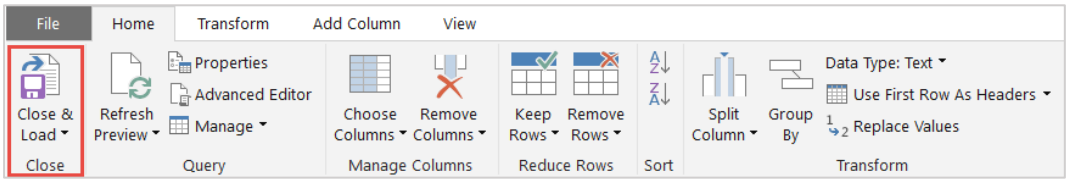


	index	name	stalls_num	address1	address2	address3
1	1	Chinatown Complex Market	226	Blk 335	Smith Street	Singapore 050335
2	2	51 Old Airport Road Food Centre	168	Blk 51	Old Airport Road	Singapore 390051
3	3	Amoy Street Food Centre	134	National Development Building	Telok Ayer Street	Singapore 069111
4	4	Taman Jurong Market & Food Centre	123	3	Yung Sheng Road	Singapore 618499
5	5	Tekka Centre	119	Blk 665	Buffalo Road	Singapore 210665
6	6	Golden Mile Food Centre	112	505	Beach Road	Singapore 199583
7	7	Maxwell Food Centre	106	1	Kadayanallur Street	Singapore 069184
8	8	Blk 79/79A Circuit Road	106	Blk 79/79A	Circuit Road	Singapore 370079/371079
9	9	Hong Lim Food Centre & Market	103	Blk 531A	Upper Cross Street	Singapore 051531

13. Before you import the hawker centres data into Excel, name the query **Hawker Centres** in the **Query Settings** pane.



14. Click on **Close & Load** to load the data into Excel.



15. Rename the worksheet as **Hawker Centres**. Save your work.

Applying Logic in Decision Making using Excel

WORKSHOP FOR NAVY

Learning Outcomes:

At the end of this session, you will be able to:

a) Build formulas with relational operators
b) Evaluate criteria using the Boolean logical functions AND, OR, and NOT
c) Apply conditional formatting to highlight key information in a worksheet
d) Apply IF functions to evaluate TRUE/FALSE values and perform calculations

Software(s): Microsoft Excel 2019

Instructions:

- Download the datasets for this hands-on: SuperStore.xlsx
- Launch Microsoft Office Excel.
- Select a blank workbook and complete the following exercises.

Using Relational Operators in Excel

In this chapter, we will learn operators and functions that allow us to decide whether a statement is **true** or **false**, we can then answer the previous question by deciding if the phrase "Last year's income is greater than this year's income" is true. ***TRUE*** and ***FALSE*** are referred to as ***Boolean values*** and can be calculated using Excel formulas and functions.

Often when analyzing data, the decision about what to do next depends on whether the data meets specific criteria. For example, if we are calculating employees' monthly paychecks, an employee may deserve a bonus *if their sales exceeded a certain amount*. This chapter expands the ability to decide whether data meets criteria by introducing Relational operators and Boolean functions.

There are six different relational operators that can be used. Examples of each are listed in the following table:

Relational Operators			
Description	*Operator*	Example	*Resulting Value*
Equal to	=	= 3+5=8	TRUE
Not equal to	<>	= Sum(3,7)<>10	FALSE
Greater than	>	=100>Max(5,10,20)	TRUE
Less than	<	= "B"<"A"	FALSE
Greater than or equal to	>=	=B2=2 where B2 contains the value 2	TRUE
Less than or equal to	<=	= 99<=8*3	FALSE

Be especially aware of the order of precedence in which these relational expressions are evaluated. As an example, the formula **=B3^2<>10*Sum(A1:A10)** would be evaluated as follows:

- First the computer would evaluate **SUM(A1:A10)**.
- Next the computer would evaluate B3^2 (^ is the exponent symbol so B2^2 is the mathematical expression $B3^2$)
- The result of **SUM(A1:A10)** would then be multiplied by 10.
- Finally, the two sides of the relational expression would be compared.

Exercise 1: Using relational operators to compare two values

Objective: To apply rules for orders to see which order is not profitable

1. Open SuperStore workbook. You will notice that the workbook contains 2 spreadsheets: **Orders** and **Returns.** Orders is as below:

Row ID	Order Priority	Discount	Unit Price	Shipping Cost	Customer ID	Customer Name	Ship Mode	Customer Segment	Product Category	Product Sub-Category
2	Not Specified	0.01	2.08	2.56	2867	Dana Teague	Regular Air	Corporate	Office Supplies	Scissors, Rulers and Trimmers
27	Critical	0.06	12.44	6.27	1821	Vanessa Boyer	Regular Air	Consumer	Office Supplies	Storage & Organization
52	Critical	0.08	155.99	8.08	1402	Wesley Tate	Regular Air	Corporate	Technology	Telephones and Communication
53	Critical	0.1	6.48	10.05	1402	Wesley Tate	Regular Air	Corporate	Office Supplies	Paper
62	High	0.02	48.58	54.11	2747	Brian Grady	Delivery Truck	Corporate	Furniture	Bookcases
63	High	0.07	39.48	1.99	2747	Brian Grady	Regular Air	Corporate	Technology	Computer Peripherals
64	Medium	0.08	124.49	51.94	553	Kristine Connolly	Delivery Truck	Corporate	Furniture	Tables
66	High	0.02	3.69	0.5	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Labels
67	High	0.09	3.85	0.7	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Pens & Art Supplies
68	Not Specified	0.06	11.7	6.96	1630	Jimmy Han	Regular Air	Home Office	Office Supplies	Appliances
78	Low	0.09	6.08	1.82	898	Harriet Hodges	Regular Air	Small Business	Office Supplies	Rubber Bands
87	Critical	0.04	3.08	0.99	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Labels
88	Critical	0.02	6.48	5.9	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Paper
89	Critical	0.04	125.99	4.2	3106	Alexander O'Brien	Regular Air	Home Office	Technology	Telephones and Communication

2. Our goal is to check which orders are profitable and which are not. In order to do so, we will create a new column at the end of the records with a header named "**Profitable**".

A	N	O	P	Q	R	S	T	U	V	W
Row ID	Product Base Margin	Region	Town	Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable
2	0.55	Central	Newton	20/2/2018	21/2/2018	-4.64	2	6.93	6	
27	0.57	Central	Novena	7/8/2016	9/8/2016	-37.04	25	312.3	193	
52	0.6	Central	Farrer Park	18/3/2018	20/3/2018	257.76	20	2634.86	322	
53	0.37	Central	Farrer Park	18/3/2018	20/3/2018	-291.59	46	281	322	
62	0.69	Central	Novena	19/9/2016	21/9/2016	-1348.06	60	2983.45	358	
63	0.54	Central	Novena	19/9/2016	19/9/2016	269.27	60	2247.04	358	
64	0.63	East	Tampines	18/12/2015	19/12/2015	-500.38	56	6831.37	359	

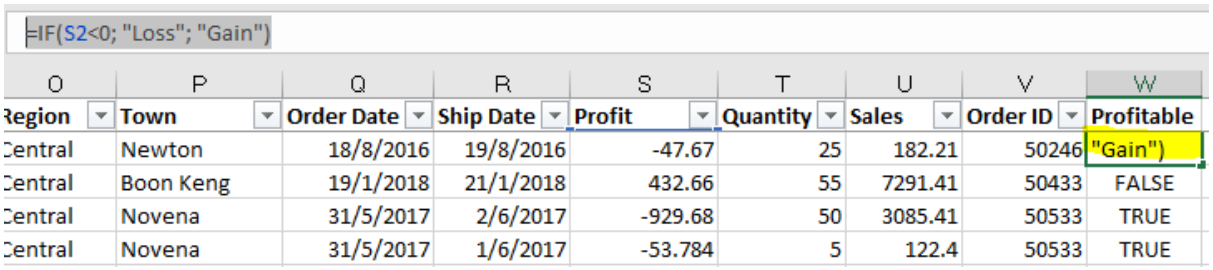
3. Select **the first cell** under "**Profitable**" and type the equal key. This allows you to select cells. Proceed to select the "**Profit**" cell on the same row. You will notice the formula stating the cell position (i.e. **S2**).

R	S	T	U	V	W
Ship Date	Profit	Quantity	Sales	Order ID	Profitable
21/2/2018	-4.64	2	6.93	6	=S2
9/8/2016	-37.04	25	312.3	193	
20/3/2018	257.76	20	2634.86	322	

4. Proceed to check if profit is less than zero, by typing the formula: **=S2 < 0.** You will see that the formula displays **FALSE.** If you **double click** on the right bottom corner of the cell, the formula will auto repeat for all rows in the column, thus displaying **TRUE** or **FALSE,** depending whether profit is less than zero.

Q	R	S	T	U	V	W
Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable
18/8/2016	19/8/2016	-47.67	25	182.21	50246	TRUE
19/1/2018	21/1/2018	432.66	55	7291.41	50433	FALSE
31/5/2017	2/6/2017	-929.68	50	3085.41	50533	TRUE
31/5/2017	1/6/2017	-53.784	5	122.4	50533	TRUE
25/10/2018	27/10/2018	-139.13	88	406.9	50656	TRUE
25/10/2015	26/10/2015	4260.112	56	29718.53	50656	FALSE
25/10/2018	25/10/2018	48.6	82	1021.78	50656	FALSE
25/10/2015	26/10/2015	-25.14	27	197.48	50656	TRUE
27/2/2018	1/3/2018	3793.703	33	9758.7	50721	FALSE

5. Instead of displaying **TRUE** or **FALSE**, we will want to have more meaningful labels such as **Gain** and **Loss**. To do so, we will use the **IF** condition to display meaningful labels.
6. In the formula bar of the cell the first row of **Profitable**, type the following:
   =IF(S2<0, "Loss", "Gain")



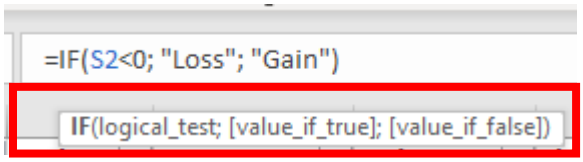
O	P	Q	R	S	T	U	V	W
Region	Town	Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable
Central	Newton	18/8/2016	19/8/2016	-47.67	25	182.21	50246	"Gain")
Central	Boon Keng	19/1/2018	21/1/2018	432.66	55	7291.41	50433	FALSE
Central	Novena	31/5/2017	2/6/2017	-929.68	50	3085.41	50533	TRUE
Central	Novena	31/5/2017	1/6/2017	-53.784	5	122.4	50533	TRUE

The **IF** condition is to evaluate and allow outcomes based on the condition give. The example above means:

IF (Value of cell **S2 less than zero**), display **Loss**.

ELSE display **Gain**

If you mouse over the IF operator in the formula bar, you see an important formula breakdown:

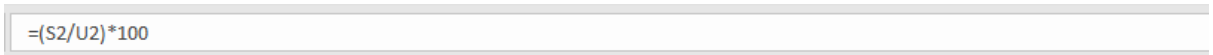


→ This means that the **1st value** will display when logical test **(S2 < 0) is true**, while **2nd value** will display when logical test is **false (S2 > 0)**.

Exercise 2: Using Nested IF conditions for more complex comparison

1. We can further create a condition that checks if **gains or losses** are more than 30%. To do that, we will need add a new column called **Margin**. In the first cell under **Margin,** add the following formula:
   =(S2/U2) * 100

This means: profit divided by sales, multiplied by 100 – **Percentage of profit over sales.**



O	P	Q	R	S	T	U	V	W	X
Region	Town	Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable	Margin
Central	Newton	20/2/2018	21/2/2018	-4.64	2	6.93	6	Loss	-66.955267
Central	Novena	7/8/2016	9/8/2016	-37.04	25	312.3	193	Loss	
Central	Farrer Park	18/3/2018	20/3/2018	257.76	20	2634.86	322	Gain	
Central	Farrer Park	18/3/2018	20/3/2018	-291.59	46	281	322	Loss	

2. You will see the first **margin** being calculated and displayed. We can now put a condition to check if this **margin** is more than 30%.
   Business Rule:
   IF margin is negative and more than 30%, display: “More than 30% Loss”
   IF margin is negative and less than 30%, display: “Less than 30% Loss”
   IF margin is positive and more than 30%, display: “More than 30% gain”
   IF margin is positive and less than 30%, display: “Less than 30% gain”

3. In the formula bar, enter:

```
=IF((S2/U2)*100<0,IF((S2/U2)*100<-30,"More than 30% Loss","Less than 30% Loss"),IF((S2/U2)*100>30,"More than 30% gain","Less than 30% gain"))
```

   The above should correctly classify the data based on the business rule, like this:

	S	T	U	V	W	X
	Profit	Quantity	Sales	Order ID	Profitable	Margin
	-4.64	2	6.93	6	Loss	More than 30% Loss
	-37.04	25	312.3	193	Loss	Less than 30% Loss
	257.76	20	2634.86	322	Gain	Less than 30% gain
	-291.59	46	281	322	Loss	More than 30% Loss
	-1348.06	60	2983.45	358	Loss	More than 30% Loss
	269.27	60	2247.04	358	Gain	Less than 30% gain
	-500.38	56	6831.37	359	Loss	Less than 30% Loss
	-0.048	4	14.96	386	Loss	Less than 30% Loss
	-2.544	4	15.69	386	Loss	Less than 30% Loss

We have used a few **Nested IF** conditions to achieve the above result. To explain, here is a step by step on how the formula works:

No	*Formula Statement*	*Meaning*
1	=IF((S2/U2)*100<0,	**If** margin is negative, go to step 2. **Else**, go to step 3
2	IF((S2/U2)*100<-30, "More than 30% Loss", "Less than 30% Loss"),	**If** margin is less than negative 30, display "More than 30% Loss" **Else** display "Less than 30% Loss"
3	IF((S2/U2)*100>30, "More than 30% gain", "Less than 30% gain"))	**If** margin is more than positive 30, display "More than 30% Loss" **Else** display "Less than 30% Loss"

Exercise 3: Using logical functions to evaluate a list of values

Relational operators can be used to compare two different values to determine if a relational expression is true or false. But how can one determine if ***all*** items in a group meet a specified criterion or if ***at least one item*** in a group meets a specified criterion? A list of Boolean values (TRUE/FALSE) can be evaluated using **And, Or, & Not** operations.

For example, Is the value in cell B2 greater than 20 ***and*** the value in cell B3 greater than 50?

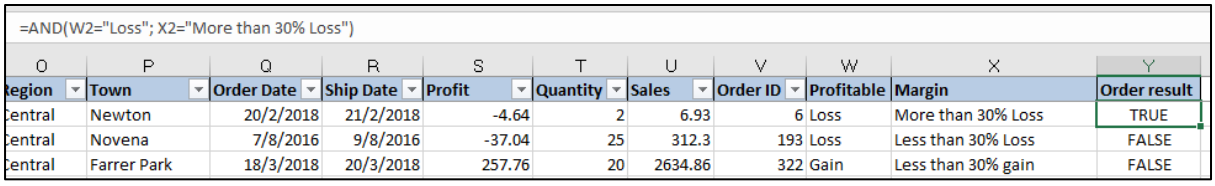
In Excel these operations are performed using the functions ***AND, OR & NOT***. These functions perform the following tasks:

Function Name	*Explanation*	*Examples*
AND	The ***AND*** function will evaluate a list of logical arguments and return **TRUE** if ***all*** of the arguments are TRUE. An **AND** function is **FALSE** if at least one of its arguments is FALSE.	=AND(60>50, 61>60) ➔ Returns **TRUE** =AND(60>50, 59>60) ➔ Return **FALSE**
OR	The ***OR*** function will evaluate a list of logical arguments and return TRUE if ***at least one*** of the arguments is TRUE. The **OR** function is only FALSE if ***all*** of the arguments in the function are FALSE.	= OR(60>50, 59>60) ➔ Returns **TRUE** =OR(49>50, 59>60) ➔ Return **FALSE**
NOT	The ***NOT*** function will evaluate only one logical argument (either a TRUE or FALSE) return TRUE if it is FALSE. The NOT function essentially changes the value TRUE to FALSE or the value FALSE to TRUE.	=NOT(60>50) ➔ Returns **FALSE** =NOT(49>50) ➔ Returns **TRUE**

To illustrate, let's go back to the SuperStore Excel. We want to first tag all the **Poor** margin orders by using a **AND** condition.

1. Let's add a third column, "**Order result**" and type in a formula that checks the **Profitable** and **Margin** columns:

=AND(W2="Loss", X2="More than 30% Loss")



=AND(W2="Loss"; X2="More than 30% Loss")

O	P	Q	R	S	T	U	V	W	X	Y
Region	Town	Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable	Margin	Order result
Central	Newton	20/2/2018	21/2/2018	-4.64	2	6.93	6	Loss	More than 30% Loss	TRUE
Central	Novena	7/8/2016	9/8/2016	-37.04	25	312.3	193	Loss	Less than 30% Loss	FALSE
Central	Farrer Park	18/3/2018	20/3/2018	257.76	20	2634.86	322	Gain	Less than 30% gain	FALSE

2. In the column, **TRUE** means that the order is a loss, and loss margin is more than 30%. This is possible from using the **AND** operator.
3. To change the label so that it displays different labels, use the **IF** condition together with the **AND** Operator. In the formula bar, key in this formula:

```
=IF(AND(W2="Loss", X2="More than 30% Loss"), "Poor", "OK")
```

This means that:
IF order has more than 30% margin loss, display **Poor**
ELSE display **OK**.

Order ID	Profitable	Margin	Order result
6	Loss	More than 30% Loss	Poor
193	Loss	Less than 30% Loss	OK
322	Gain	Less than 30% gain	OK
322	Loss	More than 30% Loss	Poor
358	Loss	More than 30% Loss	Poor
358	Gain	Less than 30% gain	OK
359	Loss	Less than 30% Loss	OK
386	Loss	Less than 30% Loss	OK

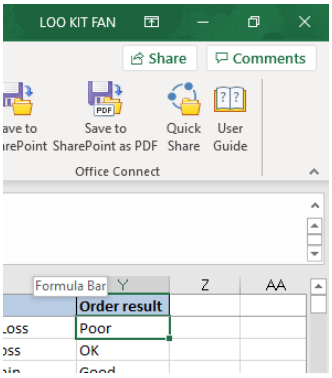
4. We can further make **Nested IF** to categorise the order results into 4 different results:

Business Rule	*Display in Order result*
IF order has more than 30% margin loss	Poor
IF order has less than 30% margin loss	OK
IF order has more than 30% margin gain	Excellent
IF order has less than 30% margin gain	Good

Are you able to figure out the formula? Take a few short minutes and try it out.

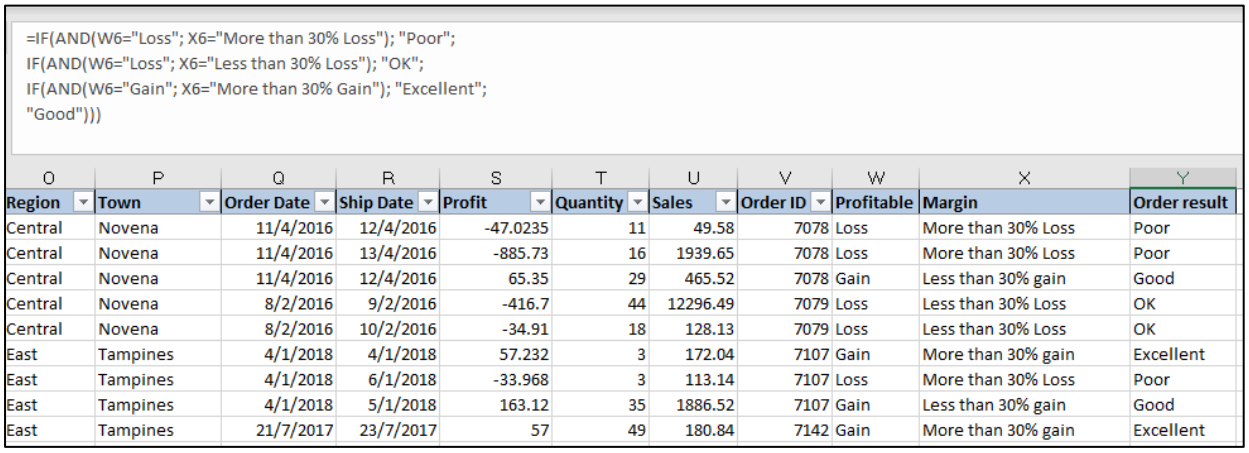
A few important things to take note when doing **Nested IF** and complicated formulas:

- You can drag the formula bar down so that it is wide enough for you to see the full formula in multiple lines.
- You can also hold the "**alt**" key and press "**enter**" to move to the next line when typing excel formulas.
- For each **Nested IF**, move to the next line so you can easily see the various conditions clearly.
- Take note of **closing round blankets** and put in the correct number of brackets to close off the formula.



5. In order to have the above categorisation based on the ***business rules***, key in the following formula, apply the formula to the whole column by double clicking the corner of the box:

```
=IF(AND(W2="Loss", X2="More than 30% loss"), "Poor",
IF(AND(W2="loss", X2="Less than 30% loss"), "OK",
IF(AND(W2="gain", X2="More than 30% gain"), "Excellent",
"Good")))
```



O	P	Q	R	S	T	U	V	W	X	Y
Region	Town	Order Date	Ship Date	Profit	Quantity	Sales	Order ID	Profitable	Margin	Order result
Central	Novena	11/4/2016	12/4/2016	-47.0235	11	49.58	7078	Loss	More than 30% Loss	Poor
Central	Novena	11/4/2016	13/4/2016	-885.73	16	1939.65	7078	Loss	More than 30% Loss	Poor
Central	Novena	11/4/2016	12/4/2016	65.35	29	465.52	7078	Gain	Less than 30% gain	Good
Central	Novena	8/2/2016	9/2/2016	-416.7	44	12296.49	7079	Loss	Less than 30% Loss	OK
Central	Novena	8/2/2016	10/2/2016	-34.91	18	128.13	7079	Loss	Less than 30% Loss	OK
East	Tampines	4/1/2018	4/1/2018	57.232	3	172.04	7107	Gain	More than 30% gain	Excellent
East	Tampines	4/1/2018	6/1/2018	-33.968	3	113.14	7107	Loss	More than 30% Loss	Poor
East	Tampines	4/1/2018	5/1/2018	163.12	35	1886.52	7107	Gain	Less than 30% gain	Good
East	Tampines	21/7/2017	23/7/2017	57	49	180.84	7142	Gain	More than 30% gain	Excellent

If you scroll down the list, you can see the **Order results** which are either Poor, OK, Good or Excellent. Let's go one step further and do a **look up** of another table in order to find if the orders are returned or not.

Exercise 4: Using the VLOOKUP function to compare values across different worksheets

VLOOKUP is an Excel function to lookup and retrieve data from a specific column in table. VLOOKUP supports approximate and exact matching, and wildcards (* ?) for partial matches. The "V" stands for "vertical". Lookup values must appear in the first column of the table, with lookup columns to the right.

The syntax for the formula is as follows:

=**VLOOKUP**(value, table, col_index, [range_lookup])

- **value** - The value to look for in the first column of a table.
- **table** - The table from which to retrieve a value.
- **col_index** - The column in the table from which to retrieve a value.
- **range_lookup** - [optional] TRUE = approximate match (default). FALSE = exact match.

VLOOKUP is designed to retrieve data in a table organized into vertical rows, where each row represents a new record. The "V" in VLOOKUP stands for vertical:

Order ID	Profitable	Margin	Order result
7078	Loss	More than 30% Loss	Poor
7078	Loss	More than 30% Loss	Poor
7078	Gain	Less than 30% gain	Good
7079	Loss	Less than 30% Loss	OK
7079	Loss	Less than 30% Loss	OK
7107	Gain	More than 30% gain	Excellent
7107	Loss	More than 30% Loss	Poor
7107	Gain	Less than 30% gain	Good
7142	Gain	More than 30% gain	Excellent
7203	Gain	Less than 30% gain	Good

Records are in Rows

If you have data organized horizontally, use the **HLOOKUP** function.
VLOOKUP requires a lookup table with lookup values in the left-most column. The data you want to retrieve (result values) can appear in any column to the **right**:

	A	B	C
1	Order ID	Status	Date of Return
2	65	Returned	21/2/2018
3	612	Returned	9/8/2016
4	614	Returned	20/3/2018
5	678	Returned	20/3/2018
6	710	Returned	21/9/2016
7	740	Returned	19/9/2016
8	775	Returned	19/12/2015

When you use VLOOKUP on Order ID, you can get Status and Date of Return values

VLOOKUP retrieves data based on column number. When you use VLOOKUP, imagine that every column in the table is numbered, starting from the left. To get a value from a column, provide the appropriate number as the "column index". For example, the column index to retrieve **Status** below is **2**:

	1	2	3			
	A	B	C	D	E	F
1	Order ID	Status	Date of Return			
2	65	Returned	21/2/2018		Order ID	775
3	612	Returned	9/8/2016		Status	Returned
4	614	Returned	20/3/2018		Date of Return	19/12/2015
5	678	Returned	20/3/2018			
6	710	Returned	21/9/2016			
7	740	Returned	19/9/2016			
8	775	Returned	19/12/2015			
9	833	Returned	26/1/2017			
10	902	Returned	26/1/2017			
11	3300	Returned	18/12/2018			

2
3

In order to retrieve **Status** value from the LOOKUP table, the column index is **2**

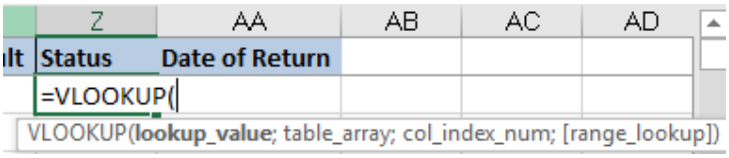
Let's start to apply **VLOOKUP** on the **Returns** worksheet. The goal is to search for the **Order ID** in the **Returns** worksheet and return the **Status and Date of Return**.

1. In the **Orders** table, add 2 more columns: **Status and Date of Return**

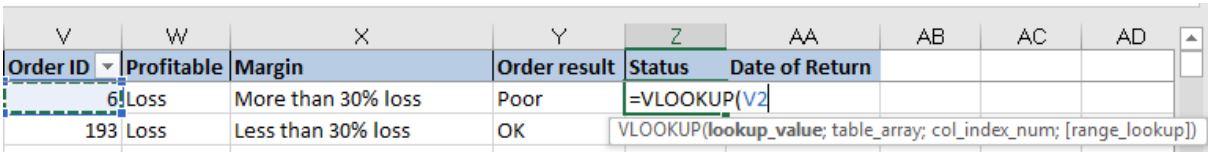
Order ID	Profitable	Margin	Order result	Status	Date of Return
6	Loss	More than 30% loss	Poor		
193	Loss	Less than 30% loss	OK		
322	Gain	Less than 30% gain	Good		
322	Loss	More than 30% loss	Poor		
358	Loss	More than 30% loss	Poor		
358	Gain	Less than 30% gain	Good		
359	Loss	Less than 30% loss	OK		
386	Loss	Less than 30% loss	OK		
386	Loss	Less than 30% loss	OK		

2. To perform a **VLOOKUP** to get the Status, we need to match the **Order ID** from **Orders** table to the **Returns** table. Under the **Status** column, type the following formula: **=VLOOKUP(**

   As you type, the formula breakdown will appear:



3. For **lookup_value**, click on the **Order ID** cell. The cell should appear inside the formula, and **lookup_value** is bold:



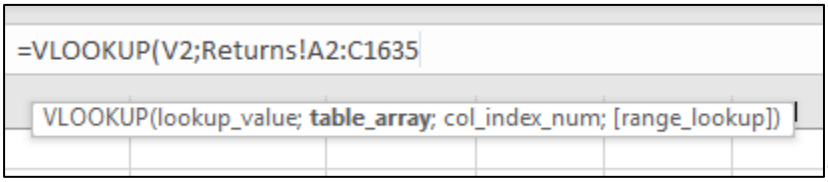
Type a comma so that the formula **bolds** the "**table_array**" parameter. Your formula should look like this now:

```
=VLOOKUP(V2,
```

4. Next, we want to select the entire **Returns** table. click on the "**Returns**" worksheet tab at the bottom of the workbook. Select the first row of the **Returns** table.

	A	B	C
1	Order ID	Status	Date of Return
2	65	Returned	21/2/2018
3	612	Returned	9/8/2016
4	614	Returned	20/3/2018
5	678	Returned	20/3/2018
6	710	Returned	21/9/2016
7	740	Returned	19/9/2016

5. Now, with the top row selected, hold on to **shift+ctrl** keys, then the down **arrow** key. This should select the entire **Returns** table and the following will be added to your formula:



→ This means that the **table_array** is from **Returns** tab, with cell values range from **A2 to C1635**.

6. Add the **$** sign to **lock the table reference** so it will not change when you apply to multiple rows: Add $ before the columns and rows for table_array like this: **$A$2:$C$1635**
   Your formula should look like this:

   =VLOOKUP(V2,Returns!$A$2:$C$1635

7. Add a comma to the end of the formula, and the next parameter, "**col_index_num**" will be **bolded.** Type **2** into the formula as status is the 2nd column in the **Returns** table.

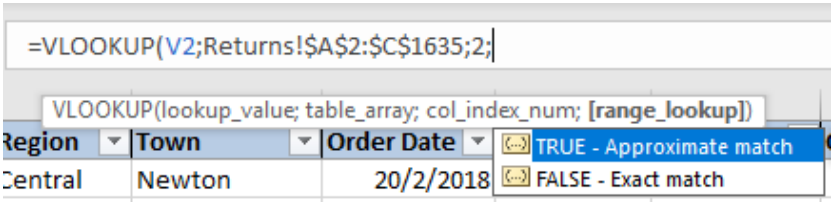
   Why do we type 2 as the **col_index_num**? This is because, based on the **Returns** table, **Status** is the 2nd column.

1 **2** **3**

	A	B	C
1	Order ID	Status	Date of Return
2	65	Returned	21/2/2018
3	612	Returned	9/8/2016
4	614	Returned	20/3/2018
5	678	Returned	20/3/2018
6	710	Returned	21/9/2016
7	740	Returned	19/9/2016
8	775	Returned	19/12/2015
9	833	Returned	26/1/2017
10	902	Returned	26/1/2017
11	3300	Returned	18/12/2018

In order to retrieve **Status** value from the LOOKUP table, the column index is **2.**

8. Add a comma to the end of the formula, and the next parameter, "**col_index_num**" will be **bolded.** Two options will be displayed: **TRUE** and **FALSE**.

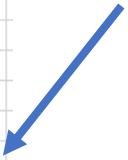


9. Select **FALSE** as **Order ID** should always be an exact match rather than an approximate match. Your formula is completed and it should look like this:

   =VLOOKUP(V2,Returns!$A$2:$C$1635,2,)

10. Your formula will now show **#N/A** if the **Order ID** is not found in the **Returns** table. If the **Order ID** is found, the Status **Returned** should be displayed.

V	W	X	Y	Z
Order ID	Profitable	Margin	Order result	Status
6	Loss	More than 30% loss	Poor	#N/A
193	Loss	Less than 30% loss	OK	#N/A
322	Gain	Less than 30% gain	Good	#N/A
322	Loss	More than 30% loss	Poor	#N/A
358	Loss	More than 30% loss	Poor	#N/A
358	Gain	Less than 30% gain	Good	#N/A
359	Loss	Less than 30% loss	OK	#N/A
386	Loss	Less than 30% loss	OK	#N/A
386	Loss	Less than 30% loss	OK	#N/A
388	Loss	Less than 30% loss	OK	#N/A
454	Gain	Less than 30% gain	Good	#N/A
548	Gain	Less than 30% gain	Good	#N/A
548	Loss	Less than 30% loss	OK	#N/A
548	Gain	Less than 30% gain	Good	#N/A
612	Loss	More than 30% loss	Poor	Returned
612	Gain	Less than 30% gain	Good	Returned



11. However, instead of **#N/A**, we want to display "**Completed**" if the order is not returned. Therefore, adding a new function **IFNA** to the formula will help to allow a different display. In the formula bar, add the following:

```
=IFNA(VLOOKUP(V2,Returns!$A$2:$C$1635,2,), "Completed")
```

12. Apply the formula again to all the rows by double clicking the right bottom corner of the cell. Now, all **#N/A** values will display "**Completed**" instead.

V	W	X	Y	Z
Order ID	Profitable	Margin	Order result	Status
6	Loss	More than 30% loss	Poor	Completed
193	Loss	Less than 30% loss	OK	Completed
322	Gain	Less than 30% gain	Good	Completed
322	Loss	More than 30% loss	Poor	Completed
358	Loss	More than 30% loss	Poor	Completed
358	Gain	Less than 30% gain	Good	Completed
359	Loss	Less than 30% loss	OK	Completed
386	Loss	Less than 30% loss	OK	Completed
386	Loss	Less than 30% loss	OK	Completed
388	Loss	Less than 30% loss	OK	Completed
454	Gain	Less than 30% gain	Good	Completed
548	Gain	Less than 30% gain	Good	Completed
548	Loss	Less than 30% loss	OK	Completed
548	Gain	Less than 30% gain	Good	Completed
612	Loss	More than 30% loss	Poor	Returned
612	Gain	Less than 30% gain	Good	Returned
613	Loss	More than 30% loss	Poor	Completed
613	Gain	Less than 30% gain	Good	Completed

13. Now, we will apply the same techniques on the **Date of Return** column.

Are you able to figure out the formula? Take a few short minutes and try it out.

14. If you repeated the steps correctly, the formula in the **Date of Return** should look like this:

```
=IFNA(VLOOKUP(V2,Returns!$A$2:$C$1635,3,), "NA")
```

Everything remains the same. However, the **col_index_num** is updated to 3 as **Date of Return** is the third column in the **Returns** table.

Order ID	Profitable	Margin	Order result	Status	Date of Return
6	Loss	More than 30% loss	Poor	Completed	NA
193	Loss	Less than 30% loss	OK	Completed	NA
322	Gain	Less than 30% gain	Good	Completed	NA
322	Loss	More than 30% loss	Poor	Completed	NA
358	Loss	More than 30% loss	Poor	Completed	NA
358	Gain	Less than 30% gain	Good	Completed	NA
359	Loss	Less than 30% loss	OK	Completed	NA
386	Loss	Less than 30% loss	OK	Completed	NA
386	Loss	Less than 30% loss	OK	Completed	NA
388	Loss	Less than 30% loss	OK	Completed	NA
454	Gain	Less than 30% gain	Good	Completed	NA
548	Gain	Less than 30% gain	Good	Completed	NA
548	Loss	Less than 30% loss	OK	Completed	NA
548	Gain	Less than 30% gain	Good	Completed	NA
612	Loss	More than 30% loss	Poor	Returned	42591
612	Gain	Less than 30% gain	Good	Returned	42591
613	Loss	More than 30% loss	Poor	Completed	NA
613	Gain	Less than 30% gain	Good	Completed	NA

Notice that instead of **Date format,** the value displayed is a number

15. We want to format the **VLOOKUP** value for **Date of Return** to normal calendar date i.e. DD/MM/YYYY. In order to do so, we will use a formula called **TEXT.** Add the function as shown below:

```
=TEXT(IFNA(VLOOKUP(V2,Returns!$A$2:$C$1635,3,), "NA"),"DD/MM/YYYY")
```

```
=TEXT(IFNA(VLOOKUP(V16;Returns!$A$2:$C$1635;3;); "NA");"DD/MM/YYYY")
```

V	W	X	Y	Z	AA
Order ID	Profitable	Margin	Order result	Status	Date of Return
548	Loss	Less than 30% loss	OK	Completed	NA
548	Gain	Less than 30% gain	Good	Completed	NA
612	Loss	More than 30% loss	Poor	Returned	09/08/2016
612	Gain	Less than 30% gain	Good	Returned	09/08/2016
613	Loss	More than 30% loss	Poor	Completed	NA
613	Gain	Less than 30% gain	Good	Completed	NA

This means that we not only **LOOKUP** the value for **Date of Return**, we also format it to **DD/MM/YYYY**.

Exercise 5: Using the Conditional Formatting to highlight

A lot of times, we want to color code Excel sheets based on the various states of the records. In this case, we want to color code the whole table based on the current business rules:

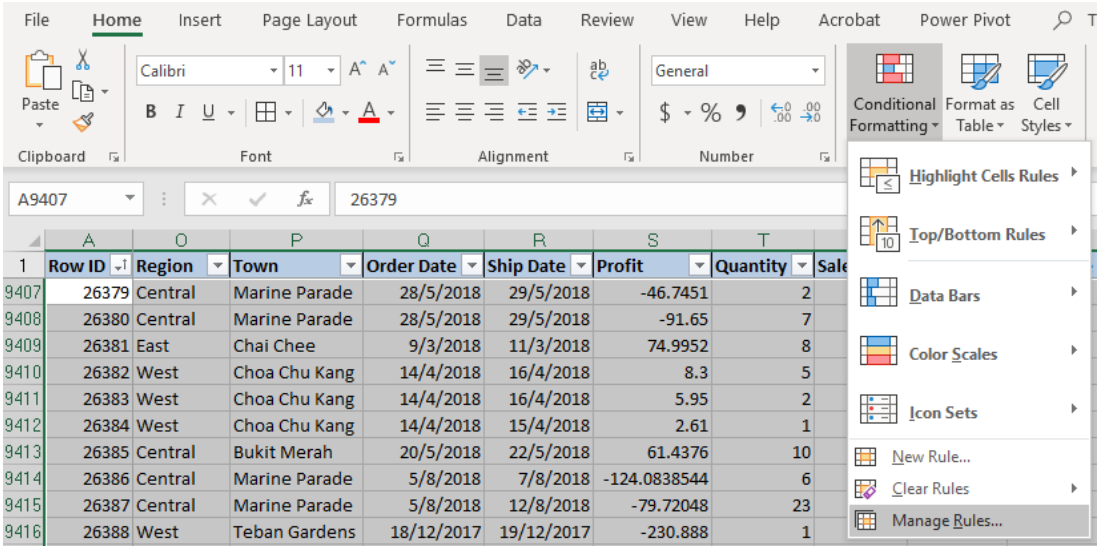
Business Rule	*Display in Order result*	*Color code*
IF order has more than 30% margin loss	Poor	Red
IF order has less than 30% margin loss	OK	Pink
IF order has more than 30% margin gain	Excellent	Blue
IF order has less than 30% margin gain	Good	Green

We will introduce the concept of **Conditional Formatting.** It is a way to color code the entire table based on different values of a single or multiple cells.

1. First, select the entire table in the **Orders worksheet,** by selecting the first cell in the first row, holding "**shift+ctrl**" keys and tapping the **arrow** keys (right and down). All the rows in the table should be highlighted.

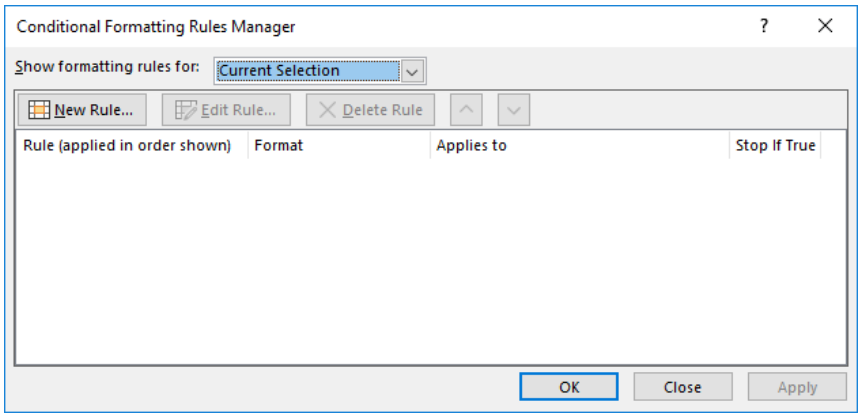
Row ID	Order Priority	Discount	Unit Price	Shipping Cost	Customer ID	Customer Name	Ship Mode	Customer Segment	Product Category	Product Sub-Category
2	Not Specified	0.01	2.08	2.56	2867	Dana Teague	Regular Air	Corporate	Office Supplies	Scissors, Rulers and Trimmers
27	Critical	0.06	12.44	6.27	1821	Vanessa Boyer	Regular Air	Consumer	Office Supplies	Storage & Organization
52	Critical	0.08	155.99	8.08	1402	Wesley Tate	Regular Air	Corporate	Technology	Telephones and Communication
53	Critical	0.1	6.48	10.05	1402	Wesley Tate	Regular Air	Corporate	Office Supplies	Paper
62	High	0.02	48.58	54.11	2747	Brian Grady	Delivery Truck	Corporate	Furniture	Bookcases
63	High	0.07	39.48	1.99	2747	Brian Grady	Regular Air	Corporate	Technology	Computer Peripherals
64	Medium	0.08	124.49	51.94	553	Kristine Connolly	Delivery Truck	Corporate	Furniture	Tables
66	High	0.02	3.69	0.5	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Labels
67	High	0.09	3.85	0.7	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Pens & Art Supplies
68	Not Specified	0.06	11.7	6.96	1630	Jimmy Han	Regular Air	Home Office	Office Supplies	Appliances
78	Low	0.09	6.08	1.82	898	Harriet Hodges	Regular Air	Small Business	Office Supplies	Rubber Bands
87	Critical	0.04	3.08	0.99	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Labels
88	Critical	0.02	6.48	5.9	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Paper
89	Critical	0.04	125.99	4.2	3106	Alexander O'Brien	Regular Air	Home Office	Technology	Telephones and Communication
95	Medium	0.1	5.28	6.26	3011	Tammy Raynor	Regular Air	Corporate	Office Supplies	Paper
96	Medium	0.01	65.99	2.5	3011	Tammy Raynor	Regular Air	Corporate	Technology	Telephones and Communication
97	High	0.03	7.3	7.72	563	Marjorie Pope	Regular Air	Corporate	Office Supplies	Binders and Binder Accessories

2. Now, at the top ribbon, click on **Home→ Conditional Formatting.** Click on **"Manage Rules"** in the dropdown selection.

