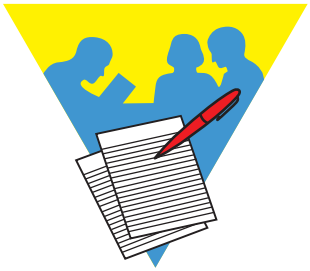
Excel Formulas

The Computer Workshop, Inc.

800-639-3535

www.tcworkshop.com
training@tcworkshop.com



Lesson Notes

Microsoft Excel Formulas

Rel 2.0, 12/12/2024

Course Number: 0200-240-ZZ-Z
Course Release Number: 2.0
Software Release Number: 365

December 12, 2024

Developed by:
Brian Ireson, Evan Hively

Edited by:
Thelma Tippie, Skye Harper, and Jeffrey DeRamus

Published by:
The Computer Workshop, Inc.
5200 Upper Metro Place, Suite 140
Dublin, Ohio 43017
(614) 798-9505

for

RoundTown Publishing
5200 Upper Metro Place, Suite 140
Dublin, Ohio 43017

Table of Contents

Table of Contents, continued

Lesson 3: Text Functions

Table of Contents,
continued

Lesson 4: Date & Time Functions

Appendix : Troubleshooting

Using This Manual

Welcome to the *Excel Formulas* course. This manual and the accompanying data files are designed to for learning, review and reference after the class. The data files can be downloaded at any time from *The Computer Workshop* website:

https://www.tcworkshop.com

No login or password required to access these files. Handouts and video links can also be found on the website in the resources section. All handouts are in PDF format and freely available. Simply open the PDF to print or save it to your computer.

Downloading Data Files

1. On The Computer Workshop's website (www.tcworkshop.com)
2. Click on ***Student Resources*** in the site's navigation bar.
3. Click the ***Data Files*** button to open a list of general application types.
4. Click the ***Microsoft Office Courses*** link.
5. Click *Excel*.
6. Click the files link related to the course.

You can choose to open or save the zipped folder's content to your computer.

A copy of the data files have been attached to this PDF.

Conventions

Conventions Used in This Manual

The hands-on exercises (***Actions***) are written in a two-column format. The left column (**Instructions**) gives numbered steps, such as what to type, keys to press, and commands to choose from menus. The right column (**Results/Comments**), contains comments descriptions of the expected results, explanations, shortcuts, and additional notes related to the instructions on the left.

- Key names and Functions are bold and enclosed in square brackets.

  [Enter], [Tab], [F5], [F10]

- Keys that you press simultaneously are separated by a plus (+) sign, they are bold and enclosed in square brackets. You do not include the plus key.

  [Shift F5]

- Keys you press in sequence are separated by a space, bold and enclosed in square brackets.

  [Home] [Down Arrow]

- Ribbon tab names are bold and italic:

  Home

- Group names are in bold.

  Font

- Dialog box names are in italic.

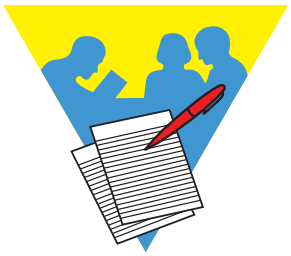
  Save As

- Button names are bold and enclosed in square brackets.

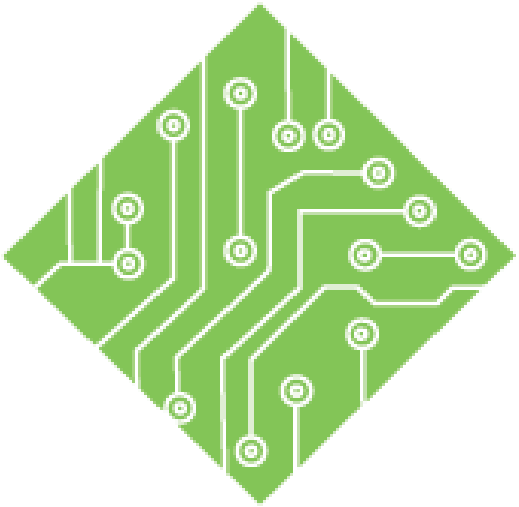
  [Sort]

- Information you are to type will be in bold.

  This is the first day of the rest of your life.



Tips and Notes



Lesson 1: Logical & Lookup Functions

Lesson Overview

The following concepts are covered in this chapter:

- Naming Cells: Review
- Logical Functions
- Lookup Functions
- LAMBDA Functions

Lesson Notes

Naming Cells

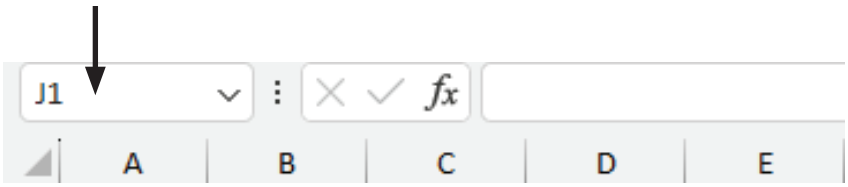
In formulas, it is easier to refer to a cell by cell name instead of by cell address. Cell naming is covered in more detail in other *Excel* courses in this series. Formulas can also be named using a similar method.

Note

Formulas referring to absolute or relative cells and ranges can be named using *Name Manager*. More complex formulas use **LAMBDA** functions, which are covered later in this chapter.

Naming Cells

- Select the cell or cell range to name.
- Click into the **Name Box** and type in the desired name.

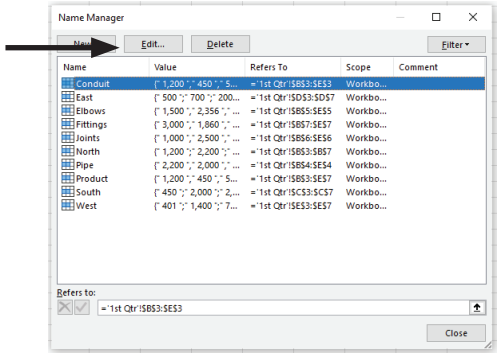


  - Do not use special characters or blank spaces in the name.
  - Use underscores, camelCase, or PascalCase to separate words in multi-word names.
  - Press **[Enter]**, or the name will not be applied.

Editing Names

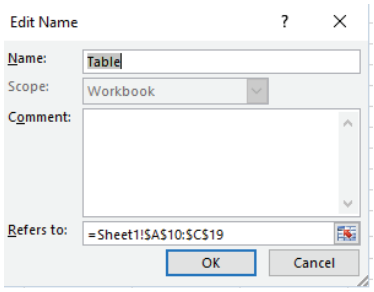
- Click the **[Name Manager]** button in the **Defined Names Group** on the ***Formulas Tab***.

- The *Name Manager* dialog box displays all names in the document. Select the name that needs to be edited and click the **[Edit]** button.



Naming Cells, continued

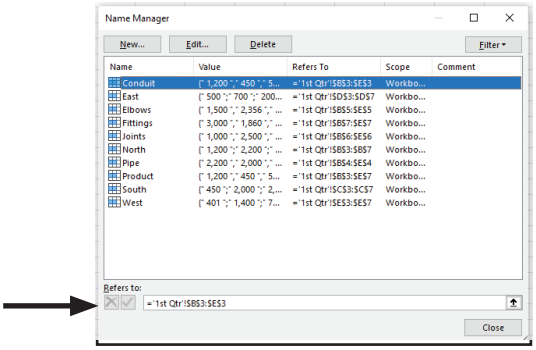
- Observe the options in the *Edit Name* dialog box:



  - **Name:** Fix typos by renaming the cell or range.
  - **Scope:** Scope is defined when the name is created and cannot be modified. By default, it is set to "Workbook."
  - **Comment:** Explain the name, if necessary.
  - **Refers to:** Redefine the address of the cell or cell range.

If only the **Refers to:** field needs editing, change it in the *Name Manager* dialog box.

- Select the name to edit.
- Click into the **Refers To:** field at the bottom of the dialog box.
- Click the **[Expand/Collapse]** dialog button and highlight the correct range of cells.
- Click **[Expand/Collapse]** to redisplay the *Name Manager* dialog box.
- Click the **[Check]** button to apply the edit.
- Click the **[Close]** button to exit the *Name Manager* dialog box.



Action 1 – Naming Cells

Instructions:	Results/ Comments:
1. Open **Loan.xlsx** from the **Data Files** folder.	**Objective**: Formulas can become quite complex. Create field names to keep better track of formulas.
2. Save the file as **My Loan.xlsx**	**F12** displays the *Save As* dialog box.
3. Select cell **B6**	**B6** is the first cell to be named.
4. Click into the **Name Box** and type: **State** Then press **[Enter]**	The **Name Box** will now display the name **State** instead of the address **B6**.
5. Repeat steps 3 and 4 for the following cells and names: **Range** / **Range Name** **A13:C22** / InterestTable **B25** / MITax **B26** / OHTax **I5** / Rate **I7** / Years	Remember to press **[Enter]** after each name to ensure it is saved.
6. Click the dropdown arrow in the **Name Box** to see a list of all cell names.	If one or more names is not listed, return to the cell and apply the corresponding name.
7. Save the file.	**[Ctrl + S]**

Logical Functions

First, it is necessary to address the difference between functions and formulas. In simplest terms, functions are built-in calculations that can be used to create formulas. This book covers many of the vital functions in *Excel* and their use in creating some commonly used formulas.

A logical function evaluates an expression and returns a value based on whether the expression is *True* or *False*. These functions are commonly used to determine whether a condition is met to choose a course of action.

Logical functions are used to check the result of a condition, such as to determine whether actual expenses are over budget or if a client is eligible for a discount. Create conditional formulas using the **AND**, **OR**, **NOT**, and **IF** functions.

All logical functions can be used as standalone functions or in conjunction with an **IF** function to expand the **IF**'s capabilities to much broader parameters.

TRUE and FALSE Logical Functions

The **[Logical]** formulas dropdown button in the **Function Library Group** on the ***Formulas Tab*** also contains **TRUE** and **FALSE** functions. These functions do not require arguments and return the same value as the function name.

- **=TRUE()** returns *True*
- **=FALSE()** returns *False*

Using the **TRUE** and **FALSE** functions is the equivalent of writing "True" and "False." These functions are most useful as part of a more complex logical formula.

Logical Functions, continued

IF Function

One of the most common logical functions is the **IF** function, which performs a comparison or *logical test* and displays the result in the active cell based on that test's outcome. The **IF** function returns one value if the condition's result is *True* and another if it is *False*.

The **IF** function takes the following arguments:

- ***logical_test:*** *[required]* any expression that evaluates to *True* or *False*
- ***value_if_true:*** *[required]* the value returned if the ***logical_test*** evaluates to *True*
- ***value_if_false:*** *[optional]* the value returned if the ***logical_test*** evaluates to *False*

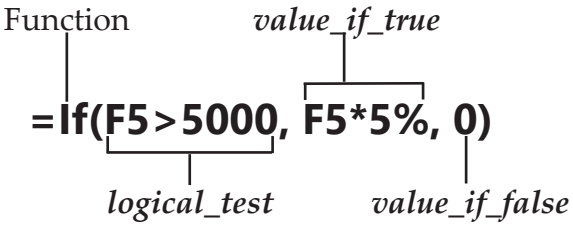
Note

Remember to wrap values in quotation marks when working with text or mathematical expressions in formulas.

The syntax of the **IF** function is as follows:

=IF(***logical_test, value_if_true, value_if_false*** **)**

For example, an **IF** function can determine if a salesperson has sold above a goal of $5000 to earn a bonus. If they sell more than the goal amount, they will receive a bonus of 5% of their total sales; if not, they will not receive a bonus. If the sales total is in cell **F5** and the bonus calculation is in cell **G5**, the syntax of the **IF** formula in cell **G5** is as follows:



This formula states that if the salesperson's sales total is greater than 5000, it will be multiplied by 5% to determine the returned bonus amount. If it is less than 5000, a zero value will be returned.

Action 2 – IF Function

Instructions:	Results/ Comments:
1. **My Loan** should still be open. If not, open it again.	**Objective**: Display the correct sales tax for the state.
2. Select cell **I9.**	**I9** will show the appropriate sales tax based on the value in cell **B6** (**State**).
3. Enter this formula: **=IF(State="OH", OHTax, MITax)** Then press **[Enter].**	The first argument of this function contains the expression to determine if cell **B6** (**State**) contains the value "OH." Since "OH" is text, it must be wrapped in quotation marks. If cell **B6** (**State**) contains the text "OH," the formula displays the second argument, **OHTax**; if not, the third argument, **MITax**, is returned. Since **B6** contains "MI," **I9** should initially display **7.0**%. The formula is not case-sensitive.
4. Click into cell **B6** (**State**).	**B6** (**State**) is the cell checked by the **IF** function in cell **I9**.
5. Type **OH.**	The value in cell **I9** now shows the value from the **OHTax** cell. The **IF** function's ***logical_test*** argument is not case-sensitive.
6. Reselect cell **B6** (**State**).	Change the state to see the sales tax in cell **I9** reflect this change.
7. Type **MI.**	The value in cell **I9** now shows the value from the **MITax** cell.
8. Save the file.	**[Ctrl + S]**

Logical Functions, continued

Note

An **IF** function can contain up to seven nested **IF**s in a single formula.

Nested Functions

There are times when it is necessary to evaluate multiple conditions. One way to do this within a formula is by nesting multiple **IF**s inside a primary **IF** function. The subsequent **IF**s are placed in the ***value_if_false*** argument of the function and act as an "otherwise if" option.

The ***logical_test*** argument of an **IF** function stops calculating as soon as any expression returns *True*. If the ***logical_test*** returns *False*, the calculation continues, and the nested **IF** function within the ***value_if_false*** argument is evaluated.

The calculation continues until any **IF** function in the statement evaluates to *True* or there are no other ***logical_tests*** to run. At that point, the statement is *False*, which returns the value in the final nested **IF**'s ***value_if_false*** argument.

The syntax of a nested **IF** function is as follows:

Primary **IF** function

=IF(F5>5000, F5*5%, IF(F5>2000, F5*3%, 0))

Nested **IF** function

In this example, the sales total is in cell **F5**. If the salesperson's sales total is exceeds $5000, the calculation stops and returns the total multiplied by 5%. If it is not greater than 5000, the nested **IF** is evaluated to determine if the sales total is greater than 2000. If it is, the calculation stops and returns the sales total multiplied by 3%. If neither of these ***logical_tests*** evaluates to *True,* the formula returns 0.

Action 3 – Nested IF Functions

Instructions:	Results/ Comments:
1. **My Loan.xlsx** should still be open. If not, open it again.	**Objective**: Add another state, Indiana, to the sales region.
2. Select cell **A27.**	
3. Type in **Indiana** and press **[Tab].**	
4. Cell **B27** should now be selected.	
5. Type in **6** and press **[Enter]**	The percentage sign will be filled in.
6. Reselect cell **B27** and apply the name **INTax.**	Continue to practice naming cells instead of using cell addresses.
7. Select cell **I9.**	The **IF** function needs to be edited to include the additional state.
8. Enter the following formula: **=IF(State="OH", OHTax,** **IF(State="MI", MITax,** **IF(State="IN",INTax,** **"Out of Region")))**	Do not include the spacing; it is for readability only. If **B6** (**State**) contains "OH," the formula returns the **OHTax** value. If not, the *False* argument begins a new **IF** function to a maximum of seven nested **IF**s. In this case, the final *False* returns a text value of "Out of Region." All text values must be wrapped in quotation marks in formulas.
9. Select cell **B6** (**State**).	Test the new **IF** function.
10. Type in **IN** and press **[Enter].**	The value in cell **I9** now shows the value from the **INTax** cell.
11. Save and close the file.	**[Ctrl + S]** to save and **[Ctrl + W]** to close.

Logical Functions, continued

IFS Function

While using nested **IF** functions is valuable, more than the limit of seven options may be needed. In the latest versions of *Excel 2019/365,* a new function called **IFS** has been added, which allows up to 127 different conditions in a single formula. The syntax is more straightforward and cleaner than a nested **IF** and has no ***value_if_false*** arguments.

The **IFS** function takes the following repeatable arguments:

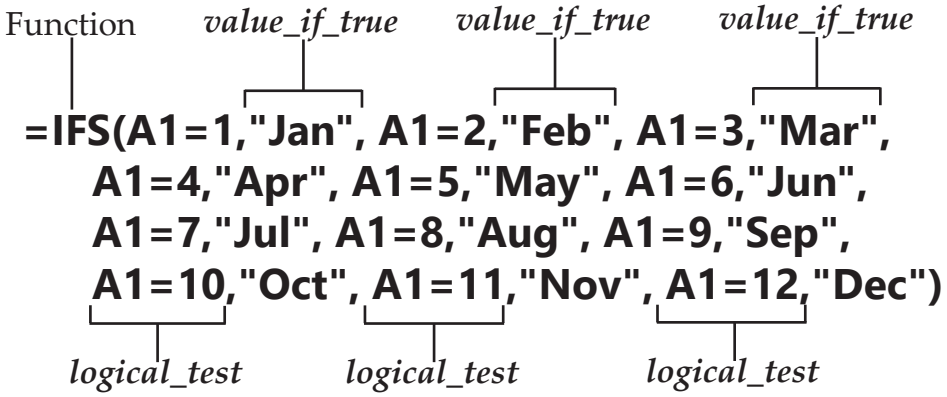
- ***logical_test:*** *[required]* any expression that evaluates to *True* or *False*
- ***value_if_true:*** *[required]* the value returned if the ***logical_test*** evaluates to *True*

These arguments can be repeated within the same function for up to 126 additional conditions.

The syntax of the **IFS** function is as follows:

=IF(***logical_test, value_if_true,*** **[*****logical_test, value_if_true*****]...)**

Each subsequent ***logical_test*** and ***value_if_true*** argument pair adds another conditional check and return option. In the following example, **IFS** simplifies the multiple-condition formula syntax to handle the twelve conditions provided effortlessly. If none of the argument pairs returns *True*, **IFS** returns **#N/A**.



Action 4 – IFS Function

Instructions:	Results/ Comments:
1. Open **Logical Formulas.xlsx**	**Objective**: Display the month abbreviation when given a month number.
2. Save the file as **My Logical Formulas.xlsx**	
3. Ensure the ***IFS Function*** sheet is selected.	
4. Select cell **A2.**	**A2** will give the text value of the numbered month in **A1**.
5. Enter this formula: **=IFS(A1 = 1, "Jan", A1 = 2, "Feb",** **A1 = 3, "Mar", A1 = 4, "Apr",** **A1 = 5, "May", A1 = 6, "Jun",** **A1 = 7, "Jul", A1 = 8, "Aug",** **A1 = 9, "Sep", A1 = 10, "Oct",** **A1 = 11, "Nov", A1 = 12, "Dec")**	The spacing is for readability only and does not need to be matched in the formula bar. Unlike a nested **IF**, **IFS** allows up to 127 conditions within the same function.
6. Select cell **A1.**	Since **A1** contains **2**, **A2** shows **Feb**.
7. Enter any value between **1** and **12.**	When a numeric month is entered in **A1**, the corresponding text month is returned in **A2**. Entering any value outside of the range returns **#N/A**.
8. Save the file and leave it open.	**[Ctrl + S]**

Logical Functions, continued

AND Function

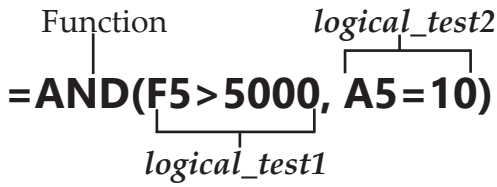
Sometimes, a logical function must test multiple conditions to return the correct value. Using an **AND** function, if all parameters evaluate to *True*, the formula returns *True*; if any condition returns *False*, the formula returns *False*. The **AND** function expands the capabilities of other logical functions.

For example, an **IF** function performs a ***logical_test*** and returns one value if the test evaluates to *True* and another if the test evaluates to *False*. By using the **AND** function in the ***logical_test*** argument of the **IF** function, many different conditions can be tested instead of just one.

The **AND** function takes the following arguments:

- ***logical_test1:*** *[required]* any expression that evaluates to *True* or *False*
- ***logical_test2,:*** *[optional, up to 254]* any expression that evaluates to *True* or *False*

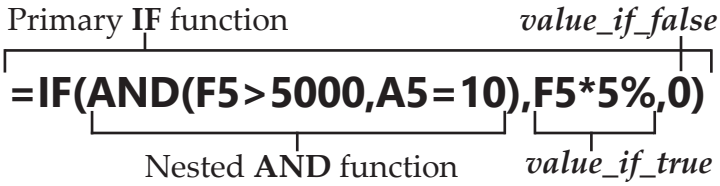
The syntax of the **AND** function is as follows:



If the value in cell **F5** is greater than 5000 and the value in cell **A5** equals 10, the formula returns *True*. If either of the values is *False* in the **AND** function, the function returns *False*.

Using AND in an IF Function

In the example below, if both conditions of **AND** are *True*, the **IF** function returns its ***value_if_true***. If either of **AND's *logical_tests*** is *False*, **IF** returns its ***value_if_false***.



Action 5 – AND Function

Instructions:	Results/ Comments:
1. **My Logical Formulas.xlsx** should still be open. If not, open it again.	**Objective**: Determine the Member Level for each salesperson.
2. Navigate to the ***Summary*** sheet.	
3. Select cell **F4.**	
4. Enter this formula: **=AND(D4 > 1800, E4 = ”yes”)** Then press **[Enter].**	This formula checks if the commission earned exceeds $1800 and the value in cell **E4** is **Yes**. If both conditions are met, the formula returns *True.*
5. Using *AutoFill*, fill cells **F5:F14.**	The cell range should have a mix of *True* and *False* values.
6. Select cells **F4:F14.**	
7. Click on the formula bar to edit the formula.	Selecting a range and editing in the formula bar applies the edit to all the selected cells simultaneously after pressing **[Ctrl + Enter].**
8. Revise the formula to include a nested **IF** function: **=IF(** AND(D4 > 1800, E4 = “Yes”), **“Gold”, IF(AND(D4 > 1600, E4 = ”Yes”), “Silver”, ”Not Eligible”))** Then press **[Ctrl + Enter]**	This nested formula uses two **AND**s in two **IF**s to check for multiple conditions and then returns the applicable membership level.
9. Save the file.	**[Ctrl + S]**

Logical Functions, continued

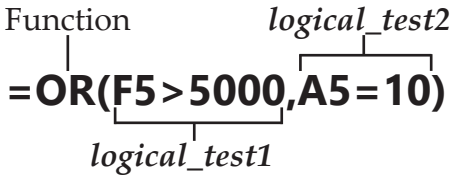
OR Function

While the **AND** function requires all ***logical_tests*** to evaluate to *True* for **AND** to return *True*, the **OR** function returns *True* if any ***logical_test*** evaluates to *True*. **OR** returns *False* only if all ***logical_tests*** evaluate to *False*.

The **OR** function takes the following arguments:

- ***logical_test1:*** *[required]* any expression that evaluates to *True* or *False*
- ***logical_test2,:*** *[optional, up to 254]* any expression that evaluates to *True* or *False*

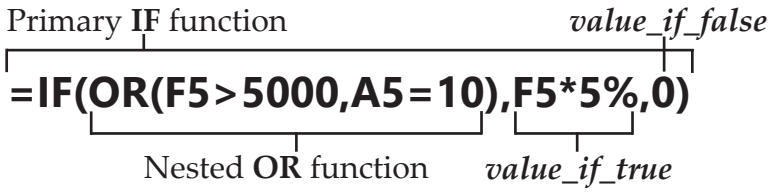
The syntax of the **OR** function is as follows:



In this example, **OR** returns *True* if either ***logical_test*** is *True*. Both conditions need to be *False* for the function to return *False*.

Using OR in an IF Function

Like the **AND** function, the **OR** function can be nested in an **IF** function to expand the **IF**'s capabilities. If any of **OR**'s ***logical_tests*** evaluates to *True*, **IF** returns its ***value_if_true***. If both ***logical_tests*** evaluate to *False*, **IF** returns its ***value_if_false***.



Action 6 – OR Function

Instructions:	Results/ Comments:
1. **My Logical Formulas.xlsx** should still be open. If not, open it again.	**Objective**: Determine if a salesperson has exceeded a $7500 sales total in any quarter, entitling them to a higher bonus.
2. Select cell **G4** on the ***Summary*** sheet**.**	
3. Build this formula: **=OR(** Click on sheet ***QT1***, highlight cell **F4,** then type: **>7500** Press **[Comma]** Click on sheet ***QT2,*** highlight cell **F4,** then type: **>7500** Press **[Comma]** Click on sheet ***QT3,*** highlight cell **F4,** then type: **>7500** Press **[Comma]** Click on sheet ***QT4,*** highlight cell **F4,** then type: **>7500)** Press **[Enter]**.	When finished, the formula will read as **=OR('QT1'!F4 > 7500, 'QT2'!F4 > 7500, 'QT3'!F4 > 7500, 'QT4'!F4 > 7500)** Added spacing for readability. If any of these *logical tests* evaluate to *True,* the formula returns *True*. Remember to close the parentheses.
4. Use *AutoFill* to fill cells **G5:G14.**	The cell range should have a mix of *True* and *False* values.
5. Select cell **G4** again.	Nest the **OR** function within an **IF** to determine the bonus amount the salesperson earned.
6. Double-click into the cell to edit the formula.	
7. Add these changes to the formula: **=IF(** OR('QT1'!F4>7500,'QT2'!F4>7500, 'QT3'!F4>7500,'QT4'!F4>7500), **C4 * 0.07, C4 * 0.03)**	Add only the bold text to the formula. Added spacing for readability. The result is $2,366.00 in **G4**.
8. Use *AutoFill* to update the formula in cells **G5:G14**	Salespeople who sell over $7500 in any quarter receive a 7% bonus. Otherwise, they receive a 3% bonus.
9. Save the file.	**[Ctrl + S]**

Logical Functions, continued

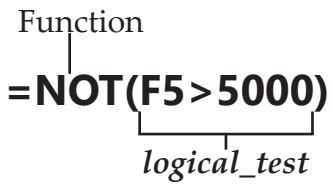
NOT Function

The **NOT** function reverses the value of its argument. Use **NOT** to ensure a value does not equal another specified value. If the ***logical_test*** evaluates to *False*, **NOT** returns *True*; if the ***logical_test*** evaluates to *True*, **NOT** returns *False*. Unlike the **AND** or **OR** functions, which can contain multiple ***logical_tests***, the **NOT** only contains one.

