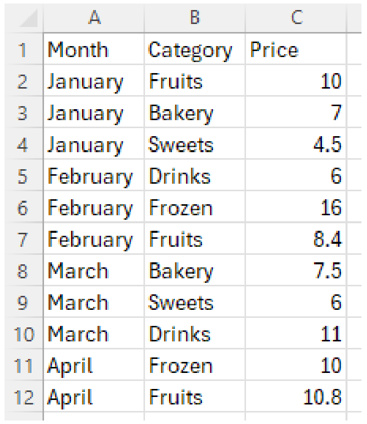
EXCEL AND POWERQUERY CHEAT SHEET

Excel

DUMMY DATA

Some concepts will be best explained via an example, in which case, the following dummy data will be used:



	A	B	C
1	Month	Category	Price
2	January	Fruits	10
3	January	Bakery	7
4	January	Sweets	4.5
5	February	Drinks	6
6	February	Frozen	16
7	February	Fruits	8.4
8	March	Bakery	7.5
9	March	Sweets	6
10	March	Drinks	11
11	April	Frozen	10
12	April	Fruits	10.8

It shows purchases of foods by category over a range of months.

A SMALL NOTE ON DEFAULT CELL ALIGNMENT

Strings are left-aligned by default.
Numeric types and Dates are right-aligned by default.

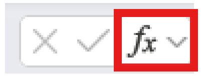
RELATIVE V.S. ABSOLUTE REFERENCING

Formulas (obtained by starting cell entry with "=") can reference other cells, either relatively or absolutely:

- Relative : Refers to a cell relative to the current position, adjusting as the formula is moved. Think: I want to reference the cell "X cells across and Y cells up."
- Absolute: refers to a fixed cell that doesn't change when the formula is moved, using dollar signs ($). For example, A1 always refers to cell A1.

Note: You can lock only the row or column by placing the dollar sign ($) before only one of them e.g., $A1 or A$1.

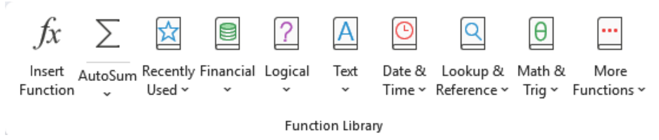
BASIC FORMULAS/FUNCTIONS



The insert function button

Functions can be added by typing "= ..." or pressing the Insert Function button and selecting a function. The Insert Function button also acts as a function helper when pressed whilst the cursor is inside the function argument brackets.
Functions may also be found by navigating in the ribbon:

Formulas → Function Library



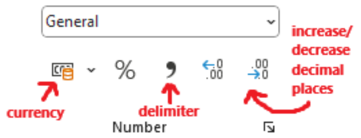
Formulas may be evaluated step-by-step (to assist with debugging, for example) via the use of

Formulas → Formula Auditing → Evaluate Formula

FORMATTING

- To format a cell as a number, navigate to

  Home → Number



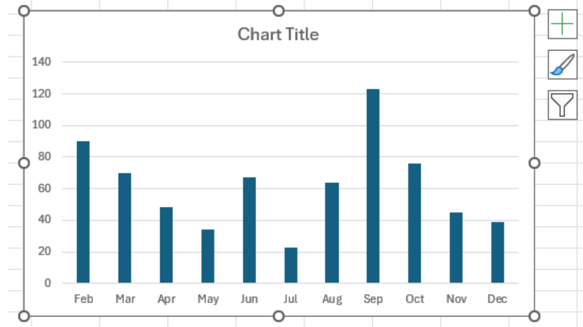
- Use the format painter (**Home → Clipboard → Format Painter**) to paint the format of the current cell onto other cells. Click once to apply once, double-click to keep active.
- Cell styles can be used/created (i.e. background colour, text colour, number format) via **Home → Styles → Cell Styles...**. Modifying a style will automatically change all cells with that style applied.
- Conditionally formatting (accessed via **Home → Styles → Conditional Formatting**) applies formatting based on conditionals. These can be edited via
  ··· → **Conditional Formatting → Manage Rules**

BASIC CHARTS

Charts may be found in the ribbon via:

Insert → Charts

- Highlight cells/select a table and choose a chart
- When selecting a chart, the Chart Design and Format tabs appear.
- Chart Design allows selection of chart style, change the data selected, change chart type etc.
- Format allows fonts to be changed, as well as the addition of shapes etc.
- Also when selecting a chart, icons allowing the addition of Chart Elements, the changing of Chart styles, and application of Chart Filters (e.g. filter some rows/columns of data out) appear.



EXCEL LISTS (SORTING AND FILTERING)

These are single columns of related data (e.g. a list of names). Excel will automatically look for headers in the first row. This is useful for sorting and filtering.

Sorting

- To sort column(s):
- Click into the column/highlight columns/click into table
- Navigate to **Data** → **Sort & Filter**
- To sort A-Z or Z-A, select the icon
- For more complicated sorts (e.g. sorting abbreviated months chronologically, click **Sort** and Order by Custom List...
- Can add multiple (levels of) sorts, they will be carried out from top downwards.

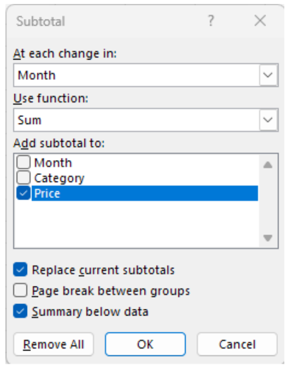
Filtering

- To filter data in column(s):
- Click into the column/highlight columns/click into table
- Navigate to **Data** → **Sort & Filter** → **Filter**
- Dropdowns will appear in the cells of the column header(s)
- Filters can be cleared via **Data** → **Sort & Filter** → **Clear**

SUBTOTAL

Used to answer questions like "What were the total sale amounts for each item?" or "How much did I spend in each month?"

- Sort the column we wish to take subtotals with respect to (this is because subtotals works by detecting changes in values in this column).
- highlight or click into list/range somewhere. Note that subtotal cannot be used with tables.
- **Data** → **Outline** → **Subtotal** gives the window in figure below (left).
- The "Use function:" dropdown lets you choose an aggregate function to use e.g. sum, average, mean...
- The "Add subtotal to:" box lets you select which column to apply subtotals to. In our dummy data, summing price is a good choice. (see figure, the left hand image)
- The result of "Ok" can be seen in the figure below (on the right).
- Note: The 1,2,3, +, and - buttons can be used to expand, and condense groupings to various levels.



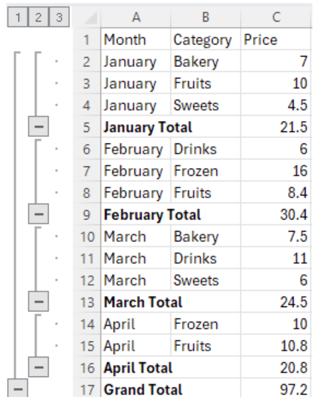
Subtotal

At each change in: Month

Use function: Sum

Add subtotal to:
- [ ] Month
- [ ] Category
- [✓] Price

- [✓] Replace current subtotals
- [ ] Page break between groups
- [✓] Summary below data



	A	B	C
1	Month	Category	Price
2	January	Bakery	7
3	January	Fruits	10
4	January	Sweets	4.5
5	**January Total**		21.5
6	February	Drinks	6
7	February	Frozen	16
8	February	Fruits	8.4
9	**February Total**		30.4
10	March	Bakery	7.5
11	March	Drinks	11
12	March	Sweets	6
13	**March Total**		24.5
14	April	Frozen	10
15	April	Fruits	10.8
16	**April Total**		20.8
17	**Grand Total**		97.2

FORMAT AS TABLE

Explicitly formatting as a table (as opposed to just having the data as a collection of columns) has many benefits, including

- Allows addition of counts which respect filtering
- Allows, for example, alternating colouring of rows, which doesn't get destroyed by sorting, which improves readability

To achieve it, click into the "table" and navigate to

Home → **Styles** → **Format as Table**

Note that a Table Design tab has appeared.