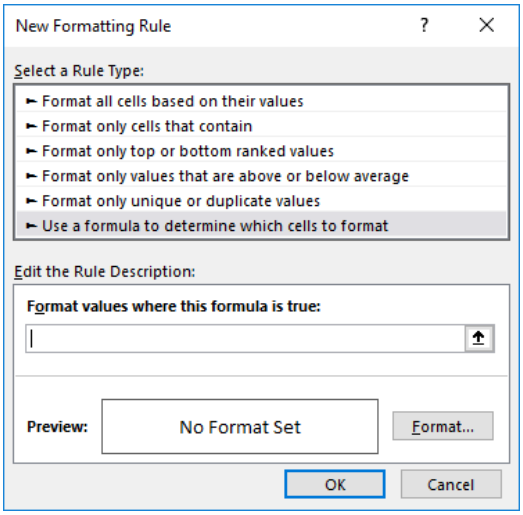


Row ID	Region	Town	Order Date	Ship Date	Profit	Quantity
26379	Central	Marine Parade	28/5/2018	29/5/2018	-46.7451	2
26380	Central	Marine Parade	28/5/2018	29/5/2018	-91.65	7
26381	East	Chai Chee	9/3/2018	11/3/2018	74.9952	8
26382	West	Choa Chu Kang	14/4/2018	16/4/2018	8.3	5
26383	West	Choa Chu Kang	14/4/2018	16/4/2018	5.95	2
26384	West	Choa Chu Kang	14/4/2018	15/4/2018	2.61	1
26385	Central	Bukit Merah	20/5/2018	22/5/2018	61.4376	10
26386	Central	Marine Parade	5/8/2018	7/8/2018	-124.0838544	6
26387	Central	Marine Parade	5/8/2018	12/8/2018	-79.72048	23
26388	West	Teban Gardens	18/12/2017	19/12/2017	-230.888	1

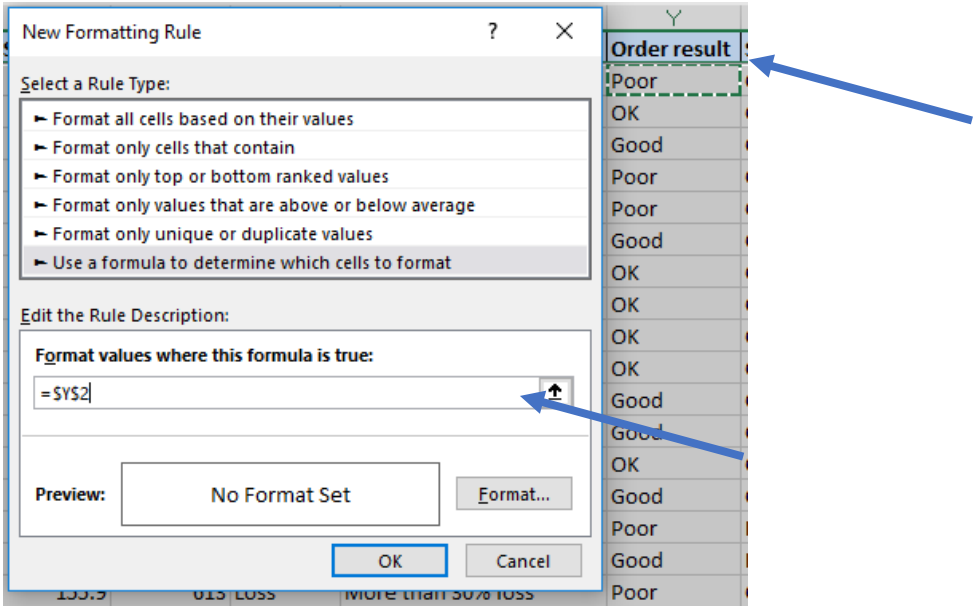
3. In the **Conditional Formatting Rules Manager**, click on **New Rule**. A new window will pop up to create a new rule.



4. In the **New Formatting Rule Window**, select "**Use a formula to determine which cells to format.**" Click into the textbox.

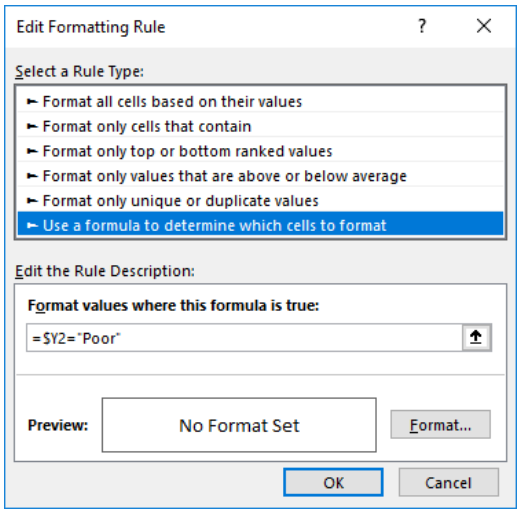


5. We will now select the first cell under **Order result.** Make sure you scroll up all the way so you are selecting the first cell. The value **=$Y$2** should be displayed:

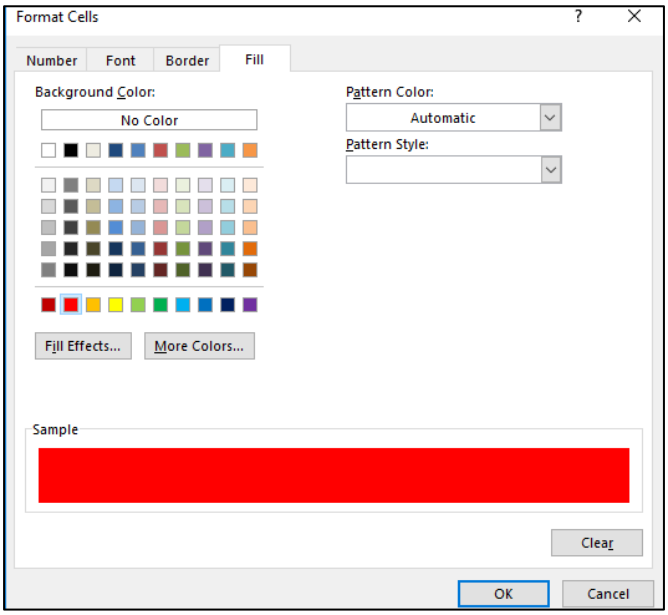


6. Based on the selected cell, we remove the $ sign behind the 2. This is because the ROW number is changing while we format the entire table. If we do not remove, the formatting will always only check the first cell value and not all the rows.
7. Type In **="Poor"** in the formula to check if cell value is equals to **Poor**. Your formula should look like this now:

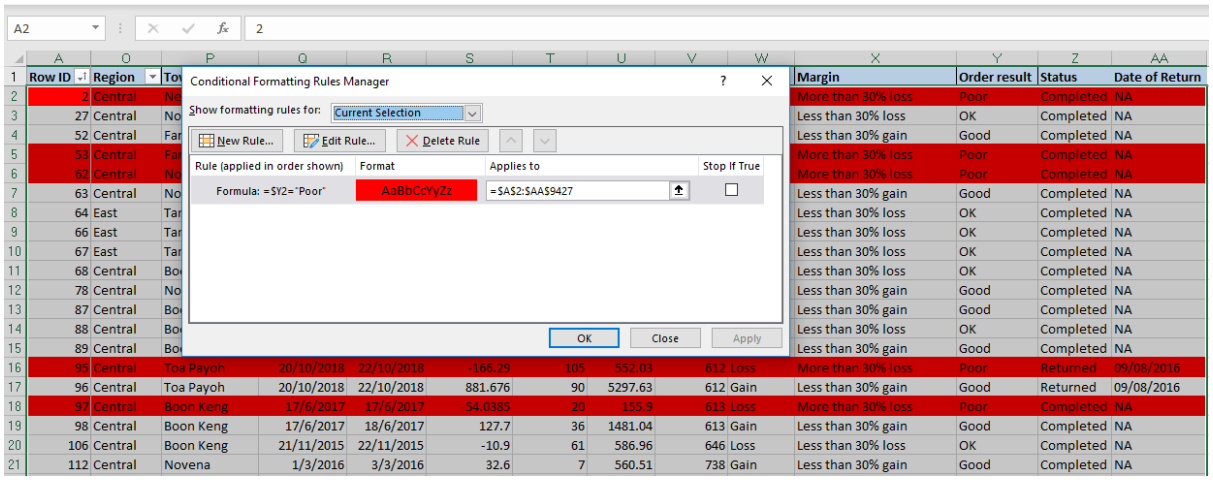
$Y2 = "Poor"



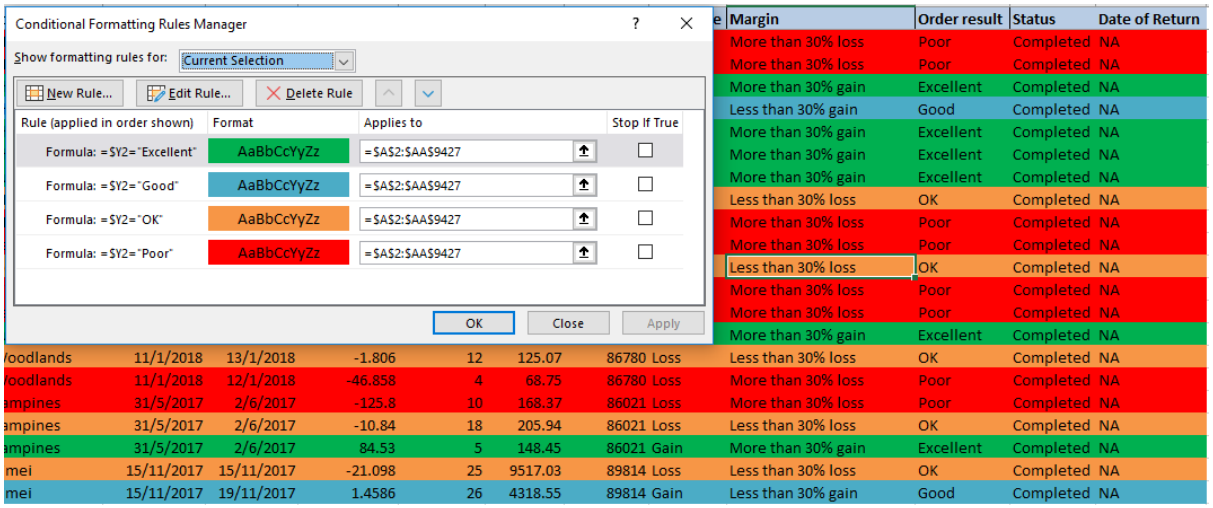
8. Now, click on **Format**. This is to change the row colour to red if the **Status** is equals to **Poor.** The following window will pop up. Select **Fill** tab, and select red as the fill **color**. Click **OK** to confirm the cell color.



9. Click on **OK**, and then **OK** again to create the rule. In the **Conditional Formatting Rules Manager**, you should see your new rule created. Click on **Apply**. You will notice all the rows with **Status** as **Poor** highlighted to red.



10. Go ahead and create 3 more **Formatting Rules,** by repeating steps 3 to 9, using different colors and comparing texts. The result should be as follows:



Day 2- Organizing Data and Effective Analysis

WORKSHOP FOR NAVY

Learning Outcomes:

At the end of this session, you will be able to:

a) Manipulate fields and perform concatenation, extraction, sorting and removing of invalid data
b) Perform data validation and checking
c) Combining multiple spreadsheets
d) Using Pivottable for summary and reporting

Software(s): Microsoft Excel 2019

Instructions:

- Download the datasets for this hands-on:
- Launch Microsoft Office Excel.
- Select a blank workbook and complete the following exercises.

Manipulating Fields

Able to manipulate fields from the worksheets you are working with is a common challenge that one faces. In order to do so, we need to change unstructured data into structured data. After having done data preparation from the previous day, fields in the excel sheet might need further manipulation.

In this section, we will do the following manipulations:

If the contents of the spreadsheet are:

	A	B	C
1	Blk 312	Ang Mo Kio Street 32	#10-113

Table 1: Manipulation Functions

Description	***Function and Syntax***	***Example***	***Result/Remarks***
Concatenation of two strings	CONCATENATE(text1,text2,…)	CONCATENATE(A1,", ", B1, ", ", C1)	Blk 312, Ang Mo Kio Street 32, #10-113
Extraction of a part of the string from the right or left, and removing spaces at the end of the string.	RIGHT(text,num_of_chars)	RIGHT(B1,2)	32
	LEFT(text,num_of_chars)	LEFT(B1,3)	Ang
	MID(text,start_num,num_of_chars)	MID(B1,8,3)	Kio
	TRIM(text)	TRIM(B1)	-return B1 with spaces taken out at the end
Finding a given text in a cell	FIND(find_text,within_text,start_num) -returns the position of find_text in within_text	FIND("Kio",B1) *-if no start_num, it assumes starting at 1*	8

Exercise 1: Manipulating fields in a worksheet

a. CONCATENATE

7. Open SuperStore workbook. Look for the spreadsheet: **Orders. Orders** can be seen below:

Row ID	Order Priority	Discount	Unit Price	Shipping Cost	Customer ID	Customer Name	Ship Mode	Customer Segment	Product Category	Product Sub-Category
2	Not Specified	0.01	2.08	2.56	2867	Dana Teague	Regular Air	Corporate	Office Supplies	Scissors, Rulers and Trimmers
27	Critical	0.06	12.44	6.27	1821	Vanessa Boyer	Regular Air	Consumer	Office Supplies	Storage & Organization
52	Critical	0.08	155.99	8.08	1402	Wesley Tate	Regular Air	Corporate	Technology	Telephones and Communication
53	Critical	0.1	6.48	10.05	1402	Wesley Tate	Regular Air	Corporate	Office Supplies	Paper
62	High	0.02	48.58	54.11	2747	Brian Grady	Delivery Truck	Corporate	Furniture	Bookcases
63	High	0.07	39.48	1.99	2747	Brian Grady	Regular Air	Corporate	Technology	Computer Peripherals
64	Medium	0.08	124.49	51.94	553	Kristine Connolly	Delivery Truck	Corporate	Furniture	Tables
66	High	0.02	3.69	0.5	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Labels
67	High	0.09	3.85	0.7	3289	Emily Britt	Regular Air	Corporate	Office Supplies	Pens & Art Supplies
68	Not Specified	0.06	11.7	6.96	1630	Jimmy Han	Regular Air	Home Office	Office Supplies	Appliances
78	Low	0.09	6.08	1.82	898	Harriet Hodges	Regular Air	Small Business	Office Supplies	Rubber Bands
87	Critical	0.04	3.08	0.99	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Labels
88	Critical	0.02	6.48	5.9	3106	Alexander O'Brien	Regular Air	Home Office	Office Supplies	Paper
89	Critical	0.04	125.99	4.2	3106	Alexander O'Brien	Regular Air	Home Office	Technology	Telephones and Communication

8. Our goal is to identify the customer and his/her purchase. To do this, create a new field in column x of **Orders** in and name "Customer Purchase".
9. In column X, row 2, add the following:

   =CONCATENATE($G2," bought ", $M2," from ",$P2)

   You should have this result:

Customer Purchase
Dana Teague bought Kleencut® Forged Office Shears by Acme United Corporation from Newton

4. Do a copy of the formula in column X2 and paste it (CTRL V) to the rest of the rows in column X. It should look like this below:

Product Name	Produc	Region	Town	Order Date	Ship Dat	Profit	Qu	Sal	Orde	Customer Purchase
Kleencut® Forged Office Shears by Acme United Corporation	0.55	Central	Newton	20/2/2018	21/2/2018	-4.64	2	6.93	6	Dana Teague bought Kleencut® Forged Office Shears by Acme United Corporation from Newton
Eldon Simplefile® Box Office®	0.57	Central	Novena	7/8/2016	9/8/2016	-37.04	25	312	193	Vanessa Boyer bought Eldon Simplefile® Box Office® from Novena
I888 World Phone	0.6	Central	Farrer Park	18/3/2018	20/3/2018	257.76	20	2635	322	Wesley Tate bought I888 World Phone from Farrer Park
Xerox 1997	0.37	Central	Farrer Park	18/3/2018	20/3/2018	-291.59	46	281	322	Wesley Tate bought Xerox 1997 from Farrer Park
O'Sullivan 2-Shelf Heavy-Duty Bookcases	0.69	Central	Novena	19/9/2016	21/9/2016	-1348.06	60	2983	358	Brian Grady bought O'Sullivan 2-Shelf Heavy-Duty Bookcases from Novena
80 Minute CD-R Spindle, 100/Pack - Staples	0.54	Central	Novena	19/9/2016	19/9/2016	269.27	60	2247	358	Brian Grady bought 80 Minute CD-R Spindle, 100/Pack - Staples from Novena
Bevis 36 x 72 Conference Tables	0.63	East	Tampines	18/12/2015	19/12/2015	-500.38	56	6831	359	Kristine Connolly bought Bevis 36 x 72 Conference Tables from Tampines
Avery 501	0.38	East	Tampines	24/1/2017	26/1/2017	-0.048	4	15	386	Emily Britt bought Avery 501 from Tampines
Avery Hi-Liter Pen Style Six-Color Fluorescent Set	0.44	East	Tampines	24/1/2017	26/1/2017	-2.544	4	15.7	386	Emily Britt bought Avery Hi-Liter Pen Style Six-Color Fluorescent Set from Tampines

b. EXTRACTION

1. In the same workbook, focus on column X – Customer Purchase, and try using the RIGHT(), LEFT(), TRIM(), FIND() and MID() to extract the content to match the original content.
2. To extract the name from 'Customer Purchase', try the following command in:

   Column Y:

   Type in the formula in cell Y2: "=LEFT($X2,FIND(" bought",$X2)-1)"

   - Copy the above formula and paste it to the remaining rows in column Y.

   Question

   1. *Why do we have a "-1" at the end of the formula?*
   2. *Did the formula work for the rest of the rows? Why?*

3. In the next exercise, extract the 'Product name' from 'Customer Purchase' in Column X.

   In *Column Z*:

   Type in the formula in cell Z2:"=MID($X2,(FIND("bought",$X2)+7),FIND("from",$X2)-(FIND("bought",$X2)+7)-1)"

   - Copy the above formula and paste it to the remaining rows in column Y.

   Question

1. *Why does this formula work?*
2. *Did it work for the rest of the rows? Why?*

c. *Challenge Question*

1. What would he formula be if you want to extract the 'Town' from Column X – Customer Purchase?

```
=RIGHT($X2,(LEN ($X2)-FIND ("from", $X2)-4))
```

2. Save your work.

More on Data Validation

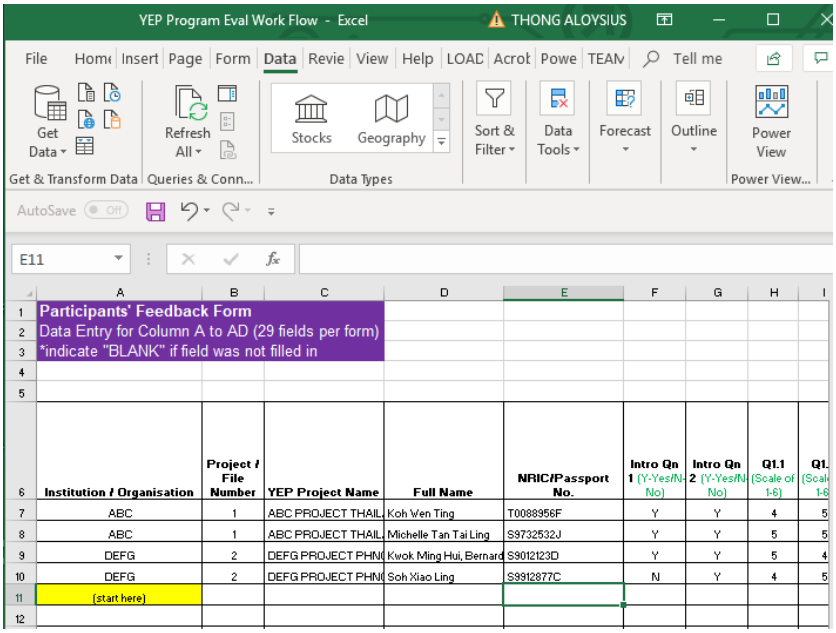
You can make Excel data entry more efficient by setting up data entry cells to accept only certain values. To do this, you can set up a cell with data validation criteria that specify the allowed value or values. This is called a data validation rule.

You can work with numbers, dates, times, or even text length, and you can set up criteria between two values, equal to a specific value, greater than a value, and so on. Excel also lets you tell the user what to enter by defining an input message that appears when the user selects the cell. You can also configure the data validation rule to display a message when the user tries to enter an invalid value.

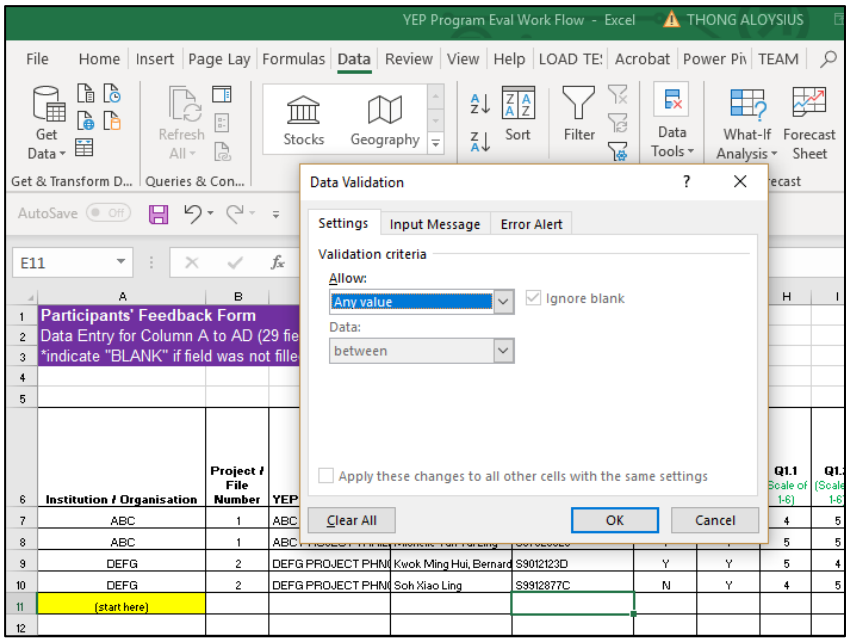
For the exercises from here on, we will be using Datasets: **"YEP_Workflow.xslx"**

Exercise 2.1: Setting Data Validation Rules for NRIC

1. Click on the cell or group of cells that you want to restrict
2. For this exercise, we will set the validation rule for the "NRIC/Passport No" field or column.
3. Select the empty cell below the "NRIC/Passport No" column.
4. Click on the Data tab of the given spreadsheet.



5. Then click on Data *Tools>Data Validation* and select *Data Validation* and the Data Validation dialog box appears.
6. Click on Settings tab.
    a. You will see Validation criteria.
        i. A drop down box appears when you select *Allow:.*

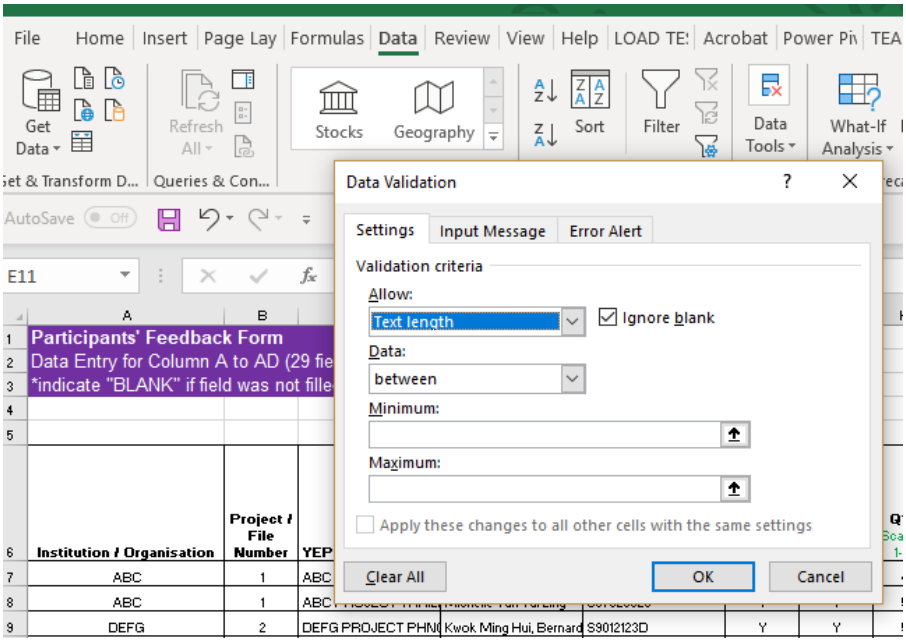


Selecting the drop down box, the following items appears-

Whole number, decimal, list, date, time, text-length and custom.

ii. We will set the "Text length" to be 9 alphanumeric characters long. For *Input Message,* you will input a *Title,* and the *Message.*

iii. Also go to the *Error Alert* tab. Select the *Style,* key in Title and the *Message*.



iv. Then save and exit.

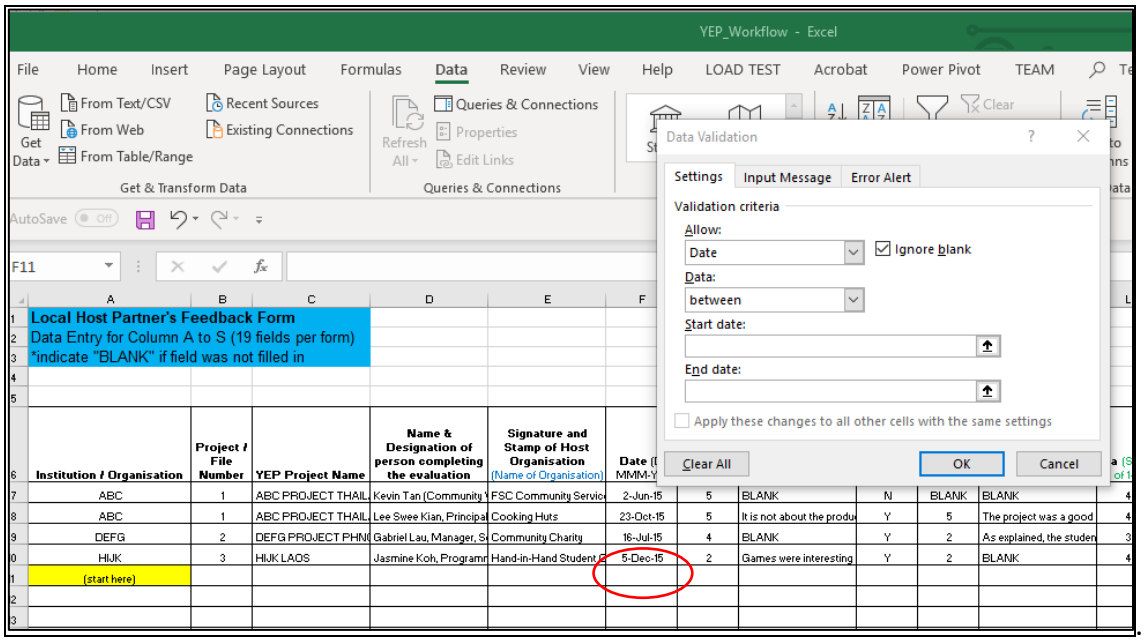
7. We have completed setting the Data Validation Rule for NRIC field.

Exercise 2.2: Setting Data Validation Rules for - Date (eg Date of Birth)

Validation for dates can be done for the following:

- Specifying a start and end date
- Dropdown list of valid dates
- Crete a rule in a custom formula

1. Make "FY16 Local Host" tab active. Place cursor at F11 cell. Go to Data tab and select Data Validation. You'll see the following:



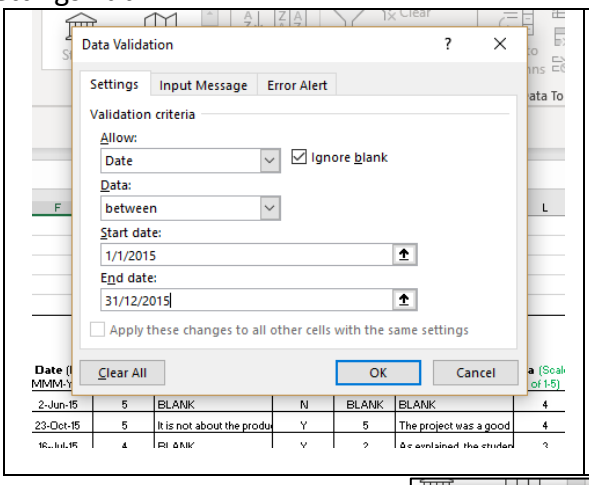
2. At the Data Validation window, do the following:
   a. "Settings" tab:
      <Allow:> choose Date
      <Data:> choose "between"
      <Start date:> enter "1/1/2015"
      <End date:> enter "31/12/2015"
      Note: This is because you are only allowing this entry for the calendar year 2015.

   b. Select "Input Message" tab:
      <Title:> enter "Validity"
      <Input message:> enter "Start date and end date must be within 2015"

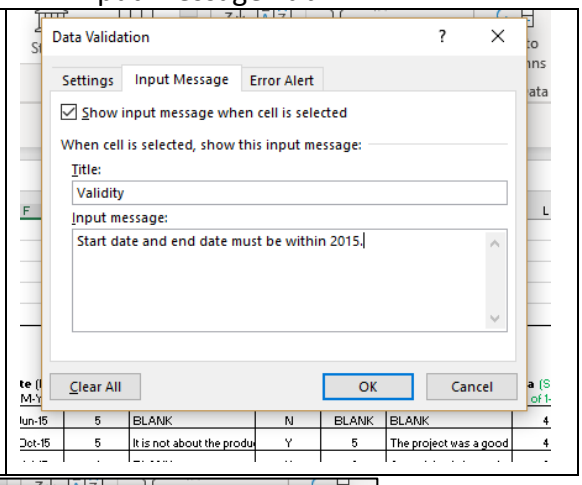
   c. and then select "Error Alert" tab:
      <Style:> select "Stop"
      <Title:>, enter "Error Entry!"
      <Error message:> enter "Not a valid entry."

      Then click on "OK".
      It should look like the images below.

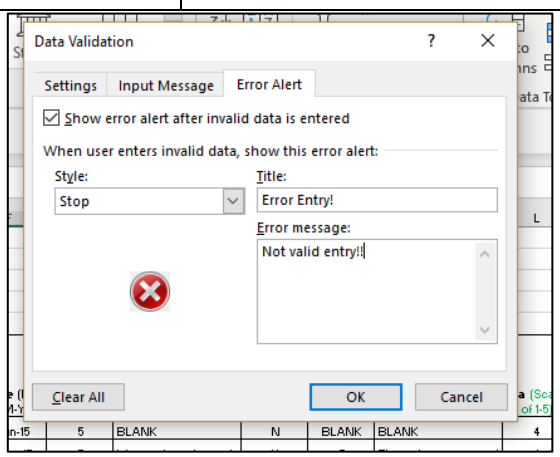
Settings Tab:



Input Message Tab:



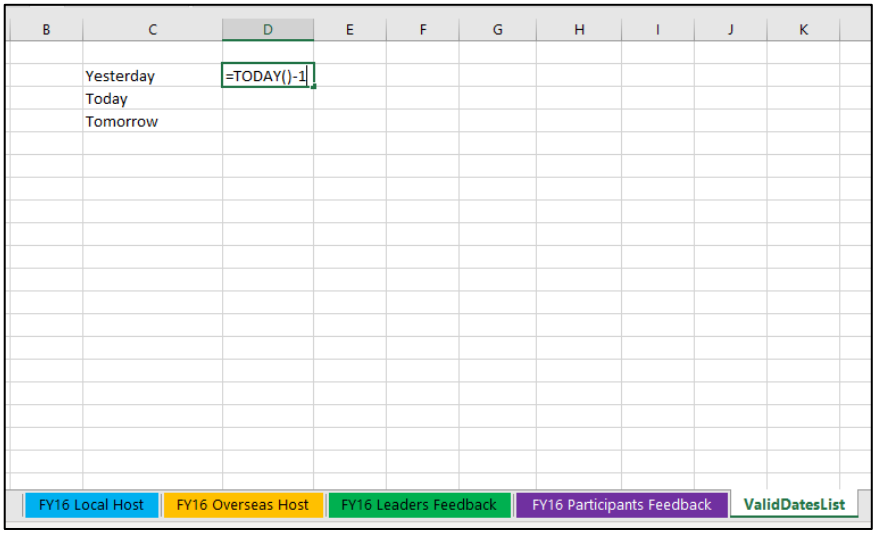
Error Alert Tab:



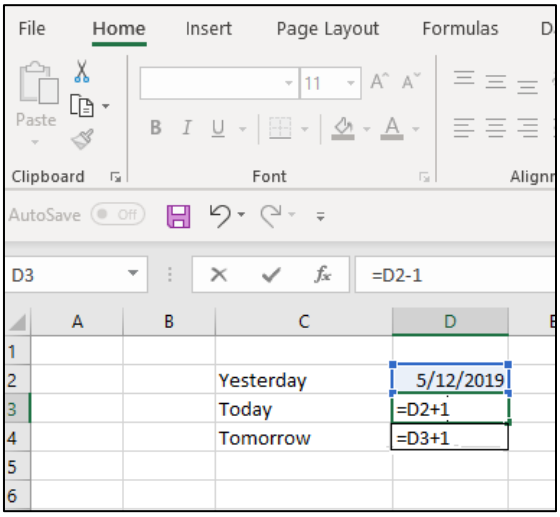
3. Test your date validation.
    a. Enter the correct dates allowed. How does the validation react?
    b. Enter wrong dates. How does the validation react?

Exercise 2.2: Setting Data Validation Rules (List)

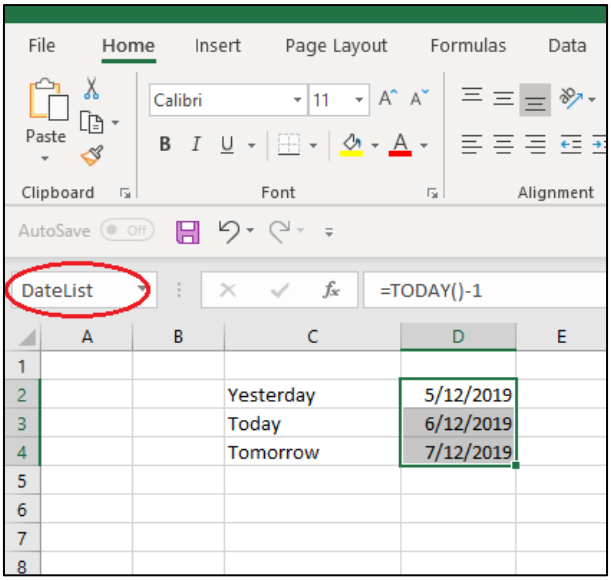
1. We will be creating a list of valid dates for this exercise.
2. On the Workbook given to you, create a new worksheet and name it “ValidDatesList”.
3. On this worksheet, go to Cell C2, key in “Yesterday” and so on as shown. In cell D2, put n the formula ‘=TODAY()-1’.



4. Then in cells D3 and D4 with formula reflecting its date. The image in the next page shows:



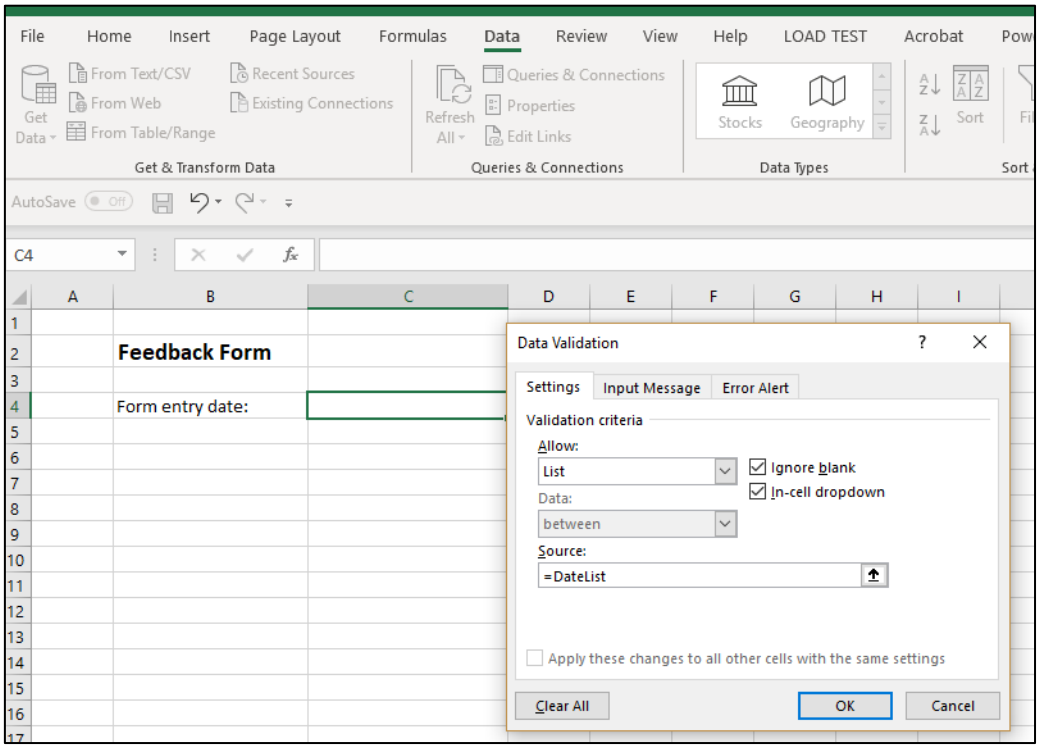
5. Name the list of 3 dates in Column D to “DateList” and then press Enter.



6. Now create the "Data Validation Drop Down List" in a new worksheet, "FeedbackForm".
    a. On the FeedbackForm worksheet select cell C4, where the drop down list of dates will be added.
    b. On the Excel ribbon, click 'Data' tab. Then select 'Data Validation'.
    c. On 'Settings' tab, do the following:

        <Allow:> choose List
        <Source:> enter "=DateList"
        Then click on OK.

        See the image in the next page.



d. You should get the drop down list with the 3 options of dates.

This completes our exercise in Data Validation.

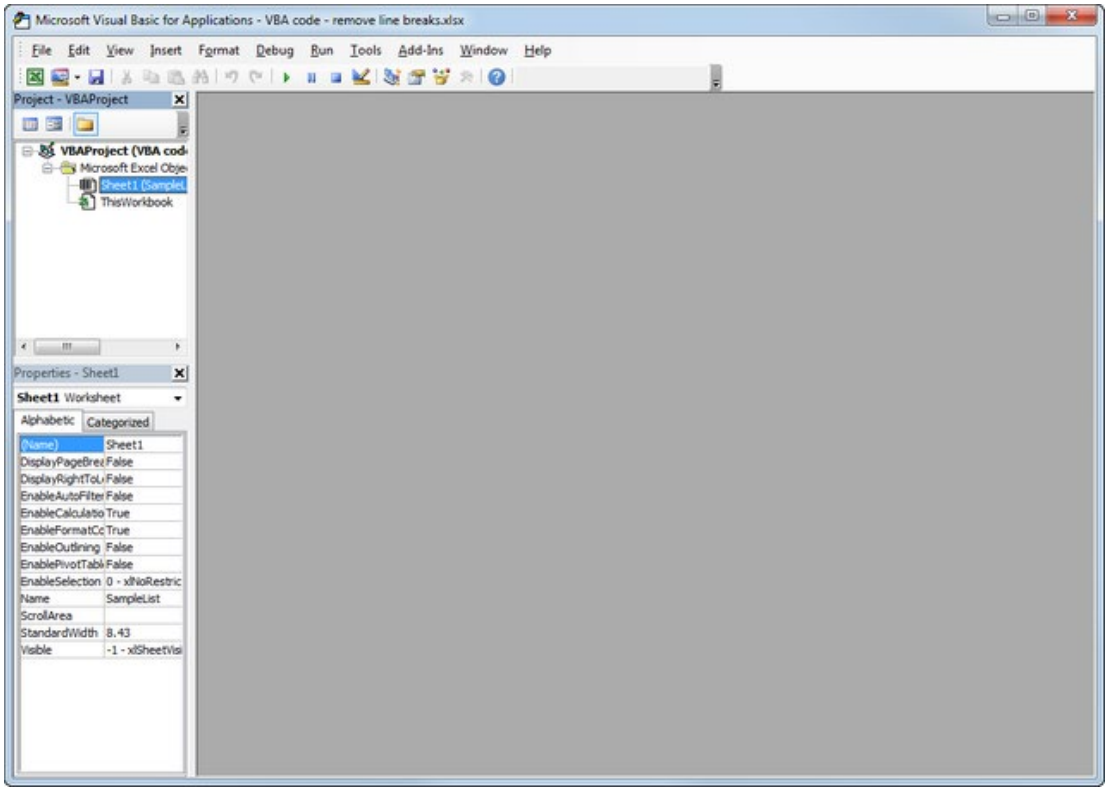
In your own time explore the other options in <Allow:> of the Data Validation dialog box. The idea is the same and you can explore the differences and the use to make your workbook workable for you.