The **NOT** function takes the following argument:

- ***logical_test***: *[required]* any expression that evaluates to *True* or *False*

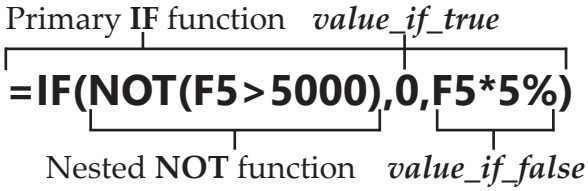
The syntax of the **NOT** function is as follows:



If the value in cell **F5** is less than 5000, the **NOT** function returns *True*. If the value in cell **F5** exceeds 5000, the **NOT** function returns *False*.

Using NOT in an IF Function

Use a combination of **NOT** and **IF** functions to ensure a specified condition is not met to get a positive result. If **NOT**'s ***logical_test*** returns *True*, **IF** returns its ***value_if_false***. If **NOT**'s ***logical_test*** returns *False*, **IF** returns its ***value_if_true.***



In the example above, if the value in cell **F5** is less than 5000, the **NOT** function returns *True,* and the **IF** function returns its ***value_if_true***, 0. If the value in **F5** is greater than 5000, **NOT** returns *False,* and the **IF** function returns its ***value_if_false***, **F5 * 5%**. When the cell containing the formula begins returning values instead of 0, the value in **F5** has exceeded 5000.

Action 7 – NOT Function

Instructions:	Results/ Comments:
1. **My Logical Formulas.xlsx** should still be open. If not, open it again.	**Objective**: Determine whether a salesperson requires a review based on their sales tier.
2. Select cell **H4.**	
3. Enter this formula: **= NOT(F4 = "Gold")** Then press **[Enter].**	If cell **F4** contains the word "Gold," the **NOT** formula returns *False*; for any other value in cell **F4,** the formula returns *True*.
4. Use *AutoFill* to fill cells **H5:H14.**	Only Gold-level salespeople should have *False* values. These salespeople do not require reviews.
5. Select cell **H4.**	Modify the formula to return a more easily understood response of *Yes* or *No*.
6. Make changes to the formula: **=IF(**NOT(F4="Gold")**,"Yes","No"**) Then press **[Enter].**	Add only the bold text to the formula. **H4** changes to **Yes**.
7. Use *AutoFill* to fill cells **H5:H14.**	Only Gold-level salespeople have a *No* value in the review column; all others have *Yes* values.
8. Save and close the file.	**[Ctrl + S]** to save and **[Ctrl + W]** to close.

Lookup Functions

When searching for specific data within a worksheet or workbook, using lookup functions can be more valuable than relying solely on filters. Lookup functions allow searches across worksheets to retrieve necessary data without duplication. Data validation lists can also be added to the lookup cell to make the search more efficient.

Lookup functions in *Excel* include:

- **LOOKUP*:*** searches a single row or column for a specific value based on a value in another row or column. It has two forms: *Vector* and *Array*.
- **VLOOKUP**: *[vertical lookup]* searches for a value in the first column of a table and returns the value from another column in the same row
- **HLOOKUP**: *[horizontal lookup]* searches for a value in the first row of a table and returns a value from another row in the same column
- **XLOOKUP*:*** searches a specified range for a value in any column and returns the first match found
- **MATCH**: searches a range for a value and returns the relative position of that item in that range
- **XMATCH*:*** extends the functionality of **MATCH** to have more search options, such as search order and partial match
- **INDEX*:*** returns a value or value reference from within a table or range. It has two forms: *Array* and *Reference*.

When a lookup value is not in a table or array's first row or column, **HLOOKUP** and **VLOOKUP** cannot be used. Instead, **XLOOKUP** or a combination of **XMATCH** and **INDEX** can search the table similarly.

Lookup Functions, continued

LOOKUP Function

The **LOOKUP** function locates the position of a value within a row or column range. Based on the position of the ***lookup_value***, data can then be pulled from a parallel row or column.

LOOKUP has two forms available: *Vector* and *Array*.

The **VLOOKUP** and **HLOOKUP** functions, which will be covered later in this chapter, are recommended instead of the *Array* form of a **LOOKUP** as they simplify the syntax.

In *Excel*, the term "vector" means a single-row or single-column range, so the **LOOKUP** *Vector* form searches a single row or column for a value in another single row or column. Once the value is located, its position within the range is noted, and **LOOKUP** returns the value in the same position from another row or column. In *Excel 365*, the *Vector* form can also be replaced with the **XLOOKUP** function.

The **LOOKUP** *Vector* function takes the following arguments:

- ***lookup_value:*** *[required]* the search value
- ***lookup_vector***: *[required]* the row or column vector to search
  - **Note:** This range must be sorted in ascending order to return the correct value. If there is a mix of upper- and lower-case text, it must also be sorted to keep cases together.
- ***result_vector***: *[optional]* a range containing the desired data. It must be the same size as and parallel to the ***lookup_vector*** range if used.

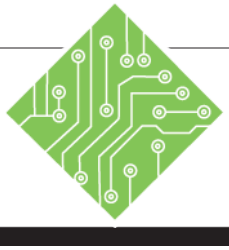
The syntax of the **LOOKUP** function is as follows**:**

= LOOKUP(***lookup_value*****,** ***lookup_vector*****, [***result_vector***])**

To only return the position of a value within the range, omit the optional ***result_vector*** argument.

If **LOOKUP** does not find a match for the ***lookup_value***, it returns the next highest value within the ***lookup_vector***. If the ***lookup_value*** is smaller than any value in the ***lookup_vector,*** the formula returns **#N/A**.

Action 8 – LOOKUP Function



Instructions:	**Results/ Comments:**
1. Open **Lookup.xlsx**	**Objective**: Look up a single part number's location, price, and availability.
2. Save it as **My Lookup.xlsx**	
3. Select cell **H2.**	**H2** will show the location of the part.
4. Enter this formula: **=LOOKUP(G2, A2:A12, D2:D12)**	The formula returns **#N/A** because **G2** is currently empty.
5. Select cell **I2.**	**I2** will show the price of the part.
6. Enter this formula: **=LOOKUP(G2, A2:A12, E2:E12)**	The formula returns **#N/A** because **G2** is currently empty.
7. Select cell **J2.**	**J2** will show the availability of the part.
8. Enter this formula: **=LOOKUP(G2, A2:A12, C2:C12)**	The formula returns **#N/A** because **G2** is currently empty.
9. Select **A7** and copy it.	**A7** contains the part being searched.
10. Select cell **G2** and paste the part number copied from **A7.**	**H2:J2** display values related to the part in question: **H2: Aisle A** **I2: $6.49** **J2: Yes**
11. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**

Lookup Functions, continued

VLOOKUP Function

The **VLOOKUP** (Vertical Lookup) function searches for specified data in the first column of a table. Once found, it returns a result from a different column in the same row. For example, a search for a student's last name in the first column of a ***table_array*** can return a phone number from the same row in the phone number column. To ensure that **VLOOKUP** returns the correct value, the first column of its ***table_array*** must be sorted in ascending order.

The **VLOOKUP** function takes the following arguments:

- ***lookup_value***: *[required]* the search value
- ***table_array***: *[required]* the range to search
- ***col_index_num***: *[required]* the column number in the ***table_array*** containing the result
- ***range_lookup***: *[optional]* an option to specify an approximate or exact match for text. "True" or "1" is the default approximate match. Enter "False" or "0" to specify an exact match.

Note

Naming the range of cells used for ***array***-type arguments makes lookup formulas easier to read.

The syntax of the **VLOOKUP** function is as follows**:**

=VLOOKUP(*lookup_value, table_array, col_index_num, [range_lookup]*)

VLOOKUP searches the first column of the ***table_array*** for the ***lookup_value*** to find the row in the ***table_array*** that contains the desired information. Once the row is established, ***col_index_num*** specifies which column of the ***table_array*** contains the desired return information. **VLOOKUP** then returns the cell content at the row and column intersection.

VLOOKUP does not need to be used on the same worksheet as the ***table_array*** it searches. The ***lookup_value*** and ***col_index_num*** arguments refer to the value passed to ***table_array***, regardless of its worksheet.

Lookup Functions, continued

VLOOKUP Example

The example below shows how to use **VLOOKUP** to retrieve the *Product Description* by *Product Number* from another sheet.

	A	B	C
1	Product #	Product Description	Price per Unit
2	101234	Gear 2 1/2"	$ 3.00
3	101235	Gear 1"	$ 1.50
4	101236	Gear 3/4"	$ 0.60
5	101237	Widget Style 1	$ 5.00
6	101238	Widget Style 2	$ 10.00
7	101239	Conduit 2"	$ 15.00
8	101240	Conduit 6"	$ 30.00
9	101241	Elbow 2"	$ 15.00
10			

Master List / Sales / Sheet3

The ***Master List*** sheet displays all products for sale. For **VLOOKUP** to work correctly, the first column of data in its lookup range must be sorted in ascending order.

B2 =VLOOKUP(A2,'Master List'!A2:C9,2)

	A	B	C	D	E
1	Product #	Product Description	Qty Sold	Price Per	Total
2	101234	Gear 2 1/2"	10	$ 3.00	$ 30.00
3	101235	Gear 1"	50	$ 1.50	$ 75.00
4					
5					

Master List / Sales / Sheet3

On the ***Sales*** sheet, the **VLOOKUP** function in cell **B2** retrieves the *Product Description* based on the *Product Number* in cell **A2**. This allows the *Product Description* to be populated on the ***Sales*** sheet without duplicating the data from the ***Master List*** sheet. Additionally, if the *Product Description* changes on the ***Master List*** sheet, it will not have to be updated in the ***Sales*** sheet. Similarly, **VLOOKUP** can automatically retrieve the *Price Per Unit* into Column D.

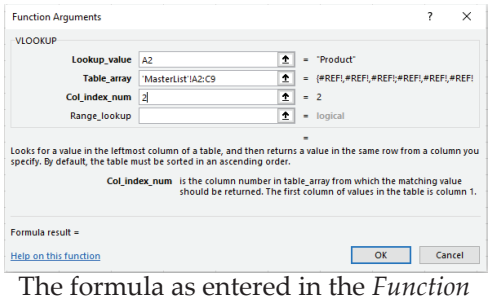
Note

VLOOKUP looks for Product #101234 (**A2**) in the ***Master List*** sheet and displays the *Product Description* in cell **B2** on the ***Sales*** sheet.

The syntax of the function in the above example is as follows:

=VLOOKUP(A2, 'Master List'!A2:C9, 2)

- ***lookup_value***: **A2** — the product number
- ***table_array*: 'Master List'!A2:C9** —the table on the ***Master List*** worksheet in the cell range **A2:C9**
- ***col_index_num*: 2** — the second column in the ***table_array*** contains the *Product Description*



The formula as entered in the *Function Arguments* dialog box.

Action 9 – VLOOKUP Function

Instructions:	Results/ Comments:
1. Open **MyLoan.xlsx**	**Objective**: Display the correct interest rate for a loan in Indiana. Interest rates depend on the state and the number of years of the loan.
2. Select cell **I5** (**Rate**).	Use **VLOOKUP** to find which row in the **InterestTable** contains the correct rate based on the number of years.
3. Enter this formula: **=VLOOKUP(Years, InterestTable, IF(State="OH", 3, 2))** Then press **[Enter].**	Each state has different interest rates per number of years, so an **IF** nested in a **VLOOKUP** can specify the ***col_index_num*** The formula returns **#N/A**. There are two errors: Indiana does not exist in the **InterestTable**, and there is no value in cell **I7 (Years)**
4. Select cell **B6** (**State**) and enter **OH**	First, test the formula with a state already in the **InterestTable**.
5. Select cell **I7** (**Years**) and enter a number between 1 and 10.	Fix the second error; the formula now returns the correct value for Ohio.
6. Select cell **D12** and type: **Indiana**	Add values for Indiana's interest rates.
7. In cell **D13**, type: **4.25%** Then press **[Enter].**	
8. In cell **D14,** type: **4.75%** Then press **[Enter]**.	
9. Select cells **D13:D14** and use *AutoFill* to populate values through cell **D22.**	Selecting both cells establishes the series *AutoFill* uses to populate the remaining cells.
10. Select cell **E13** and enter **Out of Region** Use *AutoFill* to fill the same value through **E22.**	"Out of Region" will be displayed if the formula does not find the state in the **InterestTable**.

Instructions:	Results/ Comments:
11. Select **[Name Manager]** in the **Defined Names Group** on the ***Formulas Tab***	Redefine the **InterestTable**'s range to include the new columns.
12. Select **InterestTable** and click the **[Edit]** button.	The *Edit Name* dialog box opens for the selected name.
13. Click into the **Refers to:** field and change **$C$22** to **$E$22**. Then press **[OK].**	The table range includes both new columns by changing the cell address from **C22** to **E22.**
14. Click the **[Close]** button.	
15. Select cell **I5** (**Rate**).	Fix the formula logic to account for the two new options in the **InterestTable**.
16. Add these changes to the formula: =VLOOKUP(Years, Table, IF(State="OH", 3, **IF(State="MI", 2, IF(State="IN", 4, 5))**)) Then press **[Enter].**	Only add the bold text to the formula.
17. Change the state in **B6** to **MI** or **IN** to test the new **VLOOKUP** formula.	The results in cell **I5 (Rate)** change depending on the state in **B6.** Use the **Increase Decimal** button on the **Number** group of the ***Home*** tab to show the total decimal value instead of a rounded number.
18. Change the value in the **State** cell to **MA**	Massachusetts is not part of the **InterestTable**, so it displays "Out of Region".
19. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**

Lookup Functions, continued

HLOOKUP Function

The **HLOOKUP** (Horizontal Lookup) function searches for a specified ***lookup_value*** in the first row of a table. After finding the column containing the ***lookup_value***, **HLOOKUP** returns a result from a different specified row in the same column.

The **HLOOKUP** function takes the following arguments:

- ***lookup_value***: *[required]* the search value
- ***table_array***: *[required]* the range to search
- ***row_index_num***: *[required]* the row number in the ***table_array*** containing the result
- ***range_lookup***: *[optional]* an option to specify an approximate or exact match for text. "True" or "1" is the default approximate match. Enter "False" or "0" to specify an exact match.

The syntax of the **HLOOKUP** function is as follows:

=HLOOKUP(*lookup_value*, *table_array*, *row_index_num*, [*range_lookup*])

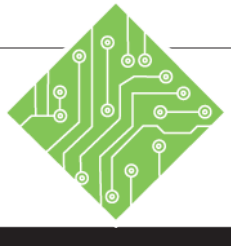
In the following example, Column C displays each student's letter grade. **HLOOKUP** finds the value of **B2**, then searches the first row of ***table_array*** **E4:M5** for the exact ***lookup_value*** or the next largest value. **HLOOKUP** returns the grade in ***row_index_num*** **2**, the second row of ***table_array***, to Column C.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Name	Grade %	Letter Grade										
2	Sara S	99	A+										
3	Mike S	81	B-		Grading Scale								
4	Scott M	60	D-		0	60	65	70	75	80	85	90	95
5	Angie H	90	A-		F	D-	D+	C-	C+	B-	B+	A-	A+
6	Jason P	83	B-										

The function in **C2**:
=HLOOKUP(*B2, $E$4:$M$5, 2*)

Using the default *approximate* option for the ***range_lookup*** argument, *Excel* does not need each number tied to a letter grade. Instead, the lookup function will return the exact match if found; if an exact match is not found, it will return the next largest value. Although the value **99** was not listed in the table, **HLOOKUP** returned the correct letter grade by using the next largest value to **99**, which was **95**.

Action 10 – HLOOKUP Function



Instructions:	Results/ Comments:
1. Open **Discount.xlsx**	**Objective**: Compare the sales total to a discount table to determine the eligible discount amount for a sale.
2. Save the file as **My Discount.xlsx**	
3. Navigate to the ***Discount Table*** sheet.	This sheet may already be open.
4. Select cells **A2:F3.**	
5. Name the range **DiscountTable.**	Use the **Name Box** to apply the name to the cell range.
6. Navigate to the ***Sales*** sheet.	The ***Sales*** sheet will use **HLOOKUP**
7. Select cell **I2.**	**I2** will display the discount applied.
8. Enter this formula: **=HLOOKUP(H2, DiscountTable, 2)** Then press **[Enter]**.	This syntax for **HLOOKUP** compares the *Total* in **H2** to the first row in the named range **DiscountTable**. When it finds an exact or next largest (approximate) match, the discount amount from **DiscountTable**'s second row is returned.
9. Use *AutoFill* to fill the remaining cells.	
10. Select cell **J2**.	
11. Enter the following formula: **=H2 - (H2 * I2)** Then press **[Enter].**	Calculate the price after the discount.
12. Format Column J as currency and use *AutoFill* to fill the remaining cells.	
13. Save the file.	**[Ctrl + S]**

Lookup Functions, continued

Note

XLOOKUP cannot be added in *Excel 2019* or earlier versions, but if it was added by someone working in *Excel 2021/365*, it still functions.

XLOOKUP Function

A newer, more powerful variation of the **LOOKUP** function is **XLOOKUP**. Unlike **HLOOKUP** and **VLOOKUP,** which must use the first row or column in a data set to search for the ***lookup_value***, **XLOOKUP** can search any column within a data set to pull information from that row. It can also return each value from an entire row if the ***return_array*** argument includes multiple columns, also called *spill*. **XLOOKUP** can pull an entire record by searching for a single ***lookup_value*** in any ***table_array*** column.

The **XLOOKUP** function takes the following arguments:

- ***lookup_value***: *[required]* the search value
- ***lookup_array*:** *[required]* the range to search
- ***return_array***: *[required]* a range containing the possible results of the search
- ***if_not_found***: *[optional]* defaults to **#N/A**. Use this argument to enter custom text to return when no match is found.
- ***match_mode:*** *[optional]* match options for the search
  - **0** - *[default]* exact match. **#N/A** is returned when no exact match is found.
  - **-1** - exact match or the next smallest value
  - **1** - exact match or the next largest value
  - **2** - match using the following wildcards: * ? ~
- ***search_mode:*** *[optional]* search order
  - **1** - *[default]* first to last
  - **-1** - last to first
  - **2** - binary search ascending. If not sorted in ascending order, invalid results are returned.
  - **-2** - binary search descending. If not sorted in descending order, invalid results are returned.

Note

Wildcard characters:
* = Any number of any characters
? = Any single character
~ = Wildcard modifier

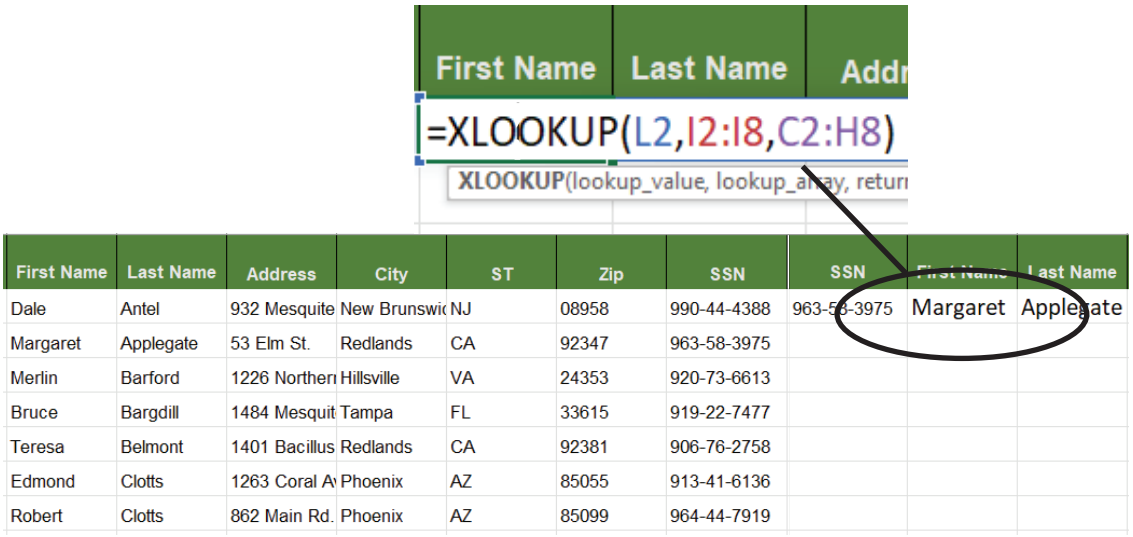
Lookup Functions, continued

The **XLOOKUP** function syntax is as follows:

=XLOOKUP(*lookup_value*, *lookup_array*, *return_array*, *[if_not_found]*, *[match_mode]*, *[search_mode]*)

In the example below, **XLOOKUP** displays an employee's record by matching a value in the *SSN* column and pulling the data on the same row from the ***return_array*** **C2:H8**.

The *SSN* ***lookup_value*** in **L2** is searched for in the ***lookup_array***, which contains the *SSN* column. When the ***lookup_value*** is found, the ***return_array*** values of that row are returned. Since the ***return_array*** includes multiple columns of data, all columns in the ***return_array*** range are returned, beginning in the cell containing the **XLOOKUP** function. If the function only needs to return a single column's value, the ***return_array*** should only contain that column.



First Name	Last Name	Address	City	ST	Zip	SSN	SSN	First Name	Last Name
Dale	Antel	932 Mesquite	New Brunswic	NJ	08958	990-44-4388	963-58-3975	Margaret	Applegate
Margaret	Applegate	53 Elm St.	Redlands	CA	92347	963-58-3975			
Merlin	Barford	1226 Northern	Hillsville	VA	24353	920-73-6613			
Bruce	Bargdill	1484 Mesquit	Tampa	FL	33615	919-22-7477			
Teresa	Belmont	1401 Bacillus	Redlands	CA	92381	906-76-2758			
Edmond	Clotts	1263 Coral Av	Phoenix	AZ	85055	913-41-6136			
Robert	Clotts	862 Main Rd.	Phoenix	AZ	85099	964-44-7919			

The syntax of the function above is as follows:

=XLOOKUP(L2, I2:I8, C2:H8)

- ***lookup_value***: **L2** — the SSN
- ***lookup_array*: I2:I8** —the *SSN* master column to search
- ***return_array*: C2:H8** — the values to return beginning in the cell containing this **XLOOKUP**

Action 11 – XLOOKUP Function

Instructions:	Results/ Comments:
1. **My Discount.xlsx** should still be open. If not, open it.	**Objective**: Return a person's name and address based on their SSN.
2. Navigate to the ***Employee Records*** worksheet.	***Employee Records*** is the third worksheet in the workbook.
3. Click the **Name Box** dropdown and select *SSN_Lookup.*	Two named ranges have already been created. The cell range **I2:I65** was named *SSN_Lookup*, so it is highlighted.
4. Click the **Name Box** dropdown and select *Records*.	Notice that *Records* is the range of columns to the left of *SSN_Lookup*, not including the *Employee ID* column.
5. Select cell **M2.**	
6. Enter the following formula: **=XLOOKUP(L2, SSN_Lookup, Records, "No Record Found")** Then press **[Enter].**	"No Record Found" is displayed since there is no value in cell **L2**
7. Select any cell in the *SSN_Lookup* range and copy it.	**[Ctrl + C]** to copy.
8. Click on cell **L2** and paste the copied value.	**[Ctrl + V]** to paste. The formula now returns the name and address for the record associated with the SSN.
9. Try entering a new SSN not on the list into cell **L2.** Removing a digit from the SSN is an easy way to do this.	"No Record Found" is returned since there is no match.

Instructions:	Results/ Comments:
10. Try a more specific range.	**Objective**: Return only a person's full name based on their SSN.
11. Select cells **C2:D65.**	This range includes the first and last name columns.
12. Click into the **Name Box** field and type: **FullName** Then press **[Enter].**	Create a named range containing only the desired results.
13. Select cell **M5.**	
14. Enter the following formula: **=XLOOKUP(L5, SSN_Lookup, FullName, "No Record Found")** Then press **[Enter].**	The result "No Record Found" should be displayed since cell **L5** is empty.
15. Copy another SSN.	Select any value in the *SSN* column. **[Ctrl + C]**
16. Select cell **L5** and paste the copied SSN into the cell.	The first and last names of the person associated with SSN in cell **L5** are now displayed.
17. Save the file.	**[Ctrl + S]**

Lookup Functions, continued

XMATCH Function

XMATCH is another new function available in *Excel 365*. It was designed to replace the **MATCH** function and offer greater versatility. **XMATCH** can perform lookups in both vertical and horizontal arrays.

Like **XLOOKUP**, **XMATCH**'s ***search_mode*** argument allows searching in ascending and descending order, and ***match_mode*** can find exact, approximate, or partial matches. **XMATCH** also uses a faster binary search algorithm than **MATCH**.

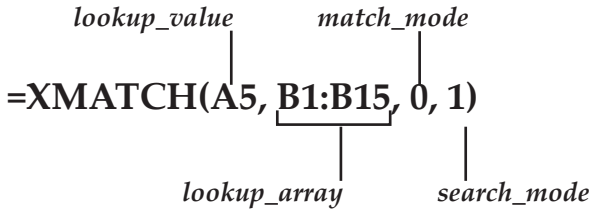
Both **MATCH** and **XMATCH** functions return the relative position of a value in an array or a range of cells.

The **XMATCH** function takes the following arguments:

- ***lookup_value***: *[required]* the search value
- ***lookup_array***: *[required]* the range to search
- ***match_mode:*** *[optional]* match options for the search
  - **0** - *[default]* exact match. **#N/A** is returned when no exact match is found.
  - **-1** - exact match or the next smallest value
  - **1** - exact match or the next largest value
  - **2** - match using the following wildcards: * ? ~
- ***search_mode:*** *[optional]* search order
  - **1** - *[default]* first to last
  - **-1** - last to first
  - **2** - binary search ascending. If not sorted in ascending order, invalid results are returned.
  - **-2** - binary search descending. If not sorted in descending order, invalid results are returned.

Lookup Functions, continued

The syntax of the **XMATCH** function is as follows:



In the example below, ***lookup_value*** uses a cell reference in the first **XMATCH** instance and a literal value in the second. Remember that if the ***lookup_value*** is text, it must be wrapped in quotation marks. Both examples omit the ***match_mode*** and ***search_mode*** arguments, using their defaults, 0 and 1.

Day		Search day	Position
Monday		Wednesday	3
Tuesday			3
Wednesday			
Thursday			
Friday			
Saturday			
Sunday			

```
XMATCH(C5,A5:A11)
XMATCH("Wednesday",A5:A11)
```

In the example below, ***match_mode*** is specified as **1** to search for partial ***lookup_values*** in the first instance and **2** to match using wildcards in the second and third instances.