REMOVING DUPLICATES

To remove duplicates from a table:

Table Design → **Tools** → **Remove Duplicates**

To remove duplicates from lists of data (i.e. data which is not formatted as a table):

Data → **Data Tools** → **Remove Duplicates**

DATABASE FUNCTIONS (DSUM, DCOUNT,...)

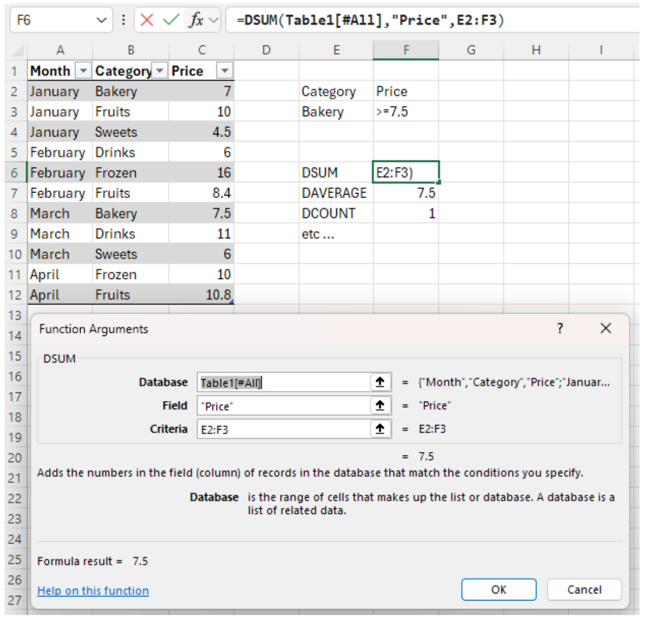
These functions apply a function to a specified field/column that match given conditions/criteria. A list of such functions is:

- DSUM
- DCOUNT
- DAVERAGE
- DMIN, DMAX
- More can be found by clicking **Insert Function** and querying for "database".

All such functions require the following arguments:

- Database, Field, Criteria
- Database: This is the range of cells that make up the data
- Field: Label of the column the function should be applied to
- Criteria: A "mini-table" containing column headers and conditions that entries must satisfy to be included in the main function

Here is an example of applying some database functions to the dummy data. Notice that E2:F3 is a "mini-table", created to house the criteria:



F6 =DSUM(Table1[#All],"Price",E2:F3)

	A	B	C	D	E	F
1	Month	Category	Price			
2	January	Bakery	7		Category	Price
3	January	Fruits	10		Bakery	>=7.5
4	January	Sweets	4.5			
5	February	Drinks	6			
6	February	Frozen	16		DSUM	E2:F3)
7	February	Fruits	8.4		DAVERAGE	7.5
8	March	Bakery	7.5		DCOUNT	1
9	March	Drinks	11		etc ...	
10	March	Sweets	6			
11	April	Frozen	10			
12	April	Fruits	10.8			

Function Arguments

DSUM

Database	Table1[#All]	= {"Month","Category","Price";"Januar...
Field	"Price"	= "Price"
Criteria	E2:F3	= E2:F3

= 7.5

Adds the numbers in the field (column) of records in the database that match the conditions you specify.

Database is the range of cells that makes up the list or database. A database is a list of related data.

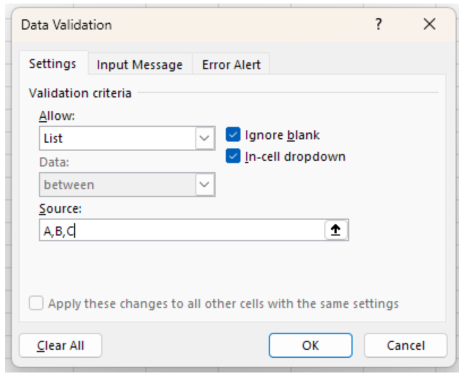
Formula result = 7.5

DATA VALIDATION

Data Validation allows us to set rules that entries to a cell must adhere to. It can be obtained by navigating to:

Data → Data Tools → Data Validation

For example, the following would only allow values from the list $[A, B, C]$ to be inputted into the cells. You can also change the cell input message and error message.


Data Validation

Settings | Input Message | Error Alert

Validation criteria

Allow: List

[✓] Ignore blank

[✓] In-cell dropdown

Data: between

Source: A,B,C

[ ] Apply these changes to all other cells with the same settings

Clear All | OK | Cancel

IMPORTING/EXPORTING DATA

Data can be imported by navigating to

Data → Get and Transform Data → Get Data

You can import data from many places, for example,

- Files e.g. csv, json, pdfs
- Databases e.g. Microsoft Access, MySQL
- Other Sources e.g. Table/Range

Importing data opens a window where there is a choice to Load, Load To, or Transform Data.

- Load - Load the data to a new worksheet
- Load To - allows loading to a specified location/worksheet. You can also load to a PivotTable report immediately.
- Transform Data - opens the data in PowerQuery.

To export data, navigate to

File → Save As

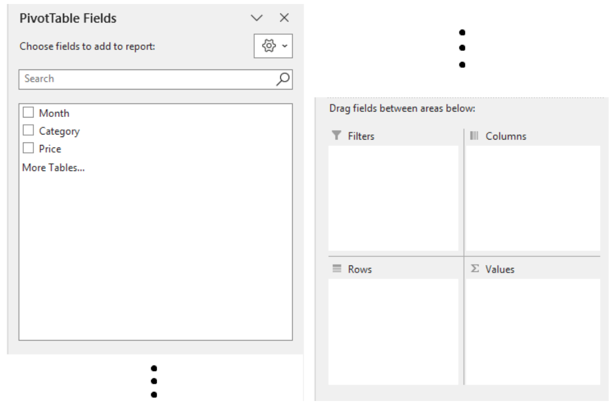
and choose the required file type from the dropdown.

PIVOT TABLES

Pivot Tables create aggregated values by grouping data from an original table. They are similar to applying an aggregate function, like SUM or COUNT, to data grouped by two or more columns. The key difference is that these field labels, used to define the groupings, appear on the rows and columns, creating a grid-like structure.

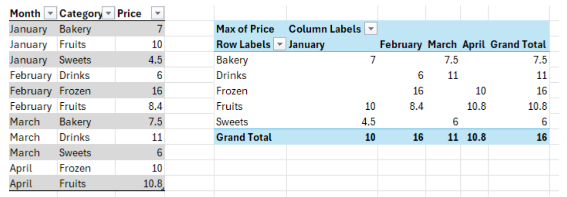
To use:

- Tip: It is better to format the range of data as a table.
- Navigate to **Insert → Tables → PivotTable** - select range of cells/table and the desired location of the pivot table.
- Using the new window on the right (see below image), drag column headers/field's into Filters/Columns/Rows/Values boxes to use them in the pivot table.