Exercise 3: Merging Multiple Worksheets to a Workbook

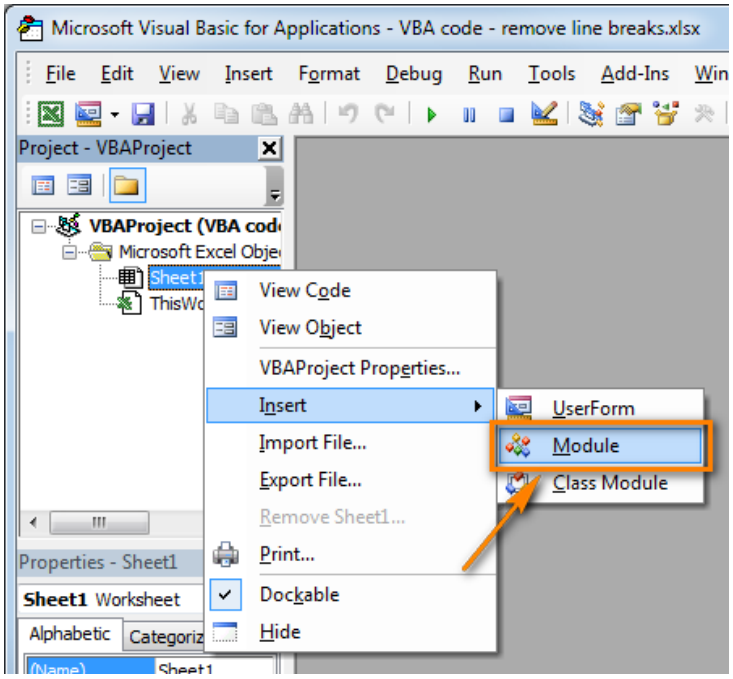
A macro is a sequence of instructions that perform one or more actions or return a result. You can save time and make the process of creating a macro easier by recording some or all of the actions you want your macro to perform. To build a macro that manipulates Excel in some way, you use the macro recorder. After you activate the recorder, you use Excel to perform the action or actions that you want in the macro, which Excel then translates into the equivalent VBA statements and stores as a macro for later use. You can store your recorded macros in any workbook.

a. Merging Excel files with VBA (macros)

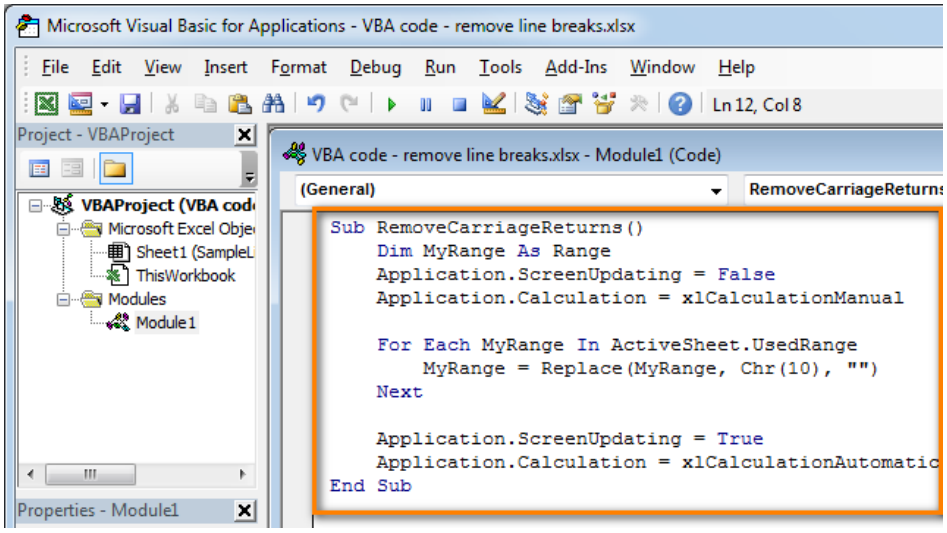
1. Merging of Excel files is possible with the use of macro. It is a faster method of merging the files rather than to perform copy and move.
2. The VBA code is listed below. Follow these steps:
    a. The VBA codes below copies all the sheets from all the Excel files that you have selected into one of the workbook. Hence, start your Excel with a new workbook.
    b. The files that are to be merged should not be opened.
    c. To add this macro into your own workbook, perform the following steps:
        i. Press <Alt + F11> to open the VB editor.



ii. Right-click on your workbook name in the *"Project-VBAProject"* pane (at the top left corner of window) and select *Insert>Module* from the context menu.



iii. Copy the VBA codes in the following page (page 56) by highlighting the codes and do a CTRL-C and paste it to the right pane of the VBA editor (*"Module1"* window)



iv. Now you can save your workbook as "**Excel macro-enabled workbook**".

v. To run this new macro, press <Alt+F8> to open the macro dialog.

vi. Then select the macro you wanted – “MergeExcelFiles” and click Run. You can select multiple workbooks by holding down the Ctrl-key and clicking the files you want to merge.

MergeExcelFiles VB Codes

```
Sub MergeExcelFiles()
   Dim fnameList, fnameCurFile As Variant
   Dim countFiles, countSheets As Integer
   Dim wksCurSheet As Worksheet
   Dim wbkCurBook, wbkSrcBook As Workbook

   fnameList = Application.GetOpenFilename(FileFilter:="Microsoft Excel Workbooks
(*.xls;*.xlsx;*.xlsm),*.xls;*.xlsx;*.xlsm", Title:="Choose Excel files to merge", MultiSelect:=True)

   If (vbBoolean <> VarType(fnameList)) Then

      If (UBound(fnameList) > 0) Then
         countFiles = 0
         countSheets = 0

         Application.ScreenUpdating = False
         Application.Calculation = xlCalculationManual

         Set wbkCurBook = ActiveWorkbook

         For Each fnameCurFile In fnameList
            countFiles = countFiles + 1

            Set wbkSrcBook = Workbooks.Open(Filename:=fnameCurFile)

            For Each wksCurSheet In wbkSrcBook.Sheets
               countSheets = countSheets + 1
               wksCurSheet.Copy after:=wbkCurBook.Sheets(wbkCurBook.Sheets.Count)
            Next

            wbkSrcBook.Close SaveChanges:=False

         Next

         Application.ScreenUpdating = True
         Application.Calculation = xlCalculationAutomatic

         MsgBox "Processed " & countFiles & " files" & vbCrLf & "Merged " & countSheets & "
worksheets", Title:="Merge Excel files"
      End If

   Else
      MsgBox "No files selected", Title:="Merge Excel files"
   End If
End Sub
```

PivotTables

Analyzing data in a excel sheet can be a challenge I you have many rows and columns to look at. In this aspect, Excel offers the user the PivotTable which allow you to summarize hundreds of records in a concise tabular format. With this, you can then manipulate the layout, i.e the pivot, to see different views.

PivotTable allows you to analyze the information by performing three operations:

i. Grouping data into categories
ii. Summarizing data using calculations
iii. Filtering data to show just the records that you want to work with

Grouping is a powerful data-analysis tool in part because it automatically groups large amounts of data into smaller, more manageable categories. For example, suppose you have a data source with a Region field where each cell contains one of four values: East, West, North, and South. The original data may contain thousands of records, but if you build your PivotTable using the Region field, the resulting table has just four rows — one each for the four unique Region values in your data. You can also nest the grouping, coming from a larger region and then taper to districts within the region.

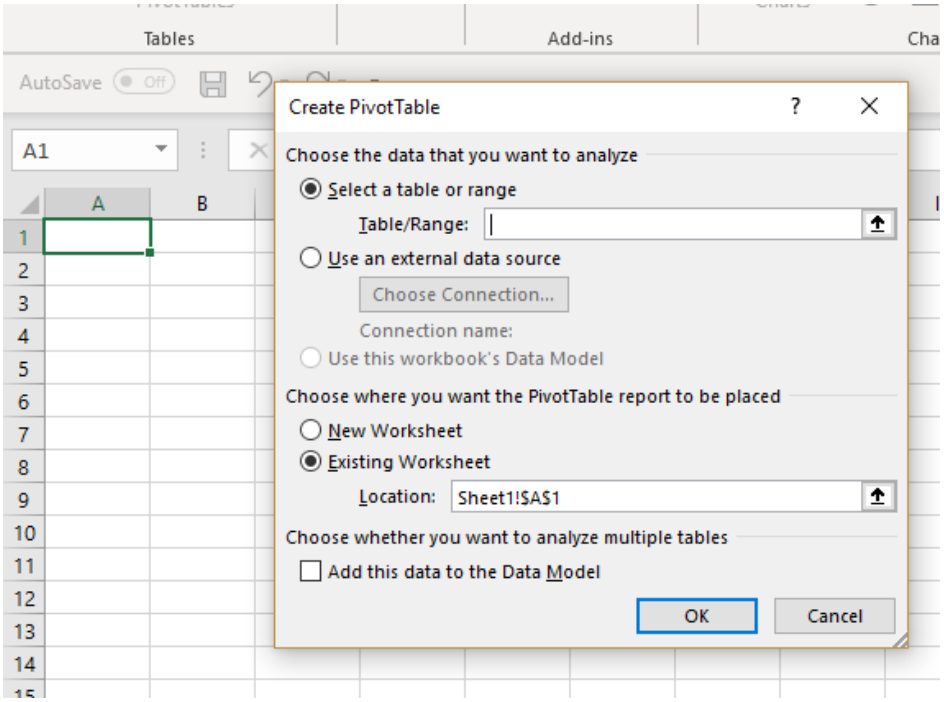
Summarizing would be to display summary calculations for each grouping. The default calculation is Sum, which means for each group, Excel totals all the values in some specified field. For example, if your data has a Region field and a Sales field, a PivotTable can group the unique Region values and display the total of the Sales values for each one. There are also other summary calculations, including Count, Average, Maximum, Minimum, and Standard Deviation. Even more powerful, a PivotTable can display summaries for one grouping broken down by another. For example, suppose your sales data also has a Product field. You can set up a PivotTable to show the total Sales for each Product, broken down by Region.

Filtering enables you to view just a subset of the data. For example, by default the PivotTable's groupings show all the unique values in the field. However, you can manipulate each grouping to hide those that you do not want to view. Each PivotTable also comes with a report filter that enables you to apply a filter to the entire PivotTable. For example, suppose your sales data also includes a

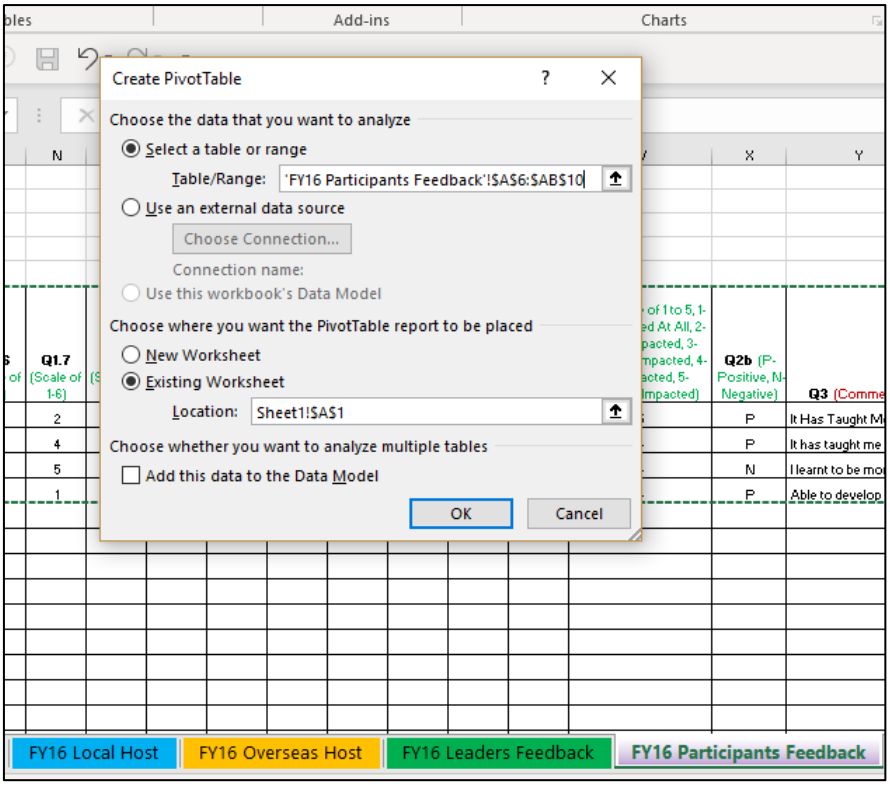
Customer field. By placing this field in the PivotTable's report filter, you can filter the PivotTable report to show just the results for a single Customer.

Exercise 4: Steps to create a PivotTable & PivotChart

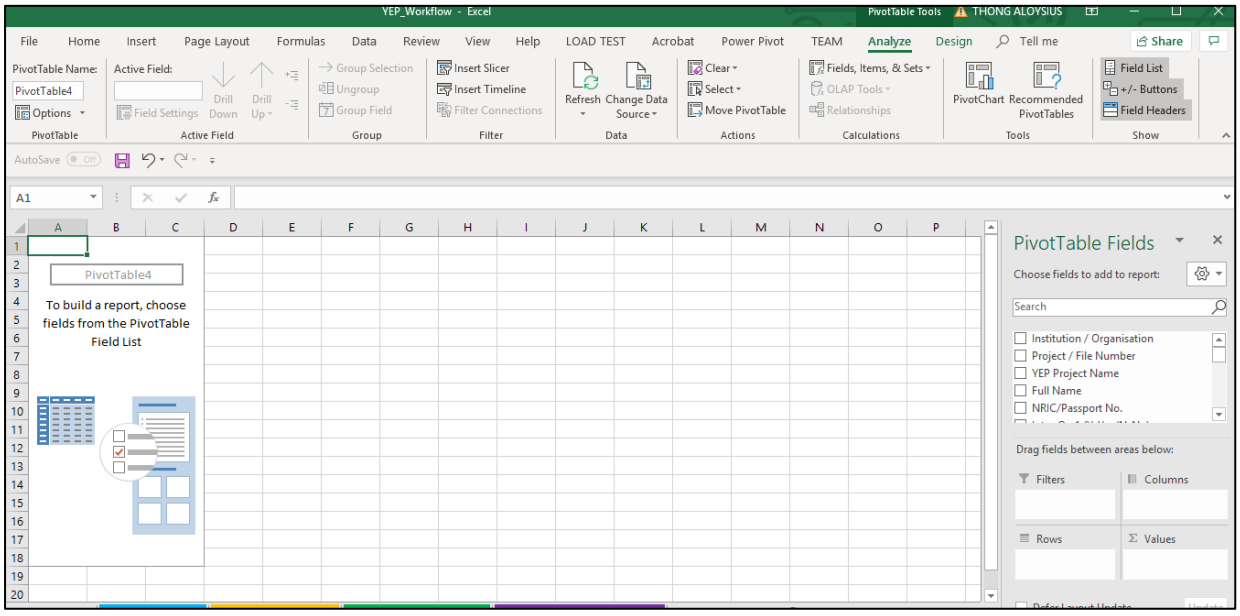
1. Use the YEP_workflow.xslx workbook. Create a new worksheet where you want your PivotTable to be located. This new Worksheet should be "Sheet1".
2. On "Sheet1", highlight cell A1. Then select the "Insert" tab from the toolbar. In the Tables group, select "PivotTable". The "Create PivotTable" window appears.



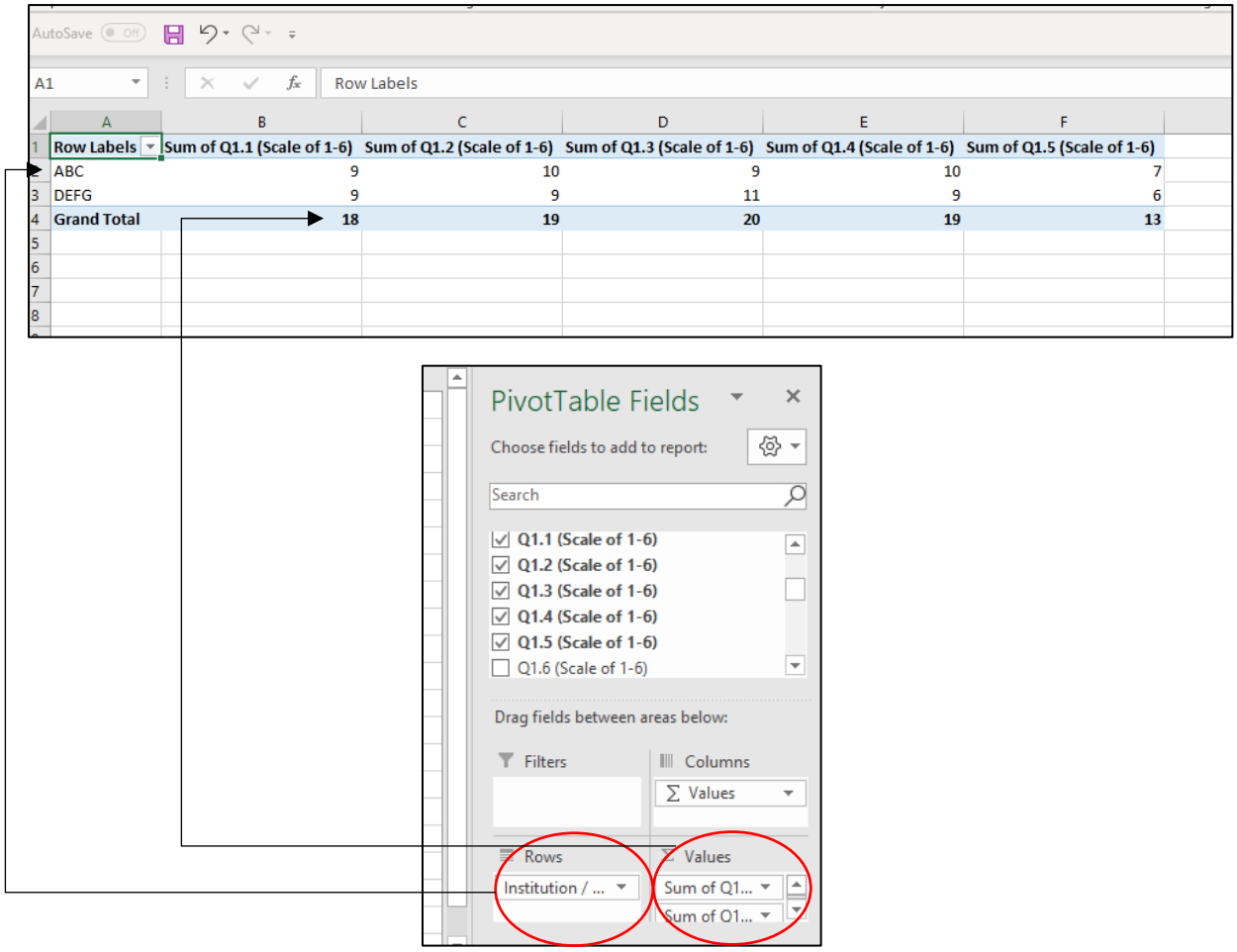
3. Then at the "Select a table or range" checkbox, select "FY16 Participants Feedback" tab, highlight from cell A6 to cell AB10.



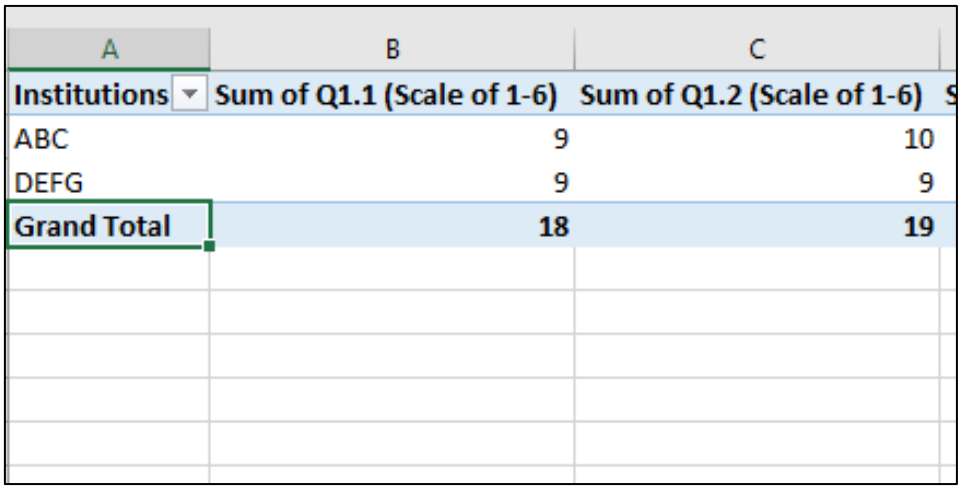
4. Your pivot table should now appear.



5. Now choose the fields you want to add to your report. Go to the PivotTable Fields Pane,. We will drag:
    i. *Institution/Organization to the "Rows" area*
    ii. *Q1.1 to Q1.5 to the "Values" area*

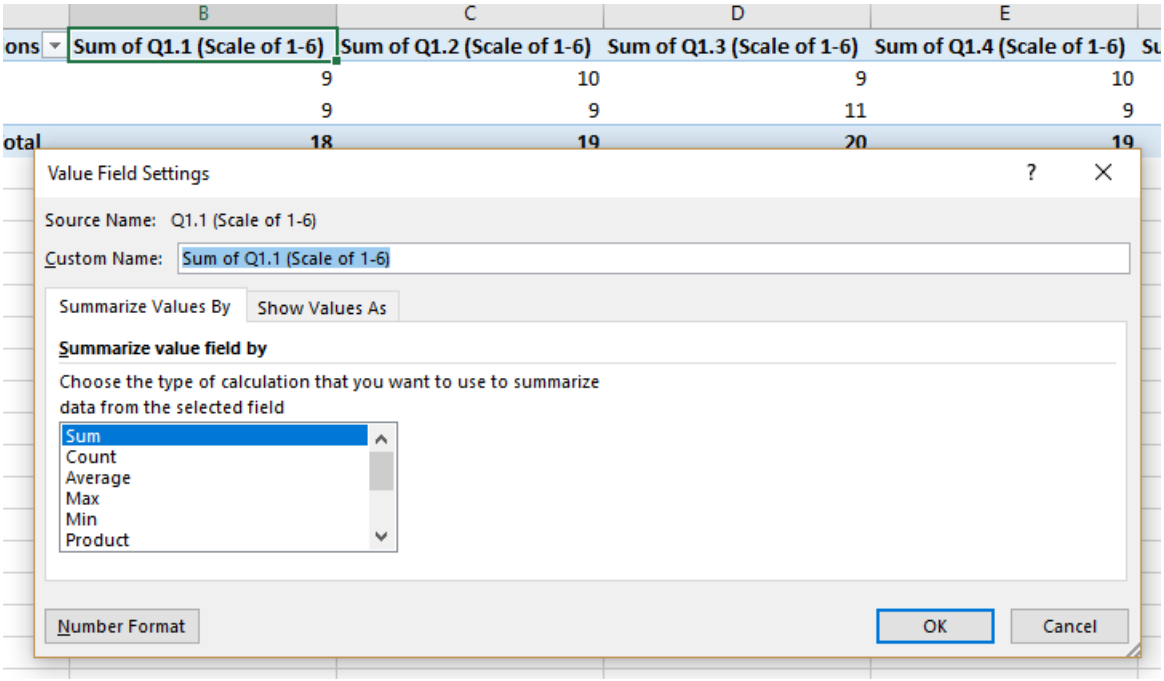


6. Go to the Pivot Table spreadsheet and change Rows Labels to Institution by double clicking the cell A1.

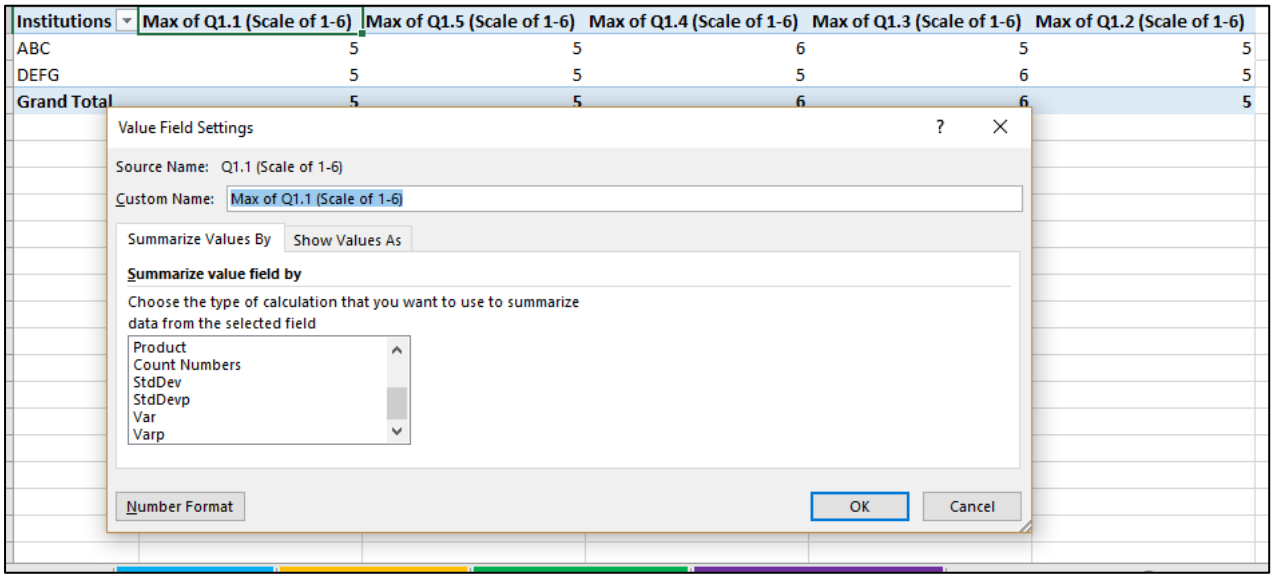


Institutions	Sum of Q1.1 (Scale of 1-6)	Sum of Q1.2 (Scale of 1-6)
ABC	9	10
DEFG	9	9
Grand Total	**18**	**19**

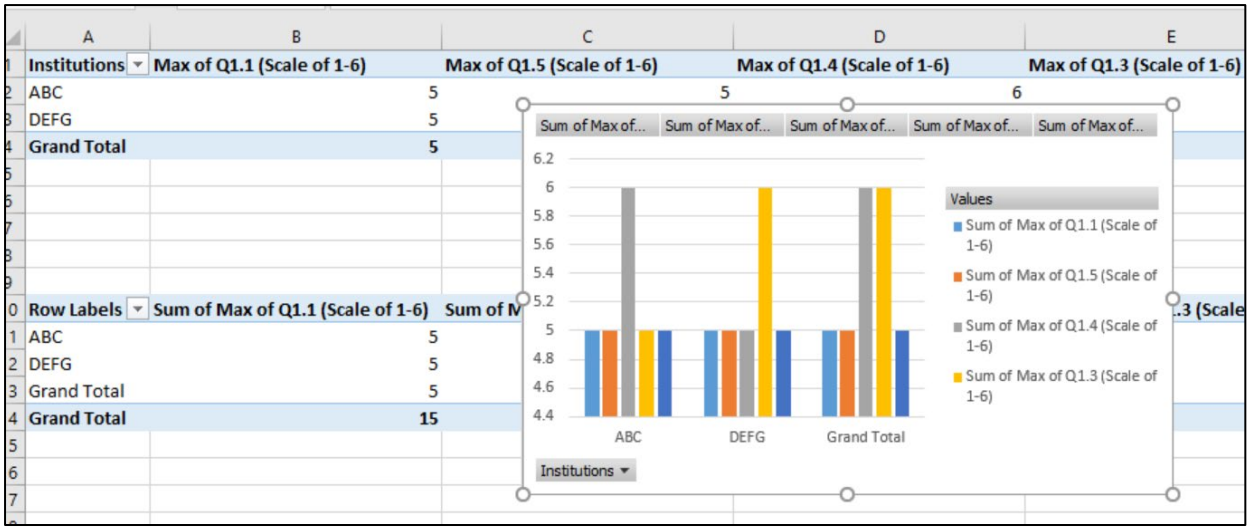
7. Now go to Column B and double click on Cell B1, you will see:



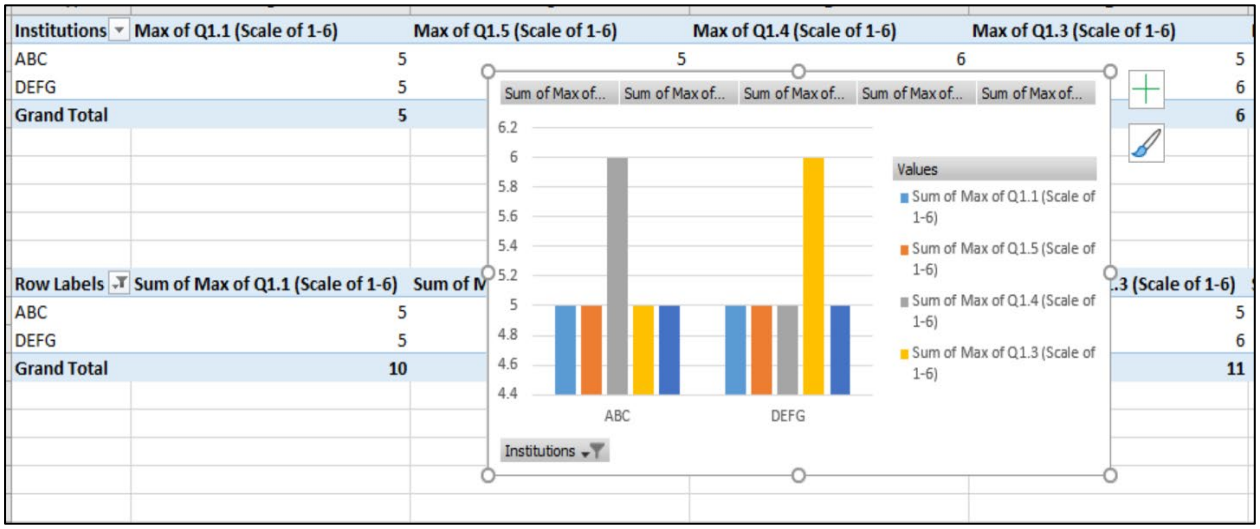
8. Notice that you can select other numerical summary types. In this case, it makes sense to select Count or even Max or Min – depending on your report purpose. There are 11 types of Summaries calculation you can choose from. Let's choose Max to see what is the highest scale that was chosen for each institution for each question.



9. Lastly, we can generate a chart. While on Sheet1, highlight cell A10 and then click on "Insert" Tab at the tools bar. Then go to PivotChart from Charts group and click on it. There are 2 options -select Pivot Chart.
10. Highlight cells A1 to F10. Then click OK. Choose your fields to be placed in the chart from the PivotChart Fields Pane.
11. As you select the fields, the bar chart appears.
12. Notice that you have an extra Grand Total which you could have omitted in the beginning when you select the cells to chart in Point 10 of this worksheet exercise.



13. To rid of Grand Total, go to “Institutions” drop down filter list on the Chart. Unselect “Grand Total”. Notice that the Grand Total is gone from the chart.

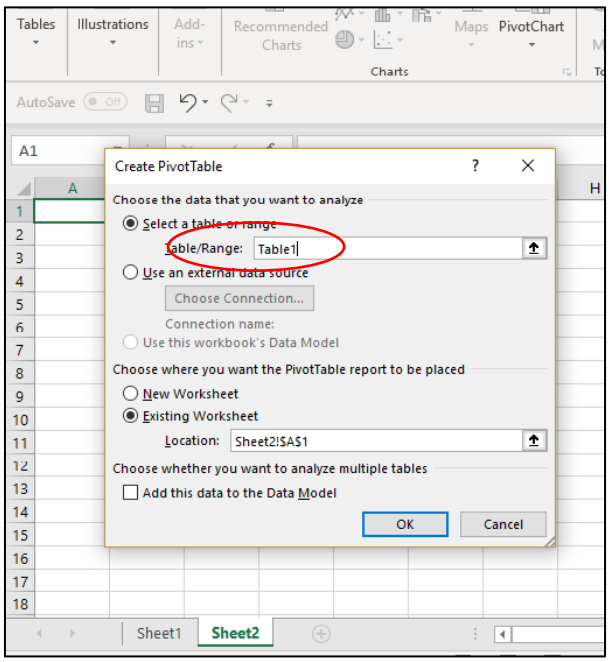


Now that you have gone through the PivotTable and PivotChart, take some time to now test out the different summary types calculations for your set of data.

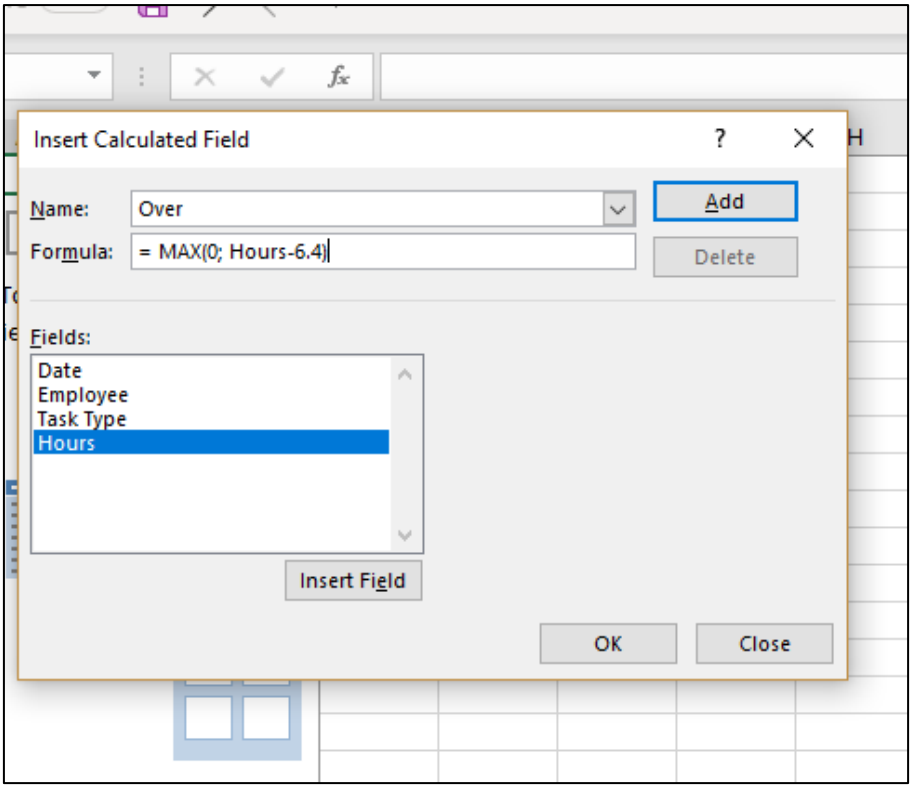
Exercise 5: Creating a Resource Utilization Chart

1. In this exercise, we will be creating a resource utilization chart with the help of PivotTable. We will use the stacked column chart found in Excel. But we have to do some calculations first since the data we receive is usually flat and needs us to do some work on it.

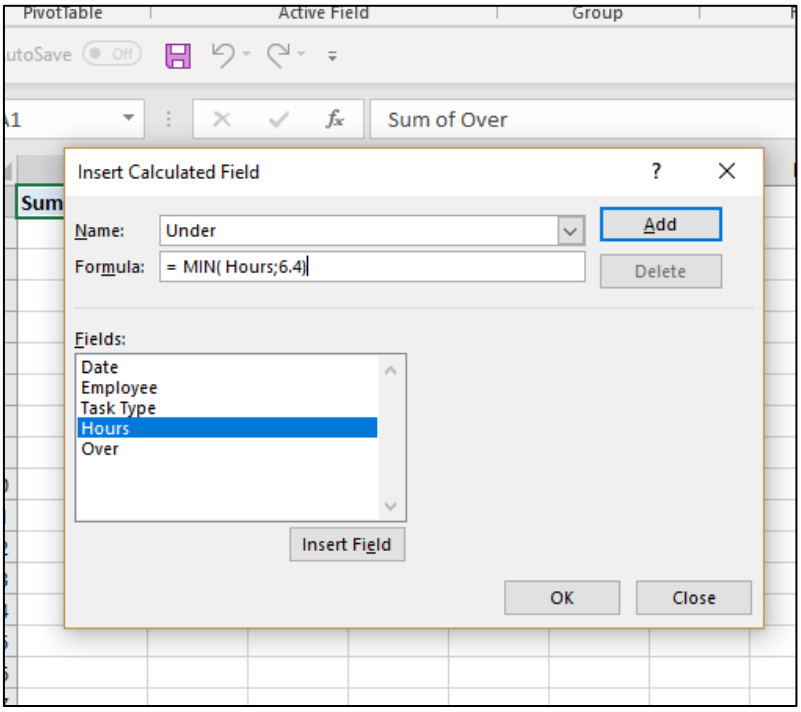
2. Here we will insert 2 calculated fields derived from the data that we have. For this exercise, we will use "ResrcUtilization.xslx" workbook. We would want to create a chart that shows when an employee works more than 6.4 hours (or 80% utilization in an 8 hour day).
3. To begin, we would want to create a PivotTable in Sheet2.
    a. Firstly, create a worksheet by clicking on the "+" at the bottom of the workbook.
    b. Then highlight cell A1 and Insert>PivotTable. In the Table/Range, key in Table1.



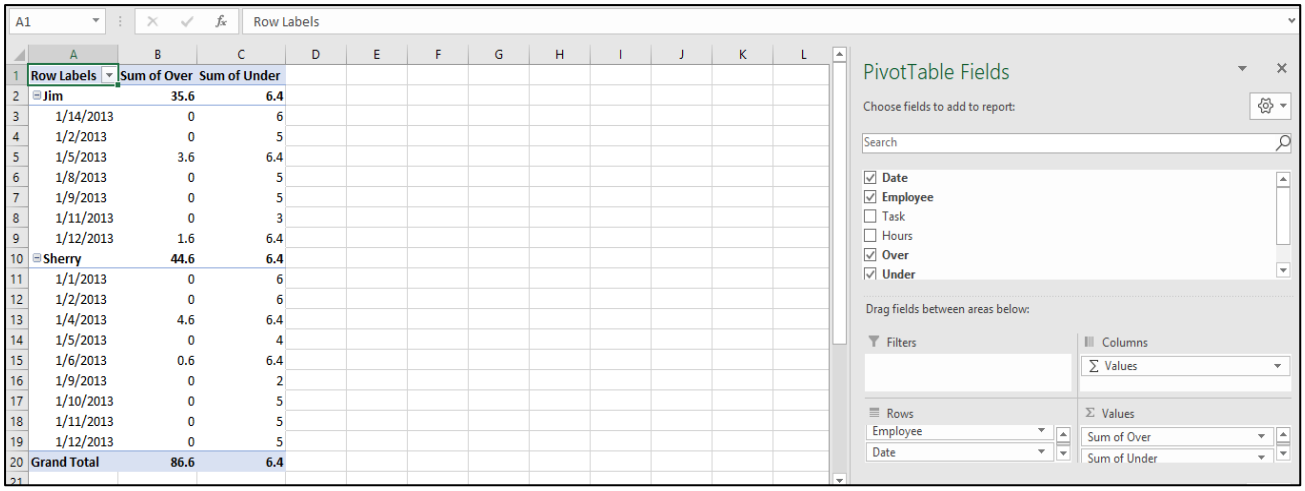
4. Then in Sheet2, click anywhere in the PivotTable and on the "Analyze" tap on the tool bar and click on "Fields, Items, & Sets". Then, click Calculated Field. The "Insert Calculated Field" window appears. Key in the fields as shown in the image below.



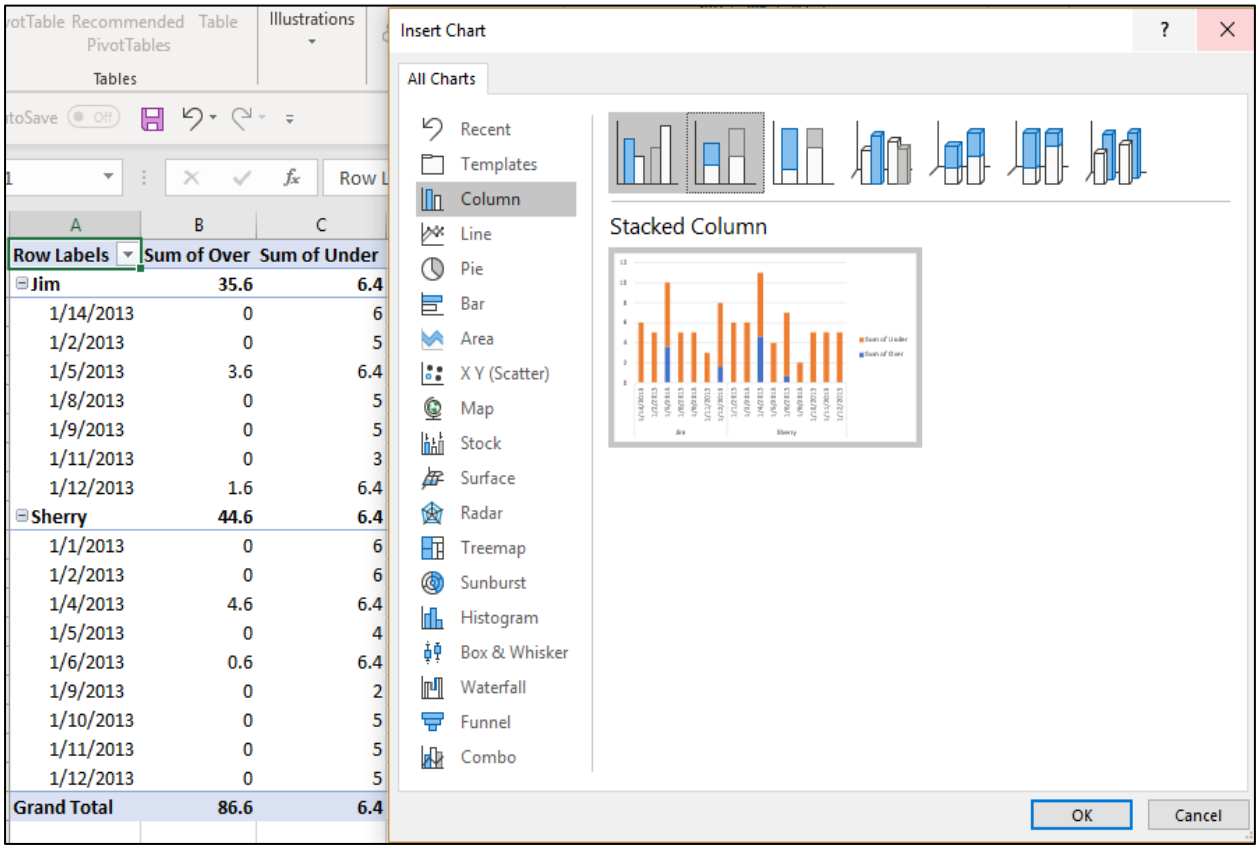
5. The calculated field is "Over" and the formula we use is *Over = MAX (0, Hours-6.4).* This will give the larger value of the number of hours over the target of 6.4 hours. It will set to zero if the calculation is negative.
6. The other calculated field will be "Under" and do the same to insert this calculated field. See image below. Formula for *Under = MIN(Hours,6.4).* Click OK.