Day		Search day	Position
Monday		Wed	3
Tuesday		Wed	3
Wednesday		Wed*	3
Thursday			
Friday			
Saturday			
Sunday			

```
XMATCH(C15,A15:A21,1)
XMATCH("wed*",A15:A21,2)
XMATCH(C17,A15:A21,2)
```

In the example below, **XMATCH** again does an approximate search, but because the ***lookup_value*** is numeric instead of text, it operates like a **COUNT** function.

Day	Sales	SearchValue	Count of
Monday	$25,000.00	$ 20,000.00	4
Tuesday	$ 8,000.00		
Wednesday	$32,000.00		
Thursday	$21,000.00		
Friday	$41,000.00		
Saturday	$16,000.00		
Sunday	$12,000.00		

```
XMATCH(C25,B25:B31,1)
```

Lookup Functions, continued

Note

row_num and ***column_num*** must point to a cell within the ***array***. Otherwise, **INDEX** returns the **#REF!** error.

Note

Array constants contain only literal values and no other arrays, formulas, or functions.

Note

When using both ***row_num*** and ***column_num***, **INDEX** returns the value at the intersection of ***row_num*** and ***column_num***.

INDEX Function

The **INDEX** function can return a value or the reference to a value from a table or range. Both the column and row within the cell range can be referenced. The results of an **XMATCH** function can be used as **INDEX**'s ***lookup_value*** to pull data from another column along the row defined by **XMATCH**. There are two forms of the **INDEX** function: *Array* and *Reference*.

The **INDEX** function's *Array* form returns the value of an element in an ***array*** using the row and column positions specified. Use **INDEX**'s *Array* form on array constants.

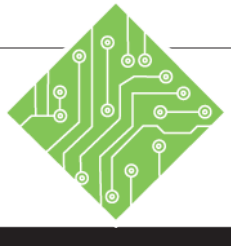
The **INDEX** *Array* function takes the following arguments:

- ***array***: *[required]* the cell range or array constant to be searched
  - If the ***array*** contains only one row or column, the corresponding ***row_num*** or ***column_num*** argument is optional.
  - If the ***array*** has more than one row and more than one column, and only ***row_num*** or ***column_num*** is used but not both, **INDEX** returns the entire row or column in the ***array***.
- ***row_num***: *[required unless* ***column_num*** *has a value]* the row in the ***array*** to return a value. This argument is similar to the ***lookup_value*** argument in previously discussed functions. If ***row_num*** is omitted, ***column_num*** is required.
- ***column_num:*** *[required unless* ***row_num*** *has a value]* the column in the ***array*** to return a value. This argument is similar to the ***lookup_value*** argument in previously discussed functions. If ***column_num*** is omitted, ***row_num*** is required.

The syntax of the **INDEX** *Array* function is as follows:

=INDEX(*array*, *row_num*, [*column_num*])

The **INDEX** function's *Reference* form returns a reference to part or all of a ***reference*** to one or several non-adjacent ranges. Other formulas using this cell can then use the cell reference instead of its literal value.



Instructions:	Results/ Comments:
1. **My Discount.xlsx** should still be open. If not, open it.	**Objective**: Use **XMATCH** and **INDEX** to find exact and approximate matches for different values in a table and practice nesting functions.
2. Navigate to the ***Locate Positions*** worksheet.	
3. Select cell **I2.**	
4. Enter the following formula: **=XMATCH(170000, A2:A28, 0)** Then press **[Enter].**	**XMATCH** returns the ***lookup_value***'s row within the ***array.*** In this case, row **15.**
5. Edit the formula in cell **I2** to search for the value **172500** instead of **170000.**	Since this ***lookup_value*** is not in the ***array***, **#N/A** is returned. **XMATCH's *match_mode*** argument is set to **0**, the default exact match.
6. Change the lookup value back to **170000.**	Since this ***lookup_value*** is in the ***array***, the row number is again returned as **15.**
7. Select cell **I3** and enter the following formula: **=XMATCH(172500, A2:A28 ,1)** Then press **[Enter].**	***match_mode*** option **1** returns the location of the cell containing the next largest value to the ***lookup_value*** if it does not find the exact ***lookup_value***. The result is row **16.**
8. Select cell **I4** and enter the following formula: **=XMATCH(172500,A2:A28,-1)** Then press **[Enter].**	***match_mode*** option **-1** returns the location of the cell containing the next smallest value to the ***lookup_value*** if it does not find the exact ***lookup_value***. The result is row **15.**
9. Select cell **I6** and enter the following formula: **=XMATCH(12%, B1:F1, 0)** Then press **[Enter].**	***match_mode*** option 0, used with the ***array***'s column headers, returns the index number of the column containing the ***lookup_value***. The column index number is **3.**

Instructions:	Results/ Comments:
10. Select cell **I9** and enter the following formula: **=INDEX(A2:A28, I3)** Then press **[Enter].**	Now that **XMATCH** has identified the column containing the value, **INDEX** can use the value returned by **XMATCH** to return the correct value within the column. **I9** displays **175000.**
11. Select cell **I10** and enter the following formula: **=XLOOKUP(INDEX(A2:A28, I3), A2:A28, A2:F28)** Then press **[Enter].**	**XLOOKUP** returns an entire row from the ***array*** based on the results of the **INDEX** and **XMATCH** functions returned previously.
12. Select cell **I12** and combine everything into a single formula: **=XLOOKUP(INDEX(A2:A28, XMATCH(172500, A2:A28, 1)), A2:A28, A2:F28)** Then press **[Enter].**	Nest the **XMATCH** inside the **INDEX** within the **XLOOKUP**. Now, this formula can be in a single cell. The results of this combined formula match the row above it exactly.
13. Save the file.	**[Ctrl + S]**
	Challenge: Notice that the range **A2:A28** is referenced several times in this exercise. For a challenge, try naming the range and replacing its references to make a cleaner formula to read in the formula bar.

Lookup Functions, continued

Using XMATCH to Compare Lists

Compare different cell ranges by using **XMATCH** with the **ISNA** and **IF** functions.

The **ISNA** function checks for **#N/A** errors in cells and formulas. As a logical function, **ISNA** returns *True* or *False* and takes only one argument, the cell being checked. If an **#N/A** error is detected, **ISNA** returns *True*; if not, it returns *False*.

The syntax of the **ISNA** function is as follows:

=ISNA(*value*)

In the comparison formula below, **XMATCH** is passed as the ***value*** for the **ISNA** function. **XMATCH** returns **#N/A** when it does not find an exact match, causing **ISNA** to return *True*. If **XMATCH** finds an exact match, **ISNA** returns *False*.

=ISNA(XMATCH(*lookup_value*, *lookup_array*))

Using this nested formula as the ***logical_test*** in an **IF** function allows different messages to return when matches are found and not found.

=IF(ISNA(XMATCH(*lookup_value*, *lookup_array*)), *value_if_true*, *value_if_false*)

In the example below, **XMATCH** compares values in a single column (***array)*** to values in a second column. When it finds a match, **XMATCH** returns its row position to **ISNA**, which returns *False* to **IF**. **IF** returns its ***value_if_false***, "Match Found." If **XMATCH** does not find a match, it returns **#N/A** to **ISNA**, which then returns *True* to **IF**, and **IF** returns its ***value_if_true***, an empty string.

nancy			IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
janet	robert	Match Found	IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
michael	laura	Match Found	IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
margaret			IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
robert	janet	Match Found	IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
anne			IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
laura			IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
steven	anne	Match Found	IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")
andrew			IF(ISNA(XMATCH($B$48:$B$56,$A$48:$A$56))," ","Match Found")

Instructions:	Results/ Comments:
1. **My Discount.xlsx** should still be open. If not, open it.	**Objective**: Find retained employees from last year to this year.
2. Navigate to the ***Employee Search*** worksheet.	
3. Select cell **F2.**	
4. Enter the following formula: **=IF(ISNA(XMATCH(B2:B65, D2:D65)), "No", "Yes")** Then press **[Ctrl + Enter]** to keep the same cell selected.	If the employee name is in both columns, this formula returns *Yes* in the Retained Employees column. If the value in Column B matches the value in Column D, **XMATCH** returns *True*. **ISNA** then returns *False* because **XMATCH** did not return **#N/A**, and so **IF** returns its ***value_if_false***, "Yes" If the names do not match, **XMATCH** returns **#N/A**, which means **ISNA** returns *True*, and **IF** returns its ***value_if_true***, "No" The list of employees retained from last year is now clearly indicated.
5. Use *AutoFill* to fill cells **F3:F65.**	
6. Select cells **A2:A65.**	
7. On the ***Home tab***, expand the **Conditional Formatting** dropdown menu and select *New Rule*	The *New Formatting Rule* dialog opens.
8. In the *New Formatting Rule* dialog, under ***Select a Rule Type***, select *Use a formula to determine which cells to format*	The ***Edit the Rule Description:*** area of the *New Formatting Rule* dialog changes to allow entering formulas and setting the formatting.

Instructions:	Results/ Comments:
9. In the **Format values where this formula is true:** field, enter the following formula: **=XMATCH(B2:B65, D2:D65)** Press **[Enter]** to apply the formatting.	This formula compares the lists and returns *True* for matches, which applies the formatting.
10. Click the **Format** button.	The *Format Cells* dialog opens.
11. Select the ***Fill*** tab in the *Format Cells* dialog.	Fill color options are displayed.
12. Choose a fill color to apply and select **OK** twice.	The formatting is applied to all employee IDs if the name is found in both lists.
13. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**

LAMBDA Functions

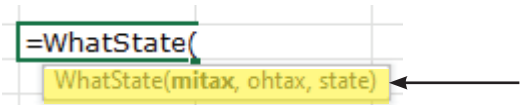
LAMBDA functions are custom, reusable formulas called by a defined name. Like built-in functions, they can use arguments (also called ***parameters***) to perform ***calculations***. **LAMBDA** functions often begin as formulas that appear multiple times throughout a worksheet or workbook that need to be named for easier access, similar to why a cell or range might be named.

As mentioned in **Naming Cells**, *Name Manager* alone can name simple formulas that refer to a static range of cells. However, if the formula requires changing values, a **LAMBDA** function must first be used before naming the function with *Name Manager*. **LAMBDA** functions are only available in *Excel 365* with an active subscription. In previous versions of *Excel*, custom functions must be created using macros and *Visual Basic for Applications* (*VBA*).

Note: Use values passed as ***parameters*** (arguments) in the calculation to make the function more reusable.

The **LAMBDA** function takes the following arguments:

- ***parameter,****: [optional, repeatable]* a name for a required value passed to the ***calculation*** as an argument
  - ***Parameter*** names will show in *AutoComplete* when the function is called.



  - ***calculation***: *[required]* any necessary functions and arithmetic to produce this function's intended return

The syntax of the **LAMBDA** function is as follows:

=LAMBDA([*parameter1*], [*parameter2*]..., *calculation*)

To create a custom **LAMBDA** function, first write the formula in a cell, as usual, to test it and ensure the logic works. This logic becomes the ***calculation***. The **LAMBDA** function will not work if the formula logic does not work.

When the formula works independently, it is time to put it into the ***calculation*** argument of the **LAMBDA** function. If it needs any values passed in for the ***calculation***, define them as ***parameters*** in **LAMBDA**'s syntax. To avoid a **#CALC!** error, call the function in the same cell to return the result. The following example describes this process in more detail.

LAMBDA Functions, continued

Creating a LAMBDA Function

Open a file containing a formula that can be reused. The following example creates a **LAMBDA** function for a simple arithmetic formula that calculates the difference between two cells. Observe the formula logic and cells containing the required information.

First Value	100
Last Value	75
Difference Value	=B1-B2

Cells **B1** and **B2** will be the arguments (***parameters***) passed to the **LAMBDA** function call. The ***calculation*** is the formula in cell B4.

- Starting in cell **B6**, create a **LAMBDA** function in any cell. Begin the formula:

  =LAMBDA(

- Enter a name representing the minuend (the number being subtracted from) in cell **B1** as the first ***parameter***. In this example, the first ***parameter*** is named **Minuend**

  =LAMBDA(Minuend

  - Notice that **B1** itself is not referenced; the formula only references the cell to use for calculation when it is called.

- Add a comma to indicate that the first ***parameter*** is defined and to begin the definition for the second ***parameter***

  =LAMBDA(Minuend,

- Enter a ***parameter*** name representing the subtrahend (the number being subtracted) in cell **B2**. In this example, the second ***parameter*** is named **Subtrahend**

  =LAMBDA(Minuend,Subtrahend

- Add a comma to indicate that the second parameter is defined. Although this example needs no additional ***parameters***, **LAMBDA** can calculate up to 253 ***parameters*** in total.

  =LAMBDA(Minuend,Subtrahend,

LAMBDA Functions, continued

- Next, add the formula's now-established ***parameter*** names for the ***calculation***, subtracting the **Subtrahend** from the **Minuend**.

=LAMBDA(Minuend,Subtrahend,Minuend-Subtrahend)

- While typing the ***parameter*** name in the ***calculation***, observe *AutoComplete*'s suggestion. ***Parameter*** names show with an *x* icon.

At this point, the function returns an error unless it is called in the same cell. A ***parameter*** is just a placeholder, and **LAMBDA** needs the value for each ***parameter*** to complete its ***calculation***.

- Call the function in the same cell by adding a second set of parentheses after the **LAMBDA** function definition. Insert the cell address that the **Minuend** ***parameter*** refers to, followed by a comma, and then the cell address for the **Subtrahend** ***parameter***.

=LAMBDA(Minuend,Subtrahend,Minuend-Subtrahend)(B1,B2)

- Do not select multiple cells.
- Use a comma to separate each cell reference.
- Do this in the order defined by the formula. The first value called is assigned to the first parameter, the second to the second, and so on.

This example returns **25** after subtracting **B2** (**75**) from **B1** (**100**).

Note: The *Insert Function* dialog and **Function Lists** only display built-in functions and does not list the names of custom functions.

Parameter constraints in **LAMBDA** functions:

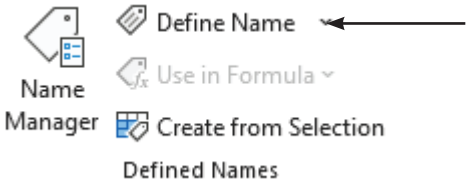
- ***Parameter*** names must not exceed 255 characters.
- ***Parameter*** names can only begin with a letter, an underscore (_), or a backslash (\ ). Periods (.) may not be used in ***parameter*** names.
- Do not use names of existing *Excel* functions; this can cause errors in syntax and logic.
- Do not create ***parameter*** names similar to cell references, such as **AAB1.**
- **LAMBDA** functions support up to 253 ***parameters***.

LAMBDA Functions, continued

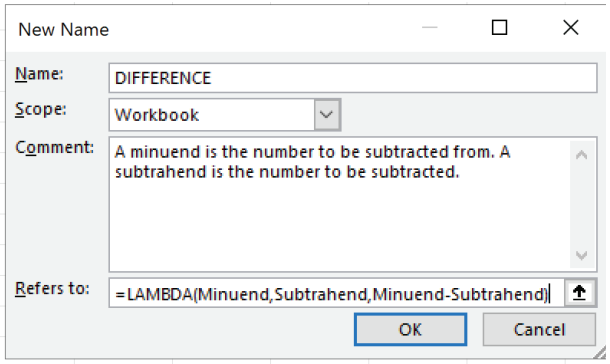
Naming a LAMBDA Function

Now that the logic works in the previous example, the **LAMBDA** function can be named for reuse. Use the same naming conventions as cells and ranges, which are covered at the beginning of this chapter.

- Navigate to the ***Formulas*** tab in the ribbon.
- In the **Defined Names** Group, click **[Define Name]**



- Create the named function in the *New Name* dialog box.



  - Name the function using a simple and descriptive name, such as **DIFFERENCE**
  - Keep the scope as *Workbook* to call this function from any sheet in the current workbook.
  - Comments can be added to explain *how* and *why* something works, but they are not required. Since the two words used as parameters are uncommon, definitions are added in the comments as an example.
  - Enter the **LAMBDA** function definition in the **Refers to:** field.

=LAMBDA(Minuend, Subtrahend, Minuend-Subtrahend)

  - Click **[OK]** to save the custom function.

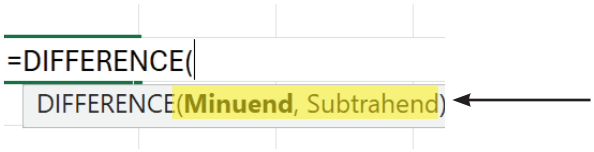
Note

No function call in a second set of parentheses is necessary here as this step defines the function for *Excel* to store in *Name Manager*. It will be called from a cell in the next step.

LAMBDA Functions, continued

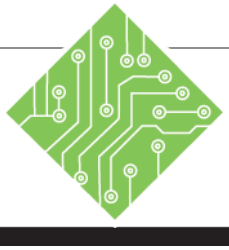
Using a Named LAMBDA Function

- Select any cell (such as **B1**) and enter any numeric value. Then, select another cell (such as **B2**) and again enter any numeric value. Either cell can be used as the minuend or subtrahend to find the difference.
- Select any cell to enter the formula and display the ***calculation***
- Begin the formula as usual with an equal sign, followed by the name given to the **LAMBDA** function in the previous example, such as **DIFFERENCE**.
- Notice that *AutoComplete* suggests the name. Select the correct function using the arrow keys and **[Tab]** or double-clicking it in the *AutoComplete* list.

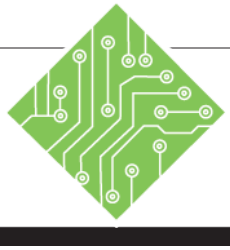


- The *AutoComplete* tooltip shows the named ***parameters*** the custom function requires, which is why they should be descriptive.
- Select the cell containing the value to use as the first ***parameter***, the **Minuend**. As indicated in the tooltip, type in a comma, then select the cell containing the value to fill the second ***parameter***, the **Subtrahend**.
- Press **[Enter]** to finish entering the formula and see the result.

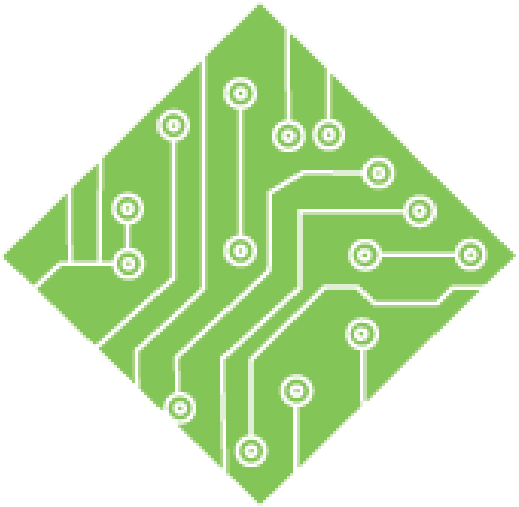
To use this function across multiple worksheets, do not use any specific cell addresses as part of the ***calculation*** in the **LAMBDA** function. *Excel* will automatically add the sheet name reference to all cell addresses. Instead, use named ***parameters***, as the above example illustrates.



Instructions:	Results/ Comments:
1. Open **Loan_LAMBDA.xlsx**	**Objective**: Create, test, name, and use a **LAMBDA** function.
2. Save as **My Loan_LAMBDA.xlsx**	
3. Navigate to the ***Loan1*** worksheet and select cell **G6.**	
4. Enter the following formula: **=LAMBDA(ST,MT,OT, IF(ST = "MI", MT, OT))(B6, B12, B13)** Then press **[Enter].** Change the value in **B6** to **OH**	The second set of parentheses calls and tests the **LAMBDA** function in the same cell. It cannot be tested in a different cell as it is not yet named. The formula returns the value in **B12**, **9.0%** The formula now returns the value in **B13**.
5. With **B6** still selected, activate the ribbon's ***Formula Tab.***	Name the **LAMBDA** function.
6. In the **Defined Names Group**, click **Define Name.**	Keep the **Scope** as "Workbook" to ensure the function can be used anywhere in the workbook.
7. In the **Name** field, enter: **StateSalesTax.**	
8. In the **Refers to:** field, enter the formula: **=LAMBDA(ST,MT,OT,IF(ST=”MI”, MT,OT))**	
9. Press **[OK].**	The function is named and ready for use.
10. Select cell **H2.**	



Instructions:	**Results/ Comments:**
11. Enter the following formula: **=StateSalesTax(B6,B12,B13)** Then press **[Enter].**	Call the function by name. In the *AutoComplete* suggestion, use the arrow keys and **[Tab]** to select the custom function. The tooltip also shows the expected ***parameters*** while typing.
12. Change the value in cell **B6** to **MI.**	
13. Navigate to the ***Loan2*** worksheet.	
14. Repeat steps **11** and **12** on the ***Loan2*** worksheet.	The custom function can be reused on multiple worksheets and passed different ***parameters*** to perform necessary ***calculations***.
15. Navigate to the ***Loan3*** worksheet.	
16. Repeat steps **11** and **12** on the ***Loan3*** worksheet.	
17. Save and close the file.	**[Ctrl S]** and **[Ctrl W]**

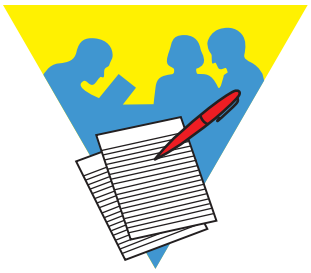


Lesson 2: Complex Summing Functions

Lesson Overview

The following concepts are covered in this chapter:

- Filtering Data
- Logical-Mathematical Functions,
- Array Functions
- SUMPRODUCT Function
- RANK Functions



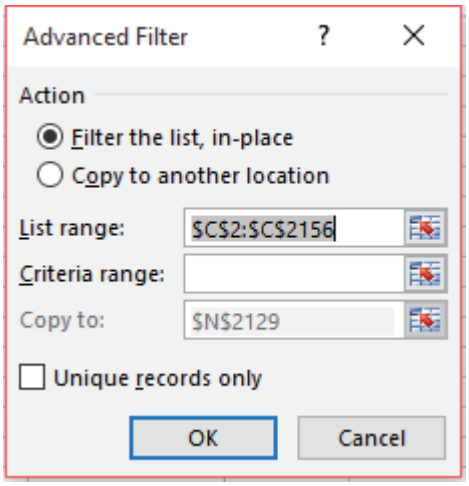
Lesson Notes

Filtering Data

Unique Values

Typically, data received for processing is raw, whether in an *Excel*, *.csv*, or other file type. As this is almost always the case, reserve a step to extract only needed data before any additional processing. The *Advanced Filter* can easily filter out unnecessary records.

Find the *Advanced Filter* on the ***Data*** tab in the **Sorting & Filter** group. The **[Advanced]** button opens the *Advanced Filter* dialog box, which displays the following filter options.



If filtering in place, the filtered range replaces the original range. However, if copying a filtered list, a copy location must be specified. Under the **Action** heading, selecting the option to *Copy to another location* makes the **Copy To** field available to type.

The *Advanced Filter* acts on the cell range specified in the **List range** field. As the name implies, the **Criteria range** field also references a cell range. The filtering criteria must be in a table format, with the top row matching headers in the **List range**. By default, *Advanced Filter* uses the *AND* operator to filter all criteria on the same row in the **Criteria range**. To use the *OR* operator, the values to filter by must be on different rows.

Check the *Unique records only* checkbox to create a list of unique values. The filter will work even if no additional criteria are specified. Unlike the **[Remove Duplicates]** button on the ***Data*** tab, the *Advanced Filter* does not delete the duplicate records; it filters them out of the visible data set.

Filtering Data

UNIQUE Function

The **UNIQUE** function returns a list of unique values from a cell range. Although the **UNIQUE** function call is made in a single cell, *Excel* automatically spills the function results into as many cells as needed. If multiple workbooks are open, **UNIQUE** can work across workbooks. However, if one of the workbooks necessary for the **UNIQUE** function is not open, it returns **#REF!**

Note

The **UNIQUE** function is available in *Excel 2021* and later on all platforms.

The **UNIQUE** function takes the following arguments:

- ***array***: [*required*] the range to search for unique values
- ***by_col***: [*optional*] an option to specify a comparison of rows or columns
  - *True*: compare columns
  - *False*: [*default*] compare rows
- ***exactly_once***: [*optional*] an option to specify desired uniqueness
  - *True*: return single instances only
  - *False*: [*default*] return all unique instances

The syntax of the **UNIQUE** function is as follows:

=UNIQUE(*array*,[*by_col*],[*exactly_once*])

Action 1 – Naming Cell Ranges

Instructions:	Results/ Comments:
1. Open **SumIf.xlsx**	**Objective**: Name ranges for easier referencing and include the column header in the range.
2. Save the file as **My SumIf.xlsx**	
3. Select any cell within the data set and press **[Ctrl + A]** to select all connected data.	The entire table is selected.
4. On the ***Formulas*** tab, select **[Create from Selection]** in the **Defined Names** group.	The *Create Names from Selection* dialog box opens.
5. Uncheck the *Left column* option and click **[OK].**	When clicking **[OK]**, only the *Top row* option should be selected. The top row contains headings that can be used as names, while the left column does not.
6. Select the [**Name Manager]** button in the **Defined Names** group on the ***Formulas*** tab.	
7. Select *Country.*	The values in the named range begin with the second row of data, and the column header is not included by default.
8. In the **Refers to:** field, change the reference from **$B$2** to **$B$1.** Then click the green checkmark to apply the edit.	The new range includes the column header.
9. Repeat steps **7** and **8** for the remaining names.	
10. Review the changes to ensure all references have been changed. Then click **[Close].**	All names include column headers.
11. Save the file.	**[Ctrl + S]**

Action 2 – Advanced Filter

Instructions:	Results/ Comments:
1. Select any cell within the data set.	**Objective**: Display each unique salesperson and customer in the data set.
2. On the ***Data*** tab, click the **[Advanced]** button in the **Sort & Filter** group.	The *Advanced Filte*r dialog box opens.
3. Select ***Copy to another location.***	This filter should not replace the existing data but should add another column. The **Copy to:** field is now available for typing.
4. In the **List range:** field, enter **Salesperson**	This is the range of cells being filtered. Using the name for the range **Salesperson** created in ***Action 1*** is more straightforward than entering the range address of **$C$2:$C$2156**. If the range address is entered for a named range, *Excel* automatically changes it to the range's name.
5. In the **Copy to:** field, enter **N1.**	
6. Select the ***Unique Records Only*** option.	Only one instance of each salesperson needs to be returned.
7. Click **[OK].**	A list of all salespeople is displayed in cells **N2**:**N11**.
8. Select cell **N15** and enter the following formula: **=UNIQUE(Customer)** Then press **[Enter].**	
9. Use *AutoFit* to resize column **N.**	Double-click between columns **N** and **O.**
10. Save the file.	**[Ctrl + S]**

Logical-Mathematical Functions, continued

Note

Remember to name ranges for easier referencing.