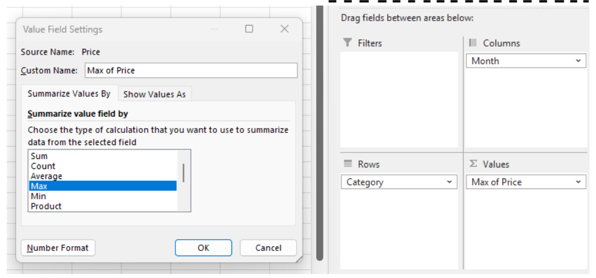
- Filters will use the data in the field's column as a filter. Row/Column uses the data in the field's column to group the data. Values uses the values in the field's column to aggregate (usually this is numeric).
- The "Value Field Settings" option for fields in the Values box allows the aggregate function to be chosen e.g. SUM, COUNT, AVERAGE etc.

For example, a pivotable for the dummy data, where data is grouped by Month on columns, and Category on rows, with aggregate data being max price:



Month	Category	Price
January	Bakery	7
January	Fruits	10
January	Sweets	4.5
February	Drinks	6
February	Frozen	16
February	Fruits	8.4
March	Bakery	7.5
March	Drinks	11
March	Sweets	6
April	Frozen	10
April	Fruits	10.8

Max of Price	Column Labels				
Row Labels	January	February	March	April	Grand Total
Bakery	7		7.5		7.5
Drinks		6	11		11
Frozen		16		10	16
Fruits	10	8.4		10.8	10.8
Sweets	4.5		6		6
Grand Total	**10**	**16**	**11**	**10.8**	**16**


Value Field Settings

Source Name: Price

Custom Name: Max of Price

Summarize Values By | Show Values As

Summarize value field by

Choose the type of calculation that you want to use to summarize data from the selected field

Sum
Count
Average
Max
Min
Product

Number Format | OK | Cancel

Drag fields between areas below:

Filters

Columns: Month

Rows: Category

Values: Max of Price

Grouping rows/columns within pivot tables

- Highlight columns/rows you want to group
- **PivotTable Analyse → Group → Group Selection**

Formatting pivot tables

- Warning: **Home → Number** formats the cells, not the table itself. If pivot table change size, all the cells may not have the formatting applied
- Instead use Value Field Settings → Number Format from within the Values box of the pivot table window.

Other aggregate functions e.g. lead/lag

- To obtain functions such as lead/lag/running total and more:
- Go to Value Field Settings → Show Values As and choose suitable options.

Drill down

Double-clicking a pivot table value produces a new table in a new worksheet displaying all data making up that value.

Charts

- To create charts using pivot table data:
- **PivotTable Analyse → Tools → PivotChart**

Filtering pivot tables

- Drag column header/field into the Filter box within the pivot table window.
- Use the dropdown menu that is now present in the pivot table to filter the data

Alternatively, a more interactive way to filter can be done using

PivotTable Analyse → Filter → Insert Slicer

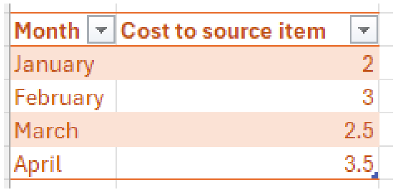
After selecting the column headers/fields you want to filter with respect to, an interactive window will appear, allowing you to filter items. This is ideal for dashboards.

POWER PIVOT

Power Pivot allows the creation of pivot tables using data from multiple tables that may be related in some way (this involves methods similar to that of an SQL Join).

- Need a data model to connect the tables
- To add tables to a Data Model, click into them and navigate to **PowerPivot → Tables → Add to Data Model**
- This opens a new window (PowerPivot for Excel) containing the table data
- Repeat this for all relevant tables.
- The tables are now present on different sheets within PowerPivot for Excel

For the rest of this section, all navigations occur in PowerPivot for Excel. We also need another table to link to our dummy data. For this we use the following, showing costs to source items for each month (the idea being that profit will be Price - Source Cost):



Month	Cost to source item
January	2
February	3
March	2.5
April	3.5

Creating relationships between tables

To create relationships/links between columns in different tables:

- **Home → Diagram View**
- Drag & drop column/field names to create a link (drag from the parent to the child)

Alternatively,

- **Home → Data View**
- Right click on the parent column header and select "Create Relationship"
- Select the table the desired child column belongs to from the dropdown menu and click the child column header.

PivotTable with linked tables

We can use pivottables using data from the linked tables together.

- **Home → PivotTable**
- This will return to Excel, choose where to place the table
- Now column headers/fields from all linked tables are available to use within the pivot table - they are able to "see each other" through the links between the tables.

PowerPivot KPI

Note: KPI stands for key performance indicator.

- Add an aggregate function to a column within PowerPivot (e.g. AVERAGE) using **Home → Calculations → AutoSum (dropdown)**.
- This is the value the indicator will be based on (e.g. is some value much lower, about equal or much greater than the average)
- Add KPI via **Home → Calculations → Create KPI**
- Add a target value and define thresholds (where red becomes yellow becomes green).
- To edit a KPI, right-click the cell with the aggregate function and select "Edit KPI settings"
- These KPI indicators are available and can be used in pivot tables.

TIPS FOR WORKING WITH LARGE DATA SETS

Freeze rows/columns

- **View → Window → Freeze Panes**
- This freezes all cells which are to the left or above the currently selected cell.

Create column/row grouping

- Select row/column headers/indices you want to group
- **Data → Outline → Group**

Linking worksheets

- We can create formulas using cells from other sheets by referring them in the following way.
- {other_sheet_name}!{cell_number}. For example, importing cell A1 from a sheet called "SupplyCostsSheet" is done via SupplyCostsSheet!A1

Consolidating data

- To consolidate data from different tables (including when the tables are in different sheets you can do the following:
- **Data → Data Tools → Consolidate**
- Choose function (i.e. what function will you use to combine data from the different sources. e.g. if you want a grant total for sales for each item, use SUM)
- Add all tables as references by highlighting them/adding their table name to the "Reference" box and pressing "Add"
- Select whether you want to use column header labels/ row index labels (or both) from the source data.

CONDITIONAL FUNCTIONS/FORMATTING

Excel name ranges

You can name a range of cells for later reference, this is some sort of middle ground between having a range of cells and having a table.

- Select cell range
- Rename in the name box (to the bottom left of the ribbon).
- Can use this name to reference this range of cells. Note that it is an absolute reference.
- Use the dropdown in the name box to select and navigate to named cell ranges
- To manage names:

  Formulas → Defined Names → Name Manager.

Conditional functions

- IF
  - Arguments:
    - Logical_test - a value or expression that can be evaluated to TRUE or FALSE
    - Value_if_true - the value returned if the test is TRUE
    - Value_if_false - the value returned if the test is FALSE
  - Example:

    =IF(A1>1,"Greater than 1", "Not greater than 1")
- AND
  - Arguments:
    - Logical1 - a value or expression that can be evaluated to TRUE or FALSE
    - Logical2 - another value or expression that can be evaluated to TRUE or FALSE
    - Logical3 - ...
    - (You can add up to 255 logical expressions)
  - Return:

    Returns TRUE if all logical expressions evaluate to TRUE, otherwise, it returns FALSE
- COUNTIF
  - Arguments:
    - Range - the range of cells from which you wish to count blank cells
    - Criteria - logical criteria the cell has to meet in order to be counted
  - Return:

    Returns the number of cells in the range that meet the given criteria
  - Example:

    =COUNTIF(A1:A5, ">5")
- SUMIF - performs similarly to COUNTIF, but can use different ranges for the cells which have to satisfy criteria, and the cells which will actually be summed up.
  - Example:

    Using dummy data, to sum total price of Bakery items we would do: =SUMIF(B2:B12, "Bakery", C2:C12)
- IFERROR
  - Arguments:
    - Value - an expression or value (which may cause an error)
    - Value_if_error - the value the function will return if there is an error with the "Value" argument.
  - Return:

    Returns Value if there's no error, and Value_if_error if there is an error.
  - Example:

    =IFERROR(5/0,"Bad: Dividing by zero")

LOOKUP FUNCTIONS

- VLOOKUP

  Looks for the first match to a value in the first column of a range of cells, and returns the value in the column specified. By default, the first column should be sorted in ascending order.
  - Arguments:
    - Lookup_value - the value you wish to search for from the first column
    - Table_array - the range of cells or table name
    - Col_index_num - the column number from the range of cells passed. Note that the first column is indexed 1.
    - Range_lookup - TRUE if you are happy with the closest match (for this, the column should be sorted). FALSE if you want an exact match (this can return N/A if no match is found).
- HLOOKUP

  Looks for the first match to a value in the first row of a range of cells, and returns the value in the row specified. The first row is indexed by 1.