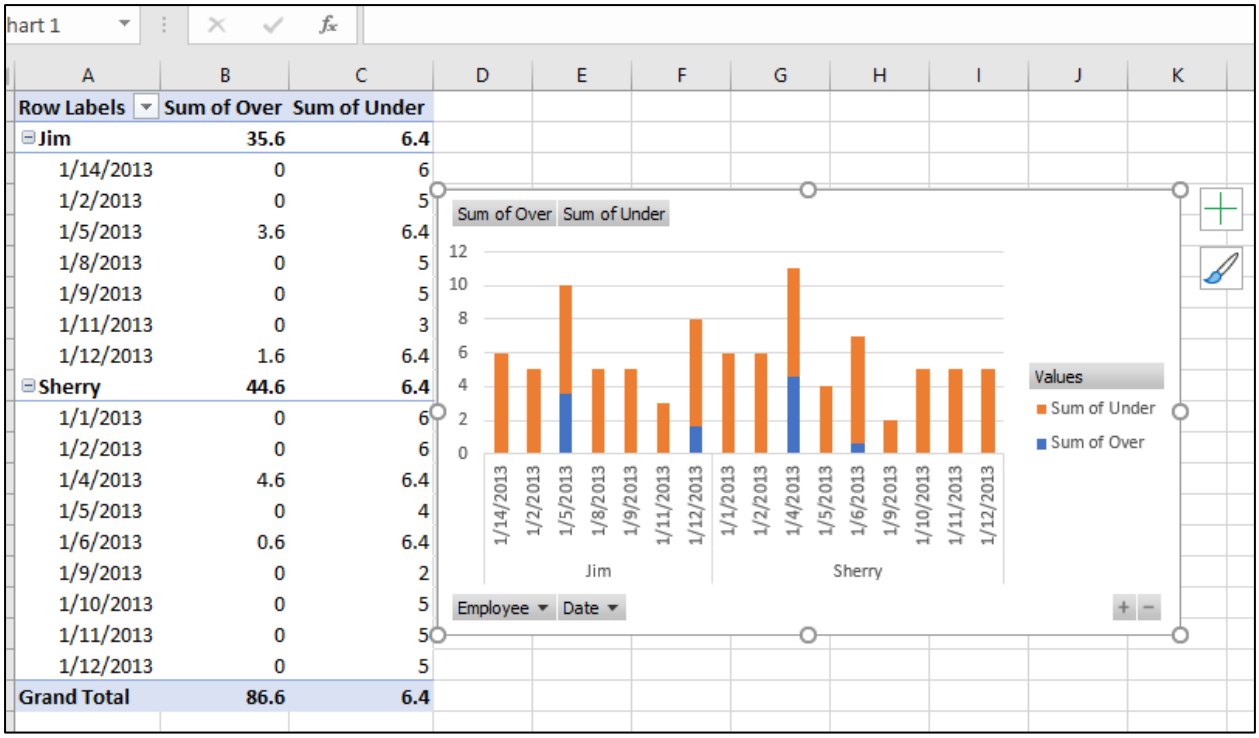
7. With these two calculated fields, we can add them to the PivotTable. Using the Date and Employee dimensions as the row values and we get the following PivotTable.



8. Ensure that the Employee label comes before the Date and tick the Over and Under fields as well.
9. Now, add the stacked bar chart. Click the Insert tab and click on the “Recommended Charts”. Select the stacked column chart as shown.



10. Click Ok and you should get this result as shown in the image.



Row Labels	Sum of Over	Sum of Under
Jim	35.6	6.4
1/14/2013	0	6
1/2/2013	0	5
1/5/2013	3.6	6.4
1/8/2013	0	5
1/9/2013	0	5
1/11/2013	0	3
1/12/2013	1.6	6.4
Sherry	44.6	6.4
1/1/2013	0	6
1/2/2013	0	6
1/4/2013	4.6	6.4
1/5/2013	0	4
1/6/2013	0.6	6.4
1/9/2013	0	2
1/10/2013	0	5
1/11/2013	0	5
1/12/2013	0	5
Grand Total	86.6	6.4

-End of Exercises-

Project Discussion 1

WORKSHOP FOR NAVY

14 DECEMBER 2023

Learning Outcomes:

This session requires you to use what you have learnt for the past few days and apply it.

a) Manipulate fields and perform concatenation, extraction, sorting and removing of invalid data
b) Perform data validation and checking
c) Combining multiple spreadsheets
d) Using Pivot table for summary and reporting

Software(s): Microsoft Excel 2019

Instructions:

- You may use your very own datasets for this case study.
- Form groups of up to 4 people and discuss.
    - If your group decides to use your own dataset, then come up with a Problem Statement and perform at least 10 reports for your problem statement.
    - If you decide to use the dataset provided by us, then the scenario is:
        - You are working for a **Superstore** and your department looks after Hardware.
        - Your department will be giving the report to management for a mock listing in the exchange and certain figures are requested by your senior management.
    - Your team has discussed and decided that these reports are useful for management to look at.
        - Sales per product per year
        - Sales per region per year
        - Sales and quantity sold per customer segment per product sub-category per year
        - Profits per year (Organised by Product Category)
        - Profits per region per year
        - Profits per product sub-category per region per year (organised by product category)
        - Profits per region per year (organised by product sub-category)
        - Quantity sold per quarter per year per region
        - Profit per customer segment per region per product sub-category
        - Loss per product sub-category per region per year
- Your team has the afternoon to prepare this and each team will share with the rest your findings based on your chosen case study.

-End-

Day 3 - Introduction to VBA

WORKSHOP FOR NAVY

Learning Outcomes:

At the end of this session, you will be able to:

a) Use VBA to write simple Excel Programs

Software(s): Microsoft Excel 2019

Instructions:

- Download the datasets for this hands-on.
- Launch Microsoft Office Excel.
- Select a blank workbook and complete the following exercises.

Before we start on the Macro Code, we must get ourselves acquainted with the VBA environment (VBA IDE). It is this environment that you will be doing most of your VBA development and test it.

VBA Environment

VBA Integrated Development Environment (IDE) is the place where you, as a developer, can create and test the VBA codes. To start the VBA IDE, there are 2 commonly used ways:

- Going to the Developer tab in your Excel and click on the Visual Basic button
- By pressing the ALT+F11 in Excel application.

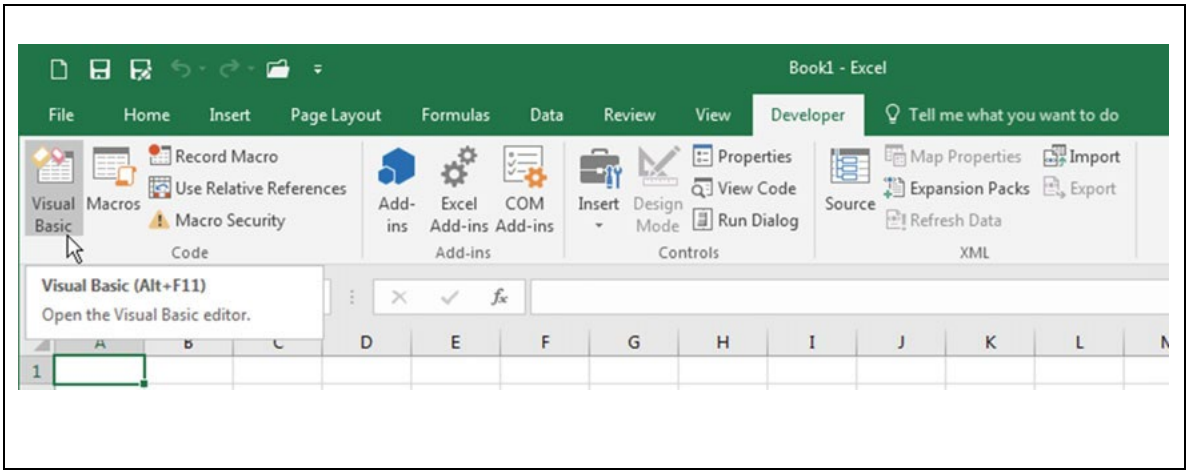


Diagram 1

To have the Developer tab made available, do the following:

1. Click on “Excel File Tab” and select “Options”. On the “Excel Options” dialog box left bar, select “Customize Ribbon” option.
2. On the right list box, check on “Developer” tab. And click “OK”.
3. The diagram (Diagram 2) below shows the “Excel Options” dialog box.

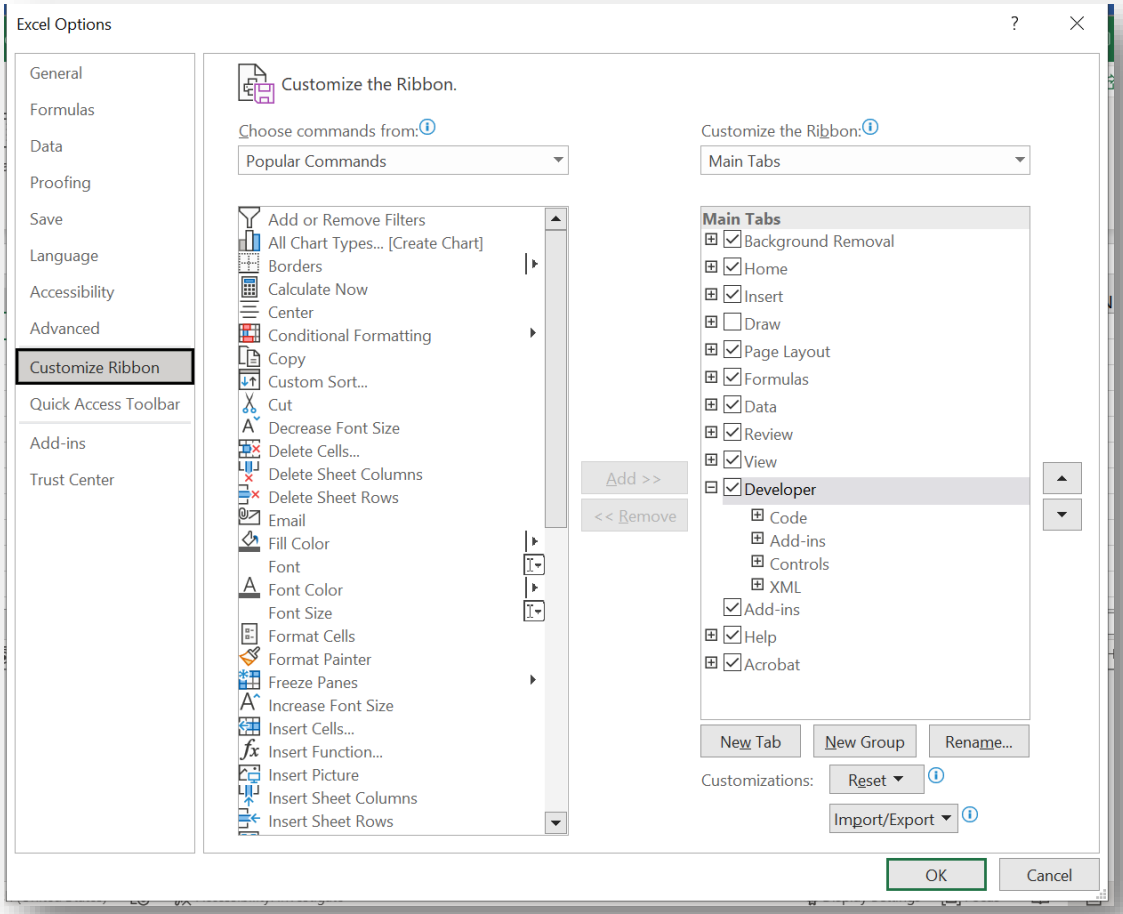


Diagram 2

The Macro Code in Excel can be sub-divided into 2 forms:

a) Excel Macros: This is the Visual Basic Application (VBA) code created by Excel when the Excel **macro recording** is activated by the user and execute any successive actions on its interface.
   - Recording a macro is like programming phone numbers into your house phone or your personal smartphone.
   - You would enter the phone manually first and then save it.
   - Then when you need it, you can redial the number using a 'shortcut' button.
   - In Excel, you can likewise do the same, you can record your actions in Excel when you perform them.
   - While recording, Excel will translate and store those keystrokes and mouse clicks to VBA codes, which you can play back time and time again.

b) VBA programming code.
   - The code that you created to implement the same automation but with more control and efficiency.

Try It! Create your first macro

In this simple exercise, we will record your first macro. To do that, you will need to find the macro recorder on the Developer tab.

The tab is usually hidden, and you will need to display it. To do so, please follow the steps:

1. Choose File -> Options.
2. In the Options dialog box, select "Customize Ribbon" fund on the left of the dialog box.
3. In the list box on the right, check on "Developer".
4. Click OK to return to Excel.
5. Now that you have the "Developer" tab showing in the Excel Ribbon, you can start up the macro recorder by selecting the "Record Macro" command found in the Code group on the Developer tab.
6. This activates the Record Macro dialog box, as shown in Diagram 3.1 below.

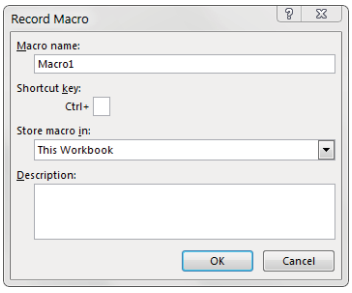
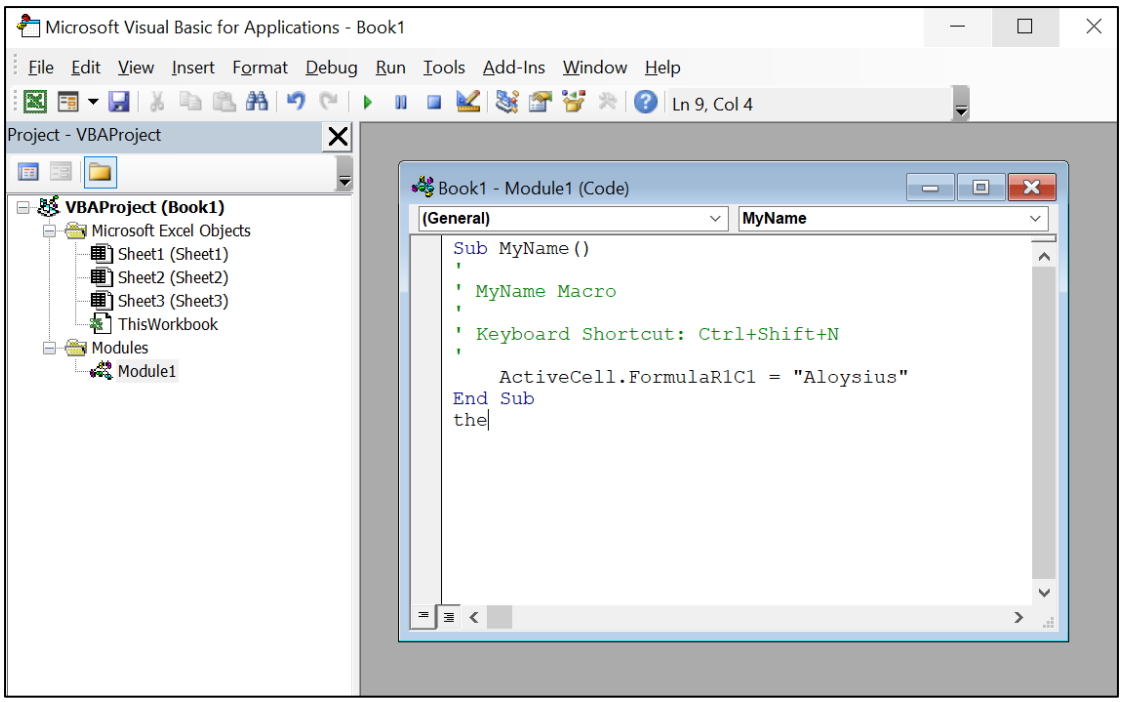


Diagram 3.1

7. Here are the four parts of the "Record Macro" dialog box
   - 7.1. **Macro Name -** Excel gives a default name to your macro, such as Macro1, but you should give your macro a name that is more descriptive of what it actually does. For example, you might want to name the macro that formats a generic table as FormatTable.
   - 7.2. **Shortcut Key**- Every macro needs an event, or something to happen, for it to run. This event can be a button press, a workbook opening, or, in this case, a combination of keystrokes. When you assign a shortcut key to your macro, entering that combination of keys will trigger your macro to run.
   - 7.3. **Store Macro In – "**This Workbook" is the default option. Storing your macro in "This Workbook" simply means that the macro is stored along with the active Excel file. So, the next time when you open this particular workbook, the macro is available to run. Similarly, if you send the workbook to another user, that user can run the macro as well (provided that the macro security is properly set by your user—more on that later in this chapter).
   - 7.4. **Description** This is an optional field, but it can come in handy if you have numerous macros in a workbook or if you need to give a user a more detailed description about what the macro does. The description is also useful for distinguishing one macro from another when you have multiple workbooks open or you have macros stored in the Personal Macro Workbook.
8. Now, that the Record Macro dialog box is open, you can start to create a simple macro that enters your name into a worksheet cell.
9. Follow the following steps:

9.1. Enter a new single-word name for the macro to replace the default Macro1 name.
9.2. Enter a good name for this example is “MyName”.
9.3. Assign the shortcut key Ctrl+Shift+N to this macro by entering uppercase **N** in the edit box labelled Shortcut Key.
9.4. Click OK to close the Record Macro dialog box and begin recording your actions.
9.5. Type your name into the active cell and press Enter.
9.6. Choose Developer -> Code -> Stop Recording. Alternatively, you can click the Stop Recording icon in the status bar (the square icon on left side of the status bar).

10. Now, examine the codes that was generated when the macro was recorded. Excel created a new module named Module1 and to view the code generated in Module1, you will have to activate the Visual Basic Editor by doing either of the action below:
    10.1. Press Alt+F11
    10.2. Choose Developer->Code->Visual Basic
11. In the VB Editor, there should be a VBA Project window. If it is not showing, go to View->Project Explorer and the menu bar will appear and select Modules. Double click on Module1 and the code that you recorded previously which is stored in Module1 will appear in the Code window.
12. The macro should look like this:

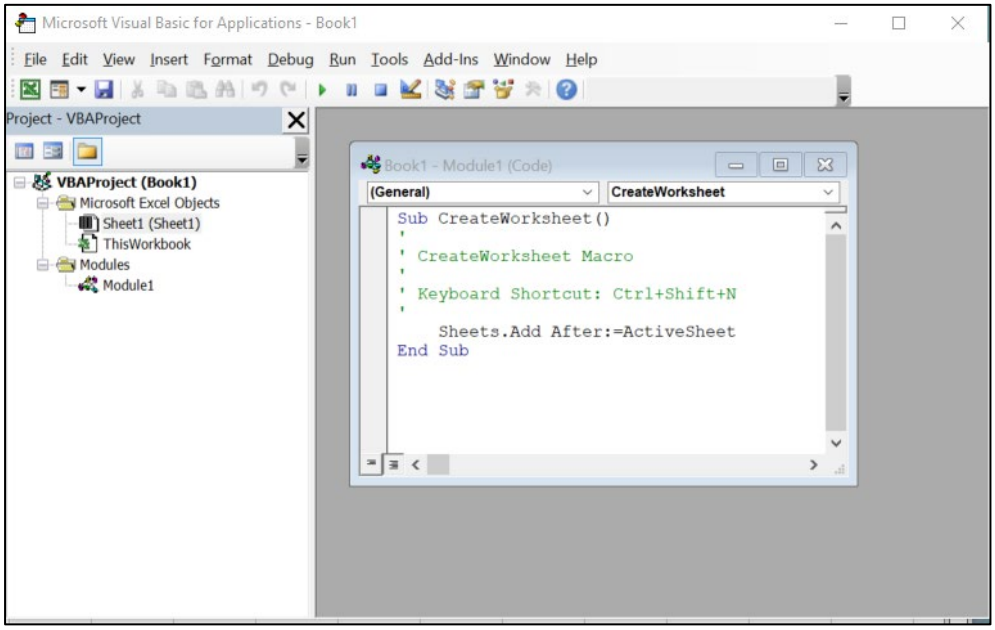


13. The macro recorded is a SUB procedure named MyName(). The first few lines below are comments inserted by Excel. Besides the comments, this Procedure only has one statement, and it is `ActiveCell.FormulaR1C1 = “Aloysius”`. This statement puts the name that was typed while recording into cell Row1 Column1, which was the active cell.
14. To start testing your simple macro, return to Excel either by:
    14.1. Pressing Alt+F11
    14.2. Click on View->Microsoft Excel in the VB Editor
    14.3. Ensure that the macro is running

15. When Excel is active, activate a worksheet. This can be in the same workbook or a new workbook.
16. Select a cell and type Ctrl+Shift+N, the shortcut key combination which you assign to the macro. The macro immediately enters your name into the active cell.

Exercise 1: Record and Test a Macro that creates a new Worksheet

1. Using the steps that was shown above to record a macro:
   1.1. Record a macro that will use the Ctrl+Shift+N to create a new worksheet.
2. Name the Macro CreateWorksheet.
3. The VBA Editor will show the following:



Assign a Macro

Try It! - Assign the Macro created in Exercise 1 to a Simple Button

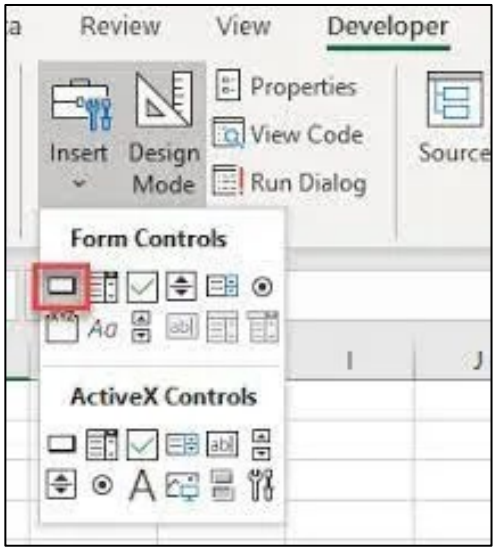
After you have created some macros, the next step would likely to be to run the macro with a click on a Button. Excel do offer a set of form controls for creating user interfaces on the spreadsheet.

Excel allows you to create form controls like Buttons to Scrollbars. The step is simple:

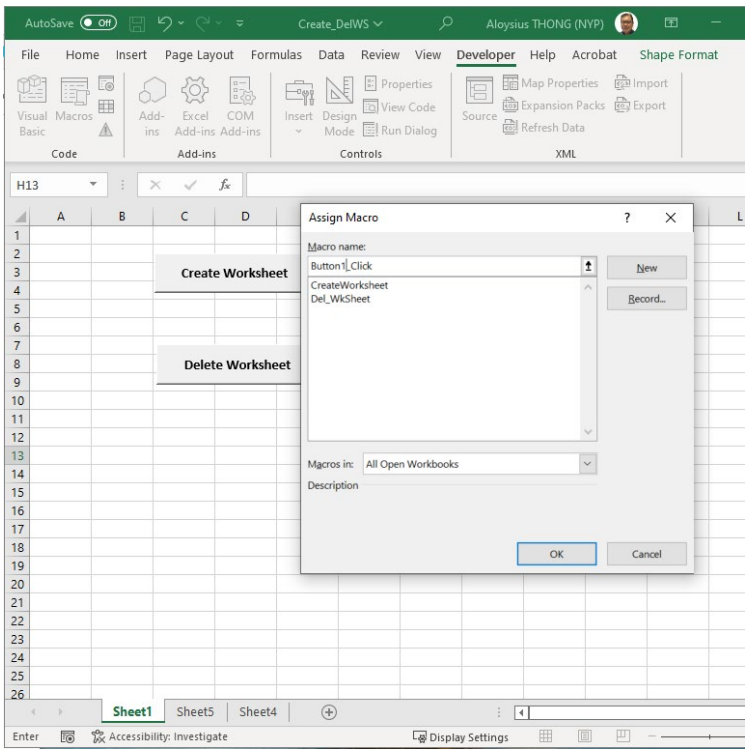
- Place the form control (like a button) on the spreadsheet
- Assign the macro to the form control
- When the macro is assigned to that form control, it will be executed (or played) when the control is clicked.

Exercise 2: Assigning a Macro to a Button

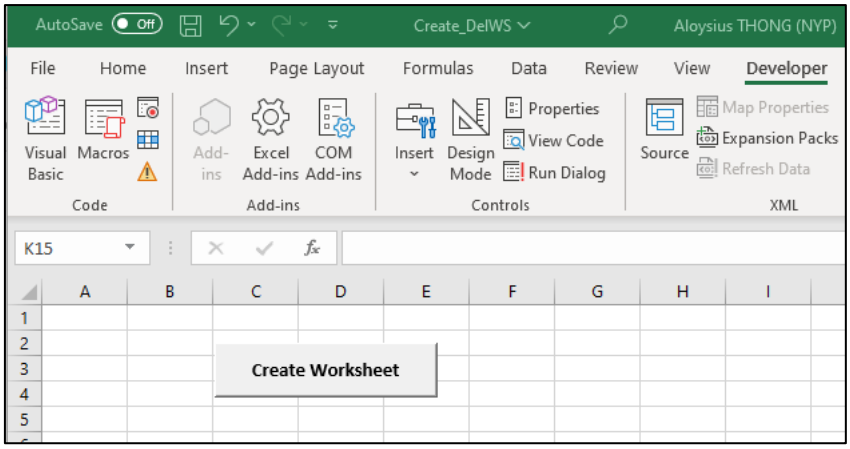
1. On the Developer tab select the Insert button. A dropdown list with groups of Form Controls and ActiveX Controls will show. On the Form Controls select the first control which happens to be the Button control as shown in the figure below.



2. Click the location on the Active sheet that you want to place the button
3. When you drop the button onto the spreadsheet, an Assign Macro dialog box appears, like the one shown below.



4. Select the macro that you want to assign to the button and then click OK.
5. Select the new button and right click and then change it to “Create Worksheet” and then enter.
6. It should look like this:

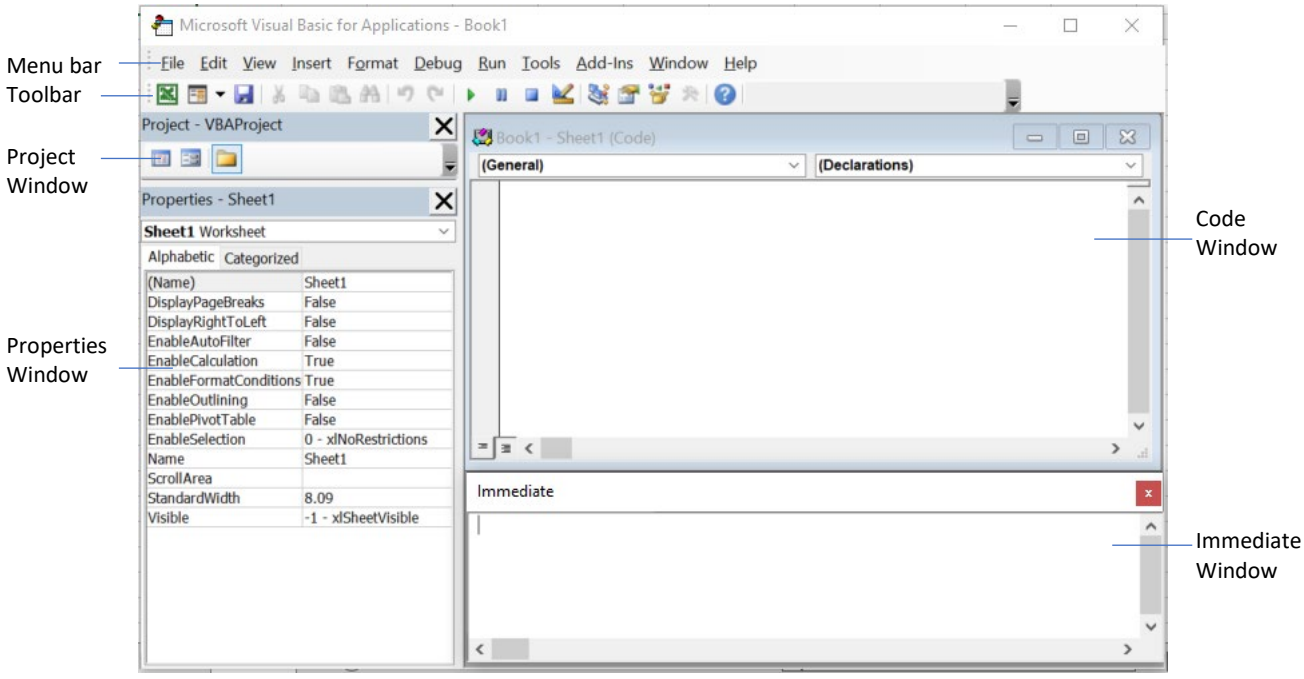


7. Test the newly created button and create a new worksheet by clicking on it.

Note
Notice the form controls and ActiveX controls in the Insert drop down list. Although they look similar, they're quite different. Form controls are designed specifically for use on a spreadsheet, and ActiveX controls are typically used on Excel user forms. As a rule, you should always use form controls when working on a spreadsheet. Form controls need less overhead, so they perform better, and configuring form controls is far easier than configuring their ActiveX counterparts.

The Visual Basic Editor (VBE)

The VBE is a separate application that runs when you start your Excel. To see the hidden VBE, you can press Alt+F11 and to return to Excel, press Alt+F11 again. The other way is to go to the Developer -> Code -> Visual Basic.



The VBE program above with some of its parts identified and yours might look differently. This is because it contains several windows and is highly customisable.

The VBE **menu bar** is like any other menu bar, and it contains commands that is useful for you. The **Project window** displays a tree diagram that shows every workbook currently open in Excel.

The Code window contains VBA codes and every and every "object" in a project has an associated Code window. To view an Object's code window, double the object in the Project window.

The Immediate window is most useful for executing individual VBA statements and for debugging your code.

Finally, the Project window – in Excel, each workbook, and add-ins that's open is a project. To expand the project, click on the '+' sign and contract a project by clicking the '-' sign. Every project expands to show at least one node called the Microsoft Excel Object. This node expands to show an item for each sheet in the workbook (each sheet is considered an object), and another object called ThisWorkbook (which represents the Workbook object). If the project has any VBA modules, the project listing also shows a Modules node.

When a macro is recorded, Excel automatically inserts a VBA module to hold the code that was recorded. Generally, a VBA module will have the following:

- Declarations
  Statements that declare the type of data for the variables
- Sub Procedures
  A set of programming instructions that will perform some action
- Function Procedures
  A set of programming instructions that will return a single value

A single VBA module can store any number of Sub procedures, Function procedures and declarations. To add a new VBA module, just select the project's name in the Project window and choose Insert -> Module. It will be added to the Modules folder in the Project window. To remove, just select the Module and do a right click Remove.

Code Window

Working with the code window is necessary as the Macros that you have recorded can be edited in this window. Furthermore, you can also create new modules and you need to write them in the code window. You can have multiple Code Windows arranged the way you like it by selecting Window-> Tile Vertically etc.

As you know, we can get the code in the code window by recording our actions and converting it to VBA code using the Macro Recorder. Another common method is to code it directly or even to copy the code from one module and paste it to another.

Even though you can enter the code for a single statement and ake it as long as possible, it is always good practice to use the underscore (_) to signal that the next line is a continuation. Below is an example:

```
Selection.Sort Key1:=Range("A1"), _
    Order1:=xlAscending, Header:=xlGuess, _
    Orientation:=xlTopToBottom
```

Notice that the codes are also indented for readability purposes.

Try It! Entering code into Code Window directly

1. Create a new workbook in Excel.
2. Press Alt+F11 to activate the VBE.
3. Click the new workbook's name in the Project window.
4. Choose Insert -> Module to insert a VBA module into the project.
5. Type the following code into the module:

```
Sub GuessName()
Dim Msg as String
Dim Ans As Long
Msg = "Is your name " & Application.UserName & "?"
Ans = MsgBox(Msg, vbYesNo)
If Ans = vbNo Then MsgBox "Oh, never mind."
If Ans = vbYes Then MsgBox "I must be clairvoyant!"
End Sub
```

6. Make sure that the cursor is located anywhere within the text you typed, and press F5 to execute the procedure.
7. This simple Macro uses the following concepts:
    a. Defining a Sub procedure (in line 1)
    b. Declaration of variables (in the Dim Statement)
    c. Assigning of values to the variables
    d. Joining the strings of text using the '&' operator
    e. Using the MsgBox, an in-built VBA fundction
    f. Using built-in VBA constants (vbYes, vbNo and vbYesNo)
    g. Using the If-Then constructs
    h. Ending the Sub procedure (End Sub)
8. The VBA environment can be customized. When the VBE is active, choose Tools->Options and the Options dialog box appears. There are 5 tabs, namely Editor, Editor Format, General and Docking. The default settings for all these tabs are normally used.

VBA Fundamentals

Understanding Objects

Under the Object-Oriented concept of Programming, everything is treated as objects. For example, a School is treated as an object, each student in the school is treated as an object and each subject taken by the student is also treated as an object.

Hence, in Excel, the Application object is the encompassing object, which is similar to the 'School' in our quoted example. Inside the Application object, Excel has a workbook,

and inside the workbook is the worksheet. Inside the worksheet is the range – and these objects are strung together in a hierarchical structure. You can point to a specific object, like the cell A1 on Sheet 1, in the following manner:

Application.ThisWorkbook.Sheets("Sheet1").Range("A1").Select

But since in most cases, the object model hierarchy is understood, you need not type every level. Excel will infer that your intention is for the active workbook and the active worksheet, and the code becomes:

Range("A1").Select

Furthermore, if your cursor is already in the cell A1, you can use *ActiveCell* object and leave out the need to spell out the range:

ActiveCell.Select

Collections

Many of Excel's objects belong to collections. For example, the School is located in a District and the District is in a collection of districts that is located in a City.

In similar explanation, in each Workbook object, there are a collection of Worksheets. This Worksheets collection is therefore an object that can be called upon in VBA. Each worksheet in the Workbook lives in the Worksheets collection, so to say. Since this is so, you can therefore refer to the specific Worksheet by the position in the collection with its index number starting with 1 or by its name in quotation.

For example, if the Worksheet has been renamed to *MySheet,* and that Workbook has only 1 worksheet, then the 2 line of codes below refer to the same thing:

Worksheets(1).Select
Worksheets("MySheet").Select

If there are 2 Worksheets in the active Workbook and they have the names "MySheet" and "YourSheet" and in that order, you can refer to the second worksheet are:

Worksheets(2).Select
Worksheets("YourSheet).Select

To reference another Workbook that is not active, you need to qualify the worksheet and workbook reference. For instance:

Workbooks("MyData.xslx").Worksheets("MySheet").Select

Properties

Properties are the characteristics or attributes of the objects. Example, each school has a certain colour or size- hence properties of the worksheet object can be changed, example the sheet name, but some cannot like the Rows.Count property which is fixed. To assign a name to the worksheet, it can be:

Worksheets("Sheet1").Name = "MySheet"

Some properties, like the *Text* property is read-only.

Some properties, on the other hand, have arguments that further specify the property value. For instance, this line of code uses the RowAbsolute and ColumnAbsolute arguments to return the address of cell A1 as an absolute reference ($A$1):
MsgBox Range("A1").Address(RowAbsolute:=True, ColumnAbsolute:=True)

Note: The ":=" sets a value of a parameter, hence in this case, *RowAbsolute & ColumnAbsolute*, the 2 parameters of the *Range Address* method is set to True.

It's important to understand that only one workbook at a time can be active, and one worksheet in that workbook is active and one cell in that worksheet is the active cell even though a multicell range is selected. Furthermore, VBA lets you refer to these active objects in a simplified manner. In this instance, VBA provides the properties of the Application object, for example the ActiveCell property of the Application object refers to the active cell. The code *ActiveCell.Value = 1,* would mean that the active cell is being assigned with the value of 1.

However, if the range is selected, the active cell is still a single and never a multicell range. Another important property of the Application object would be the *Selection* property, in which it returns the reference to whatever is being selected – namely a single cell, a range of cells, or an object such as *ChartObject, TextObject* or even *Shape*.

Some Useful Properties of the Application Object

Property	Object Returned
ActiveCell	The active cell.
ActiveChart	The active chart sheet or chart contained in a *ChartObject* on a work- sheet. This property is Nothingif a chart isn't active.
ActiveSheet	The active sheet (worksheet or chart sheet).
ActiveWindow	The active window.
ActiveWorkbook	The active workbook.
Selection	The object selected. It could be a *Range object, Shape, ChartObject,* and so on.
ThisWorkbook	The workbook that contains the VBA procedure being executed. This object may or may not be the same as the *ActiveWorkbook* object.

Methods

Methods are the actions that can be performed with an object. For example, *Select* is a method in the Range object that selects a range of cells. For a comprehensive listing of methods for each object, go to:

https://learn.microsoft.com/en-us/office/vba/api/overview/excel/object-model .
The objects are listed in the left panel of this page and selecting the object will give you the description and the methods and properties that you might need for that particular object.
Hence, the essential point to remember is that Objects have unique properties and methods, and each object has its own set of properties and methods. Some properties like Name are common with various objects and likewise some methods like Delete is common with the other various objects.

VBA Programming Fundamentals

Comments

Comments are lines of descriptive text which is not part of the executed code. It is a good practice to use comments to describe what you are doing so that it helps the next programmer maintaining your code to understand your intentions.

A comment can be a complete line, or it can be inserted after an instruction code on the same line. A comment is indicated by an apostrophe. When VBA sees an apostrophe, it ignores any text that follows it – except when the apostrophe is contained within quotation marks. Hence, the code below doesn't contain a comment:

```
Msg = "This line isn't a comment"
```

Incidentally, the following example shows a VBA procedure with 3 comments:

```
Sub CommentDemo()
'       This procedure does nothing of value
        x = 0     'x represents nothingness
'       Display the result
        MsgBox x
End Sub
```

Variables, Data Types and Constants

A variable is a named storage location in the computer's memory, and it can store various data types, such as Boolean values of True or False, large values, text as well as double precision values. The variable is assigned the value using the equal sign operator. The variable name should be made as descriptive as possible. However, you must note that there are certain rules that apply to what is allowable in those names:

- Alphabet characters, numbers and some punctuation characters are allowed but the first character must be an alphabet
- VBA does not distinguish upper case and lower case. To make it more readable, programmers would use mix case like DateOfBirth instead of Dateofbirth.
- No spaces or periods allowed and in place you can use underscore (_) instead like Date_of_birth.
- No special characters are allowed (#, $, %, !)
- Length of a variable name is 254 characters at max

Data Types

Data type refers how data is stored, that is either as integer, real numbers or strings. VBA takes care of data typing automatically it is still a good practice declare a variable with the explicit data type.

An example is shown below:

```
Option Explicit        ' Force explicit variable declaration.
Dim MyVar              ' Declare variable.
MyInt = 10             ' Undeclared variable that will generate an error.
MyVar = 10             ' Declared variable does not generate error.
```

Local variable

A local variable is one that is declared within a procedure and can only be used in the procedure that they are declared in. When the procedure ends, it no longer exists and the memory is freed up. If you need the value outside of that procedure, then declare it as *Static* instead.

The most common way to declare a local variable is to place the *Dim* statement between the Sub statement and the End statement. You can also include several variables with a single DIM statement. For example:

```
Sub MySub()
        Dim X As Integer
        Dim Y As Integer
        Dim Z As Integer
        Dim First As Long
        Dim Last As Double
        Dim MyValue                ' This variable MyValue is a variant
'       Procedure's code goes here
End Sub
```

Can also be declared in this manner:

```
Sub MySub()
        Dim X As Integer, Y As Integer, Z As Integer
        Dim First As Long, Last As Double, MyValue  'MyValue is a variant
'       Procedure's code goes here
End Sub
```

About Variant Data Type

If a variable is not declared with its data type, VBA will use the default data type, Variant. Data stored as a Variant actually acts like a chameleon: it changes type, depending on what you do with it.

However, Variant data type causes VBA to perform checks that are time consuming, and it reserves more memory than it is necessary. If VBA knows the data type, it does not need to check and investigate and instead reserve just enough memory to store the data.

Module-wide variables

If a variable is declared with a local scope, other procedures in the same module can use the same variable name but each instance is unique to its own procedure. There is another type called Module-wide variables which is available to all the procedures in the module. By putting the Dim statement as the first instruction in the module, the procedures following it in the module will have access to the variable, example:

```
Dim DateOfBirth as Date      'This is the variable that is accessible by Pro1 and Pro2
Sub Pro1 ()
' Code goes here
End Sub

Sub Pro2 ()
' Code goes here
End Sub
```

Public variables

Changing the *DIM* in the above example to *Public* will make the variable *DateOfBirth* available to all the procedures in all the VBA modules in the project. You must insert this statement, for example:

```
Public DateOfBirth as Date
```

before the first procedure in a module (any module). This type of declaration must appear in a standard VBA module, not in a code module for a sheet or a UserForm.

Static variables

To retain a value when the procedure ends, we use the *Static* variable which is declared at the procedure level. However, if the procedure is halted by an *End* statement (which is different from the *End Sub* statement), the static variable will still lose their values. Example:

```
Sub Pro3 ()
        Static Counter as Long
        ' Code goes here
End Sub
```

Declaring constants

We can declare constants for use when its value does not change. Example when we use constant like Pi (3.14159265359) in our procedure. If this value is needed only within a single procedure, then we declare it in this manner:

```
Sub Pro4 ()
        Const Pi as Double = 3.14159265359
        :
End Sub
```

However, if it is needed in all the modules in the workbook, we can use the *Public* keyword, as in:

```
Public Const Pi as Double = 3.14159265359
Sub Pro4 ()
:
End Sub
```

Strings

Strings are handled in VBA as a string of characters. There are 2 types, fixed-length strings that are declared with a specified number of characters of which the max length is 65,535 characters and variable-length strings which can hold up to a theoretic limit of 2 billion characters.

In VBA, they are declared in this manner:

```
Dim FixedString As String * 100      'FixedString is a string of length 100 characters
Dim VarString As String              'VarString is not a fixed length string
```

Date

VBA has a Date data type that uses 8 bytes of storage and can hold dates ranging from January 1, 0100 to December 31, 9999. The Date data type is also useful for storing time-related data. Dates and times are specified by enclosing them between 2 hash marks (#).

Examples of declaring Date data types:

```
Dim Today As Date
Dim StartTime as Date
Const FirstDay As Date = #1/1/2023#
Const Noon = #12:00:00#
```

Try It! – Working on Date and Time

1. Create a new workbook in Excel.
2. Press Alt+F11 to activate the VBE.
3. Click the new workbook's name in the Project window.
4. Choose Insert -> Module to insert a VBA module into the project.
5. Type the following code into the module:

```
Option Explicit

Sub FunWithDatesAndTimes()
   Debug.Print VBA.Date
   Debug.Print VBA.Time
   Debug.Print VBA.Now

   Debug.Print Date
   Debug.Print Time
   Debug.Print Now
End Sub

Sub CreateDate()
   Dim birthday As Date

   Range("A1").Value = "Samuel's Birthday :"
```

```
    birthday = DateSerial(1991, 4, 12)
    Range("B1").Value = birthday
End Sub
```

Make sure that the cursor is located anywhere within the text you typed, and press F5 to execute the procedure.