Note

Cell references can also be used as criteria, similar to the *Advanced Filter*.

SUMIF Function

The **SUMIF** function can save valuable time when summing data based on specific criteria. For example, **SUMIF** can quickly sum values in a range where only those above 100 need to be totaled.

The **SUMIF** function takes the following arguments:

- ***range***: [*required*] the range to search
- ***criteria***: [*required*] any expression that evaluates to *True* or *False*, in which *True* is added to the sum
  - Text criteria require quotation marks.
  - Text criteria can be a maximum of 255 characters.
  - Wildcard characters (? * ~) may also be used.
- ***sum_range***: [*optional*] the range to sum, if different from the range to search

The syntax of the **SUMIF** function is as follows:

=SUMIF(*range*,*criteria*,[*sum_range*])

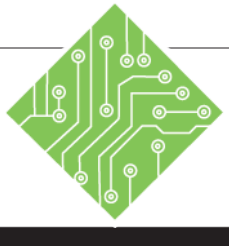
For example, a cell range detailing a salesperson's sales contains the following columns: *Date*, *Name*, and *Amount*. Use the *Name Manager* to give the ranges the same names as their column headers so that all values in the *Date* column are named "Date," all values in the *Name* column are named "Name," and all values in the *Amount* column are named "Amount." Then, the SUMIF function can find the sales total for a specific salesperson as follows.

=SUMIF(*Name*,"*Smith*",*Amount*)

This syntax shows **SUMIF** searching the *Name* column for the exact name "Smith" (modify with wildcards for a partial search) and returning the total of all values in the *Amount* column corresponding to that name. It returns the sum of all sales belonging to Smith.

Action 3 – Using the SUMIF Function

Instructions:	Results/ Comments:
1. **My Sumif.xlsx** should still be open. If not, open it again.	**Objective**: Display the sales total and the number of units sold for each salesperson and for each customer.
2. Select cell **O1.**	
3. Enter **Sales Total** Then press **[Enter].**	**Sales Total** is the header.
4. In cell **O2**, enter the following formula: **=SUMIF(Salesperson, $N2, Sale)** Then press **[Ctrl + Enter].**	**SUMIF** looks in the **Salesperson** range for each instance of the name in cell **N2**. For each instance found, it adds the value in the **Sale** range to its running total. Using **[Ctrl + Enter]** applies the formula without moving to the next cell.
5. Apply the *Accounting* format.	Right-click cell **O2** and click the dollar sign in the mini toolbar.
6. Use *AutoFill* to fill cells **O3:O10.**	
7. Select cell **P1.**	
8. Enter **Units Sold** Then press **[Enter].**	**Units Sold** is the header.
9. Select cell **O2** and use *AutoFill* to fill the formula into cell **P2.**	The **SUMIF** function along with its arguments is copied into cell **P2**. Using an absolute column and relative row address, it retains the correct column reference.
10. Double-click into cell **P2** to edit the formula. Change **Sale** to **Qty.** Then press **[Ctrl + Enter].**	The new argument to the **SUMIF** function totals the quantity of units sold by each salesperson.
11. Apply the *Number* format.	Use the dropdown in the **Number** group on the ***Home*** tab to select *Number*.



Instructions:	Results/ Comments:
12. Use *AutoFill* to complete the totals for the remaining salespeople.	
13. Select cell **O15.**	Repeat this process for the **Customer** list.
14. Enter **Sales Total** Then press **[Enter].**	**SUMIF** looks in the **Customer** range for each instance of the name in cell **N16**. Use **[F4]** to set the column as absolute. For each instance found, it adds the value in the **Sale** range to its running total.
15. In cell **O16,** enter the following formula: **=SUMIF(Customer, $N16, Sale)** Then press **[Ctrl + Enter].**	
16. Apply the *Accounting* format.	
17. Use *AutoFill* to fill cells **O17:O104.**	
18. Save the file.	**[Ctrl + S]** **Challenge**: Create another column that calculates how many units each customer has purchased.

Logical-Mathematical Functions, continued

Note

Wildcard characters may be used in the ***criteria***:
* = Any number of any characters
? = Any single character

Add a tilde (~) before either character to find a *literal* question mark or asterisk.

AVERAGEIF Function

The **AVERAGEIF** function takes similar arguments to the **SUMIF** function but returns an average instead of a sum based on the ***criteria***. This means that the values in the ***range*** must be numeric. Any values like *True*, *False*, and empty cells are ignored.

The syntax of the **AVERAGEIF** function is as follows:

=AVERAGEIF(*range*,*criteria*,[*average_range*])

The **AVERAGEIF** function takes the following arguments:

- ***range***: [*required*] the range to search
- ***criteria***: [*required*] any expression that evaluates to *True* or *False*, in which *True* is calculated as part of the average
- ***average_range***: [*optional*] the range to sum, if different from the range to search

Some things to consider when using AVERAGEIF:

- Empty cells in the ***criteria*** are treated as 0 values.
- Empty cells in the ***average_range*** are ignored.
- Cells in the ***range*** containing *True* or *False* values are ignored, but if the entire range is blank or empty, **AVERAGEIF** returns a **#DIV0!** error.
- If the ***range*** contains no cells that meet the ***criteria***, **AVERAGEIF** returns a **#DIV0!** error.
- The ***range*** and ***average_range*** do not have to be the same size and shape. **AVERAGEIF** begins calculating the top left cell in the ***average_range*** and adapts to the size of the ***range***.

Action 4 – Using the AVERAGEIF Function

Instructions:	Results/ Comments:
1. **MySumif** file should still be open.	If not, reopen it.
2. Select cell **Q1** and type: **Average Sales** then, tap the **[Enter]** key.	The next header is added to the spreadsheet. Adjust the column width by placing the cursor between columns Q and R, in the column header area. When the horizontal double arrow cursor is displayed, Double-click the line that separates the columns.
3. Enter the following formula: **=AVERAGEIF(Salesperson,N2,Sale)** then, tap the **[Ctrl + Enter]** keys.	This formula will return the average sale for the salesperson on this row.
4. Apply the *Accounting* format.	Use the Mini toolbar.
5. Use AutoFill to fill in the remaining salespeople's values.	Now you know what each salesperson's average sale is.
6. Select cell **Q15** and type: **Average Purchase** then, tap the **[Enter]** key.	You are beginning to create another subset of calculated data here.
7. In cell **Q16,** enter the following formula: **=AVERAGEIF(Customer,N16,Sale)** then, tap the **[Ctrl + Enter]** keys.	This returns the average purchase of the customer listed in cell **N16**.
8. Apply the *Accounting* format.	Use the Mini Toolbar.
9. Use AutoFill to fill in the remaining Customers values.	Now you know the average purchase each customer makes.
10. Save the file.	**[Ctrl + S]**

Logical-Mathematical Functions, continued

COUNTIF Function

Another beneficial logical-mathematical formula is **COUNTIF**. The **COUNTIF** function can conditionally count some rows to find how many sales a salesperson has made or orders a single customer has placed. Using this function combines two steps into one: filtering and counting.

The **COUNTIF** function takes the following arguments:

- ***range***: [*required*] the range to search
- ***criteria***: [*required*] any expression that evaluates to *True* or *False,* in which *True* rows are counted and *False* rows are not

The syntax of the **COUNTIF** function is as follows:

=COUNTIF(*range*,*criteria*)

Action 5 – Using the COUNTIF Function

Instructions:	Results/ Comments:
1. **MySumif** file should still be open.	If not, reopen it.
2. Select cell **R1** and type: **Number of Sales** then, tap the **[Enter]** key.	The new header is placed.
3. In cell **R2,** enter the following formula: **=COUNTIF(Salesperson,N2)** then, tap the **[Ctrl + Enter]** keys.	This formula will count the number instances the name in cell **N2** occur within the Salesperson range.
4. Use AutoFill to complete the counts for the remaining salespeople.	The other salespeople's values are entered.
5. Select cell **R15** and type: **Number of Purchases** then, tap the **[Enter]** key.	The next header is placed.
6. In cell **R16,** enter the following formula: **=COUNTIF(Customer,N16)** then, tap the **[Ctrl + Enter]** keys.	Now you are counting the number of times the customer appears in the Customer range.
7. Use AutoFill to complete the counts for the remaining customers.	We now know how often the customer has made a purchase.
8. Adjust the width of column **R** to show all the contents.	The column is now fully displayed.
9. Save the file.	**[Ctrl + S]**

Logical-Mathematical Functions, continued

SUMIFS Function

The **SUMIFS** function extends the abilities of the **SUMIF** function. Where **SUMIF** can sum data within a range based on specific criteria, **SUMIFS** can search for multiple ***criteria*** in multiple ranges. In the example file, this may be used to find out whose sales met or exceeded target goals. Unlike the **SUMIF** function, the ***sum_range*** argument is required in **SUMIFS**.

The **SUMIFS** function takes the following arguments:

- ***sum_range***: [*required*] the range containing data to be summed
- ***criteria_range1***: [*required*] the range being searched for ***criteria1***
- ***criteria1***: [*required*] any expression that evaluates to *True* or *False*, in which *True* is added to the sum

Additional ***criteria_range*** and ***criteria*** pairs can be added to narrow the data even further, up to 127 total pairs.

The syntax of the **SUMIFS** function is as follows:

=SUMIFS(sum_range,criteria_range1,criteria1, [criteria_range2,criteria2],)

AVERAGEIFS Function

Similarly, the **AVERAGEIFS** function extends the abilities of the **AVERAGEIF** function.

The **AVERAGEIFS** function takes the following arguments:

- ***average_range***: [*required*] the range containing the data to be averaged
- ***criteria_range1***: [*required*] the range being searched for ***criteria1***
- ***criteria1***: [*required*] any expression that evaluates to *True* or *False*, in which *True* is calculated as part of the average

Additional ***criteria_range*** and ***criteria*** pairs can be added to narrow the data even further, up to 127 total pairs.

The syntax of the **AVERAGEIFS** function is as follows:

=AVERAGEIFS(*average_range*,*criteria_range1*,*criteria1*, [*criteria_range2*,*criteria2*],)

Logical-Mathematical Functions, continued

COUNTIFS Function

Use the **COUNTIFS** function to count the times a value appears using multiple ***criteria*** and ***criteria_ranges***. Utilizing additional ***criteria_range*** and ***criteria*** pairs allows **COUNTIFS** to calculate more precise counts.

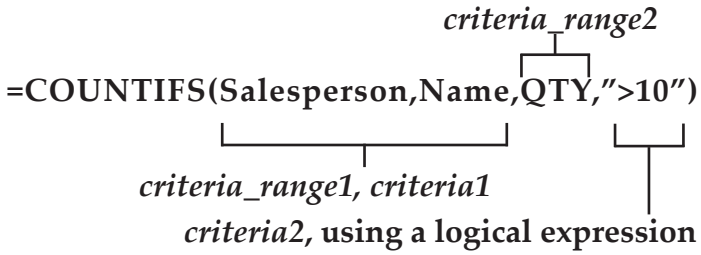
The **COUNTIFS** function takes the following arguments:

- ***criteria_range1***: [*required*] the range being searched for ***criteria1***
- ***criteria1***: [*required*] any expression that evaluates to *True* or *False*, in which *True* rows are counted and *False* rows are not

The syntax of the **COUNTIFS** function is as follows:

=COUNTIFS(*criteria_range1*,*criteria1*, [*criteria_range2*,*criteria2*],)

To find the number of sales made by a salesperson that met or exceeded a target goal, use **COUNTIFS**. Each ***criteria_range*** and ***criteria*** pair narrows down the values to be counted. In this example, the first criteri on finds the salesperson's name, and the second compares the sale to the target goal.



The ***criteria*** arguments can contain a number, a cell reference, or a mathematical or logical expression. Logical expressions such as the comparison above and any other text values must be enclosed in quotation marks.

Action 6 – Using SUMIFS with COUNTIFS

Instructions:	Results/ Comments:
1. **MySumif** file should still be open. If not, reopen it.	
2. Select cell **S1** and type: **Sales above 10** then, tap the **[Enter]** key.	The next header is in place.
3. In cell **S2**, enter the following formula: **=SUMIFS(Sales,Salesperson,N2,Qty, ">"&10)** then, tap the **[Ctrl + Enter]** keys.	This formula will add the values in the sales column only if both conditions are found to be true. The name in cell **N2** is found in the Salesperson range and the number for the record is higher than ten in the QTY range.
4. Use AutoFill to complete the calculations for the other salespeople.	
5. Select cell **T1** and type: **Sales over 10** then, tap the **[Enter]** key.	Now we will start to breakdown the data into subsets that will offer a more in depth understanding of the data.
6. In cell **T2,** enter the following formula: **=COUNTIFS(Salesperson,N2,Qty, ">"&10)** then, tap the **[Ctrl + Enter]** keys.	This will count the number of cells that have a value greater than 10 within the QTY range only if the name Salesperson range match the criteria. You now can see how many sales of more than ten units were made by each salesperson.
7. Use the AutoFill to complete the calculations for the remaining salespeople.	
8. Save the file.	**[Ctrl + S]**

Logical-Mathematical Functions, continued

MAXIFS Function

The **MAXIFS** function returns the highest value from a range based on defined criteria.

The **MAXIFS** function takes the following arguments:

- ***max_range***: [*required*] the range containing the maximum
- ***criteria_range1***: [*required*] the range being searched for ***criteria1***
- ***criteria1***: [*required*] any expression that evaluates to *True* or *False,* in which *True* is evaluated for the maximum value

The syntax of the **MAXIFS** function is as follows:

=MAXIFS(*max_range*,*criteria_range1*,*criteria1*,
[*criteria_range2*,*criteria2*],)

MINIFS Functions

The **MINIFS** function returns the lowest value from a range based on defined criteria.

The **MINIFS** function takes the following arguments:

- ***min_range***: [*required*] the range containing the minimum
- ***criteria_range1***: [*required*] the range being searched for ***criteria1***
- ***criteria1***: [*required*] any expression that evaluates to *True* or *False,* in which *True* is evaluated for the minimum value

The syntax of the **MINIFS** function is as follows:

=MINIFS(*min_range*,*criteria_range1*,*criteria1*,
[*criteria_range2*,*criteria2*],)

Action 7 – Using the MAXIFS and MINIFS Functions

Instructions:	Results/ Comments:
1. **MySumif** file should still be open. If not, reopen it.	
2. Select cell **T1** and enter: **Highest Sale.**	The new header is added.
3. Select cell **T2**.	
4. Enter the following formula; **=MAXIFS(Sale,Salesperson,N2)** then tap the **[Ctrl + Enter]** keys.	This formula will find the highest value in the Sales column of data based on the Salesperson.
5. Apply the *Accounting* format.	
6. Use the AutoFill to complete the calculations for the remaining salespeople.	Now you know the value of each salespersons best sale.
7. Select cell **S1** and enter: **Lowest Sale.**	The new header is added.
8. Select cell **S2**.	
9. Enter the following formula; **=MINIFS(Sale,Salesperson,N2)** then, tap the **[Ctrl + Enter]** keys.	This formula will find the lowest value in the Sales column of data based on the Salesperson.
10. Apply the *Accounting* format.	
11. Use the AutoFill to complete the calculations for the remaining salespeople.	Each salespersons' lowest sale value is shown.
12. Save file.	**[Ctrl + S**

Array Functions

Array Functions

Array functions are used to perform multiple calculations on multiple ranges of cells at once. They can be used either across rows or down columns. They are created by writing the formula as a regular formula but instead of using the **[Enter]** or **[Ctrl + Enter]** keys to apply the formula, it is necessary to use the **[Ctrl Shift Enter]** key combination to apply the formula. Array functions appear in the formula bar wrapped in braces like this:

{=PRODUCT(B4:B100,C4:C100,D4:D100)}

Array formulas can be used to return either a single or multiple results. When returning multiple results across an array of cells it is considered a spilled array.

A drawback to an array function is as soon as the formula is edited it ceases to be an array function unless the **[Ctrl Shift Enter]** keys are used to reapply the array. (A very common error to make.)

Dynamic and Spilled Array Functions

When a formula in a single cell is used to return a set of values across a range of cells, it uses a behavior called *spilling*. In *Excel 365,* array functions have been superseded by Dynamic Array functions that will automatically spill when the **[Enter]** key is used to apply the formula.

Formula results spill down into the cell below the cell where the formula is entered. The data array being referred to in the formula also defines the spill range size. When creating a single column dynamic array function separate from the array referred to in the formula the spill will be placed below the formula cell to the right of the formula cell. If creating a multi-column dynamic array formula the spill will move down and to the right.

Note

Only the first cell where the original formula was placed is editable.

Here is an example of a dynamic SORT formula,

$825.00	$4.80	=K2*L2	=SORT(M2:M2157,)
$26.00	$7.30	=K3*L3	
$400.00	$9.60	=K4*L4	
$91.20	$10.00	=K5*L5	

Dynamic Sort formula results Dynamic Sort formula

Note the results of the sort formula will not rearrange the data set but create a new column showing sorted data base on a column within the data set.

Array Functions, continued

FREQUENCY Function

Previously covered functions such as **COUNTIF** and **COUNTIFS** count how often a value appears within an array. The **FREQUENCY** function extends these capabilities to keep track of multiple counts simultaneously and return an array of counts.

Note

This function is now a dynamic array function available in *Excel 365*. Older versions of *Excel* use this function as an array function, requiring the use of the **[Ctrl Shift Enter]** key combination to apply the formula.

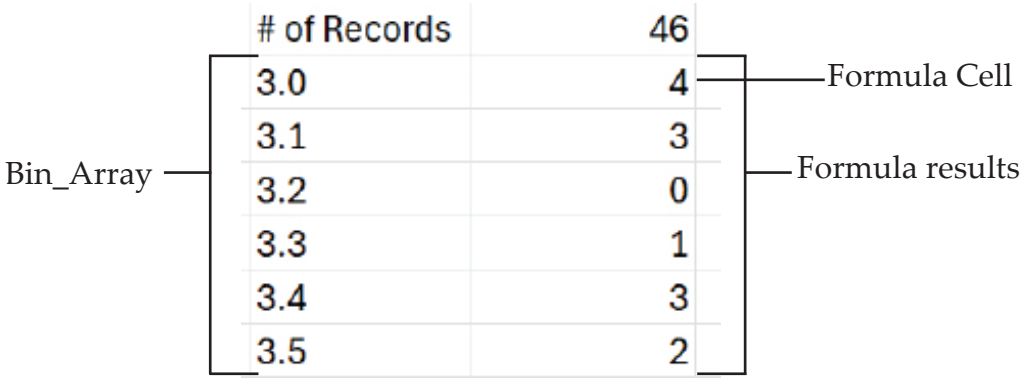
Although the **FREQUENCY** function can often replace multiple instances of COUNTIF and COUNTIFS to clean up syntax, it can only count numeric values. If a COUNTIF or COUNTIFS counts non-numeric values, FREQUENCY is not a suitable replacement for it.

The syntax of the **FREQUENCY** function is as follows:

=FREQUENCY(*data_array*,*bins_array*)

The **FREQUENCY** function takes the following arguments:

- ***data_array***: [*required*] the range being searched and counted
  - Cells containing text and empty cells are ignored.
  - An empty ***data_array*** returns an array of zeroes.
- ***bins_array***: [*required*] the values to be counted in the ***data_array***
  - An empty ***bins_array*** returns the number of elements in the ***data_array***.



Action 8 – Using the FREQUENCY Function

Instructions:	**Results/ Comments:**
1. The **MySumIfs** File should still be open.	If not re-open it.
2. Select the ***Order Turn-Around*** worksheet.	Click on the ***Order Turn-Around*** sheet tab in the lower left of the *Excel* window.
3. Select cell **M1**.	
4. Enter the following, **Turn Around Times/ In Days** then tap the **[Enter]** key.	This will be a header for the new data.
5. Auto-size the column width.	Set the cursor between the column M & N header, when the double-headed arrow appears double-click.
6. In cell **M2** enter the value; **1** then use the **[Ctrl + Enter]** key combination.	Using the **[Ctrl Enter]** key combination applies the edit and keeps the same cell actively selected.
7. Hold the **[Ctrl]** key and use AutoFill to extend the numbered list to 31.	Using the **[Ctrl]** key in conjunction with the AutoFill sets the AutoFill to ***Fill with a series*** instead of ***Copy Cells***.
8. Select cell **N2**.	
9. Enter the following formula, **=FREQUENCY(G2:G2087,M2:M33)** the tap the **[Enter]** key.	Once the formula is entered it dynamically spills into cells **N3:N33**, showing the number of times each value in cells **M2:M33** are found within cells **I2:I2087**. Cell **N33** shows the number over any other value was found within the Data_Array. In other words, you now know how many times it took X-number of days to fulfill orders.
10. Select cell M33 and enter, **Over 31 days**.	
11. Save and close the File.	**[Ctrl + S]** and **[Ctrl + W]**

SUMPRODUCT Function

SUMPRODUCT Function

This function multiplies each line in defined ranges and sums those results in one step. By default this formula multiplies, then adds but it can also perform addition, subtraction, and division calculations as needed.

Imagine an order information spreadsheet containing columns for quantity and unit price without a total column. It would be easy enough to create a total column where each order's unit price and quantity columns are multiplied then summed to give a grand total of all sales revenue. Using the **SUMPRODUCT** function allows you to find the grand total without having to create a total column.

The syntax of these formulas is;

=SUMPRODUCT(*Array1,[Array2],*)

- ***Array1***- defines the first range of cells to be calculated.
  - Within each array it is possible to add arithmetic operators to add, subtract, multiply, and divide ranges.
- ***Array2*** - is an optional argument in the formula. There can be as many as 255 optional arguments into a single formula.
  - All ranges within and across arguments must be the same size.
  - Do not use full column references as arrays.

As an example the following two formulas will yield the same result:

=SUMPRODUCT(B2:B6,C2:C6)

=(B2*C2)+(B3*C3)+(B4*C4)+(B5*C5)+(B6*C6)

SUMPRODUCT Function, continued

Referencing Table Ranges

If the data ranges are held within an *Excel* table the addressing would use the structured references found in tables.

=SUMPRODUCT(*Table1[ColumnB],Table1[ColumnC]*)

As column headers are clicked in the table during formula entry, you may see the reference as

Table1[[#Headers],[Column_Header]]

Edit the input to read as :

Table1[Column_Header]

Filtering Data Within A Range

To filter for a specific data point within a range, the array argument is treated as a nested argument, meaning that the range and criteria must be wrapped inside brackets. The criteria can be a cell reference or a value. Remember if you are searching for text it must be held in full quotation marks.

(*StartingCell:EndingCell=ReferenceCell*)

OR

(*StartingCell:EndingCell=ReferenceValue*)

OR

(*StartingCell:EndingCell="ReferenceText"*)

Combining Ranges in Arrays

When you need to filter a data set by more than a single value you will use the * or + symbols.

The asterisk can be treated as an "And". A filtered range being used to determine which values are to be calculated in a second range will be written as

((*StartingCell:EndingCell)*(StartingCell:EndingCell*))

In the example above, all arguments are treated as a single argument which searches for values within the first range, running the calculation from the data within the second range.

SUMPRODUCT Function, continued

A plus sign is treated as an "Or". In instances where there is more than one possible parameter to be considered by the formula.

((*Range=Reference)+(Range=Reference*))

This would be like searching for apples or oranges within a list of fruit as opposed to looking for apples and oranges.

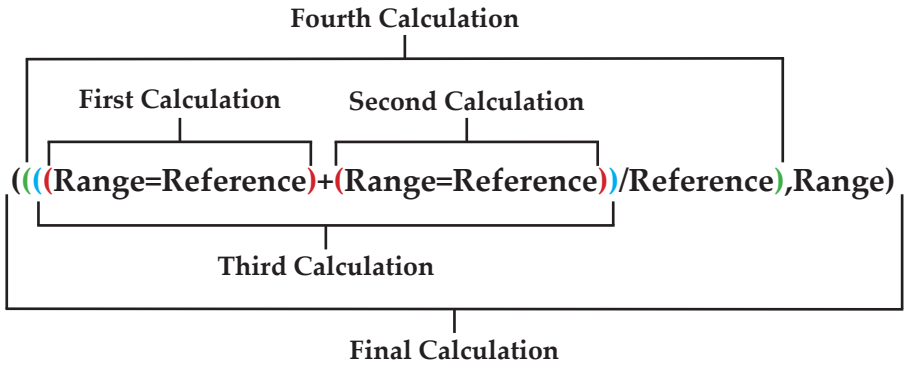
Summing Filtered Data

Using the plus and asterisk to create these type of arguments can be expanded by adding more nested arguments separated with asterisks.

((*Range=Reference)*(Range=Reference)*Range*)

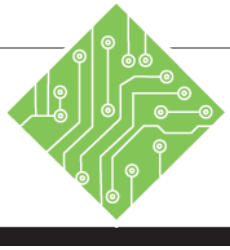
Adding Mathematic Operators

When other calculations need to be added to the formula, replace the commas with the required mathematic operator. The formula performs each nested calculation before the final sum function.