  This works the same as VLOOKUP, including the same arguments, but with the roles of columns and rows swapped.
- INDEX

  Returns the value within the cell at a given row and column number. (Both row and column numbering starts at 1).
  - Arguments:
    - Array
    - Row_num - the row from which to return the value
    - Col_num - the column from which to return the value
- MATCH

  Returns the relative position of an item in a row or column that matches a specified value.
  - Arguments:
    - Lookup_value
    - Lookup_array - must be a single row or column.
    - Match_type - one of -1,0,1. (if using ± 1, ensure the data is sorted.)
      0 = exact match,
      -1 = smallest value greater than or equal to the lookup value,
      1 = greatest value smaller than or equal to the lookup value.

TEXT FUNCIONS

- LEFT/RIGHT

  Returns the specified number of characters from the start/end of a string
  - Arguments:
    - Text
    - Num_chars - the number of characters to return.
- MID

  Returns the specified number of characters from the middle of a string, at a choosen start point
  - Arguments:
    - Text
    - Start_num - the index of the first character you want to extract
    - Num_chars - the number of characters to return.
- LEN

  Returns the length of the text
  - Arguments:
    - Text
- SEARCH

  Returns the character index at which the first occurrence of the target string is found
  - Arguments:
    - Find_text - the text you want to find
    - Within_text - the text you want to search in.

- CONCATENATE

  Joins several strings into a single string

  - Arguments:
    - Text1
    - Text2
    - (Can have up to 255 arguments, up to Text255)

"NEW" EXCEL FUNCTIONS (POST 2019)

- FILTER

  Filter a range or array

  - Arguments:
    - Array
    - Filter - an array of booleans where TRUE denotes a row or column to keep. This can be obtained via a logical condition
      e.g. 'my_table[my_column] = "some value" '.

- SORT

  Sort an array

  - Arguments:
    - Array
    - Sort_index - the index of the column/row you want to sort with respect to
    - Sort_order - +1 for ascending, -1 for descending]
    - By_col - TRUE to sort the columns, FALSE to sort the rows (default)

- UNIQUE

  Returns the unique values from an array/list

  - Arguments:
    - Array
    - By_col - TRUE for return unique rows, FALSE for return unique columns
    - Exactly_once - TRUE for returning rows/columns that appear exactly once, FALSE for returning all unique rows/columns, even if they appear multiple times (default)

- XLOOKUP

  Looks for a value in a lookup range, and returns a corresponding value from a return range.

  - Arguments:
    - Lookup_value
    - Lookup_array - the range to search for the lookup value
    - Return_array - the range from which to return the value.
    - If_not_found - what do return if no match is found
    - Match_mode - specify the match type:
      0 - Exact match. If none found, return #N/A. This is the default.
      -1 - Exact match. If none found, return the next smaller item.
      1 - Exact match. If none found, return the next larger item.

- SWITCH

  Essentially a multi-if statement, i.e. if ValueN then return ResultN

  - Arguments:
    - Expression - the value/expression to be evaluated
    - Value1 - value to be compared to expression
    - Result1 - returned if expression matches Value1
    - (Value2, Result2, etc)

- TEXTJOIN

  Essentially concatenate, but more streamlined if a custom delimiter is used

  - Arguments:
    - Delimiter
    - Ignore_empty - if TRUE it ignores empty cells if a range is passed
    - Text1 - text or range to be joined
    - (Text2, ..., Text252)

- TEXTSPLIT

  Splits text into rows or columns

  - Arguments:
    - Text - the text to split
    - Col_delimiter - character ot string to split columns by
    - Row_delimiter - character ot string to split rows by
    - There are also ignore_empty and match_mode arguments.

AUDITING

<u>Tracing precedents/dependents</u>

We can see which cells affect a current calculation, or vice versa using auditing. This is known as tracing precedents/dependents. This is done as follows:

Formulas → Formula Auditing
→ Trace Precedents/Trace Dependents

<u>Watch window</u>

This is somewhat reminiscent of having sheets in different windows with split screen. The watch window does what it says, it allows you to watch cells as you interact with the workbook. This is done via

Formulas → Formula Auditing → Watch Window

<u>Show formulas</u>

Shows formulas in the cells, rather than the final value. This is handy for debugging etc.

Formulas → Formula Auditing → Show Formulas

PROTECTING WORKSHEETS AND BOOKS

- Protecting cells:

  All cells have a lock (which is enabled by default). This is merely the ability to be locked. Before protecting a cell, disable the lock on any cells you want to keep free to edit. This is done via highlighting the cell(s), pressing Ctrl+Shift+F (format window), going to the protection tab, and unchecking Locked.

  To protect the sheet, go to

  Review → Protect → Protect Sheet

  to unprotect, go to

  Review → Protect → Unprotect Sheet

- Protect workbook structure:

  This ensures you cannot add/remove/change sheets themselves

  Review → Protect → Protect Workbook

- Adding password to a workbook

  File → Info → Protect Workbook
  → Encrypt with Password

  To remove password, re-encrypt with a blank password field.

WHAT IF ANALYSIS

What-if analysis allows you to try out different inputs (scenarios) for formulas.

Data → Forcast → What if analysis

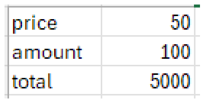
- Goal Seek: Determines what value needs to be inserted into an input cell to achieve the desired value in the output cell.

  Arguments

  - Set cell: the output cell, whose value you wish to control
  - To value: the value you want the output cell to achieve
  - By changing cell: the input cell whose values you allow to change

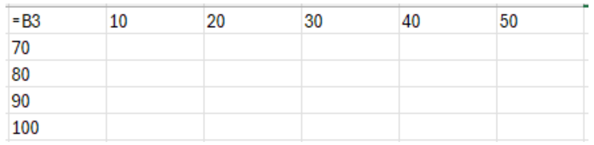
- Data Tables: A table designed to neatly arrange and calculate formulas on a range of changing arguments.

They work as follows. Suppose we have a formula with inputs B1 and B2, producing an output in B3. For example:



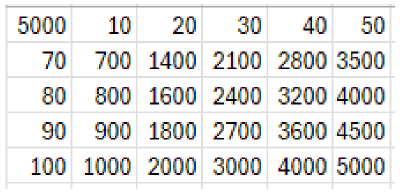
price	50
amount	100
total	5000

We then create a grid, with different values of price as column headers, and different values for amount as row indices. We make a reference to the formula output in the top-left of the newly-created grid.



=B3	10	20	30	40	50
70					
80					
90					
100					

Highlight the whole grid (including the headers and indices) and navigate to **Data Table**. Add cell B1 (the cell containing the price) as the row input cell, then add cell B2 (the cell containing the amount) as the column input cell. Press OK. This gives the following completed grid.



5000	10	20	30	40	50
70	700	1400	2100	2800	3500
80	800	1600	2400	3200	4000
90	900	1800	2700	3600	4500
100	1000	2000	3000	4000	5000

- Scenario Manager: you can add scenarios which allow you to set multiple sets of values for the same set of cells and flick between them.