6. Run the 2 macros and examine the worksheet and the "Immediate Window"

Exercise 3: Writing More Date and Time Examples

1. Open a new Excel Workbook and select a blank worksheet.
2. Type Alt+F11 to launch the VBE.
3. In the Project>Sheet 1, do a right click and Insert a Module
4. Module1 is created and so is the Code Window. Ensure the Immediate Window is also present.
5. In the Code Window type in the following codes:

```
Option Explicit

Sub MoreFunWithDateAndTime()
    Debug.Print Weekday(Now)
    Debug.Print WeekdayName(Weekday(Now))the
    Debug.Print MonthName(Month(Now))
End Sub
```

6. Run the codes.
7. Examine the results in the Immediate Window and explain what is Weekday(Now) result?
8. Now in the Code Window, add after the Sub MoreFunWithDateAndTime () statement the following codes:

```
Debug.Print Year(Now)
Debug.Print Month(Now)
Debug.Print Day(Now)
Debug.Print Hour(Now)
Debug.Print Minute(Now)
Debug.Print Second(Now)
```

9. Again, what does this new code give and explain the results.

Assignment Statements

We will now move out of Data Types and start our discussion on assignment statements before working on the exercises. Assignment statements are the most basic in a programming language as it is used all the time.

Assignment statement evaluates an expression and assigns a result to a variable or to an object. It can be a combination of keywords, operators, variables and constants that will yield a string, number or object. An expression can perform a calculation, manipulate characters or test data.

Assignment operator is a "=" sign and when in a comparison situation, it is used as an equal operator.

Exercise 4: Assignment statements & Comparisons

1. Open a new Excel Workbook and select a blank worksheet.
2. Key in the following content in Sheet1:

	A	B
1	**Student**	**Test Score**
2	Abe	62
3		66
4		90
5		89
6	Johnny	99
7	Johnny	88
8	Johnny	97
9		79
10		78
11	Mary	77
12		96
13		57
14		69
15		70
16		88
17	Sue	66
18	Sue	68
19		51
20		58
21		72

3. Type Alt+F11 to launch the VBE.
4. In the Project>Sheet 1, do a right click and Insert a Module
5. Module1 is created and so is the Code Window. Ensure the Immediate Window is also present.
6. In the Code Window type in the following codes:

```
Option Explicit

Sub WriteNames()
   Dim LastRow As Long
   ' Find the last non-empty row where Col B is non-empty
   LastRow = Cells(Rows.Count, 2).End(xlUp).Row
   '  Debug.Print (LastRow)
   Dim i As Long

   For i = 2 To LastRow
      If Cells(i, 1).Value = "" Then
         '  Cells(i, 1).Value = Cells(i - 1, 1).Value
         Cells(i, 1).FormulaR1C1 = "=R[-1]C"
      End If
   Next i
End Sub
```

7. Run the codes.
8. Examine the results in the Worksheet 'Sheet1'.
9. With your partner(s), discuss the following:
    a. Where is the "=" sign an assignment sign and where it is an equal sign?
    b. What is the effect of this statement?
        LastRow = Cells(Rows.Count, 2).End(xlUp).Row
    c. What is the equivalent statement of this: *Cells(i, 1).FormulaR1C1 = "=R[-1]C"?*
    d. What does statement in (c.) do?
10. This ends our exercise on assignment and comparison.

Arrays

An array is a group of elements of the same data type and have a common name. To get the content in the array, we refer to the name of the array and the index number of that array. For example, if the array name is CalendarMonths, you can refer to the first element of the array CalendarMonths(0), second element as CalendarMonths(1), and so on. To declare this as an array, we can declare it as:

Dim CalendarMonths (11) As String

This is where the first element (or January) is in CalendarMonths (0). If you want it to be more intuitive, then declare it as:

Dim CalendarMonths (1 to 12) As String

If you want to use only the upper bound when you declare and want to start from 1, then include this statement before any procedures in your module:

Option Base 1

Then declare the array as:

Dim CalendarMonths (12) As String

To declare a 2D Array, you can think it as Row x Columns, then the declaration is:

Dim MyArray (10, 10) As Integer

This means MyArray has 10 Rows and 10 Columns (a matrix), and the assignment of value is, for example:

MyArray (3,4) = 100

Exercise 5: Accessing the Contents of an Array

1. In this exercise, we will declare an Array and initialise it with some contents first.
2. Open a new Excel Workbook and select a blank worksheet.
3. Key in the following content in Sheet1:

	A	B
1	What is the number from 1-12 that you have in mind?	
2	The corresponding color is:	
3		

4. In cell “B2”, type a number that you have in mind. This number must be a value between 1 to 12 inclusive.
5. Type Alt+F11 to launch the VBE.
6. In the Project>Sheet 1, do a right click and Insert a Module
7. Module1 is created and so is the Code Window. Ensure the Immediate Window is also present.
8. In the Code Window type in the following codes:

```
Option Explicit

Sub MyNumColorArray()
  Dim NumColor(1 To 12), Response As String
  ‘ Dim Title As String
  ‘ Dim Val As Integer

  NumColor(1) = "Black"
  NumColor(2) = "Red"
  NumColor(3) = "White"
  NumColor(4) = "Green"
  NumColor(5) = "Brown"
  NumColor(6) = "Blue"
  NumColor(7) = "Orange"
  NumColor(8) = "Yellow"
  NumColor(9) = "Purple"
  NumColor(10) = "Gray"
  NumColor(11) = "Pink"
  NumColor(12) = "Tan"


  Response = Sheets("Sheet1").Range("B1").Value
  ‘ Title = "You have selected:"
  ' Val = MsgBox(NumColor(Response), vbOK, Title)
  Sheets("Sheet1").Range("B2").Value = NumColor(Response)

End Sub
```

9. Run the codes.
10. Examine the results in the worksheet ‘Sheet1’ cell “B2”.
11. Discuss with your partner how did the result come about.

Object Variables

An object variable is one that represents an entire object, such as a Range or Worksheet. Object variables, just like any other variables, are declared with *Dim* or *Private* or *Public*. Hence, for an example, MyCell is an object variable of Range and is declared in the following manner:

```
Dim MyCell As Range
```

By doing this, it simplifies the coding as compared to the following codes that does not use object variable.

```
Sub NoObjVarEg()
    Worksheets("Sheet1").Range("A1").Value = 64
    Worksheets("Sheet1").Range("A1").Font.Bold = True
    Worksheets("Sheet1").Range("A1").Font.Italic = True
    Worksheets("Sheet1").Range("A1").Font.Size = 12
    Worksheets("Sheet1").Range("A1").Font.Name = "Calibri"
End Sub
```

By using object variable, the code is simplified to the following:

```
Sub ObjVar()
    Dim MyCell As Range

    Set MyCell = Worksheets("Sheet1").Range("A1")
    MyCell.Value = 64
    MyCell.Font.Bold = True
    MyCell.Font.Italic = True
    MyCell.Font.Size = 12
    MyCell.Font.Name = "Calibri"
End Sub
```

On declaring MyCell as a a Range object, the Set statement assigns an object to it. Hence, subsequent statements following that can use MyCell object.

User-Defined Data Types

You can create custom data types or commonly known as user-defined data types. For example, if your application deals with student information, then you may want to create a user-defined data type called *StudentInfo*.

To do that, we must define it in this manner:

```
Type StudentInfo
    StudName As String
    AdminID As String
    PostalCode As Long
    StudyStage As Integer
End Type
```

To declare a variable of type StudentInfo, we will do the following:

```
Dim Students(1 To 100) As StudentInfo
```

In this way, each of the 100 elements in the array can be referred to in the following manner. For example, the first student can be referred as:

```
Students(1).StudName = “Johnny Lim”
Students(1).AdminID = “2312345Z”
Students(1).PostalCode = 581346
Students(1).StudyStage = 3
```

Manipulating Objects

There are 2 important constructs that can help to simplify working with objects and collections. The 2 constructs are:

- *With-End With* construct
  You would use the With-End With construct in VBA Excel when you want to repeatedly reference a single object or group of objects. This allows you to perform multiple actions on the same object(s) without having to repeat the object reference over and over again, making your code more concise and easier to read.

  Example:

```
Sub ChangeFont()
    With Selection.Font
        .Name = "Calibri"
        .Bold = True
        .Italic = True
        .Size = 11
        .Underline = xlUnderlineStyleSingle
        .ThemeColor = xlThemeColorAccent1
    End With
End Sub
```

- *For Each-Next*
  The For Each-Next construct, on the other hand, is used when you want to loop through a collection or array of objects. By using this construct, you can perform the same action on each object in the collection, without having to write code for each individual object individually. This can be a much more efficient way of processing a large number of objects.

  Example:

```
Sub ExampleForLoop()
    Dim myRange As Range
    Set myRange = Range("A1:A10")

    Dim myCell As Range
    For Each myCell In myRange
        myCell.Value = myCell.Value * 2
    Next myCell
End Sub
```

  In this example, we have a range of cells A1:A10. We use the For Each-Next construct to loop through each individual cell in that range and multiply its value by 2. The loop continues until all cells in the range have been modified.

Control Structures

Control Structures are used to control the flow of the execution in VBA Excel program. These constructs are listed below:

- For-Next loops

```
Sub SumSquareRoots()
        Dim Sum As Double
        Dim Count As Integer
        Sum = 0
        For Count = 1 To 100
                Sum = Sum + Sqr(Count)
        Next Count
        MsgBox Sum
End Sub
```

In this example, Count (the loop counter variable) starts out as 1 and increases by 1 each time the loop repeats. The Sum variable simply accumulates the square roots of each value of Count.

- Do While loops

```
Sub DoWhileLoop()
        Dim i As Integer
        i = 1
        Do While i <= 10
                MsgBox "The value of i is " & i
                i = i + 1
        Loop
End Sub
```

- Do Until loops

```
Sub DoUntilLoop()
        Dim i As Integer
        i = 1
        Do Until i > 10
                MsgBox "The value of i is " & i
                i = i + 1
        Loop
End Sub
```

In the examples above, the Do While and Do Until loops are used to execute a block of code repeatedly while a certain condition is true.

- If-Then constructs

```
Sub PassFailGrading()
        Dim Score As Integer
        Dim Grade As String
        If Score >= 50 Then
                Grade = "P"
        Else
                Grade = "F"
        End If
End Sub
```

In this example, Scores of 50 or more is a Pass grade whie anything below 50 is a Fail grade.

- GoTo statement
GoTo statements are normally used for Error Handling

```
Sub ErrorHandler()
        On Error GoTo ErrorHandler

        ' some code here that may produce an error

        Exit Sub

        ErrorHandler:

        ' handle the error here
        MsgBox "An error occurred: " & Err.Description

        ' redirect code execution to the proper location
        GoTo ExitPoint

        ExitPoint:

        'resume code execution
End Sub
```

In this example, the code has an error handling section that redirects execution to a specific label (here called `ExitPoint`) using the `GoTo` statement. This allows the program to continue execution from that point onwards, rather than just terminating the program. The `ExitPoint` section of the code is where the program resumes execution if the error has been handled properly.

Incidentally, GoTo statements are not only used in Erro Handling. In fact, it is used in programming to transfer control to a different section of code or to jump backwards and forwards within a program. However, in modern programming languages, the use of GoTo statements is discouraged because it can make code hard to follow and debug. Instead, programming languages provide structured control flow statements

like if/else statements, loops, and functions, which make code easier to read and maintain. These structured control flow statements are preferred over GoTo statements as they make code more readable, maintainable, and less error-prone.

- Select Case constructs

```
Sub SelectCaseStatement()
    Dim score As Integer
    score = 75
    Select Case score
        Case Is >= 85
            MsgBox "Your grade is an A"
        Case Is >= 70
            MsgBox "Your grade is a B"
        Case Is >= 60
            MsgBox "Your grade is a C"
        Case Is >= 50
            MsgBox "Your grade is a D"
        Case Else
            MsgBox "Your grade is an F"
    End Select
End Sub
```

The Select Case statement is used to check a variable against multiple conditions and execute a block of code based on the condition that is met.

These constructs allow you to create conditional statements, loops, and other programmatic constructs that help you automate tasks, manage data, and make your code more efficient.

MORE EXERCISES

Exercise 6.1: String Manipulation Part 1

1. In this exercise, we will attempt with some simple built-in string manipulation functions in VBA. The functions that we are working on are:
    a. To Lower Case (LCase())
    b. To Upper Case (UCase())
    c. String Length (Len())
    d. Trim spaces at the beginning of the sentence and after the last character in the sentence (Trim())
    e. Finding the position of a given substring within a string (InStr())
2. Open a new Excel Workbook and select a blank worksheet.
3. In the following cells in the worksheet, key in the following:
    a. Cell A1: “HELLO”
    b. Cell B1: “hello”
    c. Cell C1: “My number is 555-555-5555”
4. Type Alt+F11 to launch the VBE.
5. In the Project>Sheet 1, do a right click and Insert a Module

6. Module1 is created and so is the Code Window. Ensure the Immediate Window is also present.
7. In the Code Window type in the following codes:

```
Option Explicit

Sub StrFxns_1()
   Debug.Print LCase(Range("A1").Value)
   Debug.Print UCase(Range("B1").Value)
   Debug.Print Len("Excel ")
   Debug.Print Trim("     So much random space      ")
   Debug.Print InStr(18, Range("C1").Value, "-")
End Sub
```

8. Run the codes.
9. Examine the results in the Immediate Window and discuss the result.

10. In the worksheet cell D1, key in “STRAW”.
11. Let’s continue with some more string manipulation exercise:
    a. Find the 2 characters from the left of the string
    b. Find the 9 characters from the right of the string.
    c. Extract the middle part of the string from the full string. In this case from 5th position 3 characters into the string.
    d. Reverse the string
    e. Replace he string with the character “!”.
12. Append the following codes in the same Module 1:

```
Sub StrFxns_2()
   Debug.Print Left("555-555-5555", 2)
   Debug.Print Right("Some Address, SG 538761", 9)
   Debug.Print Mid("Mr. Excel", 5, 3)

   Debug.Print StrReverse(Range("D1").Value)
   Debug.Print Replace("555 555 5555", "5", "!")
End Sub
```

13. Run the codes
14. Examine the results in the Immediate Window and discuss.
15. This ends our first exercise on String Manipulation.

Exercise 6.2: String Manipulation Part 2

1. In this exercise, we will be using the Split(). This string function is used to split strings into multiple substrings based on a delimiter provided to the function and a comparison method.
2. We will be using Split() in 3 scenarios, namely:
    a. Splitting the given string into words
    b. Counting the number of words in a string
    c. Splitting a string using a delimiter other than space
3. Open the same Excel Workbook as in Exercise 6.1 and select Sheet2.
4. In the following cells in the worksheet, key in the following:

a. Cell A1: “The big brown fox jumps over the lazy dog”
b. Cell A2: “The big brown fox jumps over the lazy dog ”
c. Cell A3: “100/55/35”

5. Type Alt+F11 to launch the VBE.
6. In the Project>Sheet2, do a right click and Insert a Module
7. Module2 is created and so is the Code Window. Ensure the Immediate Window is also present.
8. In the Code Window type in the following codes:

```
Option Explicit
Sub SplitToWords()
  Dim Results() As String
  Dim Index As Integer
  Results = Split(Sheets("Sheet2").Range("A1").Value, " ")
  For Index = 0 To UBound(Results())
    Debug.Print (Results(Index))
    Next Index
End Sub
```

9. This procedure splits the string into words.
10. The UBound() actually gives us the upper bound of the array Results.
11. Run the codes and observe if this is happening.
12. Examine the results in the Immediate Window and discuss the result.

13. Add in the next procedure that counts the words in a string by appending the following codes in Module 2:

```
Sub WordCnt()
  Dim Results() As String
  Dim Counter As Integer
  Results = Split(Sheets("Sheet2").Range("A2").Value, " ")
  Counter = UBound(Results()) + 1
  Debug.Print ("The number of words in the sentence is: " & Counter)
End Sub
```

14. As in the previous procedure, the UBound() gives us the upper bound of Results and since the array starts from zero, adding 1 to the upper bound will give us the length.
15. Run the codes
16. Examine the results in the Immediate Window and discuss.
17. The next procedure uses a different delimiter from the previous 2 examples and in this case, the delimiter is “XX”. Note that it can be any character or string.
18. The codes for this new procedure will be:

```
Sub SplitUseDelim()
  Dim Results() As String
  Results = Split(Sheets("Sheet2").Range("A3").Value, "XX")
  Range("B3").Value = Results(0)
  Range("C3").Value = Results(1)
  Range("D3").Value = Results(2)
End Sub
```

19. Run the codes
20. Examine the results in the Immediate Window and discuss.

Exercise 6.3: String Manipulation Part 3

1. In this exercise, we will see how we can concatenate 2 strings (Sub JoinStr()) and how we can compare 2 strings (Sub Compare2Str()) using VBA's StrComp().
2. Open the same Excel Workbook as in Exercise 6.1 and 6.2 and select Sheet3.
3. In the Project>Sheet3, do a right click and Insert a Module
4. Module3 is created and so is the Code Window. Ensure the Immediate Window is also present.
5. In the Code Window type in the following codes:

```
Sub JoinStr()
Dim text1 As String, text2 As String
text1 = "Hi"
text2 = "Tim"

Debug.Print (text1 & " " & text2)
End Sub
```

6. Run the above codes.
7. Examine the result in the Immediate Window.
8. Now, add in the next procedure that compares 2 strings.
9. In the following cells in Sheet3, key in the following:
    a. Cell A1: "The rain in Spain"
    b. Cell B1: "THe raIN In SPAIN"
10. Select Module3 and add in the following codes:

```
Sub Compare2Str()
Dim str1, str2 As String
Dim CompareResult As Integer

str1 = Sheet3.Range("A1").Value
str2 = Sheet3.Range("B1").Value
MsgBox ("str1= " & str1 & vbNewLine & "str2= " & str2)
CompareResult = StrComp(str1, str2, vbTextCompare)
If (CompareResult = 0) Then
    Debug.Print ("The 2 strings are the same")
  Else
    Debug.Print ("The 2 strings are different")
  End If
End Sub
```

11. Run the codes.
12. Examine the results, check the MsgBox and discuss the outcome.
13. You have now completed the exercises for String Manipulation.

Exercise 7: Arrays, Ranges and Formatting Tables

1. In this exercise, we will combine the skills we learnt using For Next, With End-With, calling a Sub in another Module and passing in arguments and manipulating Arrays.
2. In a new workbook, key in the following in the Sheet1.

	A	B	C	D	E
1					
2		Sample Data Set			
3					
4		**Product Name**	**Units**	**Unit Price**	**Total Price**
5		Watch	1290	25999	33538710
6		Computer	670	80000	53600000
7		Mobile Phone	542	35000	18970000
8		Digital Camera	430	75999	32679570
9		ECG Machine	290	99999	28999710
10		Printer	3500	7000	24500000
11					
12					

3. Rename Sheet1 to “Sample Data Set”
4. Add Sheet2 and rename it to “VBA Code Results”.
5. Go to “Sample Data Set” Sheet and select B2:E10 and copy.
6. Go back to “VBA Results” and paste the copied cells to the same area, ie B2:E10.
7. Type Alt+F11 and select Sheet2 (VBA Code Results) and insert a Module.
8. Insert the following codes in Module 1:

```
Sub Using_Arrays()

'Declaring variable
Dim arr As Variant
Dim i As Long

'Calling our activesheet
arr = ActiveSheet.Range("B4").CurrentRegion

'Returns the following values for an array
For i = LBound(arr, 1) To UBound(arr, 1)
Next i

'Drawing the table by calling the sub in Module 2
Module2.Format_Table ("H4:K10")

'Showing the values in the activesheet
ActiveSheet.Range("H4:K10").Value = arr

End Sub
```

9. Select “Sheet 2 (VBA Code Results)” again and insert Module 2.

10. Insert the following codes into Module 2:

```
Sub Format_Table(rangeFrMod_1 As String)
Dim nr$, cw(), i%                          '#1

nr = rangeFrMod_1
Range(nr).ClearFormats
ReDim cw(1 To Range(nr).Columns.Count)
For i = 1 To UBound(cw)
   cw(i) = Range(nr).Columns(i).ColumnWidth
Next
With ActiveSheet                           '#2
   .ListObjects.Add(xlSrcRange, Range(nr), , xlYes).Name = "Table_" & nr
   .ListObjects("Table_" & nr).TableStyle = "TableStyleMedium2"
   .ListObjects("Table_" & nr).Unlist
End With
For i = 1 To UBound(cw)
   Range(nr).Columns(i).ColumnWidth = cw(i)
Next
With Range(nr)
   .Rows(1).Font.Bold = False
   .Rows(1).HorizontalAlignment = xlCenter
   .Range("a1") = ""
   .Columns(1).Font.Name = "Century"   ' choose a font here
End With

End Sub
```

11. Go back to Module 1 and Run the codes.
12. Examine the results and discuss.

13. Some explanations on the codes that were typed in #1 and #2:

> ***Notes (#1):*** *On the statement* ***Dim nr$, cw(), i%***
>
> *In VBA, `Dim` is used to declare a variable, `nr$` is a string variable, `cw()` is a dynamic array, and `i%` is an integer variable. The `$` and `%` characters after variable names are used for declaring the variable data type. `$` represents a string data type, while `%` represents an integer data type. `cw()` declares an empty array, where the size of the array can be defined later in the code. Overall, this line of code is declaring three variables. A string variable called `nr`, an empty dynamic array called `cw`, and an integer variable called `i`.*

Notes (#2):

ActiveSheet
.ListObjects.Add(xlSrcRange, Range(nr), , xlYes).Name = "Table_" & nr
.ListObjects("Table_" & nr).TableStyle = "TableStyleMedium2"
.ListObjects("Table_" & nr).Unlist

This VBA code adds a table to the active worksheet and applies the "TableStyleMedium2" table style.

Here's how the code works:
1. The `ActiveSheet` method refers to the active worksheet.
2. `ListObjects.Add` adds a new ListObject (i.e. a table) to the worksheet, using the `xlSrcRange` parameter to specify the data source (in this case, the `Range(nr)` range).
3. The `.Name` property sets the name of the new table to "Table_" followed by the value in the `nr` variable.
4. The `.TableStyle` property applies the "TableStyleMedium2" Table Style to the new table.
5. Finally, `Unlist` removes the ListObject and returns the data to a normal range format.

Overall, this code allows for easy creation of a formatted and styled table in Excel via VBA programming.

Day 4 – Understanding Excel Events, External Data & Files

WORKSHOP FOR NAVY

Learning Outcomes:

At the end of this session, you will be:

a) Able to understand Excel Events, Files and how to connect to data sources
b) Able to work with Userform

Software(s): Microsoft Excel 2019

Instructions:

- Download the datasets for this hands-on.
- Launch Microsoft Office Excel.
- Select a blank workbook and complete the following exercises.

Understanding Excel Events

There are several ways to execute a VBA sub procedure. Events is one of them and can be used to automatically trigger sub procedure upon the occurrence of an event. It takes place during the Excel session. Some examples are moving the mouse, clicking on objects, changing data, or activating windows.
Some common examples of events are like opening a workbook, closing a workbook, changing a cell value, and saving a workbook. In general, common event groups are as follows:

1. Workbook Events

Event Examples	When it is triggered
Activate	The workbook is activated
Open	The workbook is opened
SheetSelectionChange	The selection is changed

2. Worksheet Events

Event Examples	When it is triggered
Activate	The worksheet is activated
Change	A change is made in a cell in the worksheet

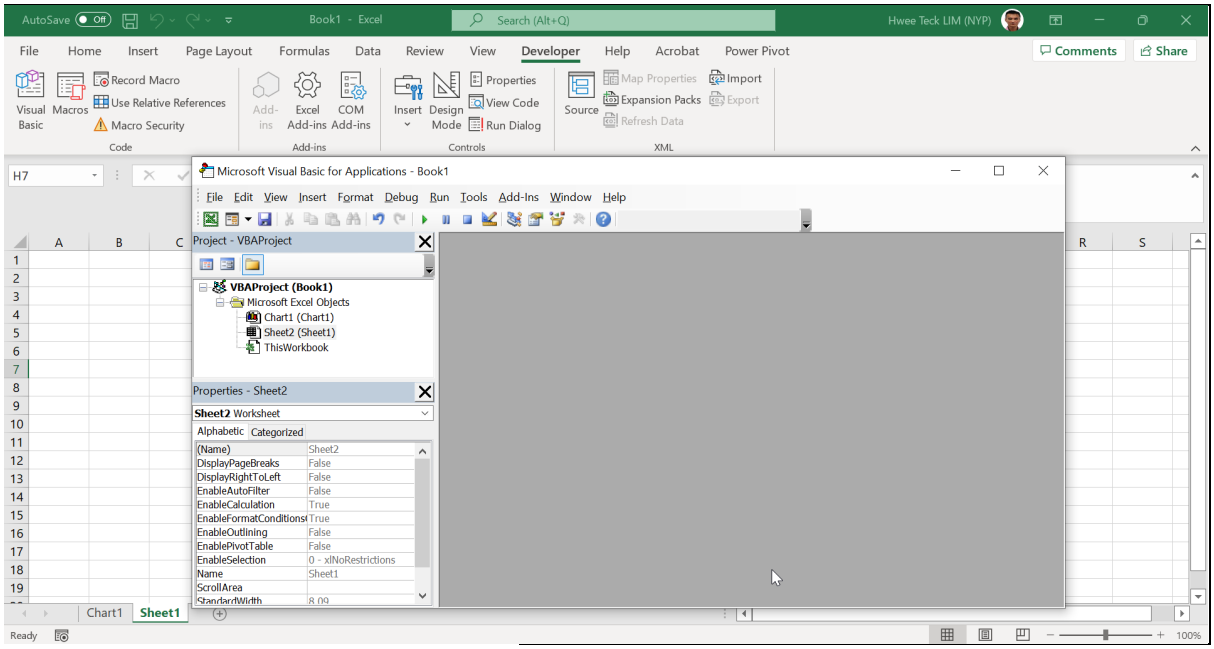
3. Chart Events

Event Examples	When it is triggered
Activate	The chart sheet is activated
Change	A chart element has been selected

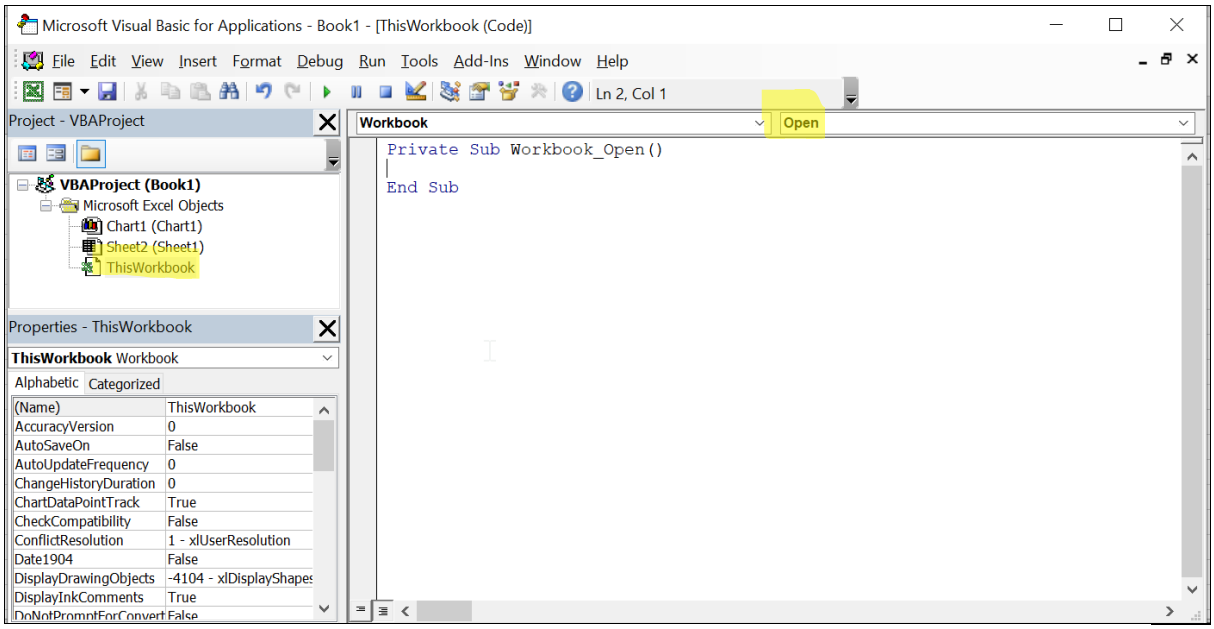
4. Userform Events

Event Examples	When it is triggered
Initialize	Event occurs before Userform is displayed
Click	A CommandButton on a Userform is clicked

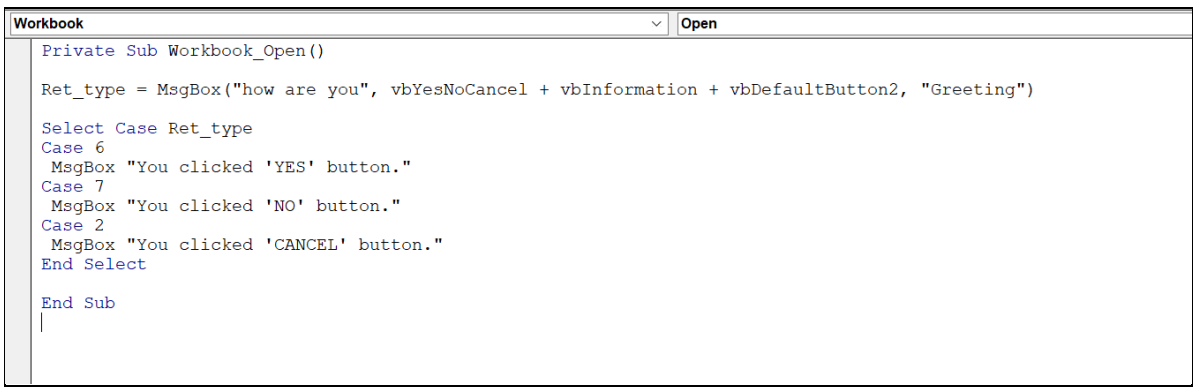
Let see some examples of processing events and understand the event sequences.
Launch Excel. Press <Alt> + <F11> simultaneously to bring up Microsoft Visual Basic for Application. See figure below:



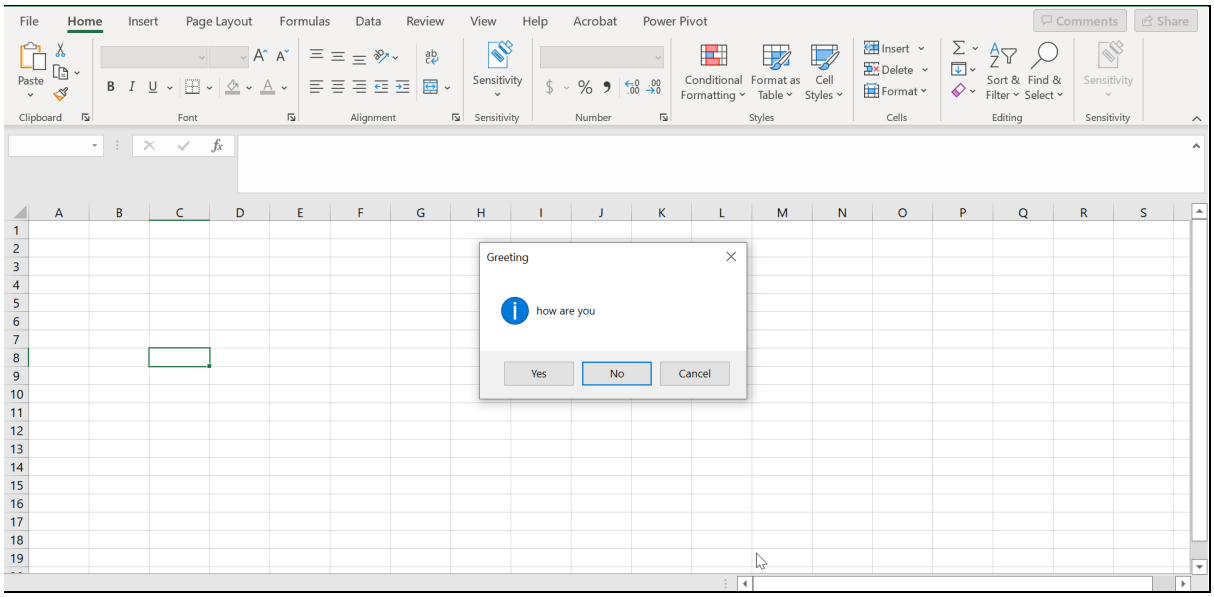
Click on ThisWorkbook and select Open



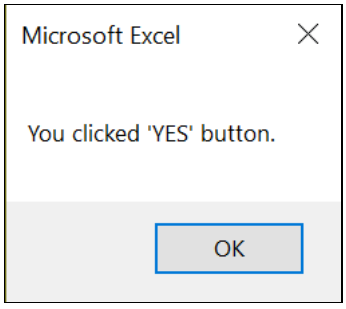
Click on the white space after the Private Sub Workbook_Open() line and type the visual basic lines as shown in the figure below.



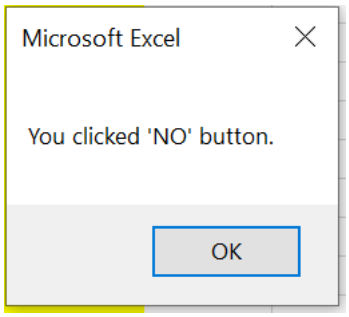
Save and close the file. Re-open the excel file. The excel file opens with a greeting message text.



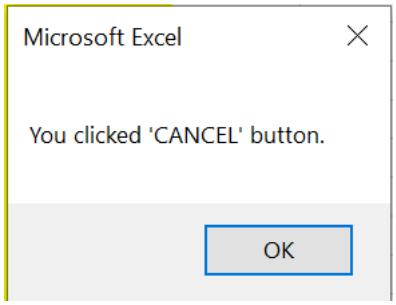
When Yes button is clicked, the msgbox will display the following



When No button is clicked, the msgbox will display the following:



When Cancel button is clicked, the msgbox will display the following:



Note:
By default, all events are enabled.
To disable all events, execute the following VBA instruction:
Application.EnableEvents = False
To enable events, use this instruction:
Application.EnableEvents = True
Sample Code
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
This is normally done to prevent Events from firing when other code is running.

Exercises

1. Create an event to display the type of excel object and sheet name

2. Create an event to display a greeting message, read and display the input button and check the day of the week.

3. Create an event to remind user to send his report

4. Create an event to track the author name and record the date and time the excel is opened and saved before closing the file

5. Create a custom function

Solution for Exercise 1

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)

MsgBox TypeName(Sh) & vbCrLf & Sh.Name

End Sub
```

Solution for Exercise 2

```
Private Sub Workbook_Open()

Ret_type = MsgBox("how are you", vbYesNoCancel + vbInformation + vbDefaultButton2, "Greeting")

Select Case Ret_type
Case 6
 MsgBox "You clicked 'YES' button."
Case 7
 MsgBox "You clicked 'NO' button."
Case 2
 MsgBox "You clicked 'CANCEL' button."
End Select

If Weekday(Now) = vbThursday Then
Msg = "Today is Thursday. Make sure that you "
Msg = Msg & "submit the TPS Report."
MsgBox Msg, vbInformation
End If
```

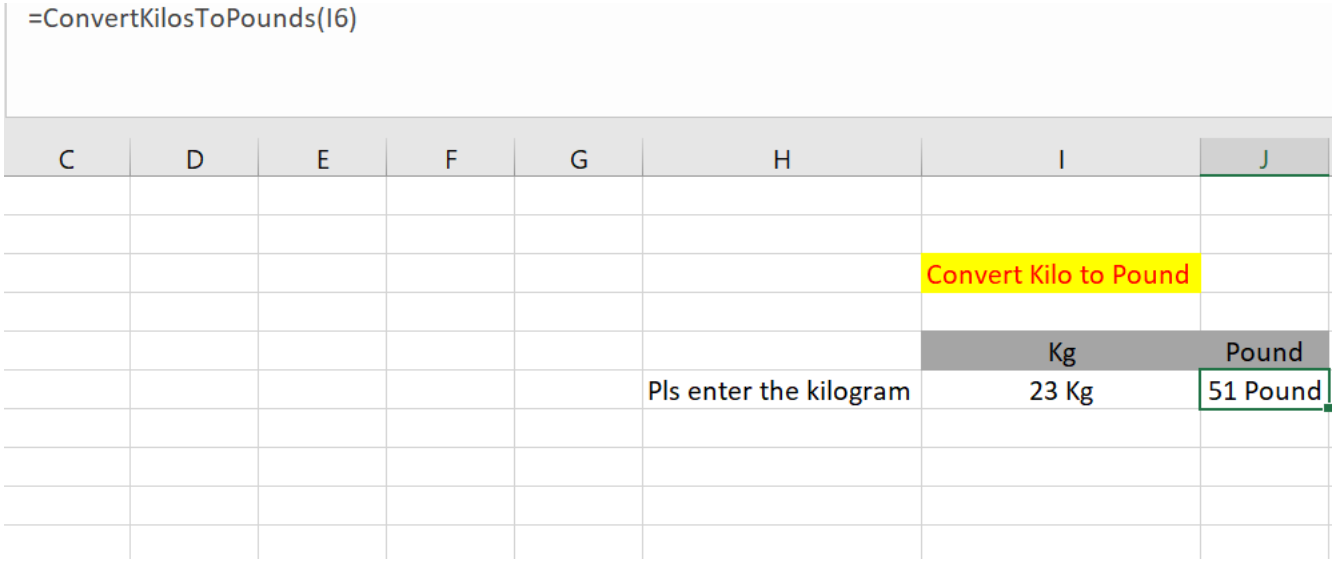
Solution for Exercise 3 (insert the Sub Procedures in the module1

```
Sub DisplayAlarm()

Beep
MsgBox "Wake up. It's time for your afternoon break!"

End Sub


Sub SetAlarm()

Application.OnTime TimeValue("13:38:30"), "DisplayAlarm"

End Sub
```

Solution for Exercise 4

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Worksheets("Log").Activate
If Range("A" & Rows.Count).End(xlUp).Value = "" Then
   Range("A" & Rows.Count).End(xlUp).Value = Now()
   Range("B" & Rows.Count).End(xlUp).Value = ThisWorkbook.BuiltinDocumentProperties("Author")
Else
   Range("A" & Rows.Count).End(xlUp).Offset(1).Value = Now()
   Range("B" & Rows.Count).End(xlUp).Offset(1).Value = ThisWorkbook.BuiltinDocumentProperties("Author")
End If
End Sub
```

Solution for Exercise 5




```
Function ConvertKilosToPounds(dblKilo As Double) As Double
  ConvertKilosToPounds = dblKilo * 2.2
  Debug.Print (ConvertKilosToPounds)
End Function
```

To display the unit of measurement, use format -> custom -> #" Kg" and #" Pound"

**VBA Function – Call. Return Value. & Parameters*

External Data and Files

External data is exactly what it sounds like: data that isn't located in the Excel workbook in which you're operating. Some examples of external data sources are text files, Access tables, SQL Server tables, and even other Excel workbooks.

There are numerous ways to get data into Excel. In fact, between the functionality found in the UI and the VBA/code techniques, there are] too many techniques to focus on in one chapter. Instead, then, in this chapter we'll focus on a handful of techniques that can be implemented in most situations and that don't come with a lot of pitfalls and gotchas. Here is an illustration using MySQL 8.0 version. The very first step is to set up MySQL connection.