Action 9 – Using the SUMPRODUCT Function

Instructions:	Results/ Comments:
1. Open the **Sumproduct** file.	
2. Save the file as **MySumproduct**.	
3. Select cell **K1**.	This cell will show the total of all sales.
4. Enter this formula **=SUMPRODUCT(G2:G15,H2:H15)** then tap the **[Ctrl + Enter]** keys.	The values in both ranges are multiplied and added returning a value of 75537. The cell also remains actively selected. If you would like to check the validity of the formula results, use column M to multiply columns G and H then run a sum function on the results.
5. On the ***Home Tab*** click the **Number Type** drop-down in the **Number Group** and choose *Currency* from the list.	The value is now shown with a more clearly understood format.
6. Select cell **K2**.	This cell will show the grand total for the month of January.
7. Enter this formula **=SUMPRODUCT((F2:F15="Jan")* G2:G15*H2:H15)** then tap the **[Ctrl + Enter]** keys.	A value of 9465 is returned and the cell remains actively selected. The first argument of the formula searches the month column and selects only cells containing the value Jan, this is wrapped in Parenthesis as it is a statement defining the search. The asterisk represents AND statements which are used to continue defining what ranges are being used by the first argument. In this case all of these make up only one array used for the calculation, we are not looking in multiple arrays as if using commas.
8. Repeat Step 5 to apply the correct formatting.	
9. Save the file.	**[Ctrl + S]**



Instructions:	Results/ Comments:
1. The **Sumproduct** file should still be open.	If not, reopen the file.
2. Select cell **K8**.	This cell will show the number of orders in a month by country defined by cell **K6** and **K7**.
3. Enter this formula **=SUMPRODUCT((B2:B15=K6)* (F2:F15=K7))** then tap the **[Enter]** key.	A value of 2 is returned. The function argument blocks can be viewed as filtering which line items in the columns are included in the calculation, the asterisk acts as an AND. Once those two argument blocks are determined the formula returns a count of where both match.
4. Select cell **K9**.	This cell will show the total units sold by both country and month.
5. Enter this formula **=SUMPRODUCT((B2:B15=K6)* (F2:F15=K7)*H2:H15)** then tap the **[Enter]** key.	A value of 129 is returned. Is this example, the first two argument blocks act as they do the formula above. The inclusion of the asterisk and third range define the values being summed from the quantity column.
6. Select cell **K10**.	This cell will show the revenue from sales by both country and month.
7. Enter this formula **=SUMPRODUCT((B2:B15=K6)* (F2:F15=K7)*G2:G15*H2:H15)** then tap the **[Enter]** key.	A value of 43305 is returned. The first two argument blocks act as filters and the last two ranges define where matching filter values will be calculated.
8. Reselect cell **K10**.	
9. On the ***Home Tab*** click the **Number Type** drop-down in the **Number Group** and choose *Currency* from the list.	The correct number formatting is applied.
10. Save the file.	**[Ctrl + S]**

RANK Functions

Rank Function

Although the basic **RANK** function is still available, new variants, **RANK.EQ** and **RANK.AVG** improves accuracy. These functions return a rank value of a number within a list of numbers; consider it as telling the position of an item within a sorted list. When multiples are in the list, they share the same rank; in other words, all instances show the same rank. After all the shared instances are counted, the next rank value will be shown as its correct position.

For example, if three values shared the rank position of 3, the next rank would be 6.

Any subsequent matching rank values are left blank; the ranking list ignores duplicates and continues to rank the remaining list items.

Note

Future versions of *Excel* will stop recognizing the **RANK** function. As of now *Excel* still recognizes it to accommodate older files that relied on the **RANK** functions.

RANK.EQ

RANK.EQ bases the rank on the value compared to other values within a list.

Note

These functions will not recognize text as a valid value and return #VALUE errors.

The syntax of this formula is:

=RANK.EQ(*Number,Ref,Order*)

- ***Number:*** is a required argument; it is the number whose rank needs to return.
- ***Ref:*** is a required argument; it is the range of cells containing the list of values. Non-numeric values in Ref are ignored.
- ***Order:*** is an optional argument; 0 or 1 defines how to rank a number.
  - Use 0 or omit the argument to treat the list as sorted in descending order.
  - Use 1 (any numeric value above zero) to treat the list as sorted in ascending order.

RANK.AVG

RANK.AVG is the same as the **RANK.EQ**, except when it finds duplicates, **RANK.AVG** shows them as a number with a .5 decimal. This result shows when the number of duplicates is an even number; otherwise, it returns whole numbers in the same way the **RANK.EQ** does.

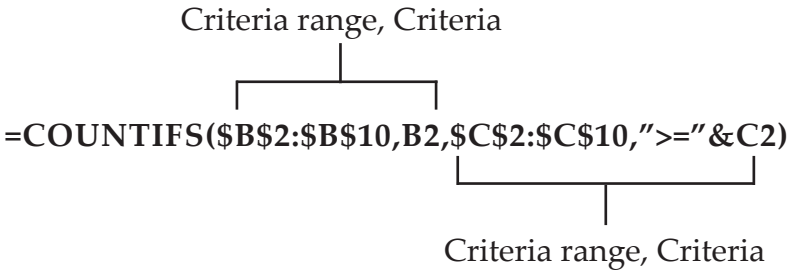
RANK Functions,
continued

Creating Grouped Rankings

Oddly, instead of using the **RANK** function, use the **COUNTIFS** function to create grouped rankings. In essence, this will generate a hierarchically organized ranking. After applying the formula, it may be necessary to run a custom sort to see the order more clearly.

Using the COUNTIFS for Ranking

The syntax of the **COUNTIFS** is the same as explained earlier, except the criteria will now be a logical test against the criteria range.



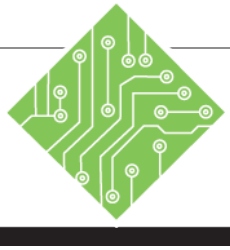
The first criteria range and criteria search a range for a specific value, effectively grouping the data in the first range. The second criteria range and criteria count variable values within the criteria ranges after the first arguments group the data.

Looking more closely at the second criteria: **">="&C2.**

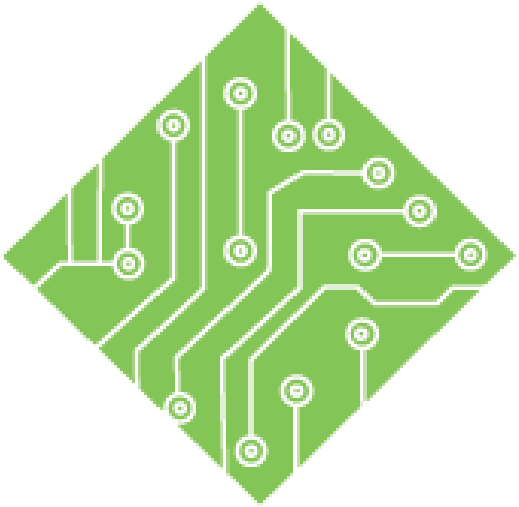
The first part of the criteria states to look for values greater than or equal to a value; the second part defines the value that the searched values must be greater than or equal to. The ranges should use absolute addresses, while the criteria use relative addresses, to check each line against the range as a whole.

Action 11 - Rank.EQ Functions

Instructions:	Results/ Comments:
1. The **MYSumProduct** file should still be open.	If not, re-open it.
2. Activate the ***Ranking*** worksheet.	
3. Select cell **J1** and enter the following **Rank by Units**.	You are adding a new header to the column.
4. Auto-adjust the column width.	Hover the cursor between columns **J** and **K** headers, when the horizontal double-headed arrow appears, double-click.
5. Select cell **J2** and enter the following formula, **=RANK.EQ(H2,$H$2:$H$15,0)** then use the **[Ctrl + Enter]** key combination.	The formula returns a value of 10 and the cell remains actively selected. Notice that there are two five reults but not a six.
6. Use AutoFill to complete the rest of the column.	
7. Re-select cell **J2** and enter the following formula, **=COUNTIFS($H$2:$H$15,">="&H2)** then use the **[Ctrl + Enter]** key combination.	Now the ranking goes from 1 to 14 correctly, there are no longer two 5 results.
8. Select cell **K1** and enter the following **Rank by Income**.	You are adding a new header to the column.
9. Auto-adjust the column width.	
10. Select cell **K2** and enter the following formula, **=RANK.EQ(I2,$I$2:$I$15,0)** then use the **[Ctrl + Enter]** key combination.	A value of 10 is returned.
11. Use AutoFill to complete the rest of the column.	You can now see the ranking based on the sales totals data.



Instructions:	Results/ Comments:
12. Re-select cell **K2** and enter the following formula, **=COUNTIFS($B$2:$B$15,B2 $I$2:$I$15,">"&I2)** then use the **[Ctrl + Enter]** key combination.	You now see how the totals are ranked and grouped by country. There are zeros shown as the highest rank. This can be somewhat confusing so it needs to be fixed.
13. Click into cell **K2** and set the cursor at the end of the existing formula, add the following to the end of the formula **+1** then use the **[Ctrl + Enter]** key combination.	The returned value changes by one . All the current results have one added and the ranking is much more clear.
14. Right-click cell **B2** and choose *A to Z* from the ***Sort*** options.	The data is grouped by countries.
15. Right-click any cell and choose *Custom Sort* from the ***Sort*** options.	The Sort window opens.
16. Click the **Add Level** button and set the fields-. **Then by**: to *Rank by Income* **Order**: to *Smallet to Largest* Then click the **[OK]** button.	The data is still primarily grouped by countries but now within each country group the data is sorted.
17. Save the file..	**[Ctrl + S]**

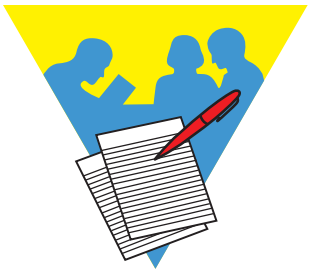


Lesson 3: Text Functions

Lesson Overview

The following concepts are covered in this chapter:

- Text Case Correction Functions
- Converting Numbers to Text
- Combining Cell Values
- Text Separation Functions
- Cleaning data



Lesson Notes

Text Case Correction Functions

While some of these functions are not new, they are important and commonly used in *Excel*. Textual data may often need to be formatted correctly, in all caps or lowercase. These formulas can make correcting these issues quick and easy.

Note

Text case is not part of text formatting in *Excel*.

PROPER Function

This function capitalizes the first letter of each text string (word) in a cell or range. As you will see, it can easily hold other nested functions, ensuring correct formatting is applied as values are returned.

The syntax of this formula is:

=PROPER(text)

- ***Text*** **-** is a required argument. This is the only argument this formula has, and it will generally be a cell reference containing incorrectly cased text.
  - It is possible to put a specific word in quotation marks to apply the proper case, though it is more common to use Find / Replace.

UPPER Function

Use this function when it is necessary to capitalize all the text in a given cell or range.

The syntax of this formula is:

=UPPER(text)

- ***Text*** - is a required argument. This is the only argument this formula has, and it will generally be a cell reference containing incorrectly cased text.

LOWER Function

Use this function when it is necessary to remove any capitalization from text in a given cell or range.

The syntax of this formula is:

=LOWER(text)

- ***Text*** - is a required argument. This is the only argument this formula has, and it will generally be a cell reference containing incorrectly cased text.

Action 1 - Correcting Text Case With Functions

Instructions:	Results/ Comments:
1. Open the **CleanUp** file.	The file can be found in the data files folder.
2. Save the file as **My_CleanUp**.	**[F12]**.
3. On the ***CaseChanges*** sheet examine the text casing for the data in column **A**.	The names in this column show a mixture or upper and lower cases.
4. Select cell **B2** and enter the following, **=PROPER(A2)** use the **[Ctrl + Enter]** keys.	The first full name is returned with the first letter of both names and initial capitalized. Since the text in cell **A2** was entered correctly there is no change to the returned text.
5. **AutoFill** the other rest of this column.	Double-click the **AutoFill** handle, the rest of column **B** now shows the full name list correctly cased.
6. Resize the width of column **B**.	Set the cursor between the column **B** & **C**, when the double-headed arrow appears, double-click to auto-adjust the column width.
7. Select cell C**2** and enter the following, **=UPPER(A2)** use the **[Ctrl + Enter]** keys.	The first name is now in all caps.
8. **AutoFill** the other rest of this column.	The rest of the list in column **C** is added.
9. Resize the width of column **C**.	
10. Select cell **D2** and enter the following, **=LOWER(A2)** use the **[Ctrl + Enter]** keys.	The name is returned in lower case.
11. **AutoFill** and resize the column **D**.	
12. Save the file.	**[Ctrl + S]**

Converting Numbers to Text

Note

If the numbers are referred to in other formulas, making a copy of those numbers may be beneficial to convert to text. This would not cause issues in those other formulas.

TEXT Function

Use this function to convert a number value into a text value. While this may cause issues with calculations using the converted numbers, it does allow the user to add formatting codes to cells, which makes the format-added content actual values.

For example, when postal codes start with a zero, *Excel* does not show those zeros. When that happens, a custom number format that shows the zero is necessary to apply; it does not add the zero as a value to the cell.

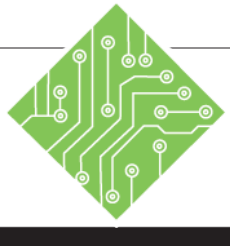
Using the Text Function converts the number to text and can add custom formatting, which adds additional formatting content. As a result of the formatting of numbers as text, numbers and symbols can be combined as one text string.

The syntax of this formula is:

=TEXT(*Value, "Format_Text"*)

- ***Value:*** a required argument; the cell containing numeric data to convert into text.
- ***Format_Text:*** a required argument; add the desired custom number formatting codes, wrapped in quotation marks, to apply to the converted text.

Action 2 - Converting Numbers to Text



Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the ***Names*** sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Enter today's date in cell **A1**.	Use the **[Ctrl + ;]** key command to enter the date.
3. Use **AutoFill** to extend the date range down to **A10**.	There should now the a range of dates going 10 days into the future.
4. Select cell **B1** and enter the following, **=TEXT(A1,"mm/dd/yyyy")** then tap the **[Enter]** key.	The formula returns a text string which looks like a date although, it is no longer a date formated number. Reselect cell **A1** to see the **Number Formatting** field in the **Number Group** on the ***Home Tab*** show it as a ***Date***.
5. Use **AutoFill** to extend the date range down to **B10**.	Double-click the **AutoFill** handle to extend the formula down to **B10**.
6. Select cell **C1** and enter the following, **=TEXT(A1,"mm/yy")** then tap the **[Enter]** key.	The formula returns a text string showing the month and year.
7. Use **AutoFill** to extend the date range down to **C10**.	Double-click the **AutoFill** handle to extend the formula down to **C10**.
8. Select cell **D1** and enter the following, **04567** then tap the **[Enter]** key.	The cell shows **4567**, *Excel* does not show leading zeros.
9. Use **AutoFill** to extend the data range down to **D10**.	Double-click the **AutoFill** handle to extend the formula down to **D10.**
10. Select cell **E1** and enter the following, **=TEXT(D1,"00000")** then tap the **[Enter]** key.	The formula returns a text string with a leading zero.
11. Use **AutoFill** to extend the data range down to **E10**.	Double-click the **AutoFill** handle to extend the formula down to **E10**.
12. Save the file.	**[Ctrl + S]**

Combining Cell Values

Data received can be broken down into the smallest usable parts, but that may not be what is required in the current file. In these situations, combining the data into a single cell is necessary. Excel offers several methods to assist in this type of undertaking, from a simple addition formula to the new **CONCAT** function or **TEXTJOIN** function.

Simple Combining Formula

This very basic formula is used to combine content in several cells into a single cell. When contents are in cells, use the cell addresses. The ampersand is used to join the cell references and/or text.

=Text1&Text2&...

- You can add strings of your own by wrapping the string in quotation marks.

=Text1&"Text2"&Text3...

Note

This function will work in *Excel 365* for both *MAC* and *Windows* and *Excel* for the web.

CONCAT Function

The new **CONCAT** function replaces the **CONCATENATE** function. Files using the older **CONCATENATE** function will still work as expected; *Excel* will recognize the function.

Note

CONCATENATE will still work, ensuring older files using that function continue to work as expected.

CONCAT will be used now when combining text strings from multiple cells. Text strings can be held in cells or added from within the formula itself. If a delimiter, such as a blank space, is required, it must be added within the formula as an argument.

The syntax of this formula is:

=CONCAT(*Text1*,*Text2*,...)

- ***Text1:*** is a required argument, the first item to appear in the combined string.
- ***Text2:*** The next element to add to the combined string.
  - If text, punctuation, or spaces are added to the final returned value, they must be held within quotation marks.

=CONCAT(*Text1*,"*Text2*",*Text3*...)

Combining Cell Values,
continued

TEXTJOIN Function

Similar to the **CONCAT** function, the **TEXTJOIN** function can be used to add a delimiter directly into the returned value. A delimiter is a character that separates each component part of the data. For example, a forward slash separates the day,month, and year in a date. Instead of having to add each delimiter as a quoted argument in the **CONCAT** or a simple combining formula, the **TEXTJOIN** function allows you to define a delimiter once.

The syntax of this formula is:

=TEXTJOIN(*Delimiter,Ignore_Empty,Text1,Text2,...*)

- ***Delimiter:*** This is a required argument; as a text entry, it should be inside quotation marks. For example: For a space, enter " "
  For a comma with a space, you enter ","
- ***Ignore_Empty:*** This will be either TRUE or FALSE.
  - TRUE will ignore empty cells in the returned value.
  - FALSE would add empty cells as blank spaces in the formula result.
- ***Text1,Text2,...:*** These are the cell addresses to be joined by the formula.

Overwriting Original Data

It may be necessary to remove the original data once it has been combined. This can be done by copying the formula results and pasting them into a combined column. When pasting, use the *Paste as Values* option in the ***Paste*** options. The formula results are pasted as static values, no longer requiring the contents defined in the formula, allowing to clean up the data without having duplicate values in the spreadsheet.

- Highlight the range containing the formula results and copy the cells.
- Highlight the original range of cells.
- Right-click the range and select *Paste as Values* option from the ***Paste*** commands.
- If using the shortcut to paste **[Ctrl + V]**, click the smart tag that appears after pasting and choose *Paste as Values.*

Action 3 - Combining Cells

Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the *Names* sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Select cell **D3**.	You will combine the first name of the list in this cell.
3. Enter the following, **=A3&", "&B3&" "&C3** **[Ctrl + Enter]** to apply the formula.	This is a simple combination formula. Using the **[Ctrl+Enter]** keys applies the formula and keeps cell **D3** selected.
4. **AutoFill** the other rest of this column.	Double-clicking the **AutoFill** handle runs the formula down.
5. Select cell **I3**.	This cell will use a **CONCAT** function to combine the names.
6. Enter the following, **=CONCAT(G3," ",H3,", ",F3)** **[Ctrl + Enter]** to apply the formula.	The " " are used to add the blank space delimiters between the cell values. When adding these to the formula, remember to include the space between the quotes.
7. **AutoFill** the other rest of this column.	Double-click the **AutoFill** handle.
8. Select cell **N3**.	This cell will use a **TEXTJOIN** function to combine the names.
9. Enter the following, **=TEXTJOIN(" ",True,L3,M3,K3)** **[Ctrl + Enter]** to apply the formula.	The first argument of this formula defines what the delimiter will be, TRUE will ignore any blank cells in the returned value, then the list of cell addresses are what will be joined.
10. **AutoFill** the rest of this column.	Double-click the **AutoFill** handle.
11. Try to make a **CONCAT** that lists the name as last name, first name and Initial.	**=CONCAT(F3,", ",G3, " ",H3)**
12. Save the file.	**[Ctrl + S]**

Instructions:	Results/ Comments:
13. Select cell **S3** and combine the name.	Use either the simple, **CONCAT**, or **TEXTJOIN** method.
14. **AutoFill** the other rest of this column.	Double-click the **AutoFill** handle.
15. Select cells **S3:S17**.	
16. Copy the range.	**[Ctrl + C]** or right-click and choose *Copy* from the menu.
17. Right-click cell **P1** and choose the *Paste as Values* from the menu.	If you used the keyboard shortcut of **[Ctrl + V]**, click the **Paste Options Smart Tag** to access the *Paste As Values* button. If a window opens informing you that are about to replace the existing contents, click the **[OK]** button to continue.
18. Select columns **Q:S**.	These columns are no longer needed.
19. Right-click the selected columns and choose *Delete* from the menu.	The columns are removed from the worksheet and the pasted values remain.
20. Save the file.	**[Ctrl + S]**

Combining Cell Values,
continued

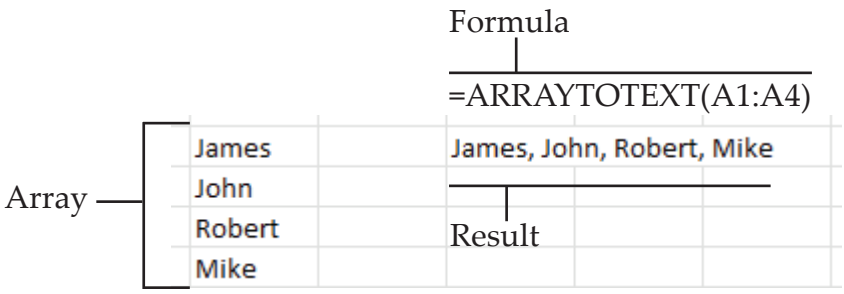
Note

This function will work in Excel 365 for both MAC and Windows and Excel for the web.

ARRAYTOTEXT

ARRAYTOTEXT is similar to the **CONCAT** and **TEXTJOIN** functions in that it combines cell values into a single string, except the combined string is in a single cell. By default, the function automatically adds a comma and space as a delimiter to separate each component of the array, but it can also show components separated by semi-colons.

As the name of the function suggests, cells within the range containing formatted numbers are converted to and treated as text in the formula result. Number formatting is also stripped away. Text is simply passed through the formula as text; it will be stripped of formatting while remaining case-sensitive. Since dates are formatted numbers, they are treated as regular numbers, and their formatting is stripped away.



The syntax of this formula is,

=ARRAYTOTEXT(*Array, [Format]*)

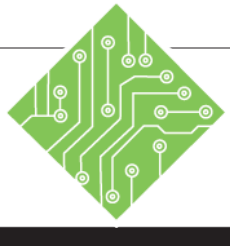
- ***Array:*** a required argument; the cell range to be combined and converted to text.
- ***Format:*** an optional argument; use one of two values to apply formatting to the returned text. This will either add a comma and space or semi-colons as delimiters.
  - 0 - This is the Concise format, which is the default option.
  - 1 - This is the Strict format, which treats the results of the formula as an array, wrapping the result within a pair of braces. Each text item is held in quotation marks. Except for numbers, errors, and booleans, those are not held within quotes.

Note

Booleans are TRUE/FALSE type values.

James, John, Robert, Mike	=ARRAYTOTEXT(Q9:Q12,0)	— Text Concise
{"James";"John";"Robert";"Mike"}	=ARRAYTOTEXT(O9:O12,1)	— Text Strict
12, 23, 45	=ARRAYTOTEXT(Q11:Q13)	— Number Concise
{12;23;45}	=ARRAYTOTEXT(Q12:Q13,1)	— Number Strict

Action 5 - Using ARRAYTOTEXT *Functions*



Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the *ArrayToText* sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Select cell **C2** and enter the following, **=ARRAYTOTEXT(A2:A4)** then tap the **[Enter]** key.	The formula combines the three cells as a text string in cell **C2**. Each string is separated with a comma space delimiter and any formatting has been removed.
3. Select cell **E2** and enter the following, **=ARRAYTOTEXT(A2:A4,1)** then tap the **[Enter]** key.	Adding the ***Format*** argument value of *1* applies *Strict* formatting to the returned string. Since cells **A2:A4** contain text strings, each string is wrapped in quotation marks; separated by semi-colons, and the entire result is wrapped in braces. The braces show the returned string as an array.
4. Select cell **C7** and enter the following, **=ARRAYTOTEXT(A7:A9)** then tap the **[Enter]** key.	The values in cell **A7:A9** have been converted to text and combined into a single cell. Each string is still separated with the comma space delimiter by default. Since cells **A7:A9** originally contained numeric data the strings are not wrapped in quotations.
5. Select cell **E7** and enter the following, **=ARRAYTOTEXT(A7:A9,1)** then tap the **[Enter]** key.	Adding the ***Format*** argument value of *1* applies *Strict* formatting to the returned string. With Strict formatting applied, the returned string is wrapped in braces.
6. Select cell **C11** and enter the following, **=ARRAYTOTEXT(A11:A13)** then tap the **[Enter]** key.	The formula understands the values in cells **A11:A13** as numeric data, stripping away the formatting, converting the values to text, and combining it into a single cell.
7. In cell **E15** enter the **ARRAYTOTEXT** formula that returns a string with *Strict* formatting.	**=ARRAYTOTEXT(A11:A11,1)**
8. Save the file.	**[Ctrl + S]**

Text Separation Functions

Another common necessity is breaking text strings into smaller, more manageable parts. Excel offers several methods to parse information held in a cell: text to Columns, Flashfill, and function formulas.

Note

This function will work in Excel 365 for both MAC and Windows and Excel for the web.

TEXTBEFORE Function

When there is a delimiter in the cell, it breaks a text string apart by the delimiter. This formula returns any text before the defined delimiter in a new location, leaving the original cells' information intact. The delimiter does not have to be in the exact location within each text string, giving this function great flexibility.

The syntax of this formula is:

=TEXTBEFORE(*Text,Delimiter,[Instance_Num],[Match_Mode],[Match_End],[If_Not_Found]*)

- ***Text:*** is a required argument; a character found within a text string in the Text cell.
  - The formula returns an empty cell if there is no text before the delimiter.
- ***Delimiter:*** is a required argument, A character found within a text string in the Text cell.
  - If searching for text delimiters, the formula requires the delimiter to be wrapped in quotation marks when searching for text.
- ***Instance_Num:*** an optional argument. If the cell has multiple instances of a delimiter, this is used to define which instance is used to determine what text is extracted.
  - Use a negative value to search from the end of the text string to the beginning.
- ***Match_Mode:*** an optional argument. Use it to determine whether the search will be case-sensitive or not.
  - 0 - makes the search case sensitive.
  - 1 - Used to make the search non-case sensitive.

Text Separation Functions, continued

Note

This function will work in *Excel 365* for both *MAC* and *Windows* and *Excel* for the web.

TEXTBEFORE Function, continued

- ***Match_End:*** an optional argument. Use it to treat the last character in the text as a delimiter. By default, the text is an exact match.
  - 0 - Do not match the delimiter to the end of the text string.
  - 1 - Match the delimiter to the end of the text string.
- ***If_Not_Found:*** an optional argument. When the formula does not find the delimiter in the text string, it allows the addition of a custom message.
  - If not used, the formula will return a #N/A message.

TEXTAFTER Function

This function is the opposite of the **TEXTBEFORE** function in that it returns text after a delimiter. Using both functions, you can split names and URLs in email addresses while maintaining the complete address in its original cells.

The syntax of this formula is:

=TEXTAFTER(*text,delimiter,[Instance_Num], [Match_Mode], [Match_End], [If_Not_Found]*)

These arguments are the same as outlined previously in the **TEXTBEFORE.**

RIGHT Functions

This function returns the right-most character or characters from a string. The number of characters specified will be the number returned, making this less flexible than the **TEXTAFTER**.