There is also a linear programming solver built into Excel

Data → Analyse → Solver

PowerQuery

GETTING STARTED

All PowerQuery related tools can be found at

Data → Get & Transform Data

PowerQuery prefers to work with tables as opposed to ranges of cells.
PowerQuery may be opened via

Data → Get & Transform Data → Get Data

→ Launch PowerQuery Editor

This opens a blank window, methods to add data can be found in the **Home** tab. However, there are better ways to get data into PowerQuery.

RETRIEVING DATA

We can open data from various sources inside PowerQuery - we mention some of the them here, all paths begin with

Data → Get & Transform Data → Get Data → . . .

- From Excel File:

  · · · → **From File → Excel Workbook**

  – Select the workbook from file explorer
  – A navigator window will appear, allowing you to choose which tables from the workbook to import

- From Excel Table within current worksheet:

  · · · → **From Other Sources → From Table/Range**

- From Text File:

  · · · → **From File → From Text/CSV**

- From DataBase:

  · · · → **From Database** → . . .

  – Can select which tables or queried tables you wish to import within the navigator window that pops up

- From Folder:

  · · · → **From File → From Folder**

  – This is useful if you wish to, for example, import and merge a collection of csv files which are in the same folder (say monthly data you wish to merge into yearly data)

- From the web:

  · · · → **From Other Sources → From Web**

  – Insert URL of the webpage
  – All <table> elements will appear in the navigator, select the tables you wish to import.

TRANSFORMING DATA

Set/Remove column headers

If the first row has not been registered as the table headers (or the first row has incorrectly been registered as headers), go to **Transform → Table → Use First Row as Headers ...**

Setting Data Types

Click the column header and navigate to

Home → Transform → Data Type ...

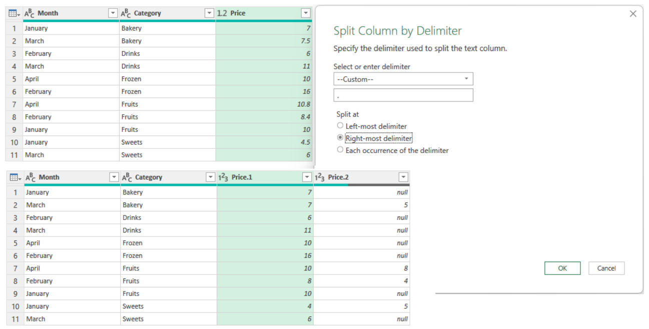
Alternatively, click the icon in the top left of the column header cell to change data type.

Splitting Column Data into multiple columns

Click the column header and navigate to

Home → Transform → Split Column

For example, transforming the price column into a column for whole pounds and another for pence is achieved as follows:



	Month	Category	1.2 Price
1	January	Bakery	7
2	March	Bakery	7.5
3	February	Drinks	6
4	March	Drinks	11
5	April	Frozen	10
6	February	Frozen	16
7	April	Fruits	10.8
8	February	Fruits	8.4
9	January	Fruits	10
10	January	Sweets	4.5
11	March	Sweets	6

Split Column by Delimiter

Specify the delimiter used to split the text column.

Select or enter delimiter: --Custom--

.

Split at
- Left-most delimiter
- [✓ Right-most delimiter]
- Each occurrence of the delimiter

	Month	Category	1²3 Price.1	1²3 Price.2
1	January	Bakery	7	null
2	March	Bakery	7	5
3	February	Drinks	6	null
4	March	Drinks	11	null
5	April	Frozen	10	null
6	February	Frozen	16	null
7	April	Fruits	10	8
8	February	Fruits	8	4
9	January	Fruits	10	null
10	January	Sweets	4	5
11	March	Sweets	6	null

Formatting Text-based columns

Transform → Text Column → Format...

Have options to set text to UPPERCASE, lowercase, Capitalised, Trim, Clean (removes all non-printable characters e.g. \n).

Can also right-click whilst in a column and select "Replace Values", this allows the replacement of special characters by selecting "Advanced options" and "Replace using special characters".

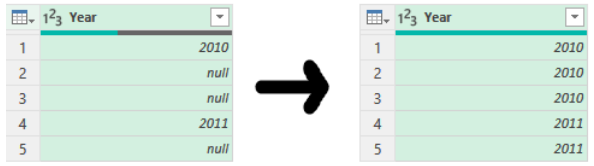
Removing duplicate rows

Home → Reduce Rows → Remove Rows...

→ Remove Duplicates

Filling columns

Right Click column, "Fill" and then choose "Up" or "Down". This replaces null values in a column with the first non-null value found below/above it.
For example, fill down achieves the following:



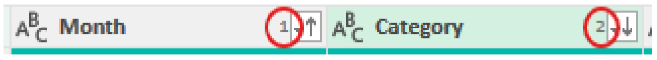
	1²3 Year
1	2010
2	null
3	null
4	2011
5	null

	1²3 Year
1	2010
2	2010
3	2010
4	2011
5	2011

SORTING AND FILTERING

Sorting

On the column header, press the down arrow (on the right hand side of the cell) and select the sort you want.
To apply a multi-level sort, sort the columns from highest priority to lowest priority - you will know a multi-sort is happening because the column headers will contain the numbers "1", "2", etc denoting the priority of the column in the multi-sort e.g.

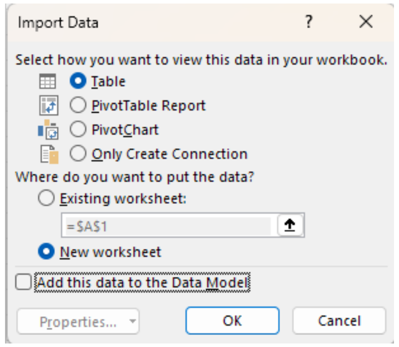


Filtering

On the column header, press the down arrow (on the right hand side of the cell). Use the checkbox to manually filter, or use the "{Data Type} Filters" option to apply conditional filters (the filters available depend on the data type).

CLOSE AND LOAD OPTIONS

- Close & Load - loads the table to a new worksheet in the Excel workbook.
- Close & Load to - opens the following window



Import Data

Select how you want to view this data in your workbook.
- [✓ Table]
- PivotTable Report
- PivotChart
- Only Create Connection

Where do you want to put the data?
- Existing worksheet:
  =A1
- [✓ New worksheet]

☐ Add this data to the Data Model

  - Can open as a Table, PivotTable Report or Only Create Connection. The last option creates a connection which can be found in **Data → Queries and Connections** but won't load the table itself into the workbook.

PIVOT, UNPIVOT AND TRANSPOSING DATA

Transposing data

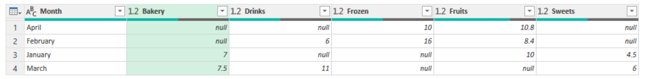
Transform → Table → Transpose

This turns rows of data into columns of data and vice versa.

Pivoting a column

- click into column you want to pivot
- **Transform → Any Column → Pivot Column**
- Choose another column as the values column
- Use advanced options to choose the aggregate function to use.

For example, pivoting our dummy data by Category with the price column as the values column results in:



	Month	1.2 Bakery	1.2 Drinks	1.2 Frozen	1.2 Fruits	1.2 Sweets
1	April	null	null	10	10.8	null
2	February	null	6	16	8.4	null
3	January	7	null	null	10	4.5
4	March	7.5	11	null	null	6

Unpivoting columns

This is the inverse of pivoting. Select multiple columns and unpivoting will replace them with two columns; one based on the column headers, and the other based on the values of the pivoted table.