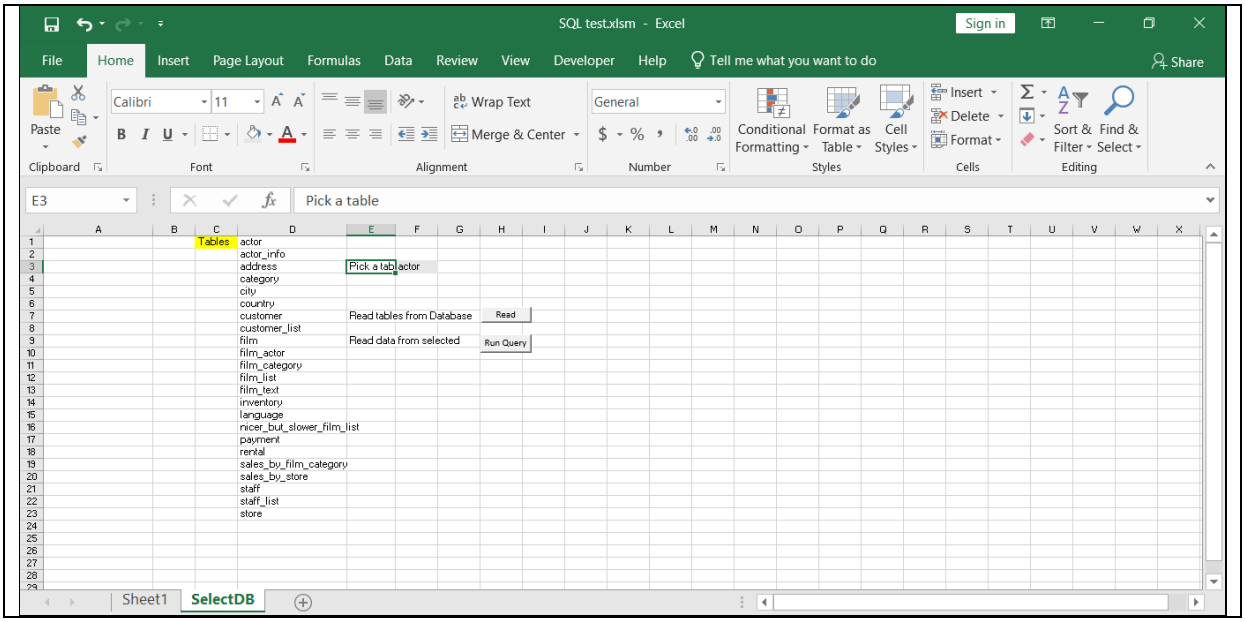
Example

```
Create the two VBA subroutines.

Sub ConnectDB()

Dim oConn As ADODB.Connection, rcSet As ADODB.Recordset


Set oConn = New ADODB.Connection
Dim str As String, n As Long, msg As String, e

str = "DRIVER={MySQL ODBC 8.0 Unicode Driver};" & _
                              "SERVER=localhost;" & _
                              "PORT=3306;" & _
                              "DATABASE=sakila;" & _
                              "UID=root;" & _
                              "PWD=password;"
''' error '''


' Const SQL = "Select *  FROM `Actor`"

SQL = "Select *  FROM " & Worksheets("SelectDB").Range("F3")

On Error Resume Next
oConn.Open str
If oConn.Errors.Count > 0 Then
   msg = ""
   For Each e In oConn.Errors
      msg = msg & vbLf & e.Description
   Next
   MsgBox msg, vbExclamation, "Error - Execute Failed"
Else
```

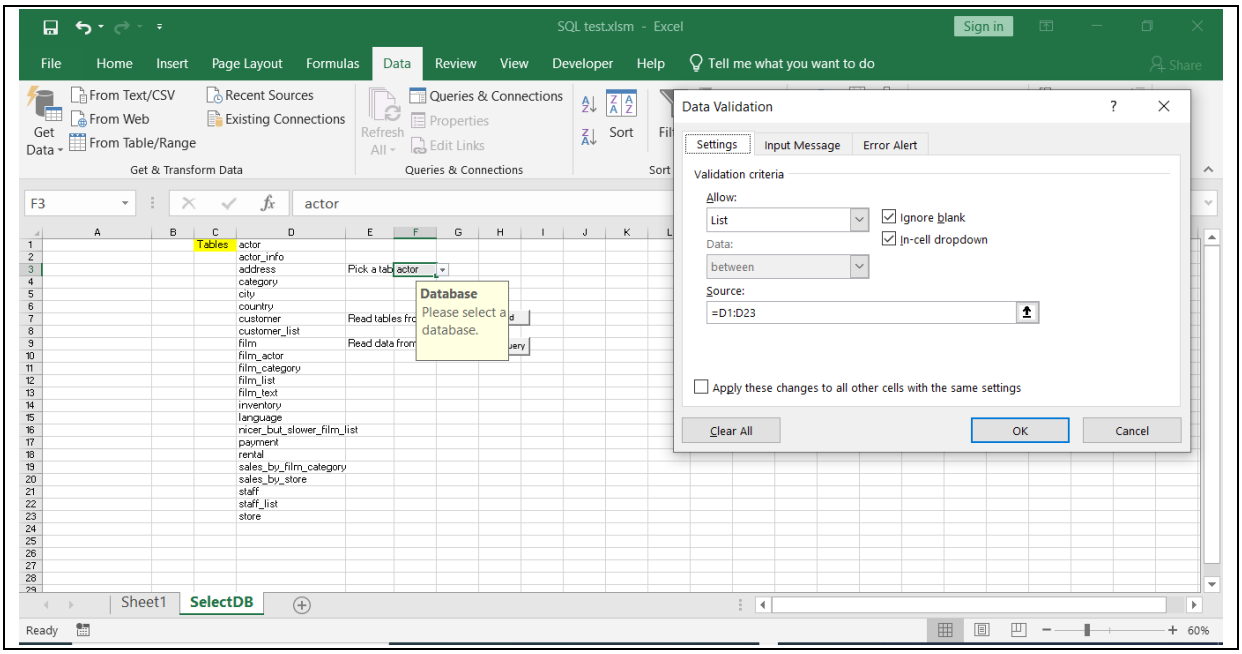
```
    MsgBox "Connected to database " & oConn.DefaultDatabase, vbInformation, "success"
End If

Set rcSet = oConn.Execute(SQL, n)
If oConn.Errors.Count > 0 Then
    msg = ""
    For Each e In oConn.Errors
        msg = msg & vbLf & e.Description
    Next
    MsgBox msg, vbExclamation, "Error - Execute Failed"
Else
    Sheets(1).Range("A1").CopyFromRecordset rcSet
    MsgBox SQL & " returned " & n & " records", vbInformation
End If

On Error GoTo 0

oConn.Close

End Sub


Sub ReadTable()

Dim oConn As ADODB.Connection, rcSet As ADODB.Recordset


Set oConn = New ADODB.Connection
Dim str As String, n As Long, msg As String, e

str = "DRIVER={MySQL ODBC 8.0 Unicode Driver};" & _
                              "SERVER=localhost;" & _
                              "PORT=3306;" & _
                              "DATABASE=sakila;" & _
                              "UID=root;" & _
                              "PWD=password;"
''' error '''


' Const SQL = "Select *  FROM `Actor`"

SQL = "Show Tables"

On Error Resume Next
oConn.Open str
If oConn.Errors.Count > 0 Then
    msg = ""
```

```
    For Each e In oConn.Errors
        msg = msg & vbLf & e.Description
    Next
    MsgBox msg, vbExclamation, "Error - Execute Failed"
Else
    MsgBox "Connected to database " & oConn.DefaultDatabase, vbInformation, "success"
End If

Set rcSet = oConn.Execute(SQL, n)
If oConn.Errors.Count > 0 Then
    msg = ""
    For Each e In oConn.Errors
        msg = msg & vbLf & e.Description
    Next
    MsgBox msg, vbExclamation, "Error - Execute Failed"
Else
    Sheets("SelectDB").Range("D1").CopyFromRecordset rcSet
    MsgBox SQL & " returned " & n & " records", vbInformation
End If

On Error GoTo 0

oConn.Close

End Sub
```

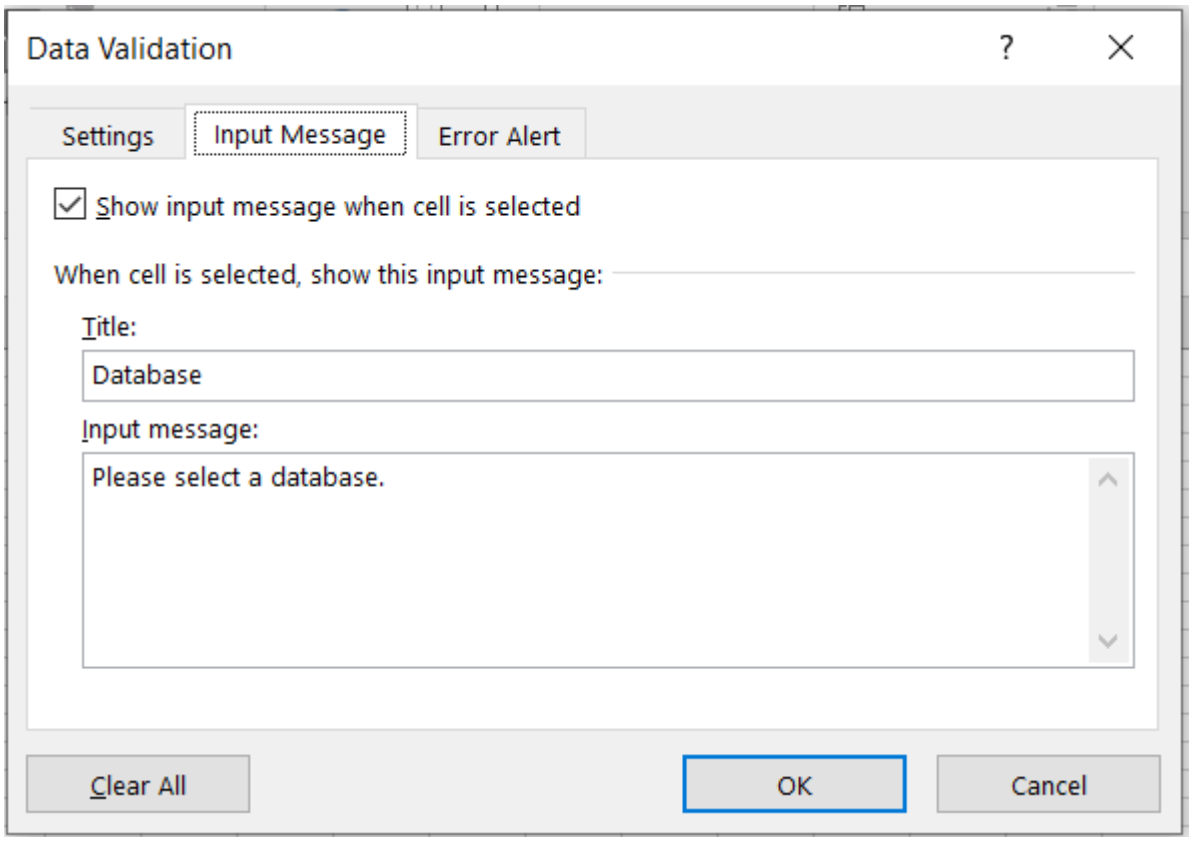
Create a worksheet and name it as SelectDB. Set up your layout accordingly.



In cell F3, set up a Data Validation rule. Use the range you selected for source input.

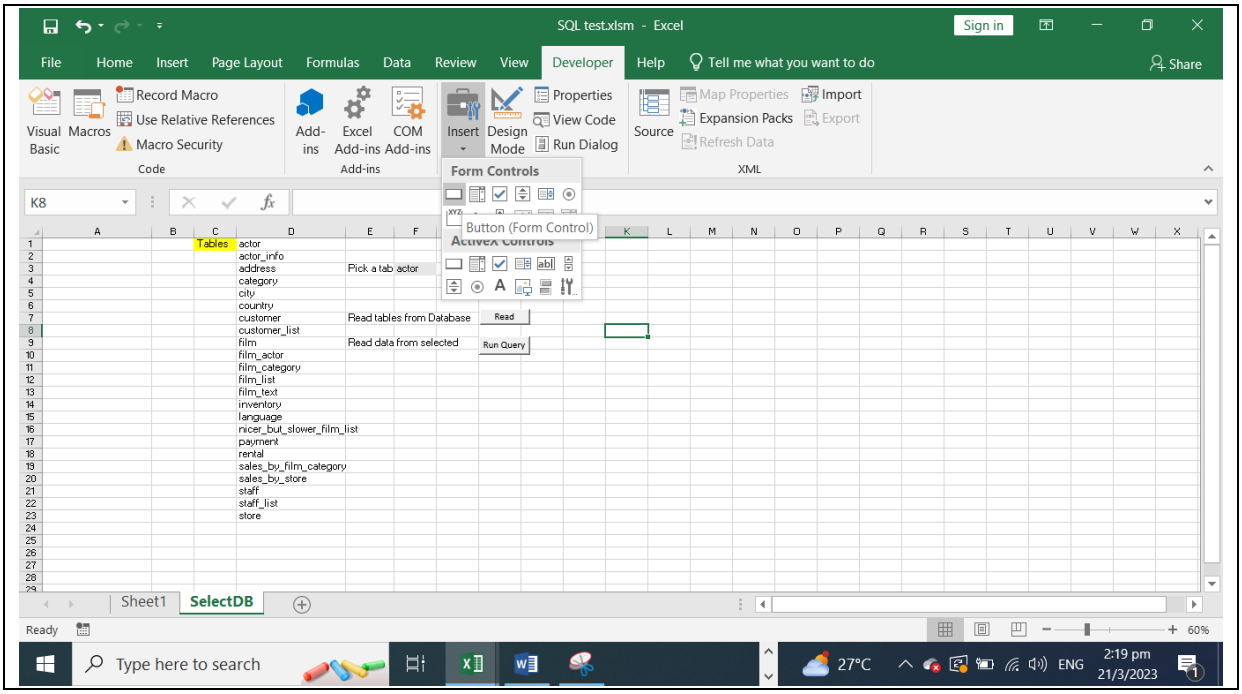


Provide a title and message for the cell F3

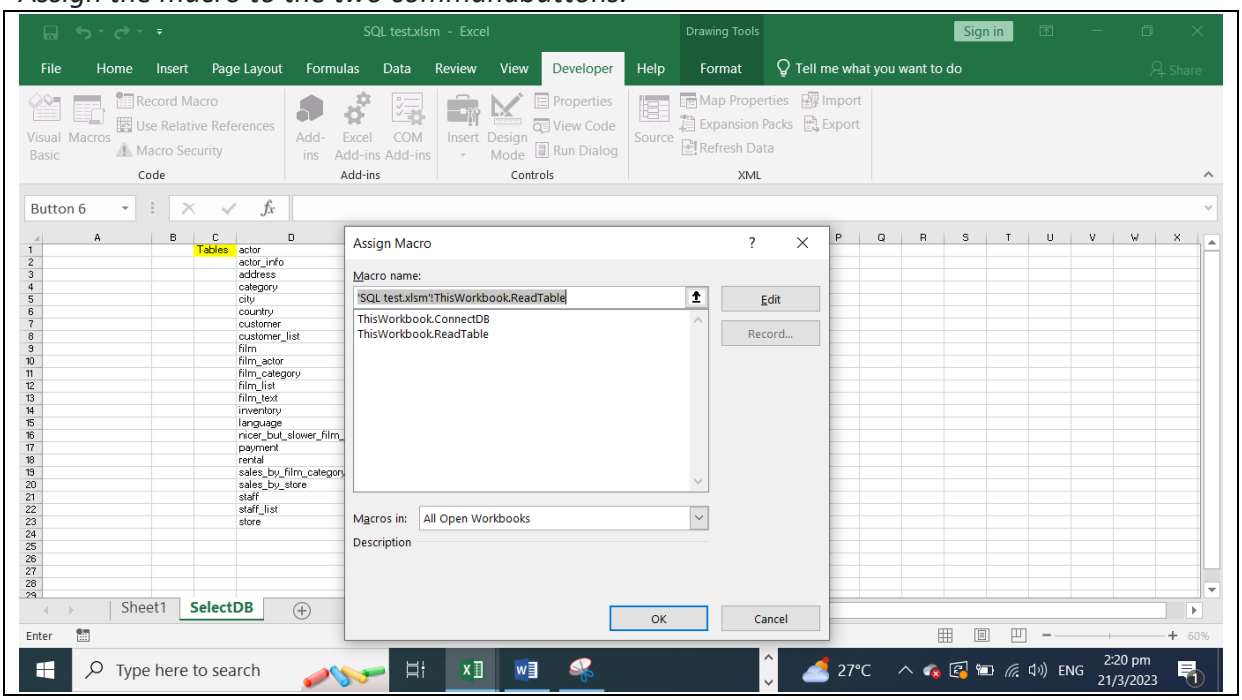


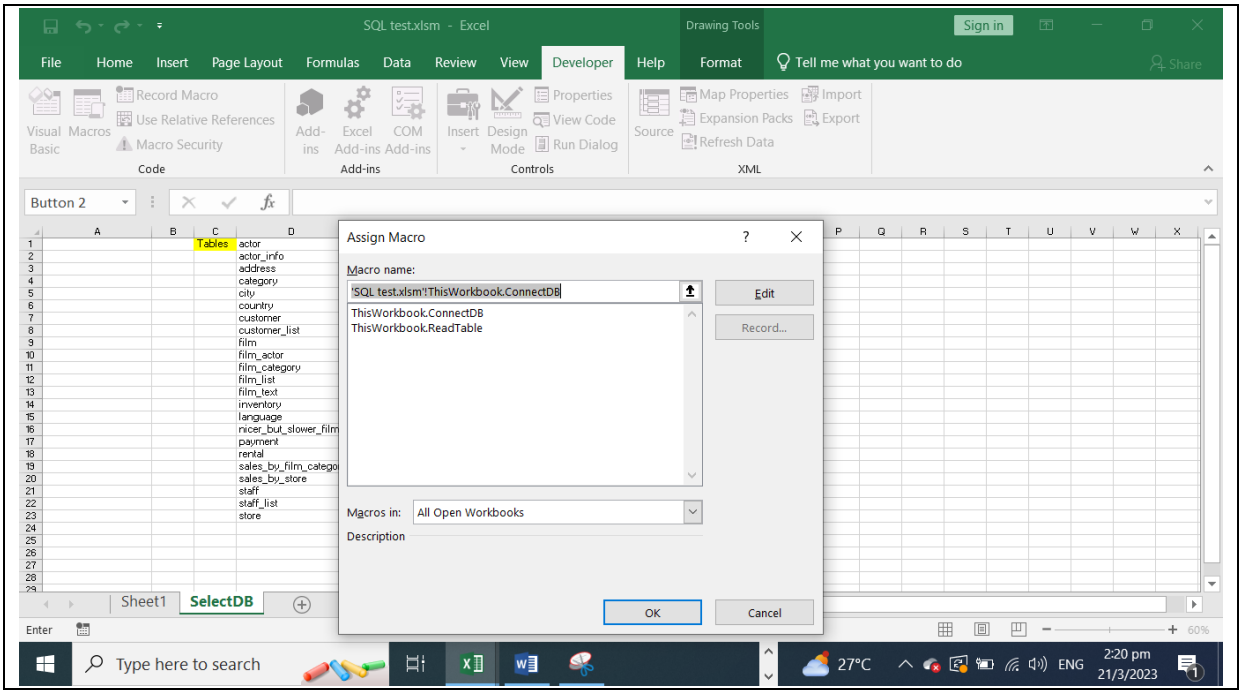
Insert two commandbuttons. The first commandbutton is to read tables from MySQL database.

The second commandbutton is to run the selected table query and display the data in Sheet1.

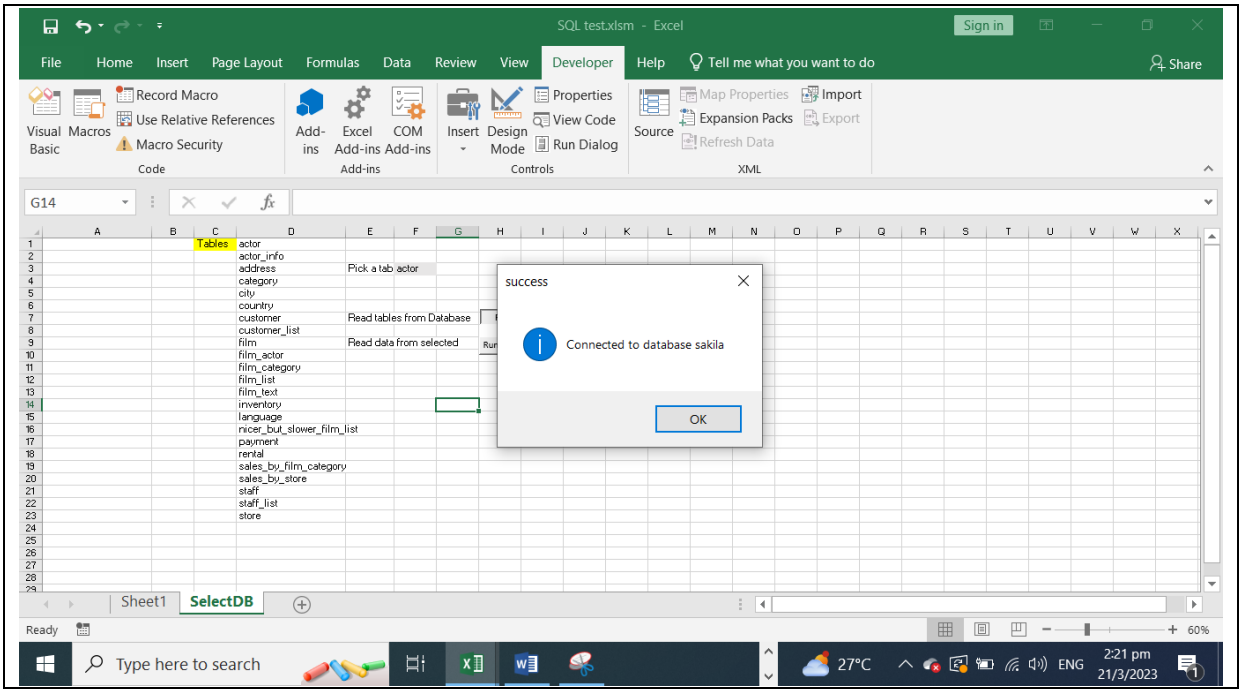


Assign the macro to the two commandbuttons.

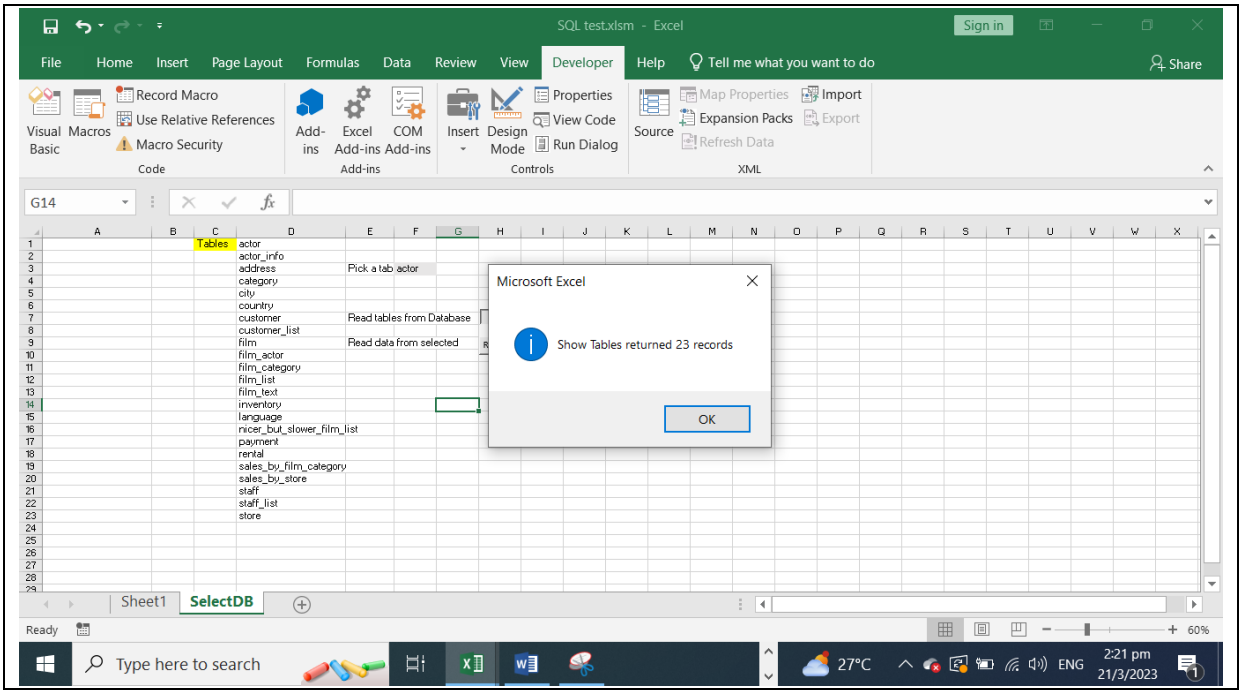




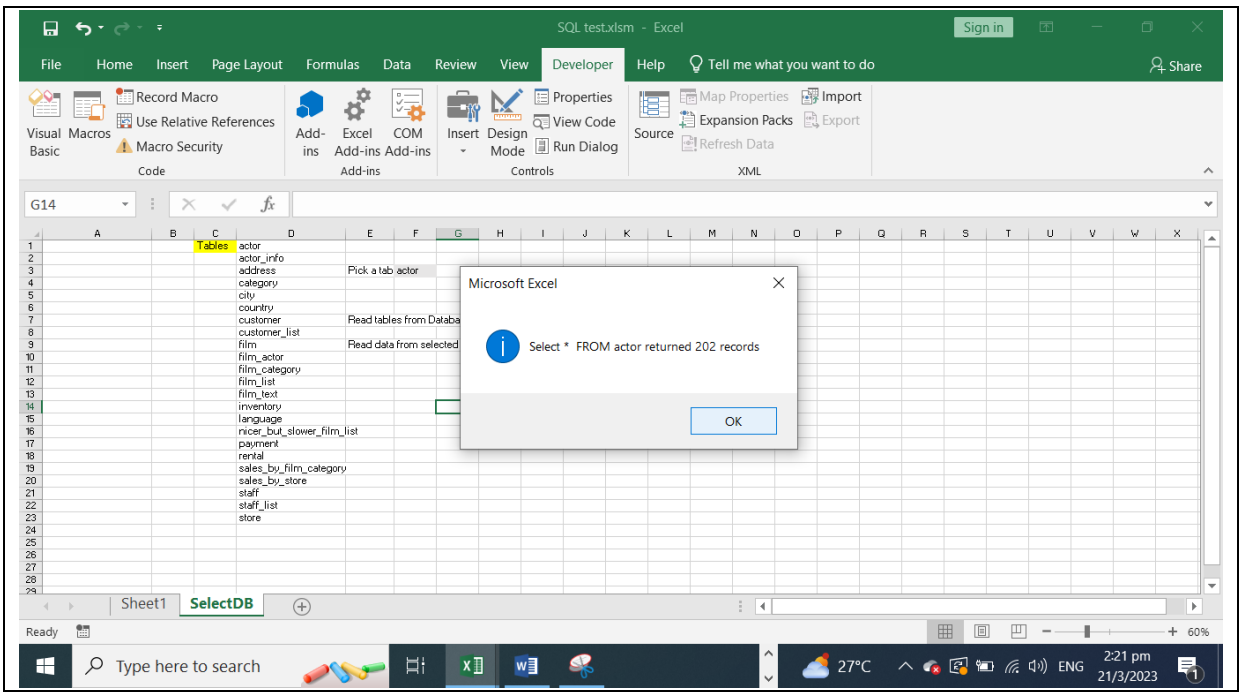
When click on the Read commandbutton, the displayed message shows "Connected to database sakila".



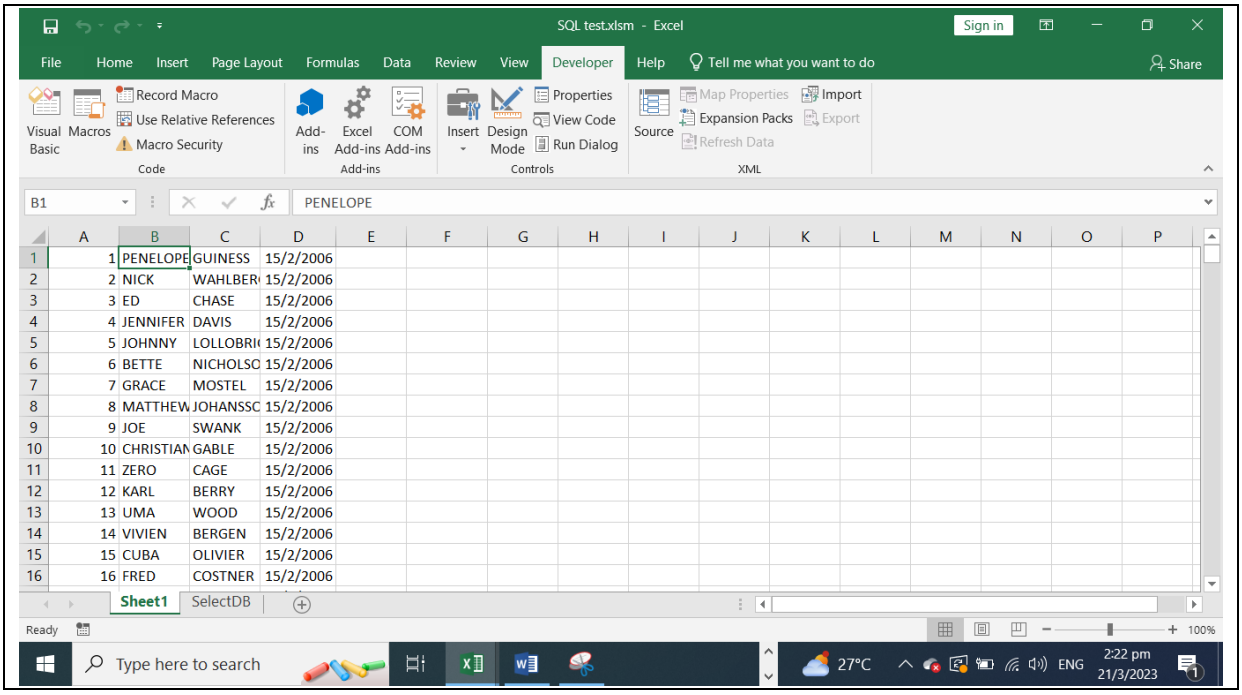
Next, it retrieves the tables in the Sakila database.



When clicked on the Run Query, it reads the data from the selected table. In the example, actor table is selected.

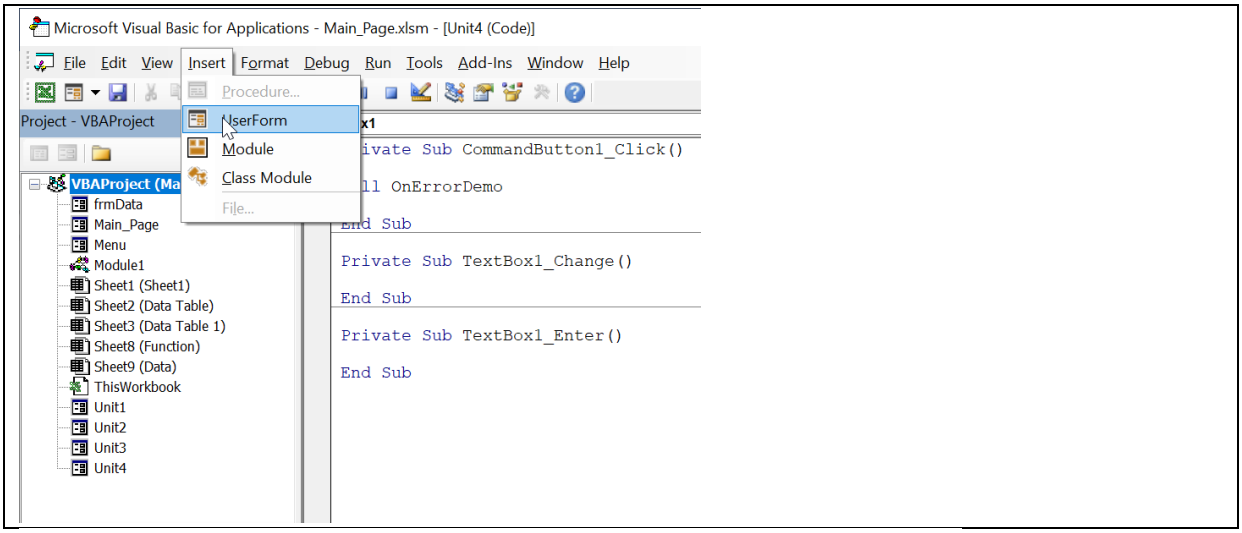


Click on Sheet1, the data will appear.

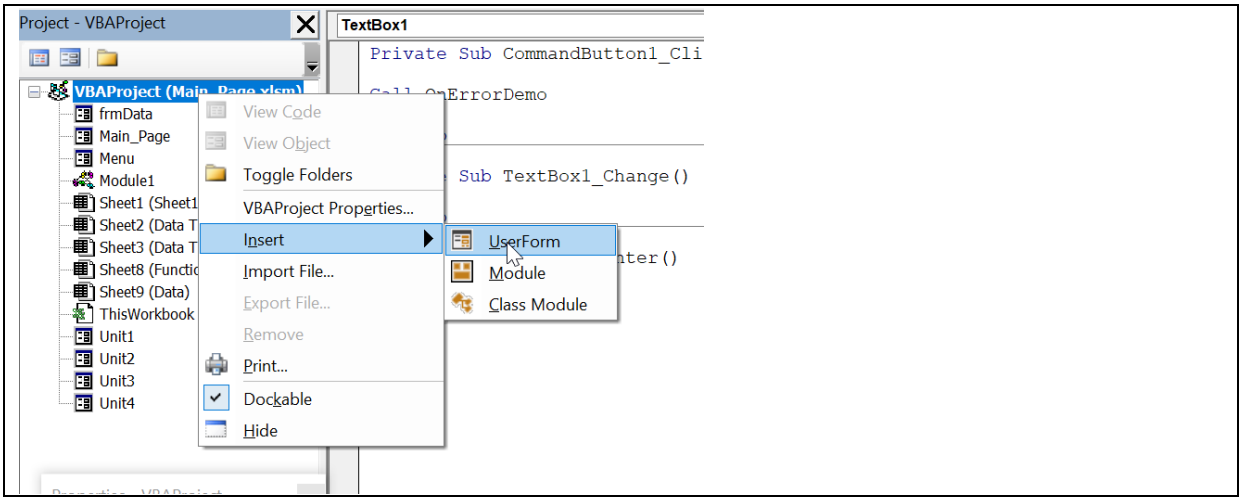


UserForm

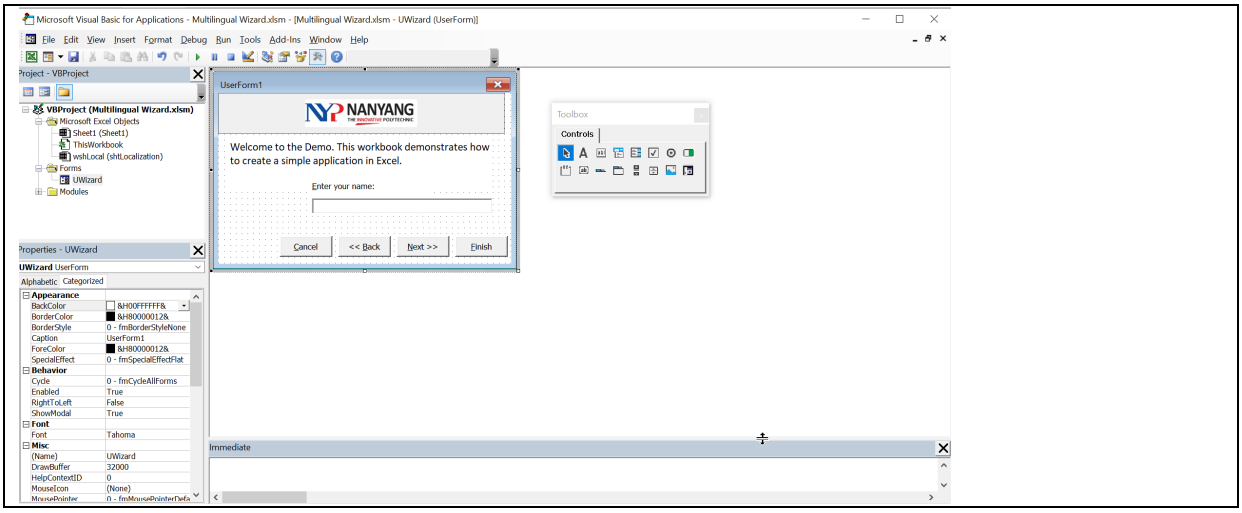
The UserForm is an useful feature of VBA Editor. It provides an interface for your macro to interact with the user and obtain information. To access the UserForm feature, you either launch the VBA Editor and navigate to Menu-item -> Insert -> UserForm.



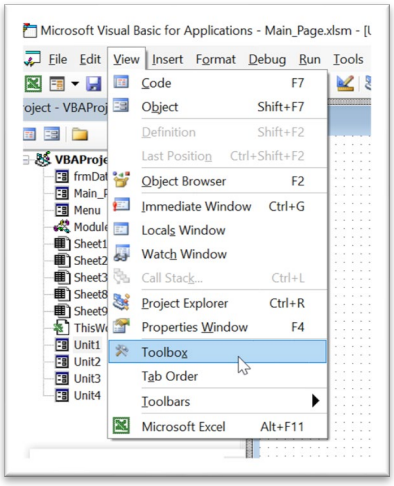
Alternatively, right click on the VBAProject and select Insert -> UserForm



In the design mode, this is where you set up the structure of your Form. You can change the size of your window, place the objects, modify their properties related with the visual part of the form. In this part, you do not really code anything; you just organize your application. The following screen shows an interactive form which prompts the user to enter name. This is created by adding the objects from the ToolBox.



If the ToolBox is not visible, you may activate it by clicking the View ->ToolBox.



Open the ToolBox. The following picture shows all the tools available at the toolbox. When you mouse over the items, the name of the item will appear. In the picture below, the item is a Label

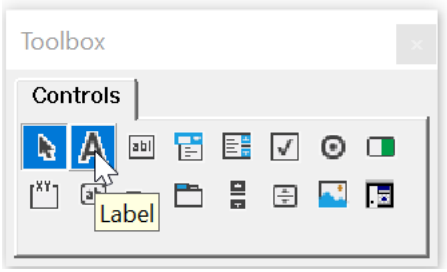
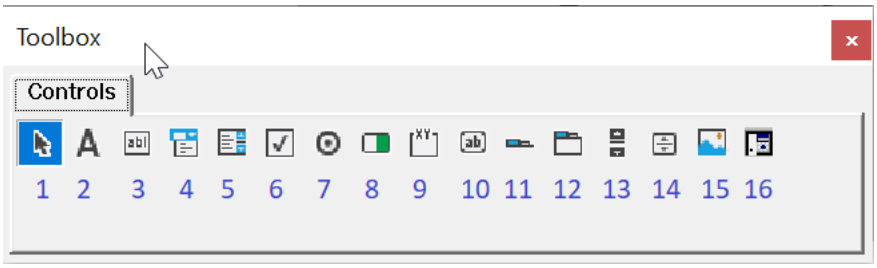
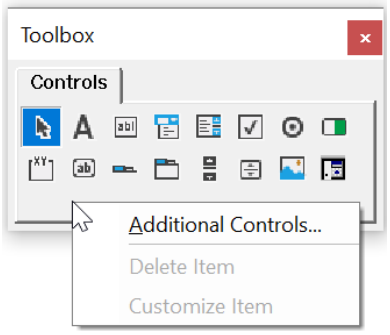


Table for the common objects found in the ToolBox

1. Select Objects 2. Label 3. TextBox 4. ComboBox 5. ListBox 6. CheckBox	7. OptionButton 8. ToggleButton 9. Frame 10. CommandButton 11. TabStrip 12. MultiPage	13. ScrollBar 14. SpinButton 15. Image 16. RefEdit



There are others hidden from the view. By right clicking on the Control and selecting "Additional Controls", you will find more.



The common objects and their descriptions as below:

Select Object

This is a selection object. You use it to select, resize, and move other objects.

Label Object

This is a text label. You can insert it in your form. It does not allow the user changes the value.

TextBox
This object provides a text box for the user to enter information.

ToggleButton
This object functions like a switch. It toggles between On and Off.

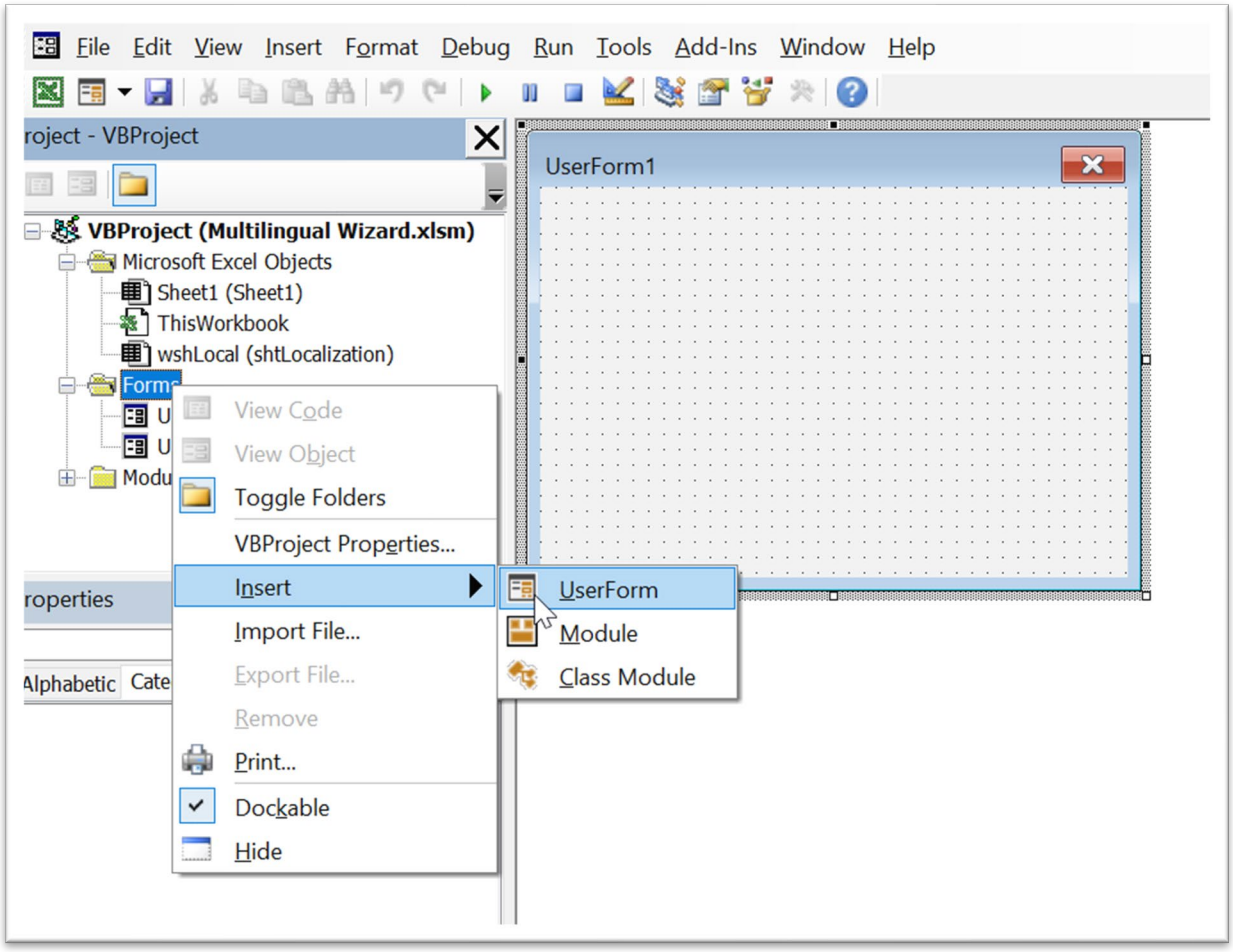
Frame
This object groups the controls inside a frame. A frame is first drawn, and then other controls are placed inside it.

TabStrip
This object views different sets of information for related controls.