The syntax of this formula is:

=RIGHT(*Text,Num_Chars)*

- ***Text:*** is a required argument. The cell address contains the text string from which the test will be extracted.
- ***Num_Chars:*** is a required argument. If this is not included in the formula, only the last character is extracted from the string. Entering a value will return the number of characters from the string; blanks are considered characters.

Text Separation Functions, continued

LEFT Functions

This function returns the leftmost character or characters from a text string. The number of characters specified will be what is returned, making this function less flexible than the **TEXTBEFORE**.

The syntax of this formula is:

=LEFT(*Text,Num_Chars*)

- ***Text:*** is a required argument. The cell address contains the text string from which the test will be extracted.
- ***Num_Chars:*** is a required argument. If not included in the formula, only the last character is extracted from the string. Entering a value will return the number of characters from the string; blanks are considered characters.

MID Functions

This function returns a specific number of characters from the middle of a text string. It allows setting the starting position, the number of characters from the left, and the number of characters extracted.

The syntax of this formula is:

=MID(*Text,Start_Num,Num_Chars*)

- ***Text:*** is a required argument, the cell address which contains the text string extracted.
- ***Start_Num:*** is a required argument for the number of characters over from the left where the extraction is to begin. (Blank spaces are characters.)
- ***Num_Chars:*** is a required argument. It sets the number of characters to be extracted from the text string.

When dealing with text strings of variable lengths that contain fixed parts, the **LEN** functions can be helpful additions to a **LEFT** or **RIGHT** function.

Text Separation Functions, continued

LEN Functions

Use this to find out how many characters are in a cell. LEN Functions help spot random blank spaces in cells that should be a fixed length, such as zip codes, product numbers, and similar information.

The syntax of this formula is:

=LEN(*cell*)

- ***Cell:*** is a required argument. Use this to count characters in a string. Spaces are included in the results as they are hidden characters.

Nesting Len Inside Left or Right

The syntax of this formula is:

=LEFT(*Cell,LEN(Cell)-Value*)

- ***LEFT(cell):*** Location where the left characters will be extracted.
- ***LEN(cell):*** Counts the number of characters in the Left functions cell.
- ***Value:*** Subtracting this from the Len result sets the starting point of left character extraction from the cell.

Action 6 - Using TEXTBEFORE & TEXTAFTER Functions

Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the ***TextBefore&After*** sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Select any cell with a value.	You will be creating a table from the data set.
3. Click the **Format As Table** button in the **Styles Group** on the ***Home Tab*** and choose one of the table style options from the gallery.	The *Create Table* dialog opens.
4. Ensure the range in the **Where is the data for your table?** field shows **$A$1:$E$16** and click the **[OK]** button.	The data is placed into a table with the chosen formatting. If the range is not correct, click into the field and then highlight the data set.
5. Select cell **D2** and enter the following; **=TEXTBEFORE([@EMAIL],"@")** then tap the **[Enter]** key.	In this column, you will extract the name from the email address. As this is a table, when entering the formula, clicking cell **C2** enters a structured reference. As formulas are entered into table columns, *Excel* applies the same formula to all the cells in that column.
6. Select cell **E2** and enter the following, **=TEXTAFTER([@EMAIL],"@")** then tap the **[Enter]** key.	In this column, you will extract the URL from the email address.
7. Select cell **A17** and enter your own name and email as a new record.	As soon as the your email address is entered the formulas perfectly breaks it apart, no matter how long the name and URL are.
8. Save the file.	**[Ctrl + S]**

Text Separation Functions, continued

Note

This function will work in *Excel 365* for both *MAC* and *Windows* and *Excel* for the web

TEXTSPLIT Function

This can be used in place of the **Text-to-Columns** command to break a single row or column of data into smaller components. Think of it as the opposite to the **TEXTJOIN** function.

The syntax of this formula is:

=TEXTSPLIT(Text,Col_Delimiter,[Row_Delimiter],[Ignore_Empty],[Match_Mode],[Pad_With])

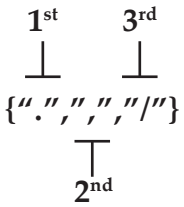
- ***Text:*** is a required argument; this is the location of the cell to split.
- ***Col_Delimiter:*** is a required argument, the repeated character that will divide the text into separate columns.
- ***Row_Delimiter:*** This is an optional argument. It is the repeated character that will be used to divide the text into separate rows.
- ***Ignore_Empty:*** is an optional argument; use TRUE to ignore consecutive delimiters. Use FALSE to break at each instance of the delimiter. The default is False.
- ***Match_Mode:*** is an optional argument; this will make the match case sensitive.
  - 1 for no case sensitivity.
  - 0 is for case sensitive, which is the default.
- ***Pad_With:*** is an optional argument, a value to display when no result is found. The default is #N/A.

Text Separation Functions, continued

Multiple Delimiters

When it is necessary to break a larger text block into smaller components and there are multiple delimiters, use the **TEXTSPLIT** function. In the ***Col-Delimiter*** or ***Row_Delimiter*** argument, use an array to combine several delimiters to determine what characters define the breakpoints.

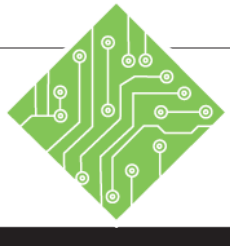
An array constant is written as:



Wrapping the ***Col_*** or ***Row_Delimiter*** argument in braces is how array constants are entered in formulas. In the example shown above, using a period comma and a forward slash as delimiters in the same formula is possible.

Array constants are used when more than a single reference is needed as an argument. They can contain cell references or values. Within the braces, each reference is separated by a comma.

The formula returns those spaces if blank spaces are not included in the delimiter definition. Hidden non-printing characters can cause many issues, so they will need to be removed. We will cover this next.



Instructions:	Results/ Comments:
1. The **My_CleanUp** file should still be open.	If the **My_CleanUp** file is not open, re-open it.
2. Activate the ***Splitting_Functions*** worksheet.	
3. Select cell **B2**.	This cell will be used to break the Course Name, City, Area, and date apart into cells **B2:E2**.
4. Enter the following formula, **=TEXTSPLIT(A2," - ")** then tap the **[Enter]** key.	The complex data in cell **A2** is broken down into manageable parts, spilling across to cell **D2**. Since there was only one delimiter defined in the formula, the data was not as broken down as needed.
5. Reselect cell **B2**.	You will continue to break apart the data in cell **A2**.
6. Edit the formula to include multiple delimiters using this, **=TEXTSPLIT(A2,{" - "," \ "})** then tap the **[Enter]** key.	By wrapping the ***Col_Delimiter*** argument in braces, you are making the single argument value into an array. Doing this allows you to search for multiple delimiters at one time. The data should now be shown as Course Name, City, Area, and Date in separate columns.
7. Use **AutoFill** to complete the rest of the column.	Double-click the **AutoFill** handle, if the double-click does not work you will have to drag **AutoFill** down to row 100.
8. Save the file.	**[Ctrl + S]**

Cleaning data

CLEAN Function

When text is exported from an external source and imported into Excel, it may include hidden characters. This is a common occurrence when copying and pasting data from Word into Excel. These hidden and non-printing characters may cause unexpected formula errors since Excel considers them text. Those hidden characters make apparently similar values behave as unique values.

Note

This function will work in *Excel 365* for both *MAC* and *Windows* and *Excel* for the web

The syntax of this formula is:

=CLEAN(*text*)

- ***Text:*** is a required argument, the cell reference where text may contain hidden or non-printable characters.

Note

Other additional non-printing character values are available in the appendix. These would not be found by the **CLEAN** function.

This function removes the first 32 ASCII non-printable characters, the most common ones. There are other characters outside the non-printable ones, including punctuation marks, letter casing, numbers, and accents, that this formula will not remove. In those instances, it will be necessary to use a modified version of the formula. Including a **SUBSTITUTE** function in the formula will allow the removal of characters that are not removed with the **CLEAN** function.

SUBSTITUTE Function

This function replaces text with other text. In this case, it replaces hidden non-printing characters that the **CLEAN** function misses with a character it will remove. It can also be used to change existing text with new text.

The syntax of this formula is:

=SUBSTITUTE(*Text,Old_Text,New_Text,[Instance_Num]*)

- ***Text:*** is a required argument, the cell reference where text may contain hidden or non-printable characters.
- ***Old_Text:*** is a required argument, and the text to be replaced can refer to hidden characters.
- ***New_Text:*** is a required argument, the new text which will replace the old.
- ***Instance_Num:*** optional argument defining which character's instance is replaced.

Note

When entering the ***CHAR*** argument be sure to capitalize the text as this is case sensitive.

Syntax when used as a nested function inside a **CLEAN** function:

=CLEAN(SUBSTITUTE(*A3,CHAR(160),CHAR(1)*))

Cleaning data, continued

The 32 ASCII symbols removed by the CLEAN function are:

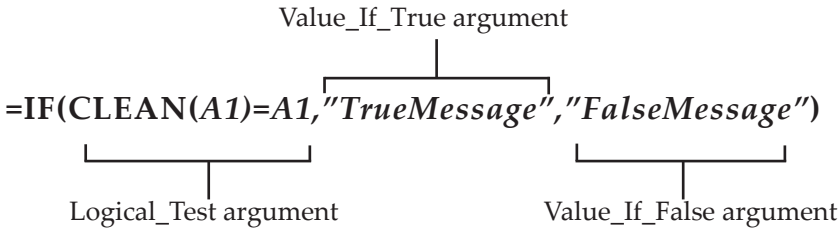
- Null Character (\0): ASCII code 0
- Start of Header (\x01): ASCII code 1
- Start of Text (\x02): ASCII code 2
- End of Text (\x03): ASCII code 3
- End of Transmission (\x04): ASCII code 4
- Enquiry (\x05): ASCII code 5
- Acknowledge (\x06): ASCII code 6
- Bell (\x07): ASCII code 7
- Backspace (\x08): ASCII code 8
- Horizontal Tab (\t): ASCII code 9
- Line Feed (\n): ASCII code 10
- Vertical Tab (\x0B): ASCII code 11
- Form Feed (\x0C): ASCII code 12
- Carriage Return (\r): ASCII code 13
- Shift Out (\x0E): ASCII code 14
- Shift In (\x0F): ASCII code 15
- Data Link Escape (\x10): ASCII code 16
- Device Control 1 (\x11): ASCII code 17
- Device Control 2 (\x12): ASCII code 18
- Device Control 3 (\x13): ASCII code 19
- Device Control 4 (\x14): ASCII code 20
- Negative Acknowledge (\x15): ASCII code 21
- Synchronous Idle (\x16): ASCII code 22
- End of Transmission Block (\x17): ASCII code 23
- Cancel (\x18): ASCII code 24
- End of Medium (\x19): ASCII code 25
- Substitute (\x1A): ASCII code 26
- Escape (\x1B): ASCII code 27
- File Separator (\x1C): ASCII code 28
- Group Separator (\x1D): ASCII code 29
- Record Separator (\x1E): ASCII code 30
- Unit Separator (\x1F): ASCII code 31

Cleaning data, continued

Finding Non-Printing Characters

To ensure that the **CLEAN** function has removed all the non-printing characters, consider using an IF formula to check by using the **CLEAN** function as the **IF's *Logical_Test***.

The syntax of a nested **CLEAN/IF** formula is:



- ***Logical_Test:*** Logical_Test: is a required argument of the IF function argument. In this case, the CLEAN function checks the results against the same cell's content.
- ***Value_If_True:*** This is an optional argument that returns a message you define to indicate if the CLEAN function has been successfully run.
- ***Value_If_False:*** an optional argument that returns a message to inform the user that the CLEAN function has not run successfully. Letting them know there are still non-printing characters in the cell.

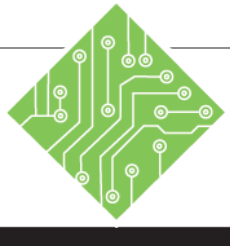
TRIM Functions

The **TRIM** Function is designed to remove blank spaces from text strings unless those spaces separate words within the string. **HTML** does not recognize consecutive spaces as such. When importing text from the web, it may include *** *** characters that **TRIM** or **CLEAN** will not remove. In those cases, use the **SUBSTITUTE** function outlined earlier.

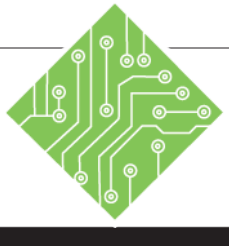
The syntax of this formula is:

=TRIM(*text*)

- ***Text:*** is a required argument; refer to the cell containing the text string with the extra spaces to be removed.



Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the ***BlankSpaces*** sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Examine the data structure in cells **A2,:A21**.	This data is has comma space combinations separating each component of the data.
3. Select cell **B2** and enter the following, **=TEXTSPLIT(A2,",")** then tap the **[Enter]** key.	The data is broken apart and spills in cells **C2:H2**. Notice that there are spaces as the first character in cell **C2:H2**. These blanks will need to be removed.
4. Reselect cell **B2** and edit the formula to, **=TRIM(TEXTSPLIT(A2," "))** then tap the **[Enter]** key.	The leading spaces in cells **C2:H2** have been removed. Another way to achieve the same result is to include the space with the comma in the ***Col_Delimiter*** argument of the **TEXTSPLIT** formula.
5. Use **AutoFill** to include the rest of the list.	Click and drag the **AutoFill** handle down to cell **B21**.
6. Examine the data structure in cells **J2:J21**.	This data was copied from a web-page and pasted into *Excel*. Use *NotePad* to examine the **names.html** file in the data folder.
7. Select cell **K2** and enter the following, **=CLEAN(J2,)** then tap the **[Enter]** key.	The formula doesn't change much since the *** *** entries are not part of the 32 ASCII codes removed by this formula.
8. Reselect cell **K2** and edit the formula to, **=CLEAN(SUBSTITUTE(J2,CHAR(160), CHAR(1)))** then tap the **[Enter]** key.	The extra spaces are removed this time. By using the **SUBSTITUTE** function to change the **CHAR**(***160***) into **CHAR**(***1***) the **CLEAN** formula was able to remove those extra spaces. Remember that the **CHAR** text is case sensitive.
9. Open the *Word* document entitled **NonPrintCharacters.docx**.	The file is in the data files folder.



Instructions:	Results/ Comments:
10. Click the **Show Hidden Characters** button in the **Paragraph Group** on the ***Home Tab***.	The hidden characters are displayed.
11. Notice the soft returns after each name in the list.	The non-printing hidden character looks like an arrow point down and left.
12. Select the entire list and copy it.	**[Ctrl + C]** or right-click and choose *Copy* from the menu.
13. Switch back to *Excel* and select cell **M2** on the ***BlankSpaces*** worksheet.	Use any method you prefer to switch between applications.
14. Paste the list.	**[Ctrl + V]** or right-click and choose *Paste* from the menu.
15. Select cell **N2** and enter the following **=CLEAN(M2)** and tap the **[Enter]** Key.	There is no discriminable difference between cells **M2** and **N2**.
16. Copy cell **N2**.	**[Ctrl + C]** or right-click and choose *Copy* from the menu.
17. Switch back over to *Word* and paste the name at the end of the list.	Notice that the soft return character has been removed.
18. Close *Word* without saving the file.	*Excel* should now be on screen.
19. Back in *Excel*, use the **AutoFill** handle to remove the non-printing characters from the remainder of the list.	Click and drag the **AutoFill** handle down to cell **M40.**
20. Rearrange the list entries so there are no empty cells separating the list entries.	Use your preferred method to accomplish this task**.**
21. Save the file.	**[Ctrl + S]**

Cleaning data,
continued

Note

This formula is case-sensitive but will ignore other formatting. Non-printing characters that are not in one of the comparison cells would also trigger false returns.

EXACT Function

Another common task in *Excel* is comparing columns or rows to see that they match. Use the **EXACT** function to run comparisons. As the comparisons are made, the formula returns either True or False values. While this function is primarily used for text comparisons, it can also be used to compare numeric and date values.

The syntax of this formula is:

=EXACT(*text1, text2*)

- ***Text1:*** is a required argument, the first cell to be compared.
- ***Text2:*** is a required argument, the second cell to be compared.

This formula can be used as a logical argument in numerous other logical formulas to determine true or false.

Myla	Myla	TRUE
Lincoln	Lincoln	FALSE
Yuvraj	Yuvraj	TRUE
Bryn	Brian	FALSE

Comparing Text

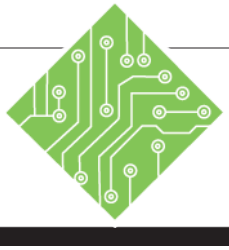
1	1	TRUE
2	5	FALSE
3	3	TRUE
4	4	TRUE

Comparing Numbers

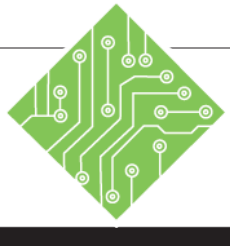
8/5/2024	8/5/2024	TRUE
8/6/2024	8/6/2024	TRUE
8/7/2024	8/5/2024	FALSE
8/8/2024	8/8/2024	TRUE

Comparing Dates

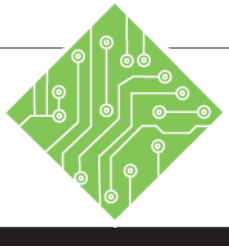
Action 9 - Comparing Lists With Functions



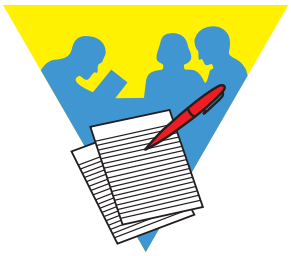
Instructions:	**Results/ Comments:**
1. In the **My_CleanUp** file, select the ***Comparisons*** sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Select cell **C2** and enter the following, **=EXACT(A2,B2)** then tap the **[Enter]** key.	Even though both cells are formatted to look the same, their true contents are now shown as not matching.
3. Use **AutoFill** to complete comparing the remainder of the lists.	Double-click the **AutoFill** handle.
4. Select cell **G2** and enter the following, **=EXACT(E2,F2)** then tap the **[Enter]** key.	Here again the text strings have been formatted to look alike but one column has some entries with non-printing characters.
5. Select cell **H2** and enter the following, **=CLEAN(E2)** and tap the **[Enter]** key.	Since we don't know which column includes the non-printing characters, we will clean both and re-compare. The first column has been cleaned.
6. Select cell **I2** and enter the following, **=CLEAN(F2)** and tap the **[Enter]** key.	The second column has now also been cleaned.
7. Select cell **J2** and enter the following, **=Exact(H2,I2)** then tap the **[Enter]** key.	The **EXACT** formula now confirms all the data in both columns match.
8. Save the file.	**[Ctrl + S]**



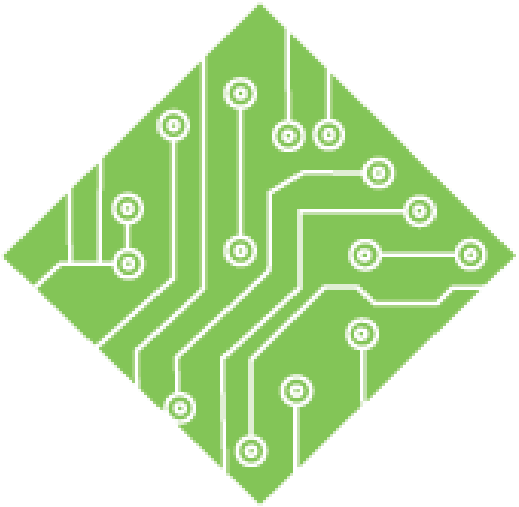
Instructions:	Results/ Comments:
1. In the **My_CleanUp** file, select the ***OverallCleanUp*** sheet.	If the **My_CleanUp** file is not open, re-open it.
2. Select cell **H1** and type in, **Full Name** then tap the **[Enter]** key.	You are adding a header for this column of data.
3. In cell **H2** enter the following, **=PROPER(TEXTJOIN(" ", TRUE,A2,B2,C2))** then tap the **[Enter]** key.	This nested formula will correct the text case as it combines the cells in the desired order.
4. Use **AutoFill** to combine and correct the rest of the list .	Double-click the **AutoFill** handle to run the formula down to cell **H35**.
5. Select cells **H1:H35** and copy the cells.	**[Ctrl + C]** or right-click the selected cells and choose *Copy* from the menu.
6. Right-click cell **A1** and choose *Paste As Values* from the menu.	**[Ctrl + V]** or right-click the selected cells and choose *Paste As Values* from the menu. If using the keyboard shortcut, click the *Paste Options Smart Tag* that appears after pasting and choose *Paste As Values*.
7. Resize column **A**.	
8. Select columns **B:C & H**.	Since these columns are no longer needed they can be removed.
9. Right-click the selected columns and choose *Delete* from the menu.	The columns are removed.
10. In cell **F2** enter the following, **=TEXT(E2,"00000")** then tap the **[Enter]** key.	This will convert the numeric data into text and add leading zeros to complete the zip codes when necessary.
11. Use **AutoFill** to combine and correct the rest of the list.	Double-click the **AutoFill** handle to run the formula down to cell **H35**.
12. Select and copy cells **F2:F35**.	**[Ctrl + C]** or right-click the selected cells and choose *Copy* from the menu.



Instructions:	Results/ Comments:
13. Right-click cell **E2** and choose *Paste as Values* from the menu. If a warning window opens asking if you want the replace the existing values, click the **[OK]** button.	**[Ctrl + V]** or right-click the selected cells and choose *Paste As Values* from the menu. If using the keyboard shortcut, click the *Paste Options Smart Tag* that appears after pasting and choose *Paste As Values* The values from the formulas now replace the data.
14. As this is now text you will see the *Formula Error Smart Tag* appear.	There are green triangles in the upper left corner of cells **E2:E35**. The smart tag appears as a yellow warning symbol.
15. Click the smart tag and choose *Ignore Error* from the menu.	The green triangle disappear.
16. Change the cell alignment from left to right.	Use the **Right Align** button in the **Paragraph Group** on the ***Home Tab.***
17. Select column **F** and delete it.	This column is also no longer needed.
18. Select cell **H1** and type in, **Name and Address** then tap the **[Enter]** key.	You are adding a header for this column of data.
19. In cell **H2** enter the following, **=ARRAYTOTEXT(A2:E2)** then the tap the **[Enter]** key.	The data in cells **A2:E2** are combined into a single text string.
20. Use **AutoFill** to combine and correct the rest of the list	
21. Resize column **H**.	
22. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**



Tips and Notes

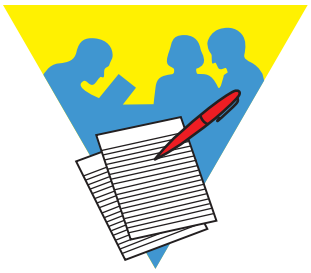


Lesson 4: Date & Time Functions

Lesson Overview

The following concepts are covered in this chapter:

- Date & Time Functions
- Date & Time Formatting
- Time
- Dates
- Calculating Working Days
- Finding Days of the Week
- OFFSET Functions
- Finding Week Number



Lesson Notes

Date & Time Functions

The first thing to understand about dates and times is how *Excel* understands and calculates them. While we see a date in a cell, we must understand this is done through formatting applied to the cell that contains the date. *Excel* is counting days as a running total beginning January 1, 1900. A date has a value based on each successive 24-hour period, with a whole number of one being added to the running count from January 1900. *Excel* understands the date of August 8, 2024, as a value of 45511, which is the number of days since January 1, 1900. Time is a fraction of one day written as a decimal. Each day has 23hrs 59mm and 59ss, and the next second tics by the fraction becomes a whole number, which is a day, and the day count goes up by 1.

Time is generally formatted as hours, minutes, and seconds, but *Excel* views it as a decimal value. 0.0 is equal to 12 a.m., and 0.99988426 is equal to 11:59:59 p.m. A time of 6 a.m. in decimal value is 0.25, while 6 p.m. is 0.75.

As you can now see, *Excel* treats all date and time entries as numeric data and can, therefore, perform date and time calculations using simple math and formulas.

Date Function

As we learned in the last lesson, date values can be converted into text and broken into more manageable parts. Sometimes, those values may need to be combined back together as a serial number. The **Date Function** allows us to achieve this.

Note

When using a **Date** or **Time Function**, the result must be formatted accordingly, as *Excel* calculates both as numbers.

The syntax of the **Date Function** is:

=Date(Year,Month,Day)

- ***Year:*** is a required argument; this can be 1 to 4 digits representing a year.
  - Using two digits is calculated as 1900+##, which could lead to confusion.
  - Using three digits is calculated as 1900+###.
  - Using four digits prevents any confusion.
- ***Month:*** This is a required argument; it can be any value between 1 and 12.
  - If a month value is higher than 12, *Excel* calculates it as years rather than months. A value of 16 months will add a year but only 4 months to the date result.

Date & Time,
contniued

- ***Day:*** is a required argument, including any value between 1 and 31.
  - Treat values exceeding 31 the same way as values over 12 months.

Time Function

As the **Date Function** combines values into a serial number, the **Time Function** combines hour, minute, and second values to create a decimal number. Format the new number using regular formatting features.

The syntax of the function is:

=TIME(Hour,Minute,Second)

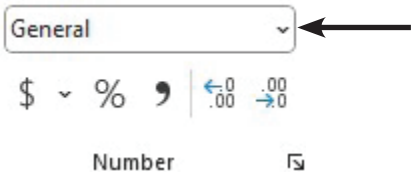
- ***Hour:*** is a required argument, any value between 1 to 23.
  - Any values above 23 are divided by 24, with whole numbers considered days.
  - The absolute highest value is 32767.
- ***Minute:*** is a required argument, any value between 1 to 59.
  - Any values above 59 are divided by 60, with whole numbers considered hours.
  - The absolute highest value is 32767.
- ***Second:*** is a required argument, any value between 1 and 59.
  - Any values above 59 are divided by 60, with whole numbers considered minutes.
  - The absolute highest value is 32767.

Date & Time Formatting

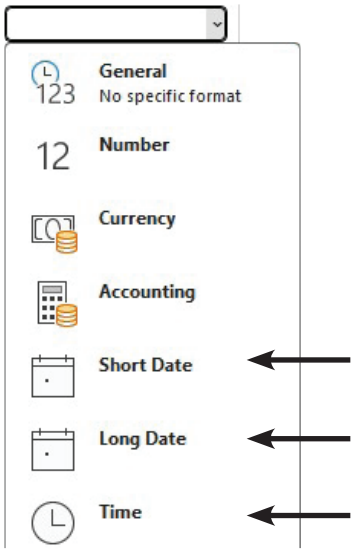
Now that we know that *Excel* views and treats dates and times as numeric data with formatting applied, let us look at the basics of date and time formatting.