Transform → Any Column → Unpivot Columns...

→ Unpivot Only Selected Columns

GROUPING DATA

Home → Transform → Group By

- Basic: Select column as grouping column from the drop-down. On the next row give the new column a name, select an aggregate function, and select the column from which to take the values (for example, this could be a price/amount column).
- Advanced: Gives the option to group by multiple columns and/or use multiple aggregate functions on multiple value columns.

ADDING COLUMNS

There are many options for adding columns, which can be found at

Add Column → . . .

Some of these options include:

- Column based on conditions (in **General**)
- An index column (in **General**)
- A duplicate/copy of an existing column (in **General**)
- Columns based on data type (e.g. length of entries in cells of text based columns, or statistics based on numeric columns)

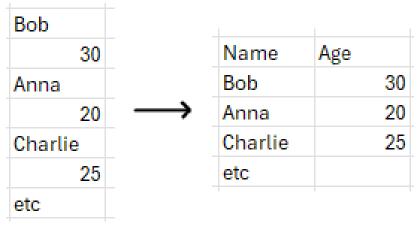
WORKING WITH MULTIPLE SOURCES

In many cases, the data we want will be split over multiple sources. We can combine these using a merge (think SQL Join) or append (stacking them vertically, note column headers must match).

- Merge
  - Open two tables in PowerQuery that you want to merge.
  - (Can open tables within PowerQuery via **Home → New Query → New Source**)
  - To merge tables and create a new table:
    Home → Combine → Merge Queries ... → Merge Queries As New
  - Select the tables, the column to join on, and the join type (e.g. left, right, inner, etc)
- Append
  - This is used to stack tables on top of each other (think, combining monthly data into yearly data)
  - The column headers in the two tables MUST match.
  - **Home → Combine → Append Queries ... → Append Queries As New**

$M \times 1$ TABLE $\rightarrow m \times n$ TABLE METHOD

Data presented in a single column i.e. stacked vertically (e.g. multiple people's name and age being stored vertically with the rows name, address, name, address, name, address, ...) is impossible to analyse.
To fix this, we want to transform this data into 2 columns of name and address, with each entry being it's own row.
For example:



Bob
30
Anna
20
Charlie
25
etc

$\longrightarrow$

Name	Age
Bob	30
Anna	20
Charlie	25
etc	

- Add index column

  (**Add Column** $\rightarrow$ **General** $\rightarrow$ **Index Column**)

- Add "modulo column" based on the index column. The modulo number will be the number of columns you want (e.g. for the above we would do modulo 2).

  (**Add Column** $\rightarrow$ **From Number** $\rightarrow$ **Standard...** $\rightarrow$ **Modulo**)

- Add "integer division" column. Again, divide by the number of columns you want.

  (**Add Column** $\rightarrow$ **From Number** $\rightarrow$ **Standard...** $\rightarrow$ **Divide (Integer)**)

- The Integer-Division column gives the row number for the new layout, and the modulo column gives the column number.

- Pivot using the modulo column (the values column should be chosen to be the column containing the original data).

- Group by the Integer-Division column, press "advanced" in the group-by window and then create a relevant name and select "max" as the aggregate function for each of the columns with data (this will choose the non-null value).

- Delete the Integer-Division column

PARAMETERS

Replace hard-coded/static filter values with a variable one a.k.a. a parameter.

Home $\rightarrow$ **Parameters** $\rightarrow$ **Manage Parameters...**

$\rightarrow$ **New Parameter**

Any created parameters will appear in the "Queries" window.

TABLE VARIABLES

Use parameters/variables from an Excel worksheet in PowerQuery.

- Format the worksheet cell containing the parameter as a table

- Open it in PowerQuery

- Right-click the value and drill down

- Select **Close and Load To...** and create a connection

This will make the cell value available as a parameter. If the parameter value in Excel is changed, you may need to refresh using

Data $\rightarrow$ **Queries & Connections** $\rightarrow$ **Refresh**

Note: A macro could be set up to refresh automatically when the cell holding the parameter changes value.

VBA for Excel

Many tasks can be automated in Excel using macros. These can be created using the macro recorder, or directly using Visual Basic.

EXCEL MACROS

You can record and reuse a recorded sequence of steps (to automate tasks such as formatting). All relevant tools can be found in

Developer → Code

Recording a macro

Macros can be recorded with **Record Macro**. Clicking this allows you to name the macro, as well as set a shortcut key (optional). Once "OK" is pressed, the recording begins. All steps are recorded until **Stop Recording** is pressed.

Running a macro

- If a shortcut key has been set, just press it
- To assign a macro to a button, first add a button via **Developer → Controls → Insert → Button (Form Control)**, and then pick which macro to assign to the button.
- Add macros to the quick access toolbar.
  Press the dropdown arrow on the toolbar, then navigate to More Commands, then Choose commands from macros. From this window you can add the macros you want.

Editing macros

Recorded macros are edited using Visual Basic directly. The Visual Basic Editor can be opened via

Developer → Code → Visual Basic.

The editor opens in a new window. Macros are stored as modules in the "Modules" folder. Double clicking a module opens the VB script.

Saving workbooks with macros

Excel files have file type .xlsx, however if a macro has been created (either via the recorder or directly using VB), then the file must be saved as a .xlsm in order to retain the macros.

VISUAL BASIC - OVERVIEW

Visual Basic (VB) is an object oriented language. Everything in Excel is an object, e.g. the workbook, the worksheets, cells. The general workflow is to select objects we want to interact with and access methods and properties related to them.

As a baby example, to change the value in cell A1 to "Hello", we first select the object (i.e. the cell) and then access its Value property to change it to "Hello". i.e.

```
Range("A1").Select
Selection.Value = "Hello"
```

In Excel, Visual Basic scripts are created in the Visual Basic Editor. The editor is opened using the shortcut Alt + F11, or by navigating to

Developer → Code → Visual Basic.

Once opened, using the View dropdown to add the windows "Properties Window", "Project Explorer" is advised (if they are not already open).

THE IMMEDIATE WINDOW

This window (enabled under the View tab) is a bit like a terminal. Commands can be written and are immediately run.

To return values using commands within the immediate window, they must be prefaced with a "?". For example, to retrieve the value in cell A1:

```
'Do not do this, it results in a compile error
Range("A1").value
'Do this instead
?Range("A1").value
```

P.S. Comments within VB scripts are started using the ' symbol.

VBA MODULES

Modules contain the functions and procedures defining the macros.

To add a module, navigate to **Insert → Module**.

Click into it and add a procedure via **Insert → Procedure** (if you are not clicked into the module fully, this will be greyed out.)

Aside: Sub vs Function

When adding a procedure, there is the choice of sub vs function. The difference is roughly the following:

- Sub (procedure): Used when you need to execute a series of steps. e.g. modifying a value in a cell. Sub procedures have no return values.
- Function: A block of code that performs calculations and returns a value.

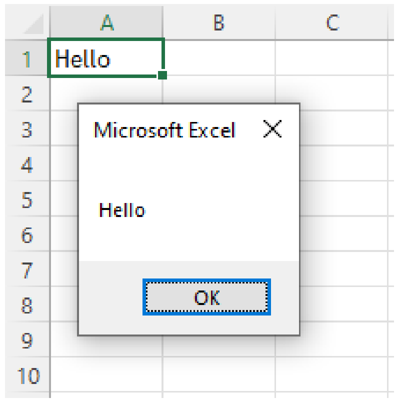
Procedures can be run by clicking into them and pressing the run button in the ribbon. Alternatively, procedures can be added to buttons (just as explained for macros recorded using the macro recorder).