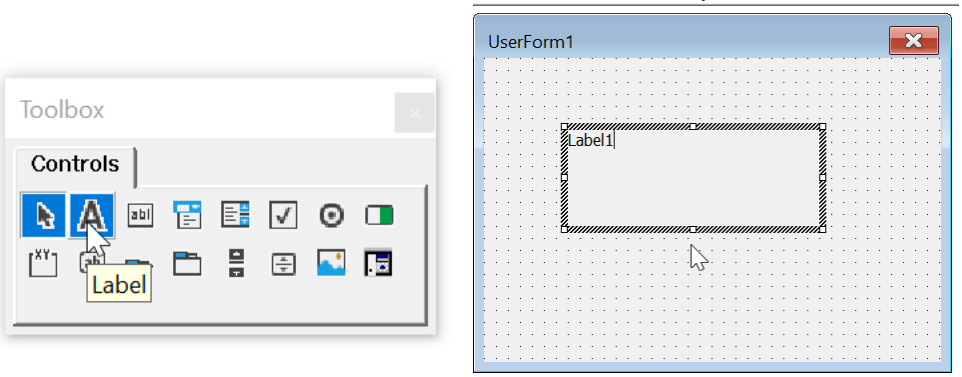
MultiPage
This object operates in a similar way like TabStrip. But, MultiPage allows different controls on each page.

ScrollBar
This object creates a Scrollbar for quick navigation. It is helpful especially with a large amount of information.

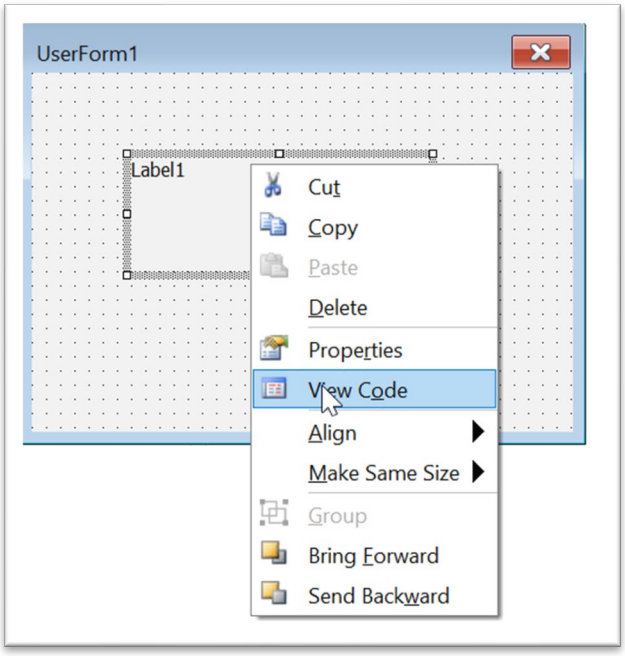
Image
This is a picture object. It allows an image from a bitmap, icon or metafile.
Now, we are ready to look at the usage of some of the objects. Start by inserting a new UserForm, you right click on the Forms and insert a new UserForm



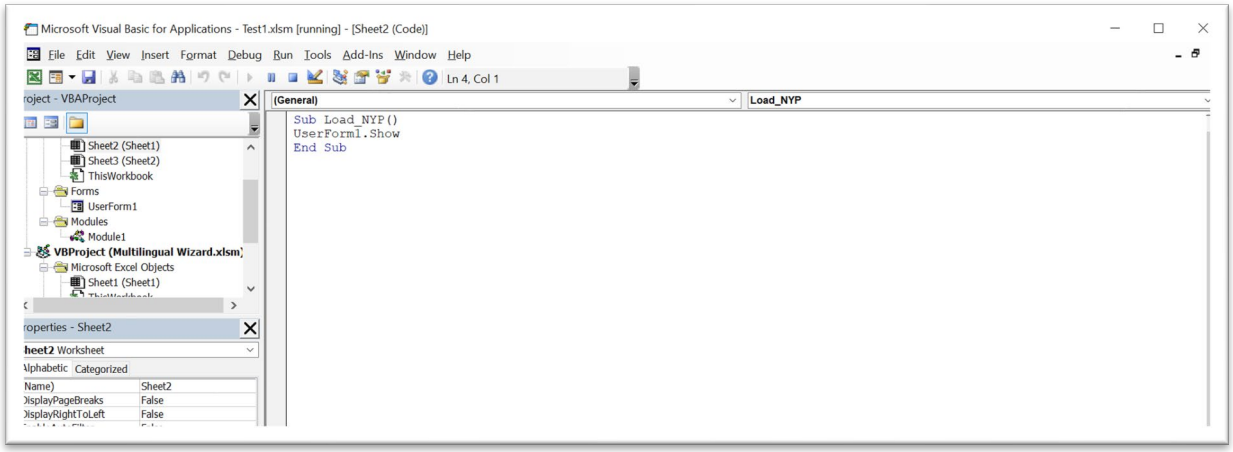
To insert a tool, in our case, we will click on the label and place it inside the Userform



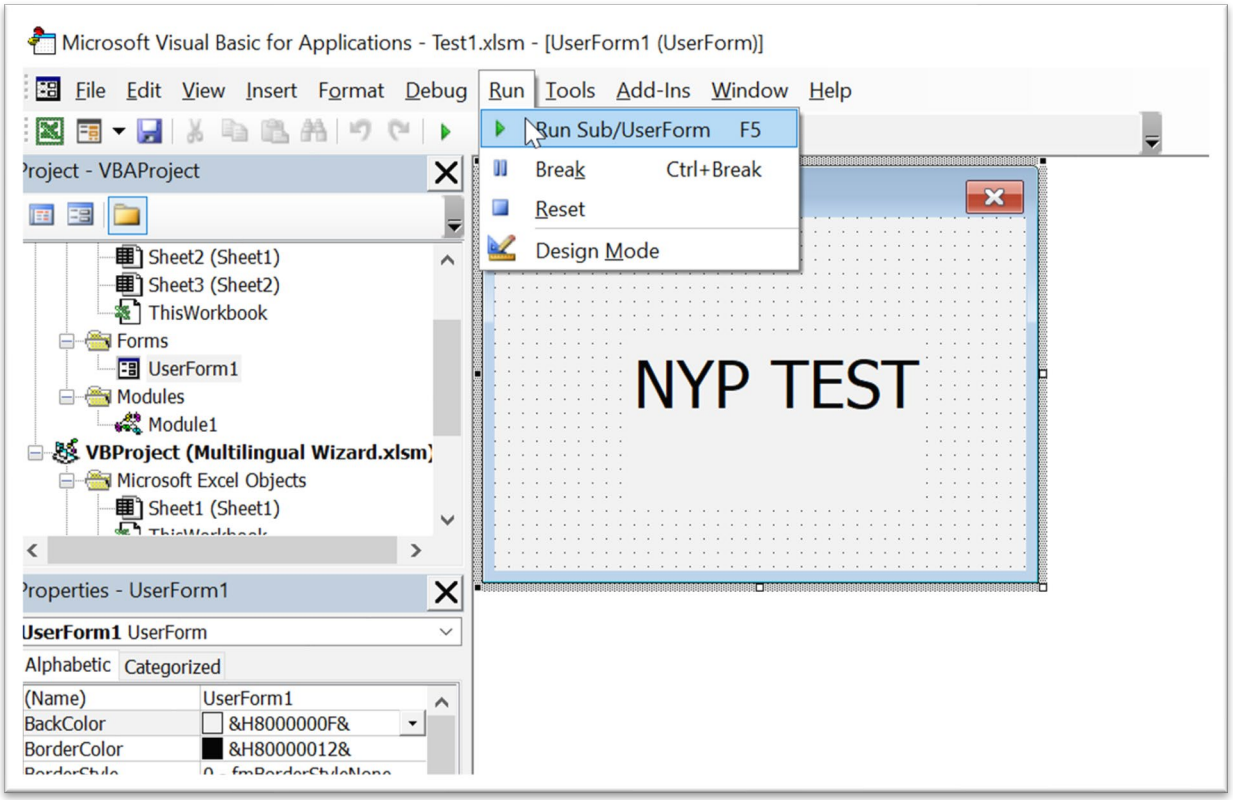
To view the Code mode, use the code mode by right clicking the Form object ("Label 1") and selecting View Code.

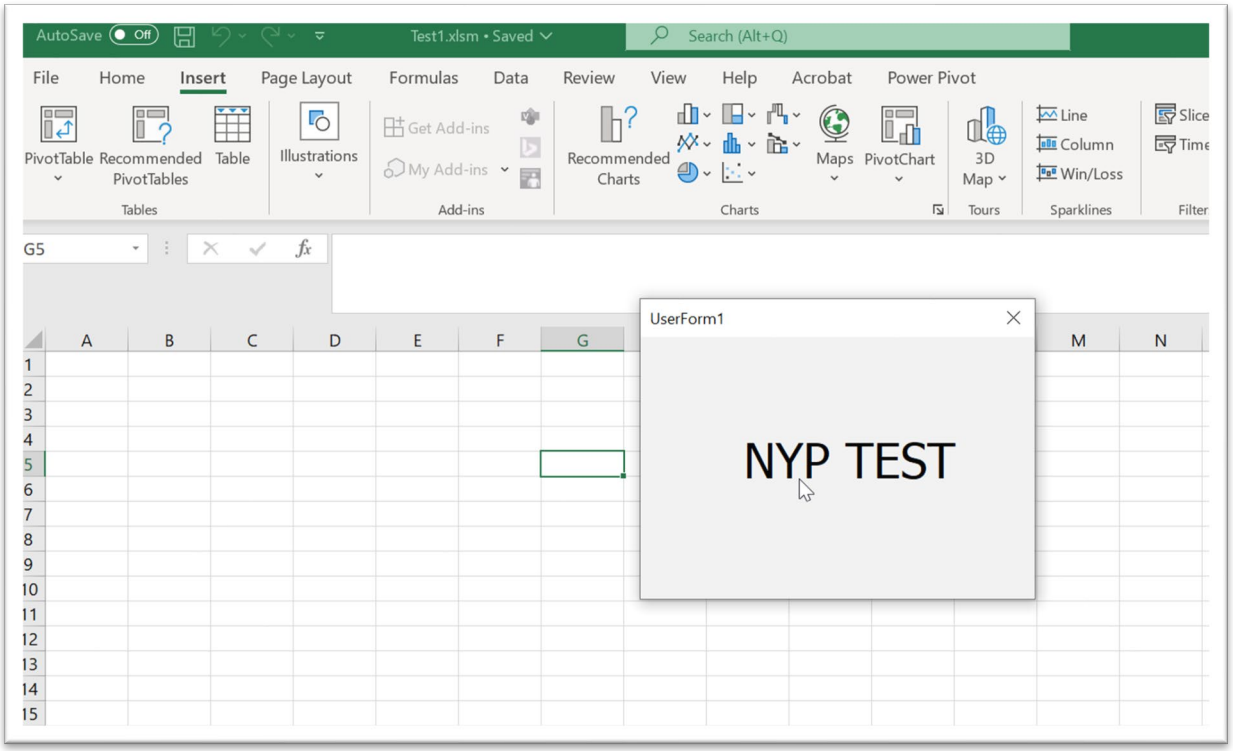


Here you create codes for any object that you created. The step is the same of any object. To activate the UserForm, we use the method “Show”. Example:



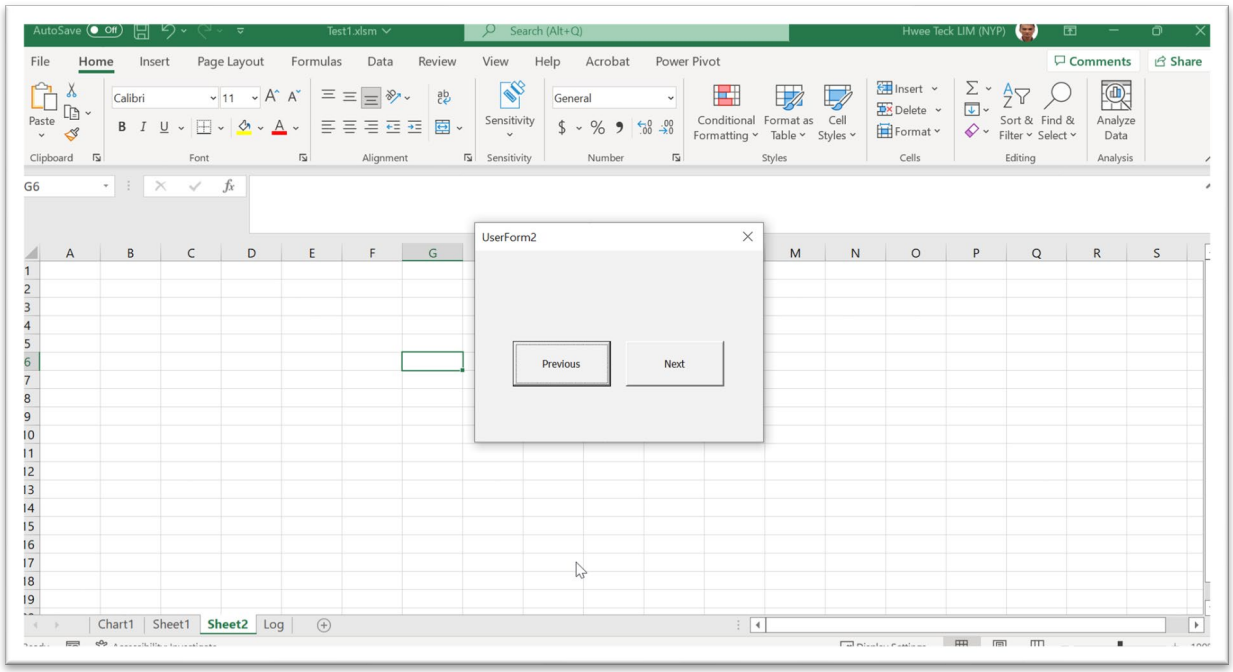
When you click Run and select run Sub/UserForm inside the Sheet2, it displays the form


Microsoft Visual Basic for Applications - Test1.xlsm - [UserForm1 (UserForm)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Run Sub/UserForm F5
Break Ctrl+Break
Reset
Design Mode
Project - VBAProject
Sheet2 (Sheet1)
Sheet3 (Sheet2)
ThisWorkbook
Forms
UserForm1
Modules
Module1
VBProject (Multilingual Wizard.xlsm)
Microsoft Excel Objects
Sheet1 (Sheet1)
Properties - UserForm1
UserForm1 UserForm
Alphabetic Categorized
(Name) UserForm1
BackColor &H8000000F&
BorderColor &H80000012&
NYP TEST

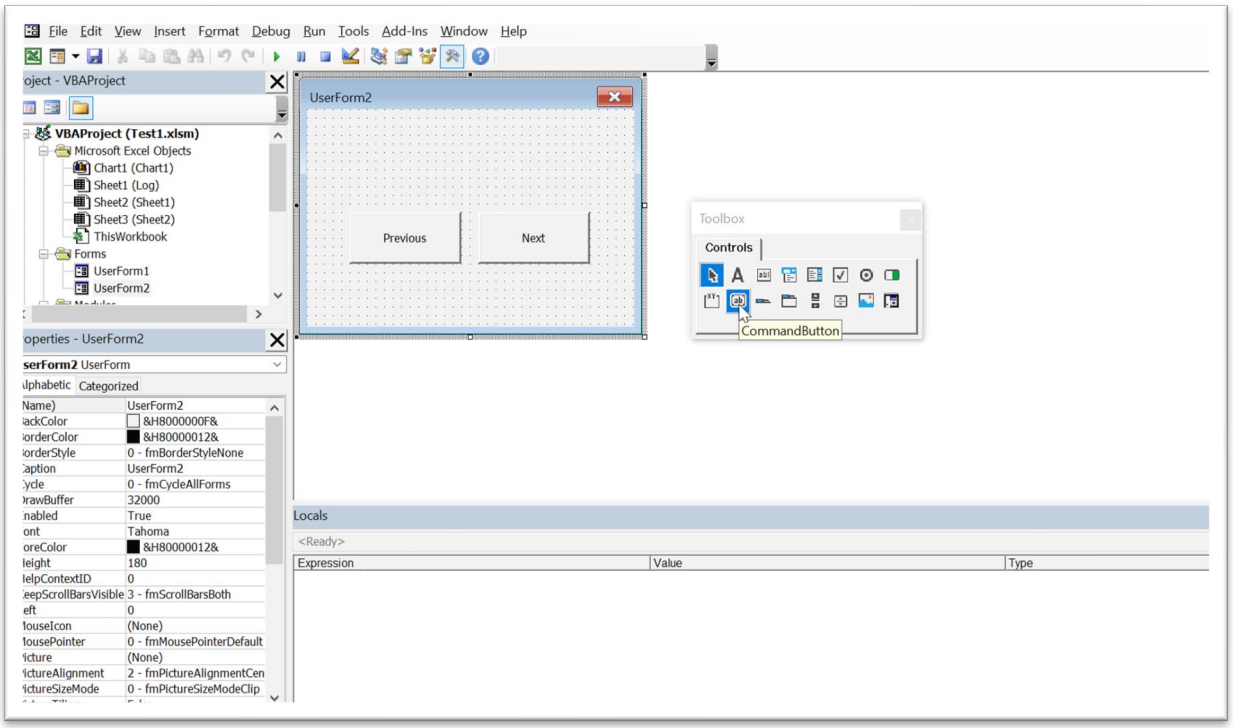

AutoSave Off
Test1.xlsm • Saved
Search (Alt+Q)
File Home Insert Page Layout Formulas Data Review View Help Acrobat Power Pivot
PivotTable Recommended PivotTables Table
Tables
Illustrations
Get Add-ins
My Add-ins
Add-ins
Recommended Charts
Maps
PivotChart
Charts
3D Map
Tours
Line
Column
Win/Loss
Sparklines
Slicer
Timeline
Filter
G5
UserForm1
NYP TEST

Example :

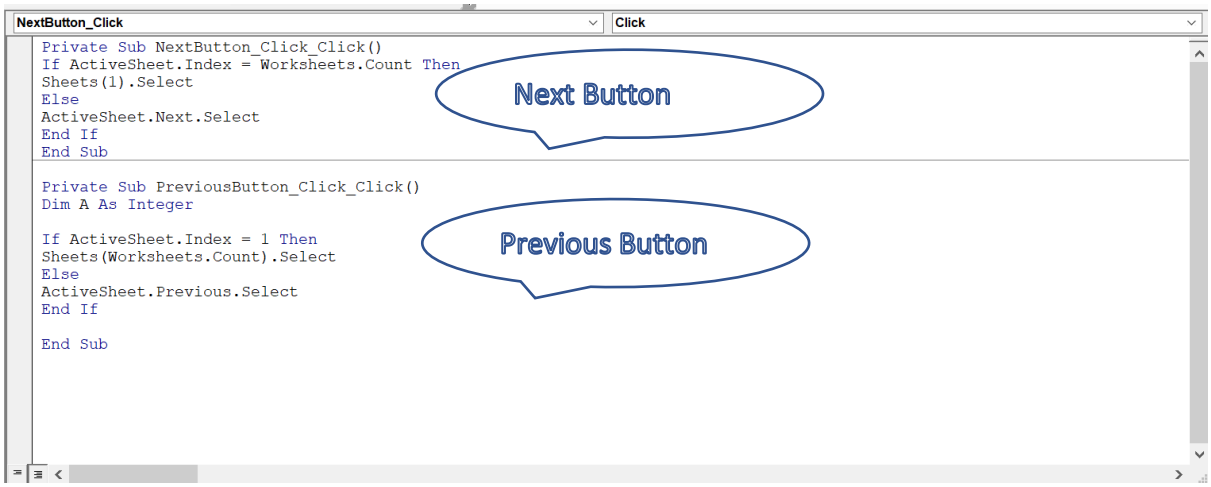
To create a Form that select the Next or Previous worksheet. Please see the UserForm below.



The first Step is insert the user interface and two Buttons (Previous and Next).
Do you still recall how to insert a new UserForm?



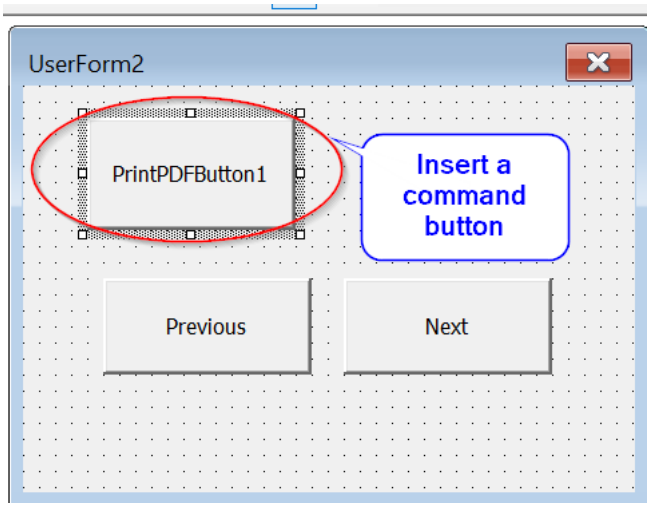
Next, you place the codes in the respective buttons.



Example

To create a button to print each worksheet to PDF

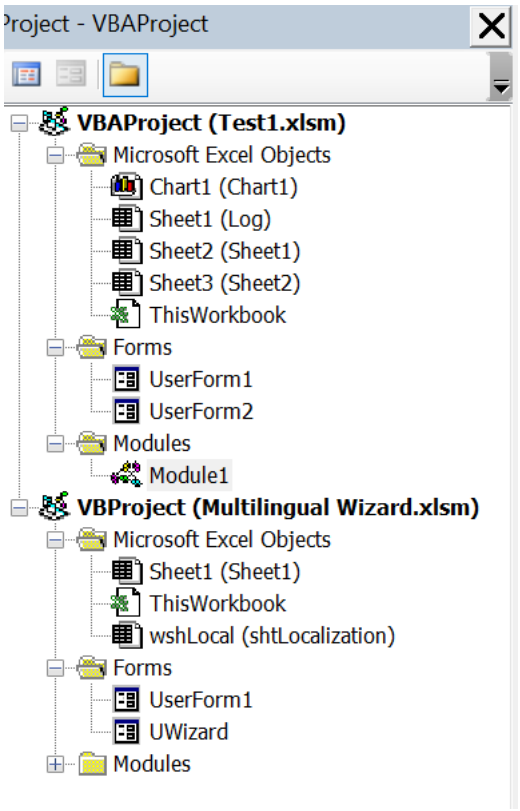
Solution for Exercise 1 (Part A)



Insert a userform and a commandbutton. Rename the commandbutton to PrintPDFButton
Right click to view code and insert the codes

Private Sub PrintPDFButton_Click()
Call SaveWorkshetAsPDF
End Sub

Solution for Exercise 1 (Part B)



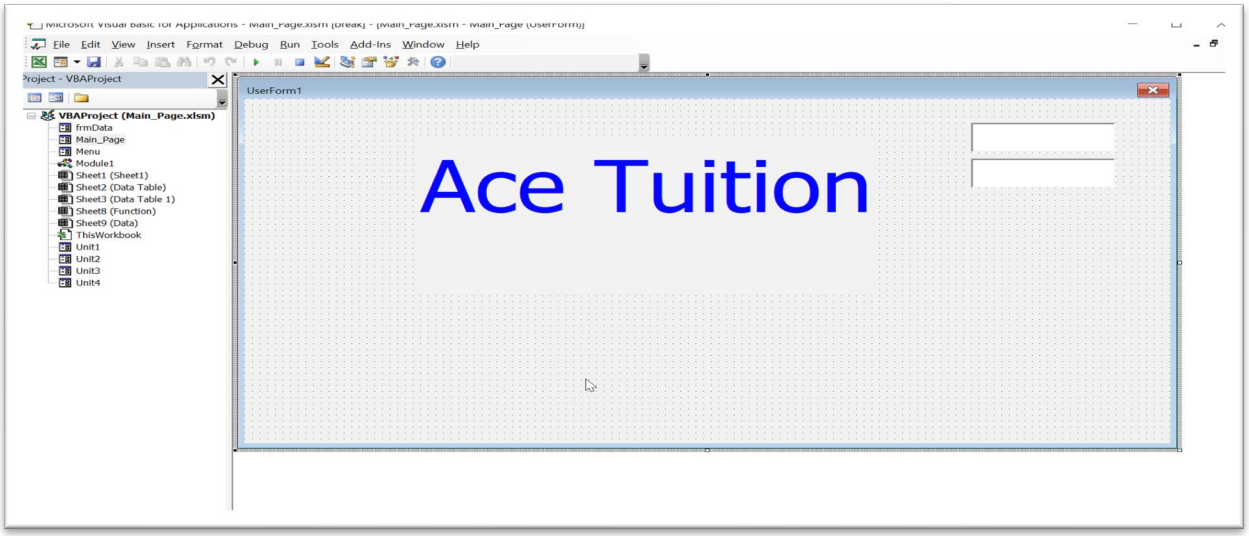

```
Sub SaveWorkshetAsPDF()
Dim ws As Worksheet
Dim PathFolder As String
PathFolder = "C:\Temp"
For Each ws In Worksheets
On Error Resume Next
ws.ExportAsFixedFormat xlTypePDF, PathFolder & "\" & ws.Name & ".pdf"
Next ws
End Sub
```

Exercises

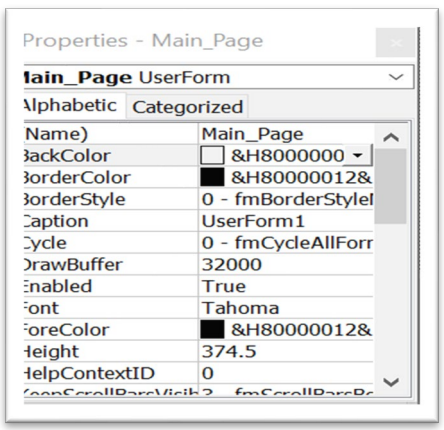
1. Display the date and time (Hints: Use Now() and counter)
2. Open a file dialog to select file (Hints: Use FileSystemObject)
3. Create different charts (Hints: Use Create Charts)
4. Create a simple selector for gender (Hints: Use options)
5. Create a userform for data collection (Hints: UserForm and VBA script)

Solution for Exercise 1

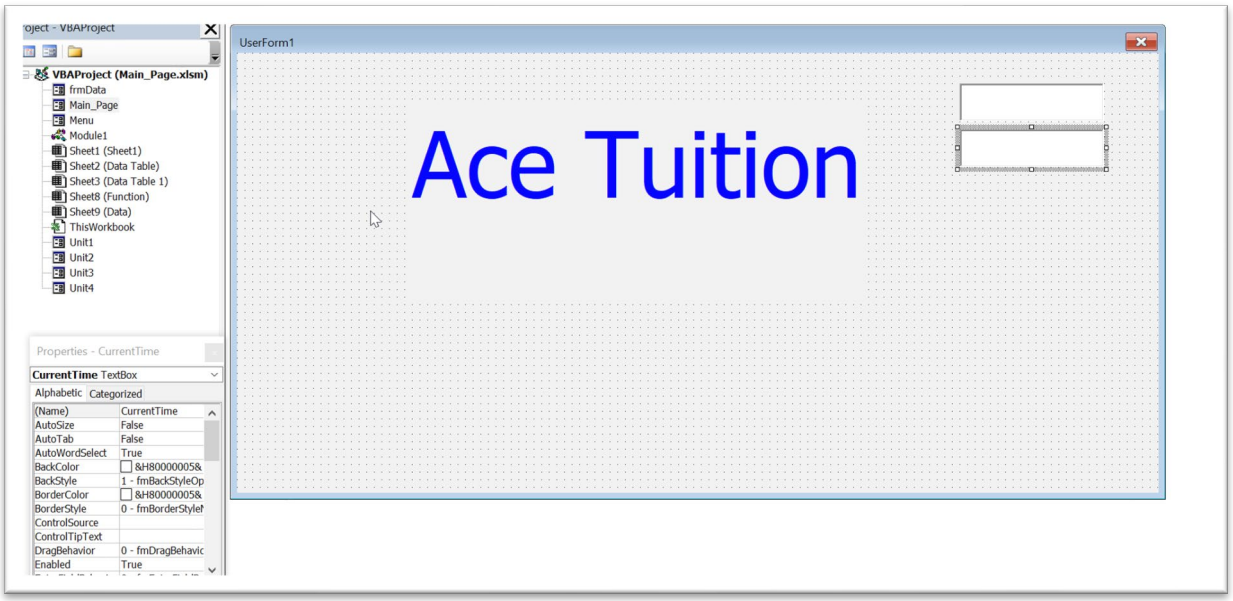
Create a userform with a label and two textboxes.



Name the userform. Here we named it Main_Page



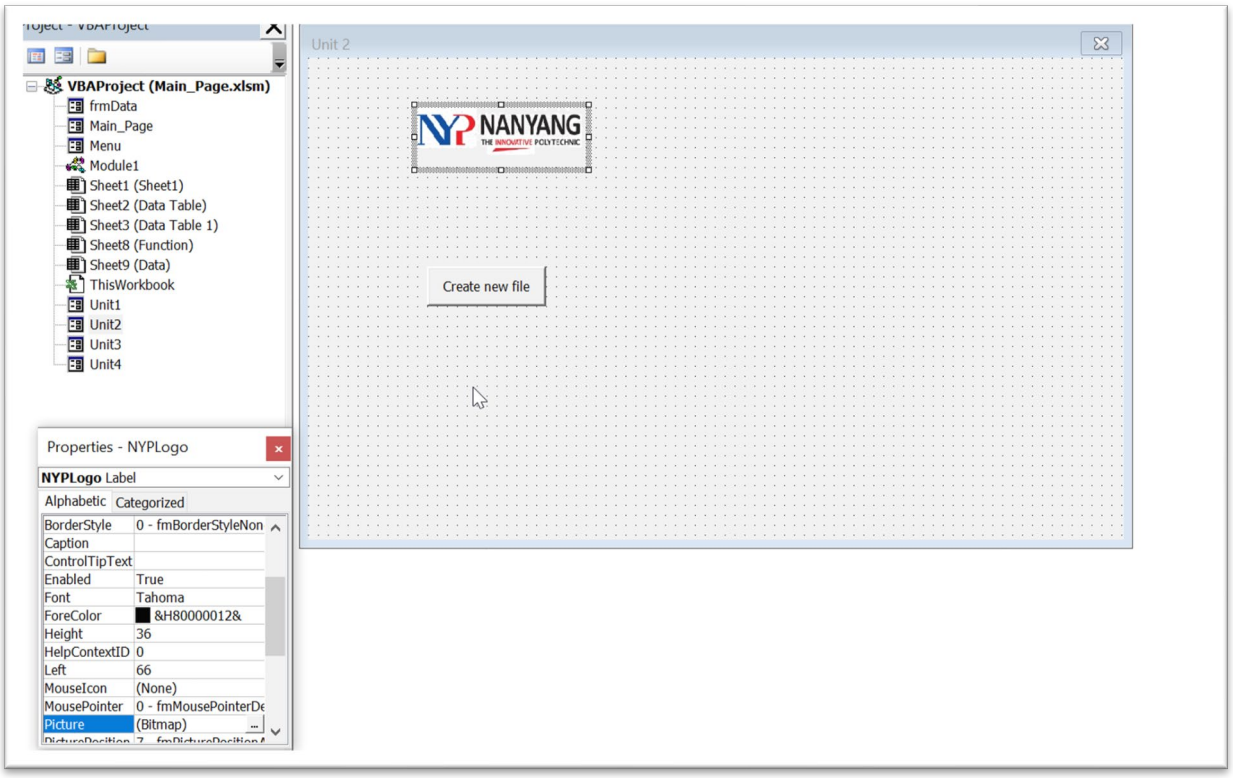
Name the two textboxes as CurrentDate and CurrentTime



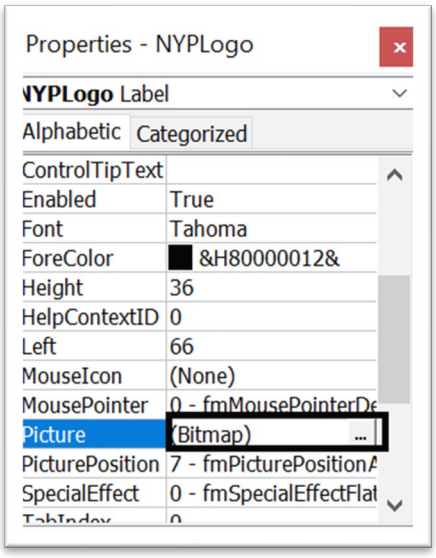

```
Sub DisplayTime()

T = Now + TimeValue("00:00:01")
Application.OnTime T, "UpdateTime"

End Sub

Sub UpdateTime()

Main_Page.CurrentDate.Value = Format(Date, "DD-MMM-YYYY")
Main_Page.CurrentTime.Value = Format(T, "hh:mm:ss")

Call DisplayTime
End Sub
```

Solution for Exercise 2

Create a userform with a label and one commandbutton.

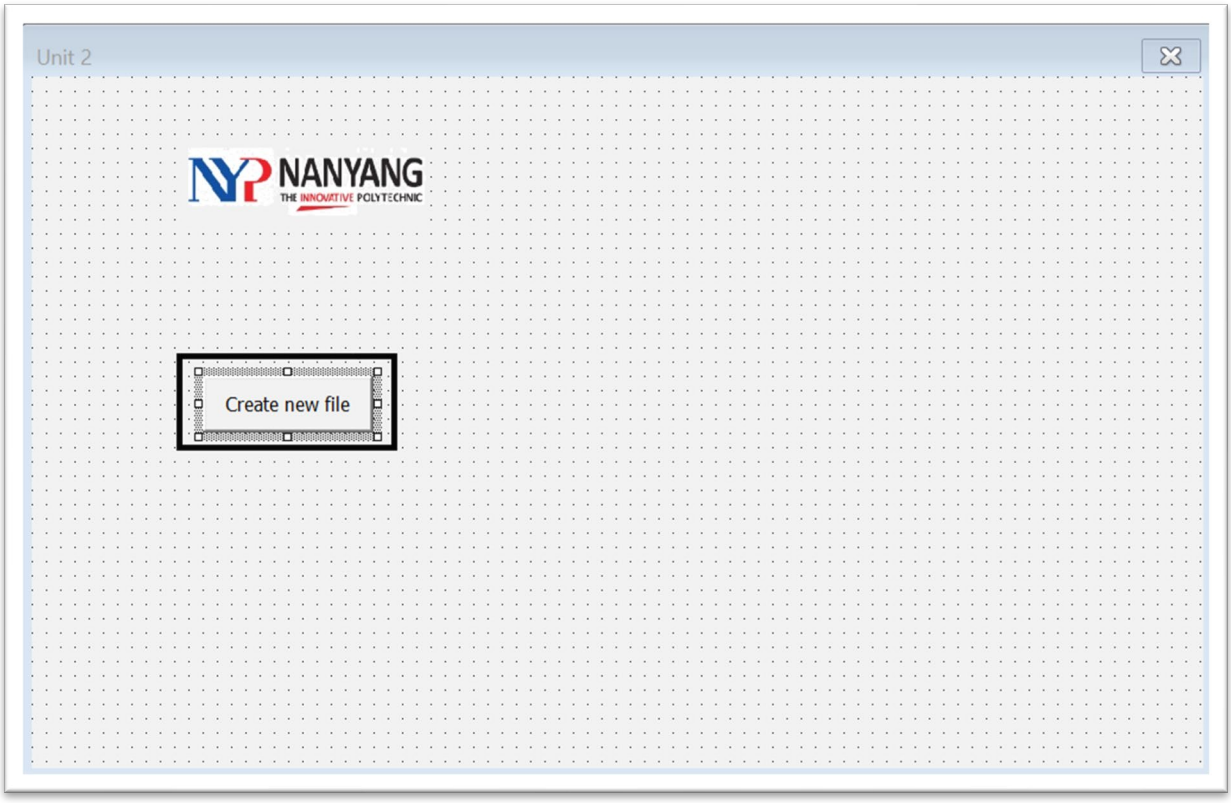


Choose a picture (Bitmap)

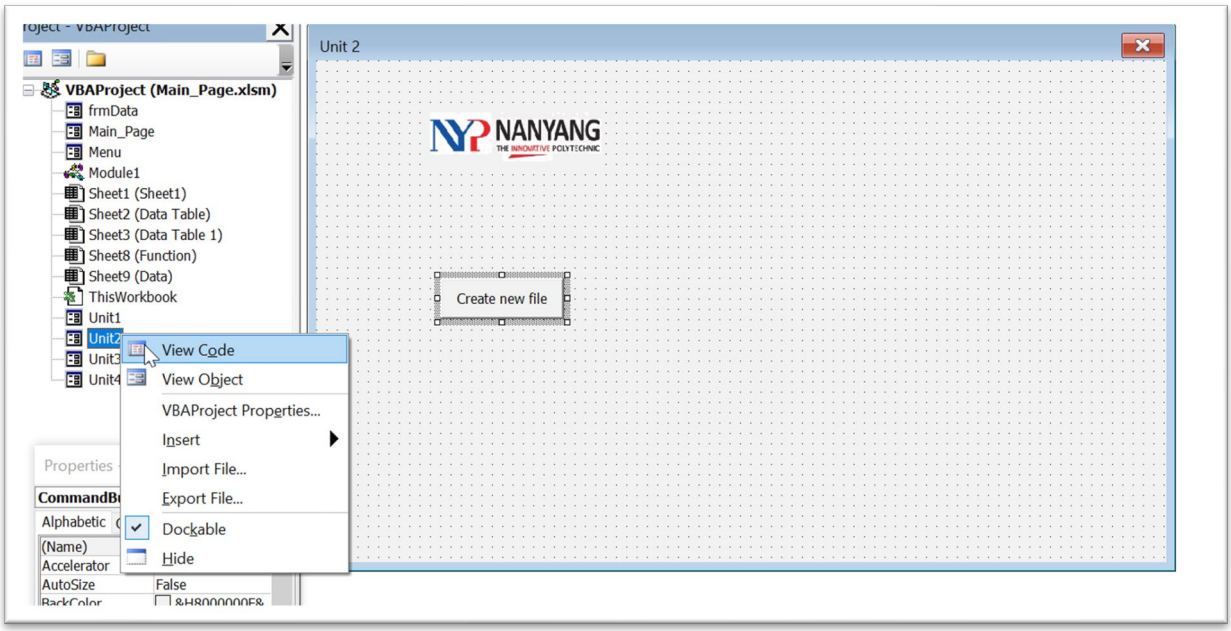


Here we choose the picture bitmap as below:

Insert a commandbutton



Right click and select View Code

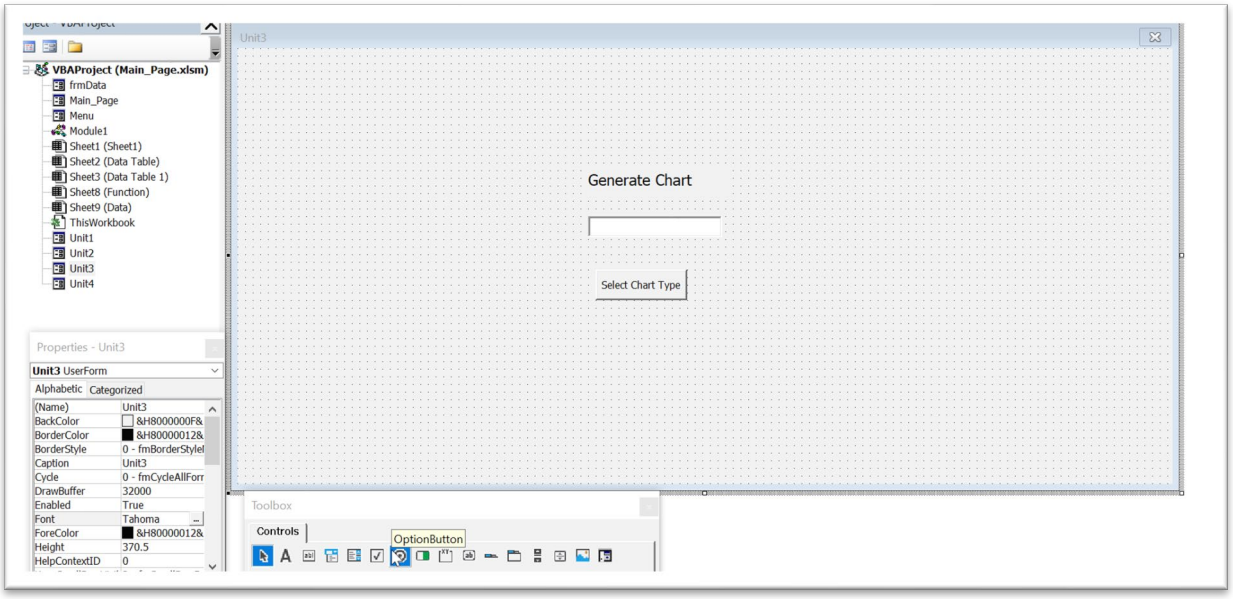



```
Private Sub CommandButton2_Click()
Set fs = CreateObject("Scripting.FileSystemObject")
Set oFile = fs.CreateTextFile("C:\Users\limhweetec\Documents\Short
Courses\Navy\testfile.txt", True)
oFile.WriteLine ("This is a test." & vbCrLf)
```