Basic Date/Time Formatting

Looking at the **Number Group** on the ***Home Tab***, notice the formatting **Type** field drop-down.



Click the drop-down arrow to apply *Short Date*, *Long Date*, or *Time* formats.



Note

To create a number format that includes text typed in a cell, insert an @ in the text section of the number format code section at the point where you want the typed text to be displayed in the cell. If the @ character is not included in the text section of the number format, any text you type in the cell is not displayed; it defaults to display only numbers.

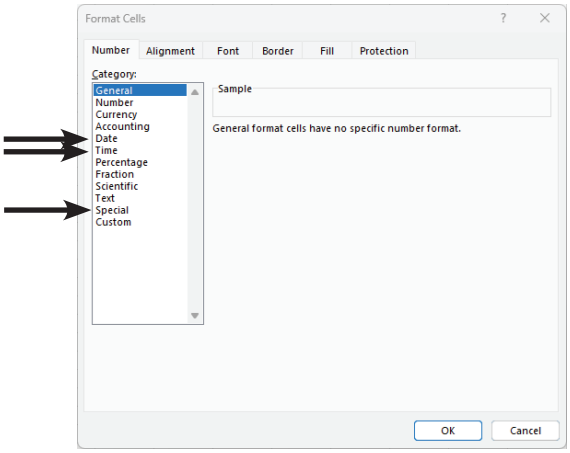
Advanced Date/Time Formatting

The *Format Cell* dialog offers a broader variety of built-in formats and allows the creation of custom formats. It can be accessed in several ways.

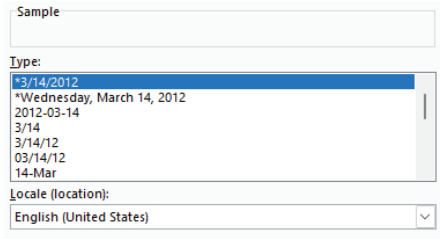
- Using the **[Ctrl + 1]** keyboard shortcut.
- Clicking the dialog launcher button in the **Font**, **Alignment**, or **Number Groups** on the ***Home Tab***.
- **Right-click** a cell or range of cells and choose *Format Cells* from the menu,

Date & Time Formatting, continued

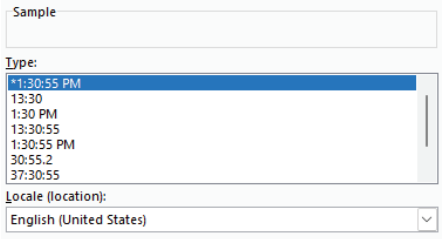
When the *Format Cells* dialog opens, choose the ***Number Tab.*** In the Category list, ***Date***, ***Time***, and ***Custom*** categories are available.



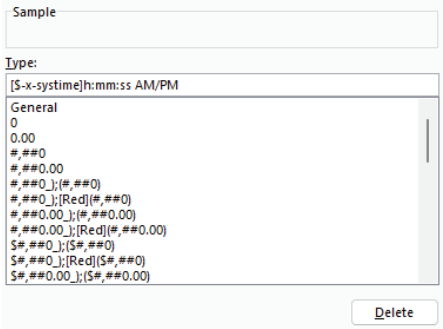
Choosing one of these will show a list of related choices available.



Date Formats



Time Formats



Custom Formats

Date & Time Formatting, continued

The ***Custom Formats*** category provides a list of choices and an option to create a custom format by typing into the **Type:** field. Below is the list of formatting codes associated with dates and times.

To display As	*Use this code*
Years as 00-99	yy
Years as 1900-9999	yyyy
Months as 1-12	m
Months as 01-12	mm
Months as Jan-Dec	mmm
Months as January-December	mmmm
Months as J-D	mmmmm
Days as 1-31	d
Days as 01-31	dd
Days as Sun-Sat	ddd
Days as Sunday-Saturday	dddd
Hours as 0-23	h
Hours as 00-23	hh
Minutes as 0-59	m
Minutes as 00-59	mm
Seconds as 0-59	s
Seconds as 00-59	ss
Time as 4 AM	h AM/PM
Time as 4:36 PM	h:mm AM/PM
Time as 4:36:03 PM	h:mm:ss A/P
Time as 4:36:03.75 PM (Fractions of a second)	h:mm:ss.00
Elapsed time (hours and minutes) as 1:02	[h]:mm
Elapsed time (minutes and seconds) as 62:16	[mm]:ss
Elapsed time (seconds and hundredths) as 3735.80	[ss].00

Extracting Parts of Dates/Times

When only one part of a date or time value is needed or needs to be referred to in another formula, use the **SECOND**, **MINUTE**, **HOUR**, **DAY**, **MONTH**, or **YEAR** functions. The syntax of all these functions has only one required argument: a date or time value that contains the desired information to extract.

The syntax of the function is:

=FUNCTION(*Serial_Number*)

Time

Entering Time

There are several methods for entering the time into a cell: manually typing it in, using a formula, or using a keyboard shortcut.

- Manually typing in the time.
  - Typing **11pm** will enter the time as a text value.
  - Typing **11 p** or **11 pm** will enter the time as a number with time formatting applied.
- Using the **=NOW()** formula, enter the time using the formatting applied.
- Using the **[Ctrl + Shift + ;]** shortcut enters the time with formatting applied.

NOW Function

Since this is a function formula, it updates as the file is opened and recalculated. Any other formulas referring to cells containing **NOW** functions will also update, making those calculations much more dynamic than using a static time value.

The syntax of this function is,

=NOW()

By default, both the date and time formats are applied, but that can be changed by applying appropriate formatting.

Note

Custom Number formats do not transfer from one file to another. New custom formats must be created for each file.

Adding and Subtracting Times

Adding or subtracting times to get an accumulated or differential time is not uncommon; Excel can use simple arithmetic to accomplish this type of task.

Adding Times

Adding Times is done by adding the cells containing the time data, as shown below.

=Time+Time+Time

When adding times, if the returned value exceeds 23hrs 59min 59sec, the result will only show a value between 0 and 23:59:59. A custom format is required to show more hours than are in a day.

The new custom format will be:

[h]:mm

Time,
continued

Subtracting Times

Another standard function is tracking time spent on projects. When a start and end time are recorded in the data, a simple subtraction formula is required, as shown below.

=End time-start time

Use the time formatting that includes AM/PM to ensure *Excel* understands how to handle times spanning days.

When the returned value could be greater than 24 hours, the cells containing the start and end times must be formatted using a different custom format, as displayed below.

m/d/yyyy h:mm AM/PM.

(*Note the empty spaces after the yyyy and mm entries.*)

The cells containing the formula should use the same custom format to allow for more than 23 hours in the result.

[h]:mm;@

These custom formats are created in the *Format Cells* dialog using the ***Custom*** category on the ***Number Tab***.

Dates

Note

Dates may be entered as text strings within quotation marks (for example, "2001/1/30"), as serial numbers (for example, 36921, which represents January 30, 2001, if using the 1900 date system), or as the results of other formulas or functions.

Entering A Date

There are several methods for entering a date into a cell: manually typing it in, with a formula, or using a keyboard shortcut.

- Manually typing in the date.
- Using the **=TODAY()** formula.
  - By default, the returned value has a short date format.
- Using the **[Ctrl + ;]** shortcut enters the date.
  - The entered date also uses the short date formatting by default.

TODAY Function

Like the **NOW** function, this updates as the file is opened and recalculated. Any other formulas referring to cells containing **TODAY** functions also update, making those calculations more dynamic than a static date.

The syntax of this function is:

=TODAY()

After adding the formula, the cell's format converts to a short date.

Finding the Difference Between Two Dates

Tracking the number of days between two dates can be as simple as creating a math formula. Subtracting one date from another returns a value formatted as a date, which must be converted to a number format for the result to make sense.

The syntax of this formula is:

=End_Date-Start_Date

The ***End_Date*** should be the most current date (more significant serial number), while the ***Start_Date*** should be the older date (lower serial number). If the date arguments use cell references and will run down a column, consider setting the ***End_Date*** cell reference as an absolute reference.

Dates,
continued

DAYS Function

The **DAYS** function returns numeric values instead of dates. Removing the additional step of reformatting the formula results are from a simple date subtraction formula.

The syntax of this function is:

=DAYS(*End_Date*,*Start_Date*)

- ***End_Date*** - is a required argument; date with the highest serial number value.
- ***Start_Date*** - is a required argument; date with the lower serial number value.

DATEDIF Function

Note: During the formula creation process, the **DATEDIF** function will not show a tooltip since it is not a Standard Excel formula and will not be listed as a built-in function.

When more specifically formatted results are required, use the **DATEDIF** function. This function is ideal since it calculates the days, months, or years between two dates.

The syntax of this function is:

=DATEDIF(*Start_Date*,*End_Date*,*Unit*)

- ***Start_Date*** - is a required argument for the starting date of the period. (lower serial number value)
- ***End_Date*** - is a required argument for the ending date of the period. (larger serial number value)
- ***Unit*** - is a required argument that defines what unit of measure the result will be in. (year, month, day)

Note: Use the formatting code related to the desired type of Unit.

Unit	Returns
"Y"	The number of complete years in the period.
"M"	The number of complete months in the period.
"D"	The number of days in the period.
"MD"	The difference between the days in ***start_date*** and ***end_date***. The months and years of the dates are ignored.
"YM"	The difference between the months in ***start_date*** and ***end_date***. The days and years of the dates are ignored
"YD"	The difference between the days of ***start_date*** and ***end_date***. The years of the dates are ignored.

The quotation marks are required when adding the Unit to the formula.

Dates, continued

When a more readable and detailed result is required, this formula can be easily modified to return the information in a desired manner.

For example, if the formula needs to return the number of years, months, and days, it can be written as follows.

=DATEDIF(*Start_date,End_date,"y"*)&
"*years,*"&DATEDIF(*Start_date,End_date,"ym"*)&
"*months,*"&DATEDIF(*Start_date,End_date,"md"*)&
"days"

This will return the following: **2 years, 4 months, 10 days**

- Each **DATEDIF** returns a specific date component: the year, month, and day between two dates.
- Using the **&**'s adds the result of the next formula to the first formula result.
- The formula adds text in quotes as written.
  - **"years,"** The next ampersand (&) adds the next section of the formula.

Defining a Formula as a Name

When a complex formula is used repeatedly throughout a worksheet, it becomes increasingly likely to make mistakes entering long and nested formulas. Consider creating names for parts of or entire formulas in cases like this. Creating names is done by defining a name and entering the formula into the Refers to: field of the defined name.

Note

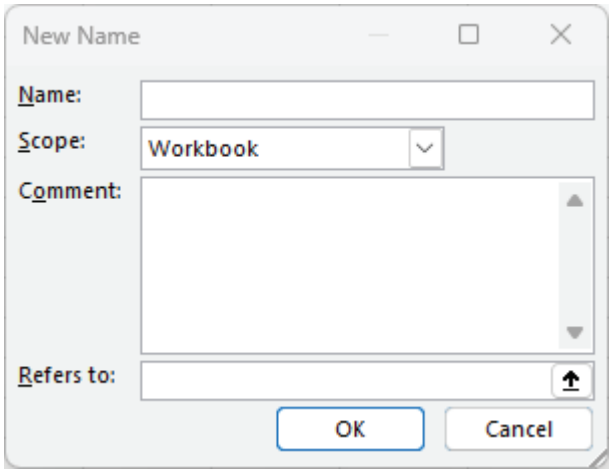
If using the named function on any worksheet, consider creating a LAMBDA function, as covered in lesson 1.

- Build and test a formula to refer to with a name.
- Select and copy the formula.
  - Use the formula bar or double-click into the cell to access the formula.
- Click the **Define Name** button in the **Defined Names Group** on the ***Formulas Tab***.
- In the *Define Name* dialog,
  - Enter a name in the **Name:** field.

Dates, continued

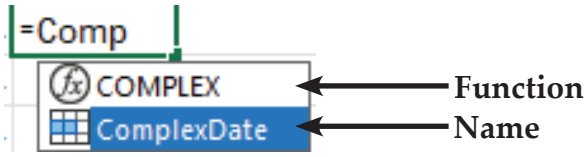
Defining a Formula as a Name, continued

- Set the **Scope:** field to the current worksheet. The formula will only work on this worksheet since the cell addresses will be absolute references.
- Paste the copied formula into the **Refers to:** field.
- Tap the **[OK]** button.



Using a Named Formula

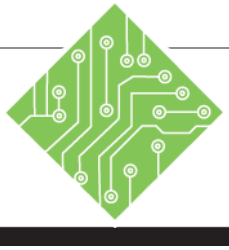
Select the cell where the formula needs to be entered and type in the equal sign. Then, begin typing the name of the named formula. The IntelliSense drop-down will show the names among the available functions and names. To enter the named formula, double-click the named formula or tap the [Tab] key of the highlighted name.



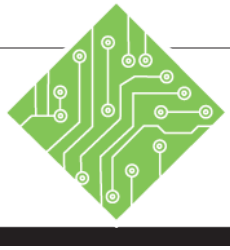
The named formula will run using the same offset as the original formula used in its arguments.

Action 1 - Adding and Formatting Date and Time

Instructions:	**Results/ Comments:**
1. Open the **DateFormulas.xlsx** file.	
2. Save the file as **MyDateFormulas**.	Use the **F12** key command to open the *Save As* dialog.
3. Select the ***DateTime*** sheet.	Look at the formatting applied to the cells containing the times.
4. Select cell **B2** and enter the current date then tap the **[Enter]** key.	Simply type the date. Once entered, *Excel* applies the short date formatting.
5. Select cell **B3** and enter the current time as shown on the computer's clock.	This is somewhat unclear as to the exact time.
6. Edit cell **B3** to read as, **=#:## p** then tap the **[Enter]** key.	By adding a space and either an *a* or *p* changes the formatting to show AM or PM accordingly. Use lower case letters. The # sings here represent the time already entered.
7. Select cell **C2** and use the **[Ctrl + ;]** keyboard shortcut.	The current date is entered as a static value.
8. Select cell **C3** and use the **[Ctrl + Shift + ;]** keyboard shortcut.	The current time is entered as a static value, it should match the formatting in cell **B3**.
9. Select cell **D2** and enter the following, **=TODAY()** then tap the **[Enter]** key.	The current date is entered. This is a dynamic date entry, meaning it will update as the file is re-opened and recalculated.
10. Select cell **D3** and enter the following, **=NOW()** then tap the **[Enter]** key.	A dynamic time is entered. The cell is formatted to include both date and time
11. Right-click cell **D3** and choose *Format Cells* from the menu.	The *Format Cells* dialog opens.



Instructions:	Results/ Comments:
12. Select the *Time* category on the left, choose the *1:30 PM* formatting option then click the **[OK]** button.	The *Format Cells* dialog shows what formatting is applied to the active cell. The cell is now correctly formatted.
13. Click the **Show Formulas** button in the **Formula Auditing Group** on the ***Formulas Tab***.	**[Ctrl + ~].** The formula results are replaced with the formulas and the date and times are shown as actual unformatted values.
14. Examine the way the dates are shown.	As a serial number, showing the number of days since January 1st 1900.
15. Examine the way the times are shown.	As a fractional value of a single day formatted as a decimal.
16. Click the **Show Formulas** button in the **Formula Auditing Group** on the ***Formulas Tab***.	The formula results and formatted date and times are re-displayed.
17. Save the file**.**	**[Ctrl + S]**

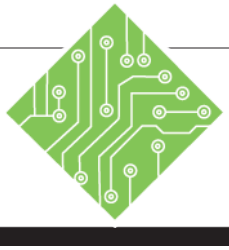


Instructions:	Results/ Comments:
1. The **MyDateFormulas** file should still be open with the ***DateTime*** sheet active.	If not, reopen it.
2. Select cell **H2** and enter the following, **=Year(D2)** then tap the **[Tab]** key.	The year is extracted from the date and is using the *General* type of formatting.
3. Select cell **I2** and enter the following, **=Month(D2)** then tap the **[Tab]** key.	The month is extracted from the date and is using the *General* type of formatting.
4. Select cell **J2** and enter the following, **=Day(D2)** then tap the **[Tab]** key.	The day is extracted from the date and is using the *General* type of formatting.
5. Select cell **K2** and enter the following, **=DATE(H2,I2,J2)** then tap the **[Enter]** key.	The text values in cells **H2:J2** are combined with the date formatting applied.
6. Save the file.	**[Ctrl + S]**
7. Try extracting the hour and minutes from cell **D3** into cells **H3:I3**.	Use the **HOUR** and **MONTH** functions.
8. Try combine those values as time in cell **K3**.	Use the **TIME** function.
9. Save the file.	**[Ctrl + S]**

Action 3 - Naming a Formula

Instructions:	Results/ Comments:
1. The **MyDateFormulas** file should still be open.	If not, reopen it.
2. Activate the ***DateDif*** sheet.	
3. Select cell **A1** and enter the current date.	Use the key command, **[Ctrl + ;]**.This needs to be a static date.
4. Use the *Auto-fill* handle to drag down a second date.	Tomorrows date appears in cell **A2**.
5. Double click into the **A2** and edit the date 14 days forward and tap the **[Enter]** key.	The dates are now 14days apart.
6. Select cells **A1:A2** and use Auto-fill to complete the rest of the date list to cell **A15**.	The list of dates separated by two week increments is created.
7. Select cell **C1** and enter the following, **=DATEDIF(A1,B1,"y")&"Years, "&DATEDIF(A1,B1,"ym")&"Months, "&DATEDIF(A1,B1)&"md")&"Days"** then tap the **[Enter]** key.	The formula tells you how many year, months, and days exist between the two dates. The result is a mixture of numeric and textual data as defined by the formula.
8. Reselect cell **C1**.	
9. Click into the formula bar, highlight the entire formula and copy it, then tap the **[Esc]** key.	The formula will be used to create a named formula. By tapping the **[Esc]** key, you exited editing the cell without making any changes.
10. Click the **Define Name** button in the **Defined Names Group** on the ***Formulas Tab***.	The *Define Name* dialog opens.
11. In the **Name:** field type in **CountDown**, Set the **Scope:** field to *This Worksheet*, Paste the formula into the **Refers to:** field, then click the **[OK]** button.	The formula has been saved as a name.

Instructions:	Results/ Comments:
12. Click into cell **C2** and enter the following, **=CountDown** then tap the **[Enter]** key.	As you are entering the formula, the IntelliSense options appears showing the named formula, double-clicking the name quickly enters it.
13. Use ***Auto-fill*** and drag down to **C15** to complete the countdown list.	Double-click the ***Auto-fill*** handle.
14. Save the file.	**[Ctrl + S]**



Instructions:	Results/ Comments:
1. The **MyDateFormulas** file should still be open.	If not, reopen it.
2. Activate the ***HR*** sheet.	
3. Select cell **A2** and enter the following, **=TODAY()** then tap the **[Enter]** key.	The current date is entered.
4. Select cell **F5** and enter the following , **=YEAR($A$2)-YEAR(E2)** then tap the **[Enter]** key.	This calculates the age of the first employee in the list by subtracting the year value in cell **E2** from the year value in cell **A2**. Since this formula will be repeated down the column the **A2** cell reference is absolute.
5. Use ***Auto-fill*** to calculate the ages of the remaining employees.	Double-click the ***Auto-fill*** handle.
6. Save the file.	**[Ctrl + S]**

Calculating Working Days

WORKDAY Function

Calculating a day in the future can be as simple as adding the number of days in the future to the current date since the date is a number.

=Date+Value

Use the **WORKDAY** formula if only working days are to be added to a date to calculate a future date. This formula assumes the work week runs Monday through Friday. Like the **NETWORKDAYS** formulas, it can also exclude holidays.

The syntax of this function is:

=WORKDAY(*Start_Date,Number_Of_Days,[Holidays]*)

- ***Start_Date*** - is a required argument when adding days.
- ***Number_Of_Days*** - is a required argument, the number of working days being added to the start date.
- ***Holidays*** - an optional argument, a list of dates to be excluded by the function.

EDATE Function

The **EDATE** function provides a simple solution to add months to a date. It returns both future and past dates with equal ease. Format the cell containing the formula as a date.

The syntax of this function is:

=EDATE(*Start_Date,Number_Of_Months*)

- ***Start_Date*** - is a required argument when adding months.
- ***Number_Of_Months*** - is a required argument for the number of months to add or subtract from the start date.
  - To subtract, type a minus sign before the Number_Of_Months value

Calculating Working Days, continued

NETWORKDAYS Function

When it is necessary to calculate the number of working days, excluding weekends and holidays, Excel has two built-in formulas to run this type of calculation: **NETWORKDAYS** and **NETWORKDAYS.INTL**. These functions are very similar except that the **NETWORKDAYS** function uses Saturday and Sunday as the weekend while the **NETWORKDAYS.INTL** allows the user to define what days of the week are considered weekends.

The syntax of this function is:

=NETWORKDAYS(*Start_Date,End_Date,[Holidays]*)

- ***Start_Date*** - is a required argument for the starting day of a date range.
- ***End_Date*** - is a required argument, the last day of a date range.
- ***Holidays*** - an optional argument, a list of dates to be excluded as working days by the function. The argument can be a range of cells containing dates or an array of constant serial numbers representing specific dates.

The syntax of the **NETWORKDAYS.INTL** function is:

=NETWORKDAYS.INTL(Start_Date,End_Date, [Weekend],[Holidays])

- ***Start_Date*** and ***End_Date*** - are required arguments that define the date range.
- ***Weekend*** - an optional argument used to define the weekend's day or days.
  - Refer to the table on the next page for accepted values and meanings this argument uses.
- ***Holidays*** - an optional argument, a list of dates to be excluded as working days by the function.

Calculating Working Days, continued

Table of Weekend Argument variables

Weekend number	Weekend days
1 or omitted	Saturday, Sunday
2	Sunday, Monday
3	Monday, Tuesday
4	Tuesday, Wednesday
5	Wednesday, Thursday
6	Thursday, Friday
7	Friday, Saturday
11	Sunday only
12	Monday only
13	Tuesday only
14	Wednesday only
15	Thursday only
16	Friday only
17	Saturday only

Finding Days of the Week

WEEKDAY Function

Use this function when it is important to know on what day of the week a date falls. This function returns a numeric value representing the day of the week.

The syntax of this function is:

=WEEKDAY(Serial_Number,[Return_Type]

- ***Serial_Number*** - is a required argument, the cell containing a date or the serial number of a date.
- ***Return_Type*** - an optional argument that defines the first day of the week. If omitted, the default is applied.

The list of available return types used by WEEKDAY functions.

Value	Description
1 or omitted	Numbers 1 (Sunday) through 7 (Saturday).
2	Numbers 1 (Monday) through 7 (Sunday).
3	Numbers 0 (Monday) through 6 (Sunday).
11	Numbers 1 (Monday) through 7 (Sunday).
12	Numbers 1 (Tuesday) through 7 (Monday).
13	Numbers 1 (Wednesday) through 7 (Tuesday).
14	Numbers 1 (Thursday) through 7 (Wednesday).
15	Numbers 1 (Friday) through 7 (Thursday).
16	Numbers 1 (Saturday) through 7 (Friday).
17	Numbers 1 (Sunday) through 7 (Saturday).

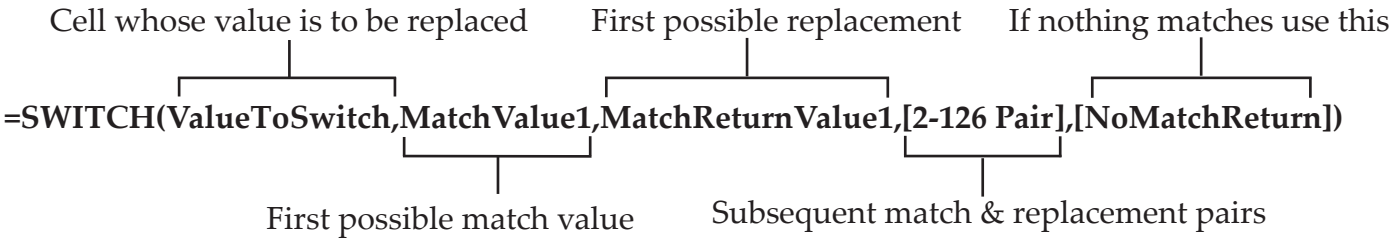
Using the **SWITCH** function with the **WEEKDAY** function nested inside would show the day of the week as text instead of a number.

Finding Days of the Week,
continued

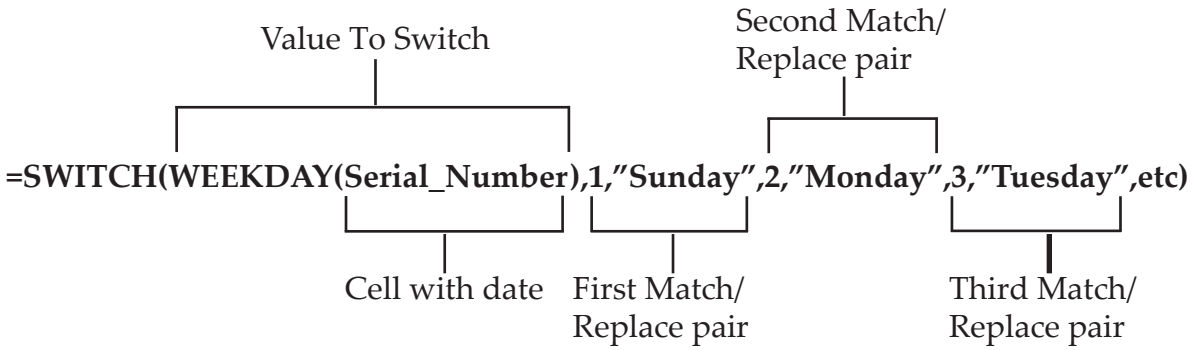
SWITCH Function

The **SWITCH** function can be seen as a merged **VLOOKUP** and **IFS** formula. It changes a cell's existing value based on its content. This function can use up to 126 value pairs. Finding a match causes the match return value to replace the original value. Should no matches be found, returning an optional value is possible.

The syntax of this function is,



The first argument of the formula (***ValueTo Switch***) is where the **WEEKDAY** function can be nested so that the day of the week on which the matched date falls is returned.



Using the WEEKDAY function negates the need to add lookup tables and write multiple formulas in cells to get a desired answer.