Documentation for any method can be opened by clicking into the method and pressing F1.

Breakpoints can be added at any point in the procedure by clicking in the left hand grey bar, in line with the line(s) you wish to break at. This is very useful for debugging. Break points are removed by clicking again.

As an example of a (sub) procedure, here is a procedure that creates a pop-up message box displaying the value in the cell A1:

```
Public Sub PopUpCellA1()
    MsgBox (Range("A1").Value)
End Sub
```



Hello

Microsoft Excel

Hello

VARIABLES

We do not usually wish to hard code values, instead we want to make use of variables. In VB, variables are declared using the "Dim" keyword. After declaration, they can be initialised with a value. For example,

```
Public Sub MyFirstVariable()
    Dim Person As String
    Person = "Bob"
    'A msg box containing the word "Bob".
    MsgBox(Person)
End Sub
```

Here, String is a data type. Some common types are String, Integer, Boolean (True or False) and Variant, which is a "catch all" type (wiht the drawback of taking up more memory).

Furthermore, variables may be combined with text using the ampersand symbol (&). For example,

```
Public Sub MyFirstVariable()
    Dim Person As String
    Person = "Bob"
    Dim Age As Integer
    Age = 25
    MsgBox (Person & " is " & Age & " years old")
    'Bob is 25 years old
End Sub
```

USEFUL OBJECTS, METHODS AND PROPERTIES

- ActiveSheet - accesses the ActiveSheet
  - .UsedRange - a range object containing all cells that have been used
    - Rows - a collection of rows within the Used Range. You can access the number of used rows via the .Count attribute.
    - Columns - a collection of columns within the Used Range. You can access the number of used columns via the .Count attribute.
- ActiveCell - accesses the current active/highlighted cell in the worksheet
  - .Value - the value of the active cell. This can be used to change it, or for use in conditionals.
  - .Offset(a,b) - points to the cell ‘a’ rows below and ‘b’ columns right of the active cell.
    For example, to select the cell below the active cell we would do

    ```
    ActiveCell.Offset(1,0).Select
    ```

  - .Address - the reference for a cell. By default this is an absolute reference e.g. "A2". To obtain a relative reference use `ActiveCell.Address(false,false)` instead.
- Application - accesses the application i.e. Excel itself.
  - .CutCopyMode - a boolean (True/False). Setting this equal to False clears the clipboard.
  - .ScreenUpdating - a boolean (True/False). Whether or not to graphically display the running of the macro on screen.
    Turning this off before the body of the script (and turning it back on at the end) removes "screen flicker".

IF AND SELECT CASE

The format of If statements is as follows

```
If (condition) Then
    'do something
ElseIf (condition) Then
    'do something else
Else
    'do something otherwise
End If
```

For Select Case, we first select the expression, and then run some steps depending on its value (or in other words, depending on what its case is). The syntax is as follows:

```
Select Case expression
    Case Is =value1
        ' do when expression = value1
    Case Is =value2
        ' do when expression = value2
    'etc
End Select
```

A concrete example of Select Case:

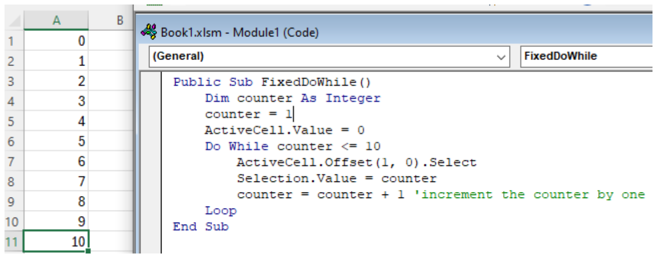
```
Dim MyValue As String
MyValue = ActiveCell.Value
Select Case MyValue
    Case Is = "Hello"
        MsgBox ("Active cell says Hello")
    Case Is = "Hey"
        MsgBox ("Active cell says Hey")
 End Select
```

DO WHILE LOOPS

To implement a Do While loop which runs a fixed number of times:

```
Dim counter As Integer
counter = 1

Do While counter <= 10
    'do something
    counter = counter + 1 'increment the counter
Loop
```

For example, to create an index column (incrementing by 1) we could do



	A	B
1	0	
2	1	
3	2	
4	3	
5	4	
6	5	
7	6	
8	7	
9	8	
10	9	
11	10	

Book1.xlsm - Module1 (Code)

(General) FixedDoWhile

```
Public Sub FixedDoWhile()
    Dim counter As Integer
    counter = 1
    ActiveCell.Value = 0
    Do While counter <= 10
        ActiveCell.Offset(1, 0).Select
        Selection.Value = counter
        counter = counter + 1 'increment the counter by one
    Loop
End Sub
```

FOR EACH & FOR NEXT

A For Each loop can be used to do something for each item in a collection (e.g. a range of cells).

For example, suppose cells A1:A10 contained numbers from 1 to 10 inclusive and we want to fill the cell next to each value with "It’s a 1" if the value is 1, and otherwise fill it with "It’s not a 1". This can be done as follows:

```
Sub IsItA1()
    Dim cell As Range 'a single cell is technically a
      range.
    For Each cell In Range("A1:A10")
        If cell.Value = 1 Then
            cell.Offset(0, 1).Value = "It's a 1"
        Else
            cell.Offset(0, 1).Value = "It's not a 1"
        End If
    Next cell
End Sub
```

A For Next loop is a numerical counter based loop (usually the integers). the syntax is as follows

```
    Dim i As Integer
    For i=1 To 10
        'do something
    Next i
```

To make loops more dynamic you could, for example, use the active sheet’s UsedRange object to count the number of columns and/or rows in use. i.e.

```
    'number of rows used
    ActiveSheet.UsedRange.Rows.Count

    'number of columns used
    ActiveSheet.UsedRange.Columns.Count
```

SORTING WITH VBA

We can sort using the Range.Sort() method. There are many arguments, we will only discuss three. Namely, Key1 (the column to sort by), Order1 (the order), and Header (does the range contain headers?).

For example, to select data in columns A:F, and then sort using the data in column C we would do:

```
Range("A:F").Sort Key1:=Range("C1"), Order1:=
    xlAscending, Header:=xlYes
```

Four things to observe/note

- Arguments are passed to a method/function by listing them (no brackets needed). Also, arguments are comma separated. i.e. my_method arg1, arg2, arg3
- Named arguments are given values using the assignment operator (:=)
- Arguments such as order and headers (a yes or no answer) are given "xl" values, i.e. xlAscending, xlDescending, xlYes, xlNo.
- The Key1 argument is set to be the cell C1 (i.e. Range("C1")). This can be any cell in column C. e.g. Range("C100") would have worked just as well.

USER INPUT

User input is obtained using the InputBox object, the first argument is the prompt for the user. i.e.

```
Dim UserChoice As String
UserChoice = InputBox("What do you choose?")
```

Multi-line prompts/strings

In many cases, it is useful for the prompt text to the user to be on multiple lines (for example, when creating a list of choices). There are two characters of interest here.

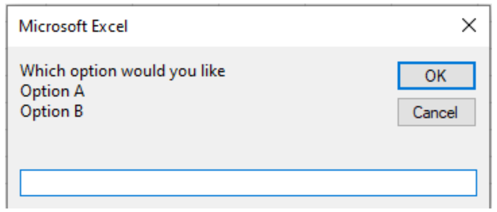
The first is the vbCrLf (carriage return and line feed). This creates a new line

The second is the underscore (_) character. It is the line continuation character in VBA, and it allows you to break a long line of code into multiple lines (for example, for readability reasons).