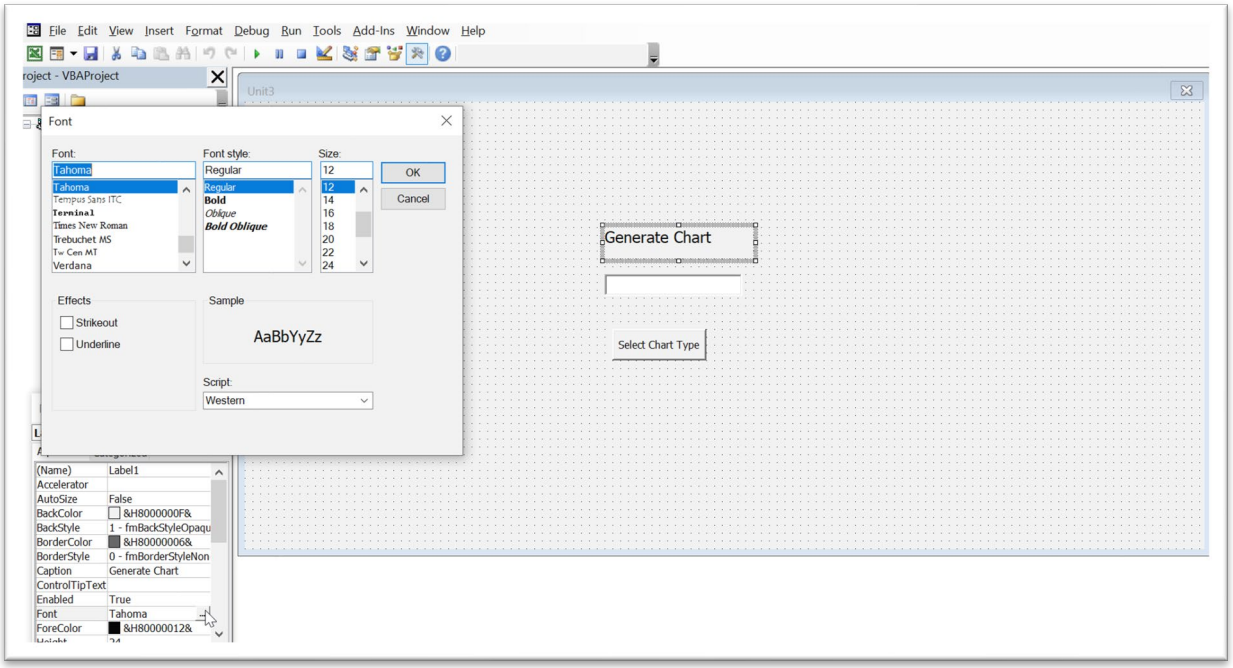
oFile.Close
Call TextStreamTest
End Sub

Solution for Exercise 3

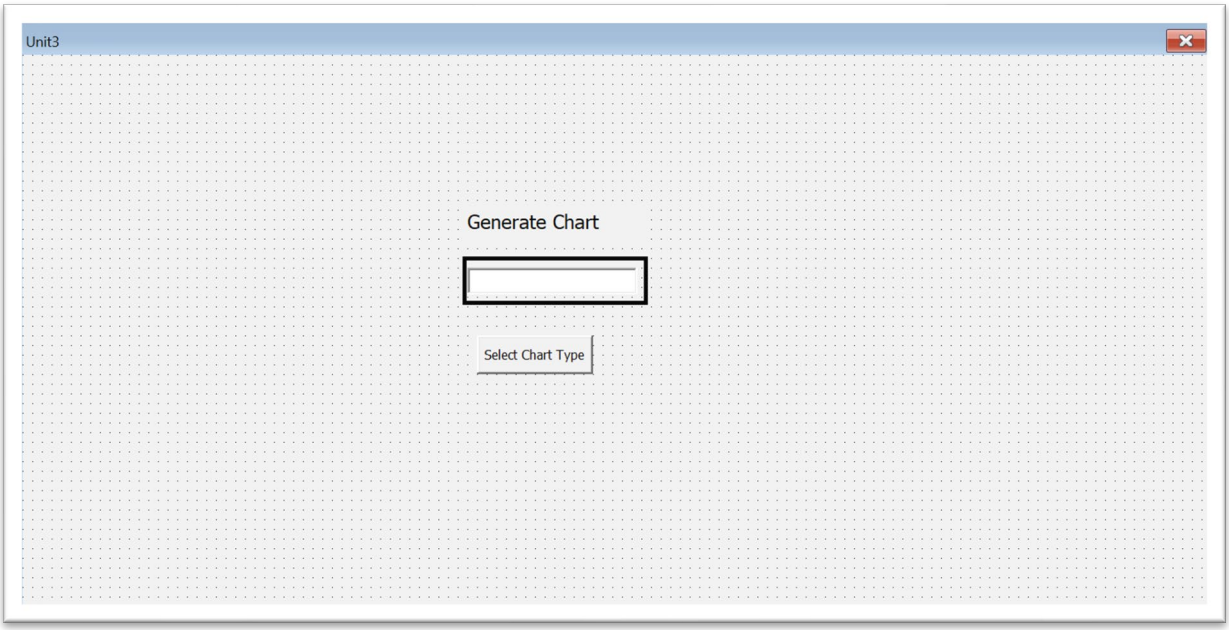
Create a userform with a label, a listbox and a commandbutton



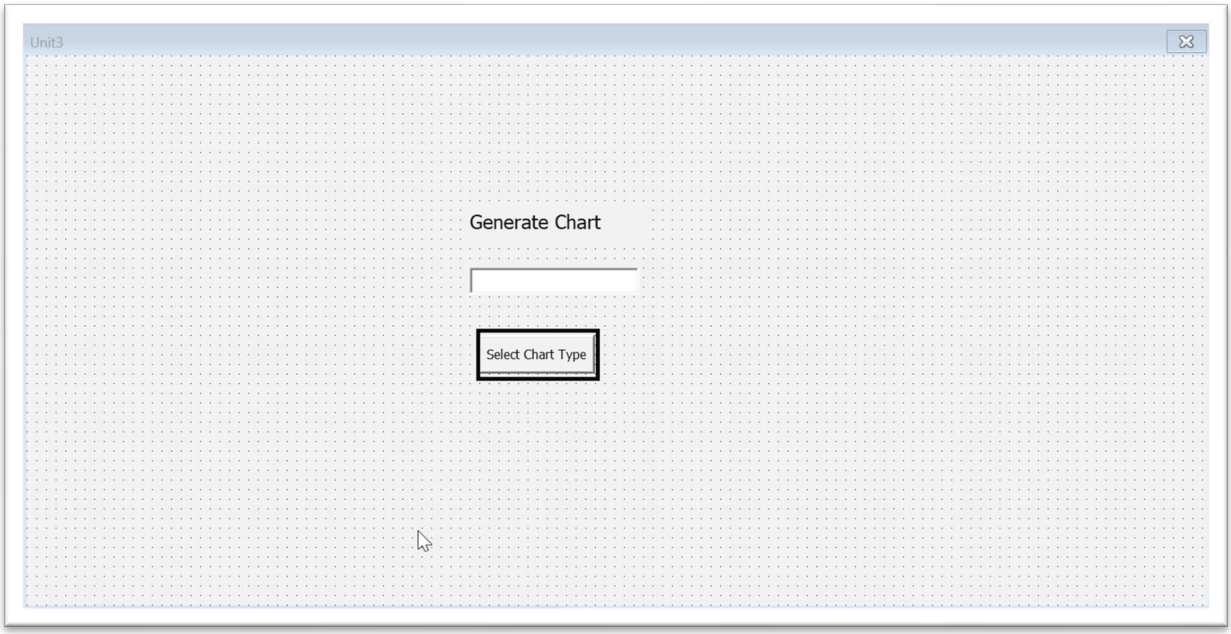
Change the label text font size to 12



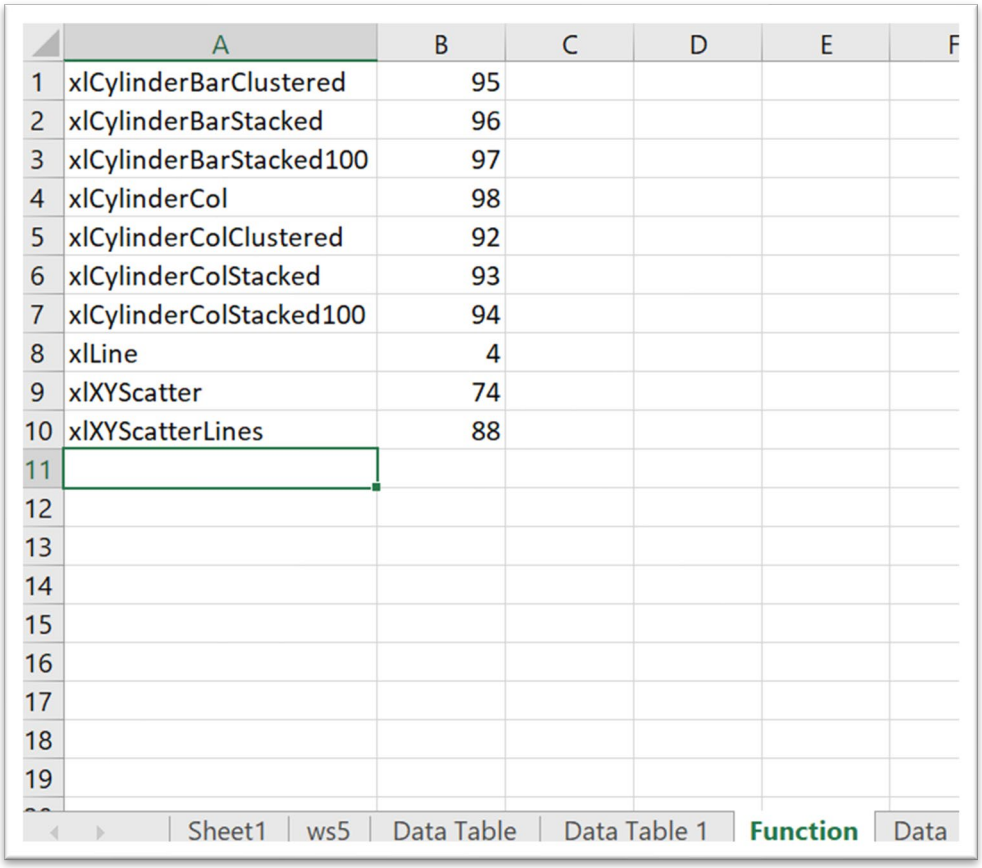
Insert a listbox



Insert a commandbutton and name the caption to Select Chart Type

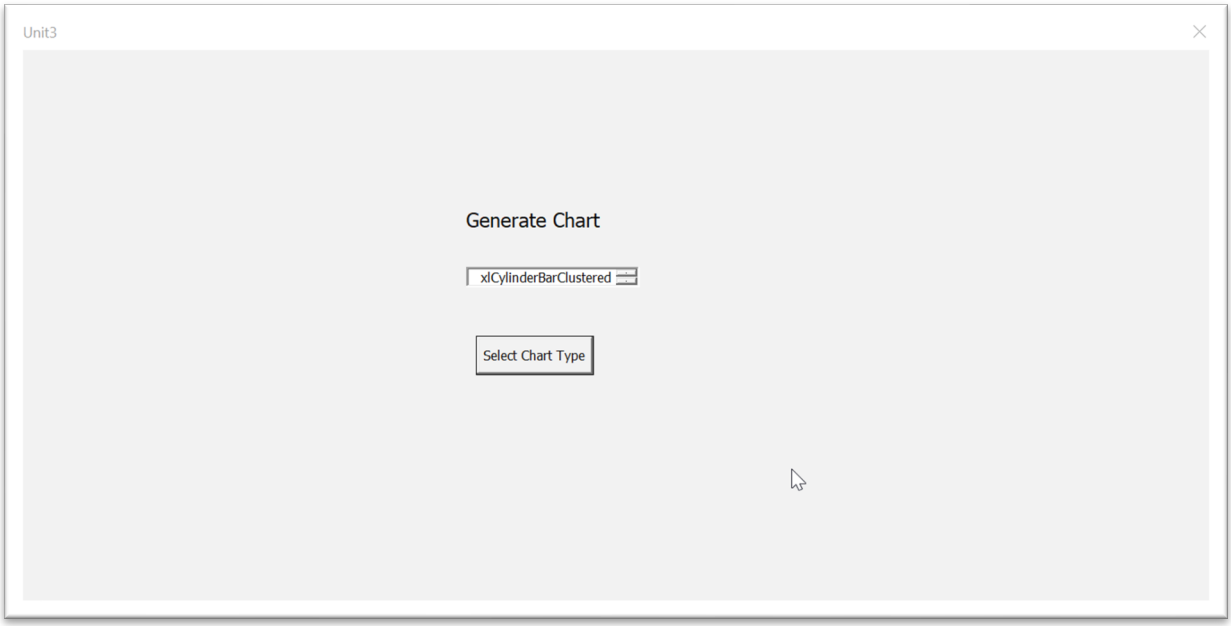


Create a worksheet with list of functions and name the worksheet as Function



	A	B	C	D	E	F
1	xlCylinderBarClustered	95				
2	xlCylinderBarStacked	96				
3	xlCylinderBarStacked100	97				
4	xlCylinderCol	98				
5	xlCylinderColClustered	92				
6	xlCylinderColStacked	93				
7	xlCylinderColStacked100	94				
8	xlLine	4				
9	xlXYScatter	74				
10	xlXYScatterLines	88				
11						
12						
13						
14						
15						
16						
17						
18						
19						

Sheet1 | ws5 | Data Table | Data Table 1 | **Function** | Data

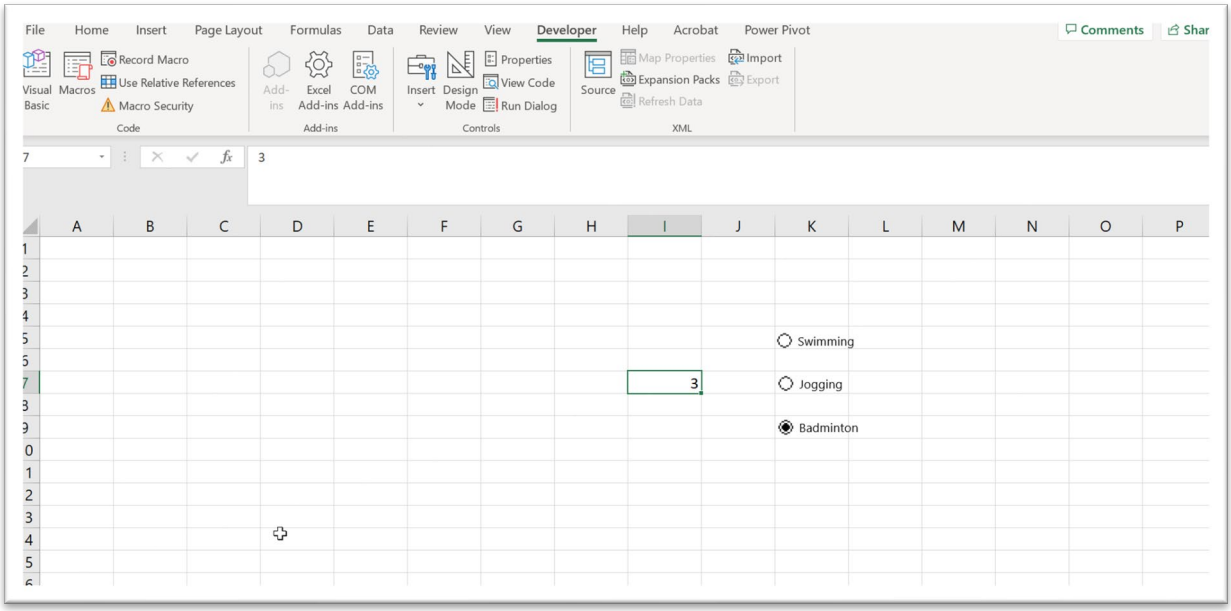


```
Rightclick on the userform and select View Code

Dim ISelection As Integer

Sub CommandButton1_Click()

' MsgBox ListBox1.Value
' Call CreateChart

Dim rng As Range
Dim cht As Object

'Your data range for the chart
 Set rng = Worksheets("Data Table").Range("C5:E7")

'Create a chart
 Worksheets.Add.Name = "ws" & Worksheets.Count

 Set cht = ActiveSheet.Shapes.AddChart2

'Give chart some data
 cht.Chart.SetSourceData Source:=rng

'Determine the chart type
 cht.Chart.ChartType = ISelection

End Sub


Sub ListBox1_Click()

Select Case Unit3.ListBox1.Value
Case "xlCylinderBarClustered"
   ISelection = 95
Case "xlCylinderBarStacked"
   ISelection = 96
Case "xlCylinderBarStacked100"
   ISelection = 97
Case "xlCylinderCol"
   ISelection = 98
Case "xlCylinderColClustered"
   ISelection = 92
Case "xlCylinderColStacked"
   ISelection = 93
Case "xlCylinderColStacked100"
   ISelection = 94
Case "xlLine"
```

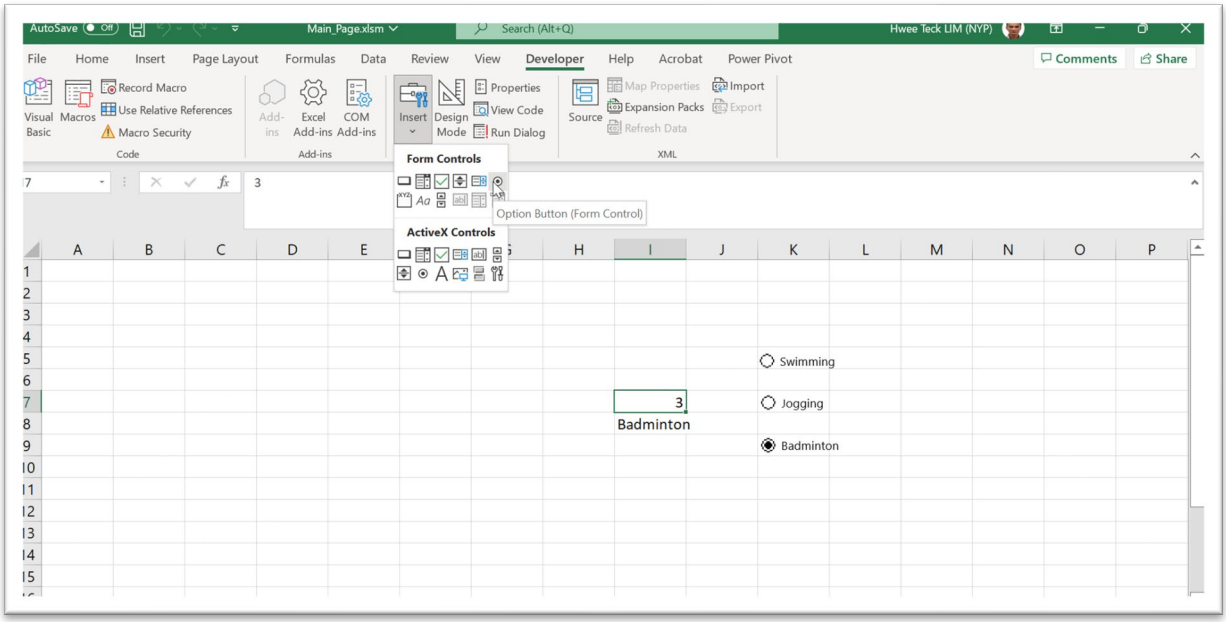
```
    ISelection = 4
Case "xlXYScatter"
    ISelection = 74
Case "xlXYScatterline"
     ISelection = 88
End Select

End Sub

Sub UserForm_Initialize()
Dim rg As Range
 Set rg = Worksheets("Function").Range("A1:A10")

 ListBox1.RowSource = rg.Address

End Sub
```

Solution for Exercise 4

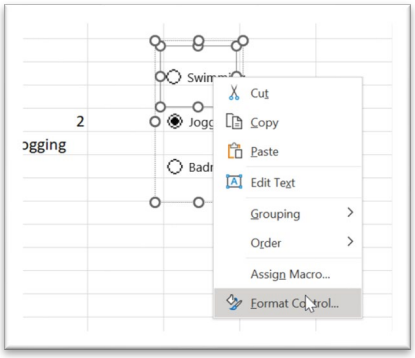
Create a worksheet with OptionButton
Add a new worksheet
Click on Developer



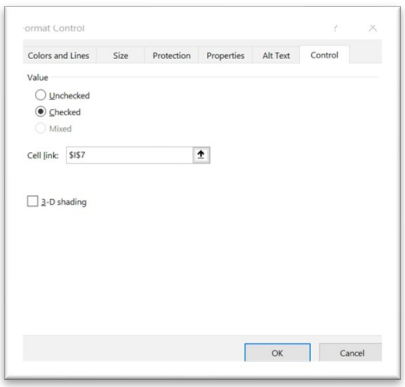
Insert OptionButton three times



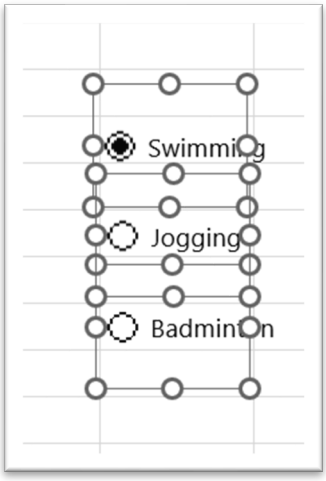
Right click on Option Button and select Format Control



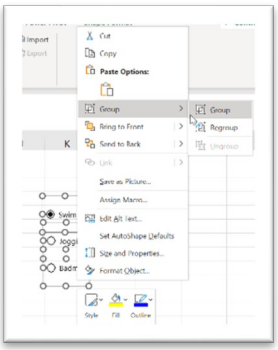
Click and enter the cell link. Repeat the step for the other two option buttons



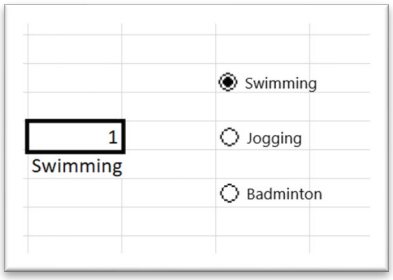
Press Control and Select all the three Option Buttons.



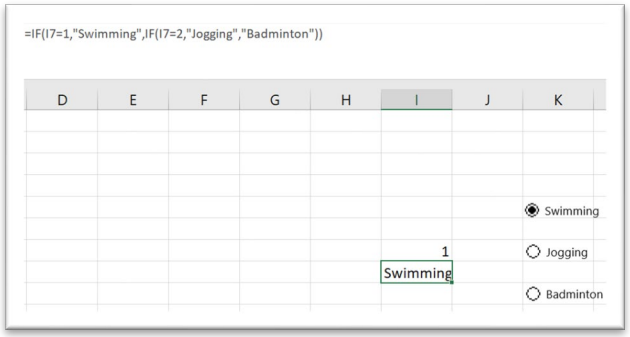
Right click and select Group-> Group



When the option button is clicked, the number will appear at I7.

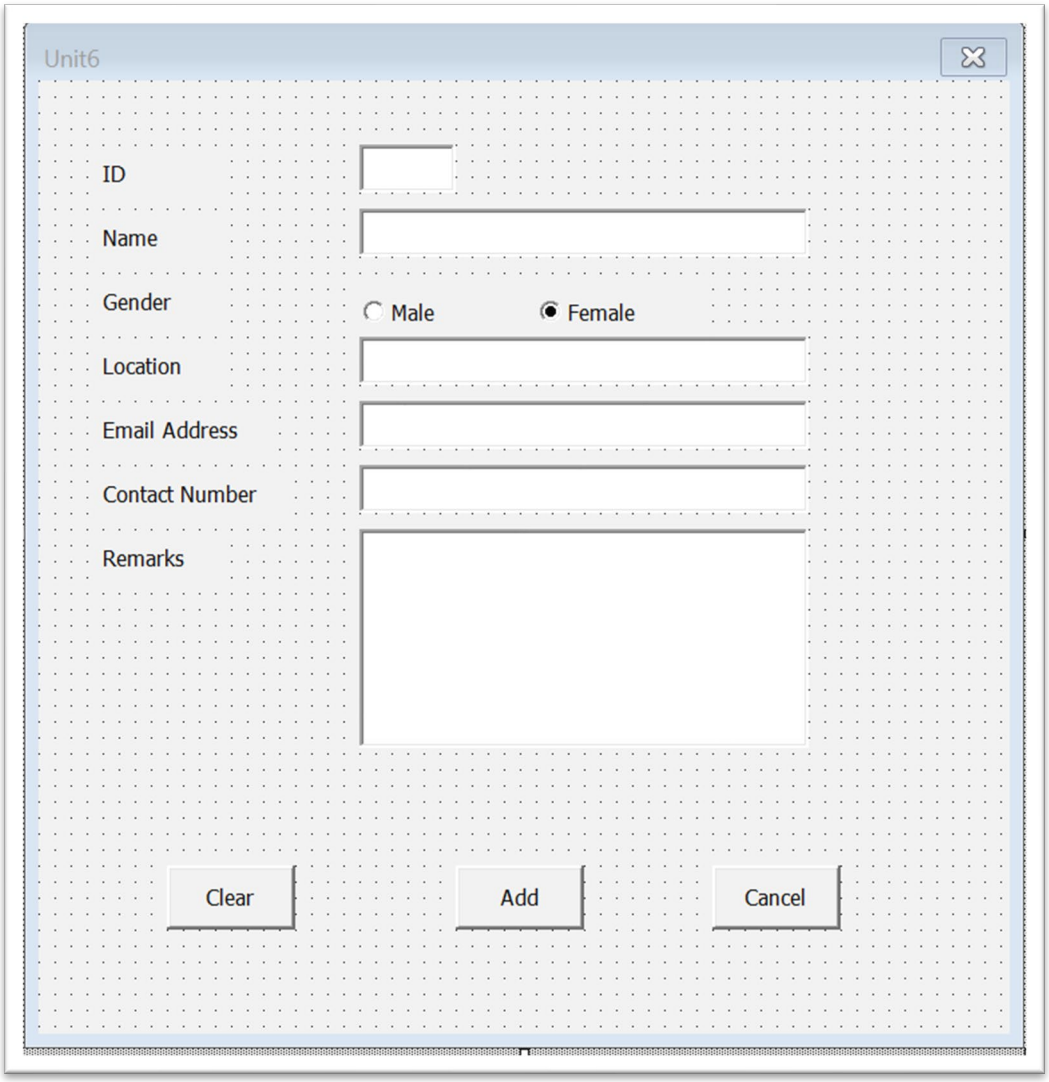


Enter the formula =IF(I7=1,"Swimming",IF(I7=2,"Jogging","Badminton")) at cell I8



Solution for Exercise 5

Add a new userform. Insert and layout the labels, textboxes,commnandbuttons and optionbuttons as per your design




```
Private Sub cmdAdd_Click()
 On Error GoTo ErrOccured
   'Boolean Value
   BlnVal = 0

   'Data Validation
   Call Data_Validation

   'Check validation of all fields are completed are not
   'If BlnVal = 0 Then Exit Sub

   'TurnOff screen updating
   With Application
      .ScreenUpdating = False
```

```
    .EnableEvents = False
  End With

  'Variable declaration
  Dim txtId, txtName, GenderValue, txtLocation, txtCNum, txtEAddr, txtRemarks
  Dim iCnt As Integer

  'find next available row to update data in the data worksheet
  iCnt = fn_LastRow(Sheets("Data")) + 1

  'Find Gender value
  If frmData.obMale = True Then
    GenderValue = "Male"
  Else
    GenderValue = "Female"
  End If

  'Update userform data to the Data Worksheet
  With Sheets("Data")
    .Cells(iCnt, 1) = iCnt - 1
    .Cells(iCnt, 2) = frmData.txtName.Value

    .Cells(iCnt, 3) = GenderValue
    .Cells(iCnt, 4) = frmData.txtLocation.Value
    .Cells(iCnt, 5) = frmData.txtAddress.Value
    .Cells(iCnt, 6) = frmData.txtCNumber.Value
    .Cells(iCnt, 7) = frmData.txtRemarks.Value

Application.ScreenUpdating = False
    frmData.txtId.Value = ""
    frmData.txtName.Value = ""
    frmData.obMale.Value = True
    frmData.txtLocation.Value = ""
    frmData.txtAddress.Value = ""
    frmData.txtCNumber.Value = ""
    frmData.txtRemarks.Value = ""
  Application.ScreenUpdating = True


    'Diplay headers on the first row of Data Worksheet
    If .Range("A1") = "" Then
      .Cells(1, 1) = "Id"
      .Cells(1, 2) = "Name"
      .Cells(1, 3) = "Gender"
      .Cells(1, 4) = "Location"
      .Cells(1, 5) = "Email Addres"
      .Cells(1, 6) = "Contact Number"
```

```
        .Cells(1, 7) = "Remarks"

        'Formatiing Data
        .Columns("A:G").Columns.AutoFit
        .Range("A1:G1").Font.Bold = True
        .Range("A1:G1").LineStyle = xlDash

    End If
  End With


'Display next available Id number on the Userform
'Variable declaration
  Dim IdVal As Integer

  'Finding last row in the Data Sheet
  IdVal = fn_LastRow(Sheets("Data"))

  'Update next available id on the userform
  frmData.txtId = IdVal
ErrOccured:
  'TurnOn screen updating
  Application.ScreenUpdating = True
  Application.EnableEvents = True

End Sub

Private Sub cmdCancel_Click()
  Unload Me

End Sub

Private Sub cmdClear_Click()
 Application.ScreenUpdating = False
    txtId.Value = ""
    txtName.Value = ""
    obMale.Value = True
    txtLocation.Value = ""
    txtAddress.Value = ""
    txtCNumber.Value = ""
    txtRemarks.Value = ""
  Application.ScreenUpdating = True
End Sub

Private Sub UserForm_Initialize()
'Variable declaration
  Dim IdVal As Integer
```

```
    'Finding last row in the Data Sheet
    IdVal = fn_LastRow(Sheets("Data"))

    'Update next available id on the userform
    frmData.txtId = IdVal

End Sub

Function fn_LastRow(ByVal Sht As Worksheet)

    Dim lastRow, lRow As Long
    lastRow = Sht.Cells.SpecialCells(xlLastCell).Row
    lRow = Sht.Cells.SpecialCells(xlLastCell).Row
    Do While Application.CountA(Sht.Rows(lRow)) = 0 And lRow <> 1
        lRow = lRow - 1
    Loop
    fn_LastRow = lRow

End Function
```

-End-

NANYANG THE INNOVATIVE POLYTECHNIC School of Information Technology

Project Discussion 2

WORKSHOP FOR NAVY

14 DECEMBER 2023

Learning Outcomes:

This session requires you to use what you have learnt for VBA in the past 2 days and apply it.

a) To familiarise yourself with the VBA IDE
b) Using VBA programming to implement certain functions for future excel needs
c) Assigning created macros to form controls

Software(s): Microsoft Excel 2019

Instructions:

1. In pairs, discuss and write out the VB codes using the VBA IDE in Excel.
2. The descriptions are listed and the Macro name provided. Write as many as you can and at the end of the discussion, share your findings with the class.

Writing Basic Codes

- Write macros that does the following:
    - Automatically add serial numbers ("S/No") to the selected cell in the worksheet starting from "1" to "N".
        - **AddSerialNumbers()**:
            - This macro is useful for you to serialise the number from the selected cell and should show a message box where the highest number for the serial numbers is entered.
    - Add multiple columns to the right of the selected cell in a single click.
        - **InsertMultipleColumns():**
            - When this code is run, it will prompt for the number of columns that you want to add to the right of the selected cell.
            - What must you change to if you want it to be to the left of the selected cell?
    - Add multiple rows to the selected cell below.
        - **InsertMultipleRows():**
            - When this code is run, it will prompt for the number of rows that you want to insert below of the selected cell.
            - What must you change to if you want to insert on top of the selected cell?
    - Auto fit all the columns in your worksheet.
        - **AutoFitColumns():**
            - When this code is run, all the cells in the worksheet will be selected and instantly auto-fit all the columns
    - Open a calculator in Excel.
        - **OpenCalculator():**

- Open a calculator in Windows Excel.
- *[Hint:] Use "Application.ActivateMicrosoftApp"*

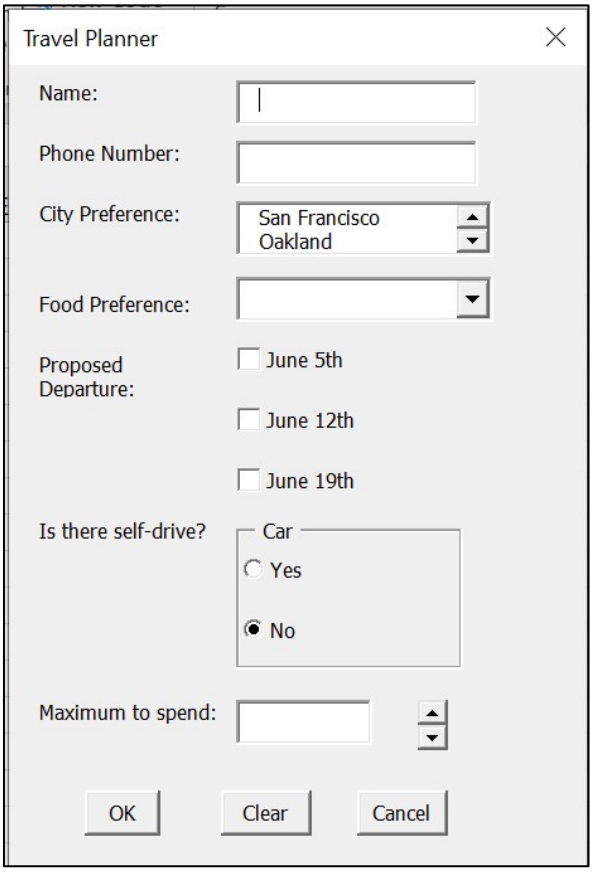
- Add date to the header and also page number to the footer
  - **DateInHeader():**
    - When this code is run, current system date (use tag "&D") will be added to the centre of page. You can check by "printing" the active worksheet.
    - Try now putting the page number at the center of the footer using "&P".

Formatting Codes

- Write macros that format cells and ranges using some specific criteria and conditions.
  - Highlight the cells that have duplicate values with different colour.
    - **HighlightDupVal()**:
      - This macro will check each cell in the selection and highlight the duplicate values with yellow.
  - Highlight the top 10 values with the green colour.
    - **HighlightTopTen()**:
      - Just use this macro to highlight in green the top 10 values of the selected range.
  - Highlight negative numbers.
    - **HighlightNegativeNumbers():**
      - By selecting a range of cells, this macro will highlight the cells who have negative numbers in red.
  - Highlight cells with a specific text in a worksheet.
    - **HighlightSpecificValues():**
      - This code will count the cells which have a specific value that you have input, and the cells will also be highlighted
  - Highlight the cell with the maximum value.
    - **HighlightMaxValue():**
      - This code checks the selected cells and highlight the cell with the maximum value
      - For minimum value, you can use *WorksheetFunction.Min(Selection)*

Creating Forms for Input

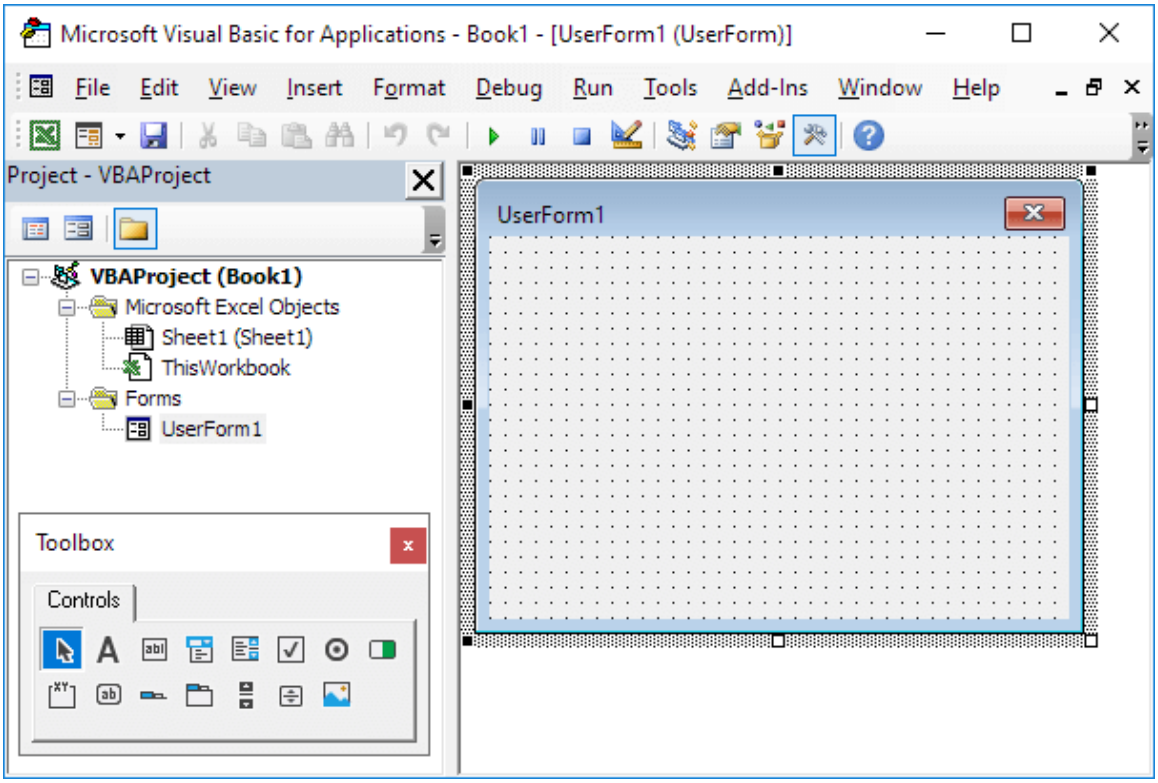
3. In pairs, discuss and create userform for input with requirements shown below.
4. Create a Userform that looks like the following:



Add the Controls

To add the controls to the Userform, execute the following steps.

1. Open the Visual Basic Editor. If the Project Explorer is not visible, click View, Project Explorer.
2. Click Insert, Userform. If the Toolbox does not appear automatically, click View, Toolbox. Your screen should be set up as below.



3. Add the controls listed in the table below. Once this has been completed, the result should be consistent with the picture of the Userform shown earlier. For example, create a text box control by clicking on TextBox from the Toolbox. Next, you can drag a text box on the Userform. When you arrive at the Car frame, remember to draw this frame first before you place the two option buttons in it.

4. Change the names and captions of the controls according to the table below. Names are used in the Excel VBA code. Captions are those that appear on your screen. It is good practice to change the names of controls. This will make your code easier to read. To change the names and captions of the controls, click View, Properties Window and click on each control.

Control	Name	Caption
Userform	TravelPlannerUserForm	Travel Planner
Text Box	NameTextBox	
Text Box	PhoneTextBox	
List Box	CityListBox	
Combo Box	DinnerComboBox	

Check Box	DateCheckBox1	June 5th
Check Box	DateCheckBox2	June 12th
Check Box	DateCheckBox3	June 19th
Frame	CarFrame	Car
Option Button	CarOptionButton1	Yes
Option Button	CarOptionButton2	No
Text Box	MoneyTextBox	
Spin Button	MoneySpinButton	
Command Button	OKButton	OK
Command Button	ClearButton	Clear
Command Button	CancelButton	Cancel
7 Labels	No need to change	Name:, Phone Number:, etc.

Show the Userform

1. To show the Userform, place a command button on your worksheet and add the following code line:

```
Private Sub CommandButton1_Click()

TravelPlannerUserForm.Show

End Sub
```

2. We are now going to create the Sub UserForm_Initialize. When you use the Show method for the Userform, this sub will automatically be executed.
3. Open the Visual Basic Editor.
4. In the Project Explorer, right click on TravelPlannerUserForm and then click View Code.
5. Choose Userform from the left drop-down list. Choose Initialize from the right drop-down list.
6. Add the following code lines:

```
Private Sub UserForm_Initialize()

'Empty NameTextBox
NameTextBox.Value = ""

'Empty PhoneTextBox
PhoneTextBox.Value = ""

'Empty CityListBox
CityListBox.Clear

'Fill CityListBox
With CityListBox
   .AddItem "San Francisco"
   .AddItem "Oakland"
   .AddItem "Richmond"
End With

'Empty DinnerComboBox
DinnerComboBox.Clear

'Fill DinnerComboBox
With DinnerComboBox
   .AddItem "Italian"
   .AddItem "Chinese"
   .AddItem "Frites and Meat"
End With

'Uncheck DataCheckBoxes
DateCheckBox1.Value = False
DateCheckBox2.Value = False
DateCheckBox3.Value = False

'Set no car as default
CarOptionButton2.Value = True

'Empty MoneyTextBox
MoneyTextBox.Value = ""

'Set Focus on NameTextBox
NameTextBox.SetFocus

End Sub
```

Explanation: This initialisation ensures that the Text boxes are emptied, list boxes and combo boxes are filled, check boxes are unchecked, etc.

Assign the Macros

We have now created the first part of the Userform. Although it looks neat already, nothing will happen yet when we click the command buttons on the Userform.

1. Open the Visual Basic Editor.
2. In the Project Explorer, double click on TravelPlannerUserForm.
3. Double click on the Money spin button.
4. Add the following code line:

```
Private Sub MoneySpinButton_Change()

MoneyTextBox.Text = MoneySpinButton.Value

End Sub
```

Explanation: This code updates the text box when you use the spin button.

1. Double click on the OK button.
2. Add the following code lines:

```
Private Sub OKButton_Click()

Dim emptyRow As Long

'Make Sheet1 active
Sheet1.Activate

'Determine emptyRow
emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1

'Transfer information
Cells(emptyRow, 1).Value = NameTextBox.Value
Cells(emptyRow, 2).Value = PhoneTextBox.Value
Cells(emptyRow, 3).Value = CityListBox.Value
Cells(emptyRow, 4).Value = DinnerComboBox.Value

If DateCheckBox1.Value = True Then Cells(emptyRow, 5).Value = DateCheckBox1.Caption

If DateCheckBox2.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value
& " " & DateCheckBox2.Caption

If DateCheckBox3.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value
& " " & DateCheckBox3.Caption
```

```
If CarOptionButton1.Value = True Then
   Cells(emptyRow, 6).Value = "Yes"
Else
   Cells(emptyRow, 6).Value = "No"
End If

Cells(emptyRow, 7).Value = MoneyTextBox.Value

End Sub
```

Explanation: We have to activate a sheet and in this case we are activating “Sheet1”. Next, we determine emptyRow. The variable emptyRow is the first empty row and increases every time a record is added. Finally, we transfer the information from the Userform to the specific columns of emptyRow.

1. Double click on the Clear button.
2. Add the following code line:

```
Private Sub ClearButton_Click()

Call UserForm_Initialize

End Sub
```

Explanation: This code calls the Sub UserForm_Initialize when you click on the Clear button.

1. Double click on the Cancel Button.
2. Add the following code line:

```
Private Sub CancelButton_Click()
Unload Me
End Sub
```

Explanation: This code closes the Userform when you click on the Cancel button.

Test the Userform

Exit the Visual Basic Editor, enter the labels shown below into row 1 and test the Userform.

Result:

	A	B	C	D	E	F	G
1	**Name**	**Phone Number**	**City**	**Food Preference**	**Departure Date**	**Car**	**Maximum to Spend**
2	Angie Tan	+65 8114 2334	San Francisco	Chinese	June 12th	Yes	2000
3	Leonard Pang	+65 9238 2156	Richmond	Italian	June 5th June 19th	No	1500
4							

-End-