One way to create a user input box with multiple lines of prompt is as follows:

```
Dim UserChoice As String
Dim LongPrompt As String
LongPrmpt = "Which option would you like" & vbcrlf & _
            "Option A" & vbcrlf & _
            "Option B
UserChoice = InputBox(LongPrmpt)
```

This results in the following:


Microsoft Excel

Which option would you like
Option A
Option B

Changing buttons on message boxes

By default, the message box (MsgBox) has an "OK" button. The buttons displayed can be customised. For example we can create a Yes/No button as follows:

```
Dim Choice As Integer
Choice = MsgBox("Yes or No", vbYesNo)
```

Note: The choice (i.e. clicking either Yes or No) is stored as an Integer. The integer 6 corresponds to the choice of Yes, whereas 7 corresponds to the choice of No.

Handling errors

Pressing "OK" with an empty input or pressing "Cancel" in an InputBox results in an error. To handle this, we need to tell the script what to do when an error occurs.
This is done by specifying an error handler location using the 'On Error GoTo' statement, which should appear before the code we want to error handle. For example:

```
    On Error GoTo err
    Dim Choice As Integer
    Choice = InputBox("Yes or No")

    If False Then ' Can't get in here unless
        through err
  'place err where the code should go in case of an
      error. In this case, inside the if block
err:
        MsgBox ("There was an error")
    End If
```

WORKING ACROSS MULTIPLE SHEETS

The "Worksheets" object is a collection of all the worksheets. We can use a For Each loop to apply a set of commands/a macro to each sheet. For example:

```
Dim ws As Worksheet
For Each ws In Worksheets
    ws.Select
    'ws is now the active sheet
    MsgBox (ActiveSheet.Name)
Next ws
```

DYNAMIC SELECTION OF COLUMNS

Suppose we want to sum the values column A for a number of worksheets. We want to be able to dynamically find the range, rather than hard coding the range for each sheet.

In Excel, we have the following four key shortcuts:

- Ctrl + Down
  Jumps to the last filled cell in the current column before an empty cell
- Ctrl + Right
  Jumps to the last filled cell in the current row before an empty cell.
- Ctrl + Shift + Down
  Selects all the filled cells in the current column from the active cell downward until an empty cell is met.
- Ctrl + Shift + Right
  Selects all the filled cells in the current row from the active cell rightward until an empty cell is met.

Their corresponding VBA code is:

- `ActiveCell.End(xlDown).Select`
- `ActiveCell.End(xlToRight).Select`
- `Range(ActiveCell, ActiveCell.End(xlDown)).Select`
- `Range(ActiveCell, ActiveCell.End(xlToRight)).Select`

Tip: Finding corresponding VBA code for shortcuts.

To find the corresponding VBA code for a keyboard shortcut, record a macro using the macro recorder, press the desired keyboard shortcuts, stop the recording, then view the new VBA script in the Visual Basic Editor.

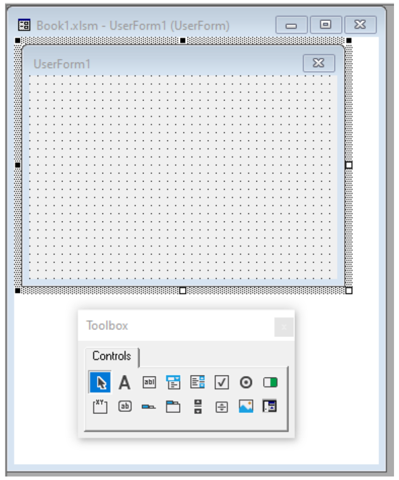
COPYING AND PASTING

```
'To copy
Selection.Copy

'To paste
ActiveCell.Paste

'To clear the clipboard
Application.CutCopyMode = False
```

USER FORMS

User forms provide a very flexible and customizable way to take user input and provide interaction within an Excel workbook.

Within Visual Basic, go to **Insert** → **User Form**. This opens a window which looks like:



The toolbox contains components such as labels, buttons, combo boxes (a.k.a. dropdown menus), list boxes and more.

Once components are added to the user form, use the "Properties" window to edit them. The property "(name)" is especially important, as this is how the component will be referenced in scripts.

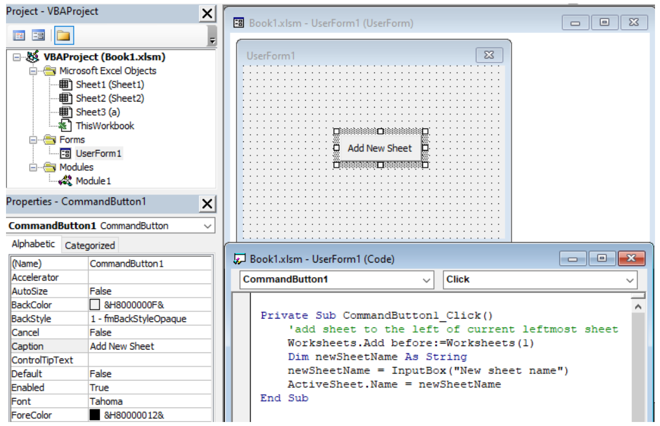
Event handling with user forms using script

Double clicking anywhere in the user form opens a new window where event handling procedures can be added. Use the dropdown menus at the top of the window to choose components and events. Some common examples are:

- **ComboBox – Change**
  Runs when the selected item changes
- **Button – Click**
  Runs when the button is clicked
- **UserForm – Activate**
  Runs when the form becomes active

Within this script, the user form itself can be referred to using the "me" keyword. (This is analogous to "self" in Python, or "this" in C#).

As an example, here is a user form with a button that when clicked will create a new sheet and allow the user to name it.



(Name)	CommandButton1
Accelerator	
AutoSize	False
BackColor	&H8000000F&
BackStyle	1 - fmBackStyleOpaque
Cancel	False
Caption	Add New Sheet
ControlTipText	
Default	False
Enabled	True
Font	Tahoma
ForeColor	&H80000012&

```
Private Sub CommandButton1_Click()
    'add sheet to the left of current leftmost sheet
    Worksheets.Add before:=Worksheets(1)
    Dim newSheetName As String
    newSheetName = InputBox("New sheet name")
    ActiveSheet.Name = newSheetName
End Sub
```

Clicking on "ThisWorkbook" in the VBA project explorer allows events that involve the workbook itself to be handled, such as **Workbook – Open** (which will run when the workbook is opened).

IMPORTING DATA FROM TEXT FILES

```
Dim TextFile As Workbook
Set TextFile = Workbooks.Open("C:\Users\...\.txt")
```

This opens the text file at the given path within an Excel workbook (in a new window).

The Set keyword:

The Set keyword is used (and required) when setting Excel object references. This is in contrast to simply assigning a value (i.e. i = 1) where the set keyword is not needed.

A potential situation when opening text files could be that you want to copy the information and paste it into the original workbook that was open. This is done as follows:

```
Dim TextFile As Workbook
Set TextFile = Workbooks.Open("C:\Users\...\.txt")
'Copy all content linked to cell A1 of the first
    sheet of the newly opened workbook
TextFile.Sheets(1).Range("A1").CurrentRegion.Copy
'Make the original workbook the activate one
'As it was open first, it is at index 1 in the
    workbooks collection
Workbooks(1).Activate
ActiveSheet.Paste
TextFile.Close
```

Opening (multiple) files using the File Explorer

To open a file explorer window and allow the user to select multiple files to open, we can do:

```
Dim OpenFiles As Variant
OpenFiles = Application.GetOpenFilename(
    MultiSelect:=True)
```

then we can loop through them, open them, and do whatever

```
For Each file in OpenFiles
    Set TextFile = Workbooks.Open(file)
    'do something with TextFile
Next file
```