OFFSET Functions

OFFSET Functions

The **OFFSET** function is used to pull data form a cell or range of cells a set distance away from a reference cell. The distance from the reference cell is determined by defining the number of rows and columns away, positive value move Right and Down while negative values move Left and Up.

This function is very useful when nested inside of other functions like a sum, making **SUM** act like a **SUMIF** function.

The syntax of this formula is,

=OFFSET(Reference, Rows, Cols, [Height], [Width])

- Reference - is a required argument, the cell from which the offset will be based. .
- Rows - is a required argument, this is where you define how many rows, Up or Down, from the reference cell to pull data from.
  - If using negative values, make sure that the number of rows will not lead to a location above Row1, as that location does not exist. The #REF error will be returned in those cases.
- Cols - is a required argument, use this argument to define the number of columns, Left or Right, from the reference cell to pull data from.
  - If using negative values , make sure that the number of columns will not lead to a location left of column A, as that location does not exist. The #REF error will be returned in those cases.
- Height - an optional argument, use this when a range spanning multiple rows is to be returned instead of the value of a single cell. This must be a positive number, the formula can only pull down from a cell location..
- Width - an optional argument, use this when a range spanning multiple columns is to be returned instead of the value of a single cell. This argument must also only contain positive values.

OFFSET Functions, cotinued

Calculated OFFSETs

Adding an OFFSET as the second cell reference that defines the range being summed can make a regular SUM become much more dynamic. Image creating a dashboard worksheet where users can change a variable to control what data is summed.

The syntax of the formula is,

=SUM(cell:OFFSET(SameCell,Rows-1,Cols-1))

Including the minus 1 after the Row or Col argument is done to include the value of the OFFSET cell as the first row or column in the sum range.

Calculating a row of data is written as

=SUM(B4:OFFSET(B4,0,Cols-1))

Instead of using a fixed values in the Rows or Cols arguments consider using cell references, that way a user can redefine that argument within the formula and thereby choose how much data is being summed.

Finding Week Number

WEEKNUM Function

This function returns the week number of a specific date. For example, the week containing January 1 is the first week of the year, numbered week 1 up to a possible week 52.

There are two systems used for this function:

- **System 1:** The week containing January 1 is the first week of the year and is numbered week 1.
- **System 2:** The week containing the first Thursday of the year is the first week of the year and becomes week 1. This system is the methodology specified in ISO 8601, commonly known as the European week numbering system.

The syntax of this function is:

=WEEKNUM(Serial_Number,[Return_Type])

- ***Serial_Number:*** Required. - a date within the week. Use the DATE function to enter dates or as results of other formulas or functions. For example, use **DATE(2008,5,23)** for the 23rd day of May, 2008. Entering dates as text can cause problems to occur.
- ***Return_Type:*** Optional - a number that determines on which day the week begins. The default is 1.

Return_type	Week begins on	System
1 or omitted	Sunday	1
2	Monday	1
11	Monday	1
12	Tuesday	1
13	Wednesday	1
14	Thursday	1
15	Friday	1
16	Saturday	1
17	Sunday	1
21	Monday	2

Finding Week Number,
continued

ISOWEEKNUM Function

There may come a time when someone needs to know the week number in the year in which a specific date falls. *Excel* has added a new formula, the **ISOWEEKNUM** formula, to make this easy to determine.

The syntax of this function is:

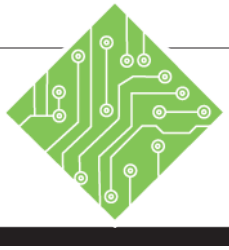
=ISOWEEKNUM(Date)

- Date: Required - any date after January 1, 1900 is valid. Dates before January 1, 1900, will return a #NUM error.
  - The date cell must be formatted as a date in order for the function to work.

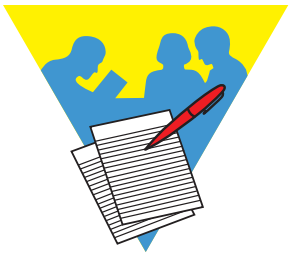
The standard issued in 1986, prescribes amongst other things,

- Formats of Dates must either:
  - A non-separated form of **yyyymmdd** (20160130 for 30-Jan-2016)
  - A separated form of **yyyy-mm-dd** (2016-01-30 for 30-Jan-2016)
- Week numbering must use one of the two following systems:
  - Week 1 begins on the Monday where 4th January falls.
  - Week 1 begins on the Monday which contains the first Thursday of the calendar year.

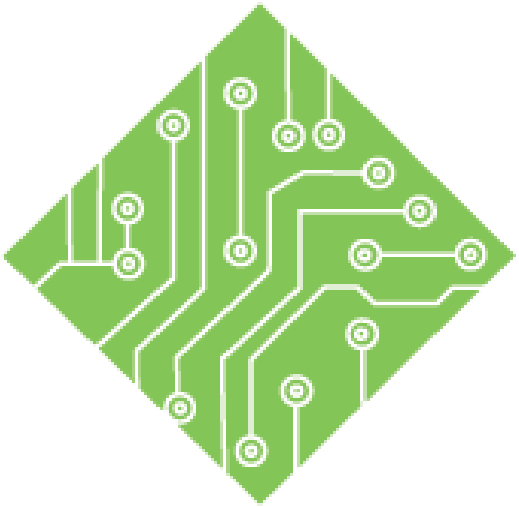
Which of these functions to use is determined by the requirements desired.



Instructions:	Results/ Comments:
1. The **MyDateFormulas** file should still be open with the ***HR*** sheet active.	If not, then re-open it.
2. Select cell **H5** and enter the following, **=G5+90** then tap the **[Enter]** key.	Ninety days have been added to the hire date. While this is accurate in the addition, it does not reflect what ninety working days would be.
3. Reselect cell **H5** and enter the following, **=WORKDAYS(G5,90)** then tap the **[Enter]** key.	Now the probation period spans ninety working days all be it, without discounting holiday dates.
4. Select cell **I5** and enter the following, **=DATE(YEAR(G5)+30),MONTH(G5), DAY(G5))** then tap the **[Enter]** key**.**	This formula is extracting each component of the date in cell **G5** adding thirty years and recombining the constituent parts as a new date.
5. Use **AutoFill** to complete calculating the other retirement dates.	Double-click the ***AutoFill*** handle.
6. Save the file.	**[Ctrl + S]**
7. Activate the ***Project*** sheet.	
8. Select cell **D2** and enter the following, **=NETWORKDAYS(A2,A4,B7:B17)** then tap the **[Enter]** key.	This calculates exactly how many working days exist between the start and end dates of the project, taking into account both weekends and holidays.
9. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**



Tips and Notes

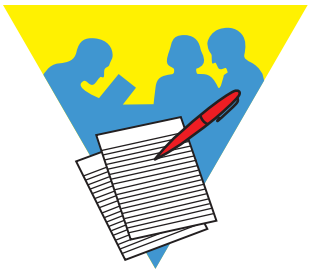


Appendix : Troubleshooting

Lesson Overview

The following concepts are covered in this chapter:

- Formula Auditing
- Tracing Formulas
- Errors
- Error Checking
- Evaluating Formulas
- Watch Window
- Calculations



Lesson Notes

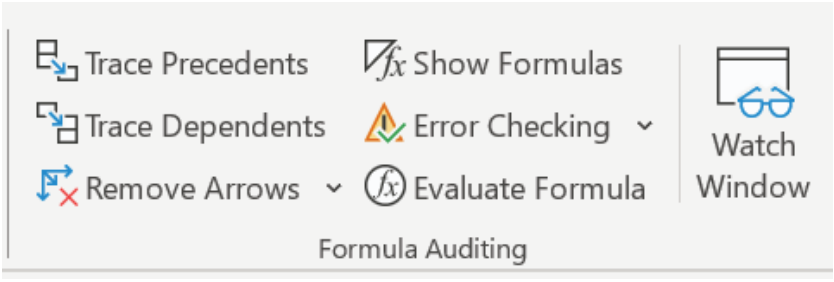
Formula Auditing

Spreadsheets created by other users or created in previous versions of Excel may have formulas that no longer function or show error messages. In this lesson, learn to use *Excel's* tools to help audit and correct these formula errors and issues.

The Formula Auditing Group

The **Formula Auditing Group** on the ***Formulas Tab*** has tools to facilitate management of your formulas. A variety of tools allow you to:

- ***Trace Precedents and Dependents*****:** displays blue lines to show where formulas draw their data.
  - ***Precedents*** show where the selected cell gets its data.
  - ***Dependants*** show connections to cells that are referencing the selected cell.
- ***Remove Arrows*****:** clears the lines from the spreadsheet.
- ***Show Formulas*****:** displays the formulas in their cells instead of the formula results.
- ***Error Checking***: examines the spreadsheet for formula errors similar to spell check.
- ***Evaluate Formulas*****:** will open a dialog to move through the formula step-by-step.
- ***Watch Window*****:** opens a dialog that dynamically display results of multiple formula changes as data is modified.

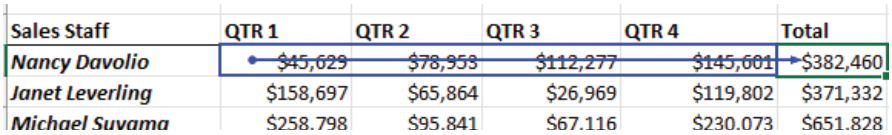


Tracing Formulas

Tracing Precedents

The **[Trace Precedents]** button will indicate which cells the active cell draws its data. A **Precedent** is a cell that contributes to the result.

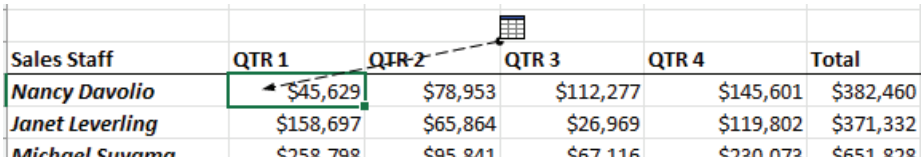
When clicking the button, a blue line appears. It traces backward to the cells that influence the current cell's formula, and dots on the line show where the source cells are.



Sales Staff	QTR 1	QTR 2	QTR 3	QTR 4	Total
Nancy Davolio	$45,629	$78,953	$112,277	$145,601	$382,460
Janet Leverling	$158,697	$65,864	$26,969	$119,802	$371,332
Michael Suyama	$258.798	$95.841	$67.116	$230.073	$651.828

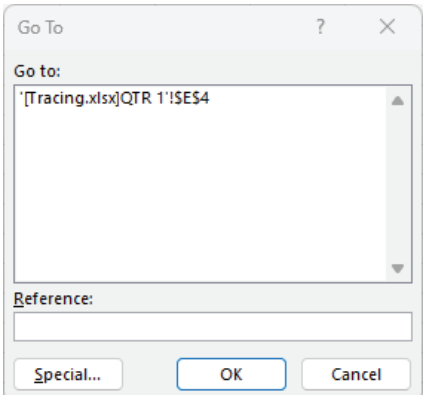
Clicking the button again will show the precedents of any preceding cells that contain formula references. Continue clicking the **[Trace Precedents]** button to continue examining formula connections.

If any cells reference content from other worksheets or workbooks, a black dotted line and worksheet icon will appear. Double-click the dotted line (not the icon) to follow the precedents and to open the *Go To* dialog box.



Sales Staff	QTR 1	QTR 2	QTR 3	QTR 4	Total
Nancy Davolio	$45,629	$78,953	$112,277	$145,601	$382,460
Janet Leverling	$158,697	$65,864	$26,969	$119,802	$371,332
Michael Suyama	$258.798	$95.841	$67.116	$230.073	$651.828

The *Go To* dialog box will list any 3D references. Select the one to follow and click the **[OK]** button to move to the sheet or file. If the file is closed, it will open.



Tracing Formulas, continued

Tracing Dependents

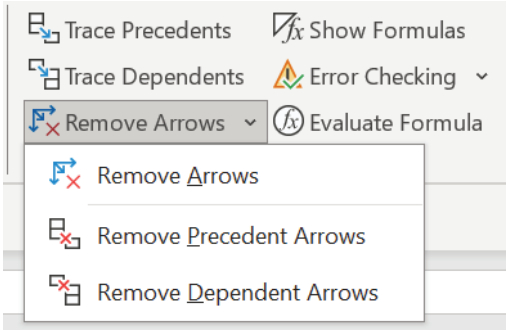
Use this tool to see what other formulas are drawing data from the active cell. The **Dependent** cell does not have to contain a formula. It can be a static value. The arrows will now point to cells that the active cell contributes to. Just like the **[Trace Precedent]** button, if you continue to click the button it will show the next level of cells that the active cell contributes to.

Sales Staff	QTR 1	QTR 2	QTR 3	QTR 4	Total
Nancy Davolio	$45,629	$78,953	$112,277	$145,601	$382,460
Janet Leverling	$158,697	$65,864	$26,969	$119,802	$371,332
Michael Suyama	$258,798	$95,841	$67,116	$230,073	$651,828
	$463,124	$240,658	$206,362	$495,476	$1,405,620

Removing Arrows

Since Excel can show both types of arrows simultaneously, the **[Remove Arrows]** button will remove all **Precedent** or **Dependent** arrows with a single click. The drop-down for the **[Remove Arrows]** button will offer three choices:

- ***Remove Arrows***: This will remove all the arrows at once.



- ***Remove Precedent Arrows***: Use to remove the last level of precedent arrows applied.
- ***Remove Dependent Arrows***: Use to remove the last level of dependent arrows applied.

Action 1- Tracing Formulas

Instructions:	Results/ Comments:
1. Open the **Tracing.xlsx** file.	The file is stored in the class data files folder, the instructor will point out where to find the folder.
2. Save the file as **MY Tracing**	
3. Make the ***QTR 1*** sheet active.	Click the ***QTR 1*** sheet tab below the spreadsheet.
4. Select cell **F13**.	This cell is the result of a summing formula.
5. Click the **[Trace Precedents]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	A blue line indicates what cells are referred to by the formula in cell **F13**.
6. Click the **[Trace Precedents]** button again.	A new set of lines indicate the cells being referred to by the first set of cells that are being calculated in cell **F13**.
7. Click the **[Remove Arrows]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	The arrows are removed from the spreadsheet.
8. Select cell **B6**.	This cell contains a value that another cells' formula is referring to.
9. Click the **[Trace Dependents]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	Blue lines point to the cells that include the active cell in their formulas.
10. Click the **[Trace Dependents]** button again.	More lines appear, pointing to the next cells that are dependent on the active cell in their formulas. A black dotted line pointing to a spreadsheet icon also appears, indicating that formulas in other locations depend on the cell in their calculations.
11. Click the **[Remove Arrows]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	The arrows are removed from the spreadsheet.

Instructions:	Results/ Comments:
12. Make the ***Summary*** sheet active.	Click the tab below the spreadsheet.
13. Select cell **F13**.	This cell contains a formula referring to other cells in the spreadsheet.
14. Click the **[Trace Precedents]** button three times.	Each time the button is clicked the formula is traced back another level.
15. Double-click any of the dotted lines.	The *Go To* dialog box opens, showing the reference or related references.
16. Select the reference and click the **[OK]** button.	The reference location is now active.
17. Make the ***Summary*** sheet active.	Click the tab below the spreadsheet.
18. Click the **[Remove Arrows] drop-down** button and choose *Remove Precedent Arrows*.	One level of Precedent lines is removed from the spreadsheet.
19. Click the **[Remove Arrows]** button.	The arrows are removed from the spreadsheet.
20. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**

Errors

Errors

Errors can be marked and corrected in two ways:

- The **[Error Checking]** button (like a spelling checker) checks for errors throughout the spreadsheet.
- While working, use the ***Error Checking Options*** immediately when they occur on the worksheet.

When an error occurs, a green triangle appears in the upper-left corner of the cell.

Error Messages

Some errors cannot return a result and will display an error value in the cell. The error messages are simple ways in which *Excel* explains problems in formulas.

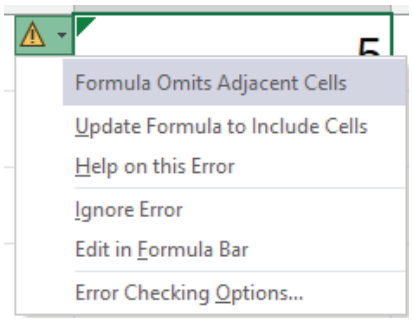
#NAME?	occurs when *Excel* does not recognize text in a formula if using a name in a formula and the name is misspelled or does not exist
#VALUE!	occurs when a formula has the wrong type of argument if the formula is expecting numeric values but a cell contains text or dates
#DIV/0!	occurs when a formula tries to divide a number by 0 or an empty cell
#REF!	occurs when a formula refers to a cell that is not valid, if the formula refers to a cell in a deleted row or column
#NULL!	occurs when specifying an intersecting range that does NOT intersect; the value in a cell may be negative, percent, or currency value
#N/A	occurs when the wrong type of argument or operand is used in the formula, most commonly relating to lookup formulas

Error Checking

Error Checking Options

Other errors return values but show error warnings, such as inconsistency with adjacent formulas. The error may advise examining the formula. When the cell with the error is active, a small warning icon appears to the left of the cell.

When the user clicks the icon, a drop-down menu appears with suggestions for resolving the potential error. For example, there could be a potential error because the user may not need to refer to the same relative range for this instance. In that case, "Ignore Error" would be the best option to avoid having the icon appear when the cell is selected.



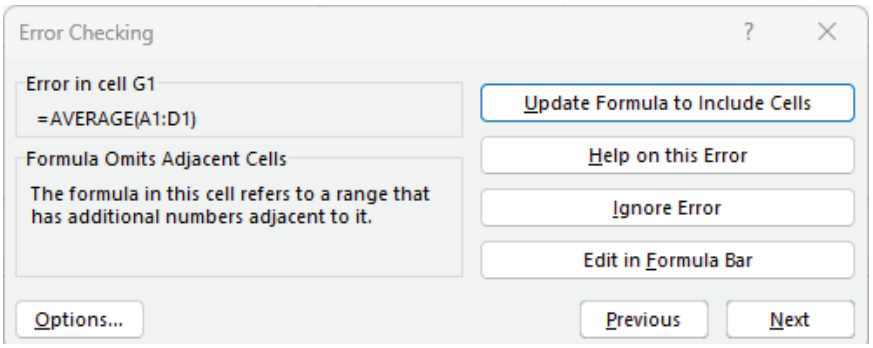
- ***Inconsistent Formula***: This choice is the basis error (it can change depending on the formula) that *Excel* has determined.
- ***Copy Formula from Above***: This choice will copy the formula from the cell above to remove the perceived error.
- ***Help on This Error***: This selection opens the Help dialog box to the specific error type help page.
- ***Ignore Error***: This option leaves the formula as is without Excel continuing to display the **Error Checking Options** smart tag.
- ***Edit in Formula Bar:*** This choice displays the formula in the formula bar.
- ***Error Checking Options***: This selection opens the Options dialog box, where users can select the rules governing error checking.

Error Checking, continued

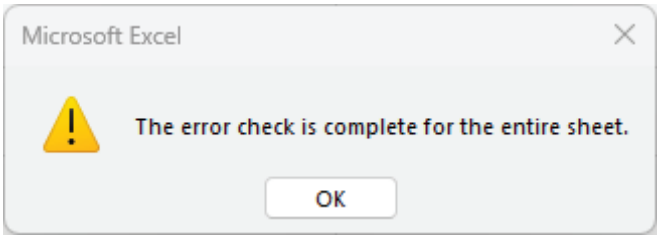
Error Checking

The tool works in a similar fashion to running a spell check in other programs. It inspects the entire spreadsheet, looking for errors that may not have been noticed while creating it.

Clicking the **[Error Checking]** button opens the *Error Checking* dialog box. The user can go through the spreadsheet, checking all errors at one time. These options are similar to those found in the **Error Checking Options Smart Tag** menu.



As one formula is corrected, click the [Next] button to advance to the next error found. When all errors have been checked, Excel displays a message to that effect.



While still in the process of checking for errors in the *Error Checking* dialog, clicking the **[Options]** button will open the *Excel Options* dialog, which contains more controls regarding how *Excel* will treat errors.

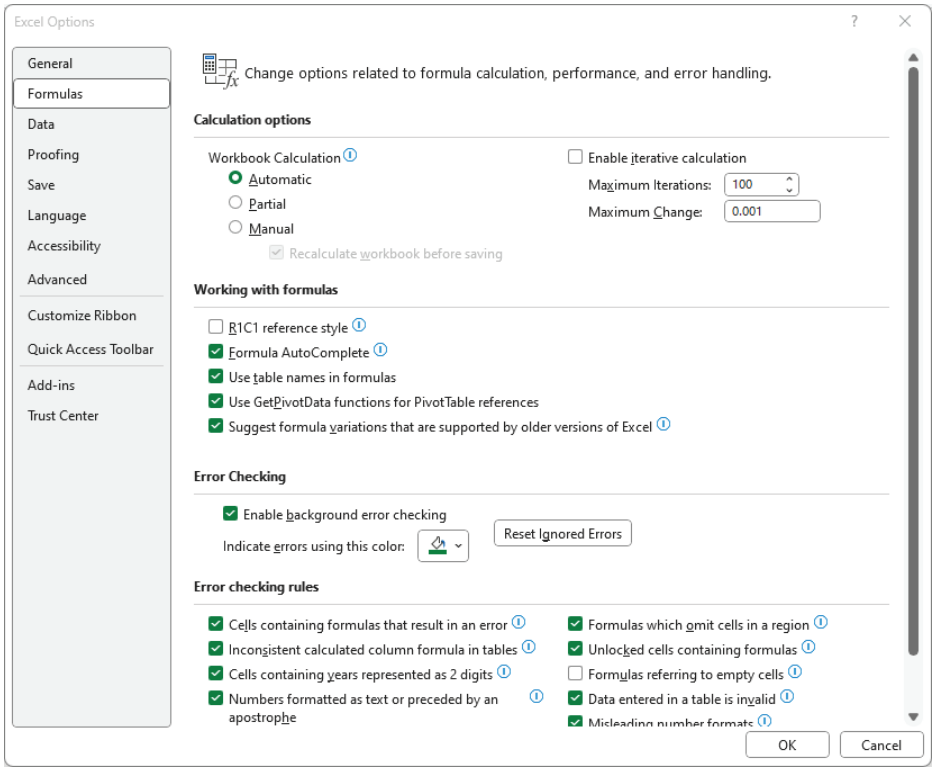
Error Checking,
continued

Formula Options

Turning off error checking is done through the program options by clicking the **[Options]** button in the *Error Checking* dialog. A more direct method to access the program options is to click the ***File Tab***, choose ***Options***, and then the *Formulas* category.

Note

Try using the keyboard to navigate the program to access the **Options** by sequentially tapping the **ALT, F, T** keys.



In the options, *Excel* users can turn error checking on or off and even change the color of the error warning triangle within the **Error Checking** group of controls. They can also determine what errors will be ignored or tracked in the **Error Checking Rules** group of commands.

- ***Cells containing formulas that result in an error***: Turning this off will stop the display of error messages.
- ***Inconsistent calculated column formula in tables***: Turning this off will stop displaying **Error Checking Options** smart tags for any formula perceived as inconsistent inside an *Excel* table.

Error Checking, continued

Exceptions within a calculated column in a table are as follows:

- Typing data other than a formula in a calculated column cell,
- Typing a formula in a calculated column cell and then click the **[Undo]** button on the *Quick Access Toolbar*,
- Typing in a new formula within a calculated column cell containing one or more exceptions,
- If copying data into the calculated column using a different formula,
- If a cell on another worksheet area referenced by a row within a calculated column is moved or deleted.

- ***Cells containing years represented as two digits:*** Turn off this rule to prevent *Excel* from checking the misinterpretation of ambiguous dates, such as an incorrect century.
- ***Numbers formatted as text or preceded by an apostrophe:*** Turning this rule off will stop *Excel* warnings that a numeric value could have a format for text. When this occurs, *Excel* will display the ***Error Checking Options*** smart tag triangle in the upper left corner of the cell. When the *Excel* user stores numbers as text, it will cause unexpected sorting behaviors and prevent the values from being calculated.
- ***Formulas inconsistent with other formulas in the region:*** Turning this rule off will stop *Excel* from warning the *Excel* user when a formula does not match the pattern of other formulas nearby. In many cases, formulas adjacent to other formulas differ only in the references used.

Action 2- Error Checking

Instructions:	Results/ Comments:
1. Open the **Errors** file.	
2. Save the file as **My Errors**.	
3. Select cells **E6:E17**.	We will add a formula to average the grades here.
4. Click the **[Autosum]** button drop-down in the **Function Library Group** on the ***Formulas Tab*** and choose *Average*.	The formula is added to all the cells. You can also find the **[Autosum]** button in the **Editing Group** on the ***Home Tab***.
5. Select column **E**.	Click the column header **E**.
6. Right-click the column and choose *Insert* from the menu.	A new column is added after column **D**.
7. Select cell **E5** and type in: **Photo Corrections** press the **[Enter]** key.	The project header is added. If you want to add a soft return, use the **[ALT Enter]** key combination.
8. In cell **E6**, type: **85** then, press the **[Enter]** key.	
9. In cell **E7**, type: **78** then, press the **[Enter]** key.	
10. Select cells **E6:E7**.	
11. Use AutoFill to fill in the remaining grades. Click the ***AutoFill Options*** smart tag and choose *Copy Cells*.	Choosing to copy the cells will fill the cells with the number pair.
12. Select cell **F8** and click the *Error Checking Options* smart tag. Choose *Update formula to include cells*.	The original formula reads as: **=AVERAGE(B8:D8)**. After updating the formula, now reads as: **=AVERAGE(B8:E8)**.
13. Click the **[Error Checking]** button in the **Formula Auditing Group** on the ***Formula Tab***.	The *Error Checking* dialog box opens, this acts like a spell check dialog box. You can check every error that potentially exists in the spreadsheet.

Instructions:

14. Click the **[Update formula to include cells]** button, repeat this to correct all the errors. When all have been updated, click the **[OK]** button to close the dialog box.

15. Click the ***File Tab*** and choose ***Options***. Choose the ***Formulas*** category and examine the ***Error Checking*** options.

16. Click **[OK]**.

17. Save and close the file.

Results/ Comments:

Each error is displayed and you are given options on how to correct them. When all errors have been checked a dialog box lets you know the checking is completed.

The ***Formula Error*** options can also be accessed from within the *Error Checking* dialog box by clicking the **[Options]** button.

To close the options.

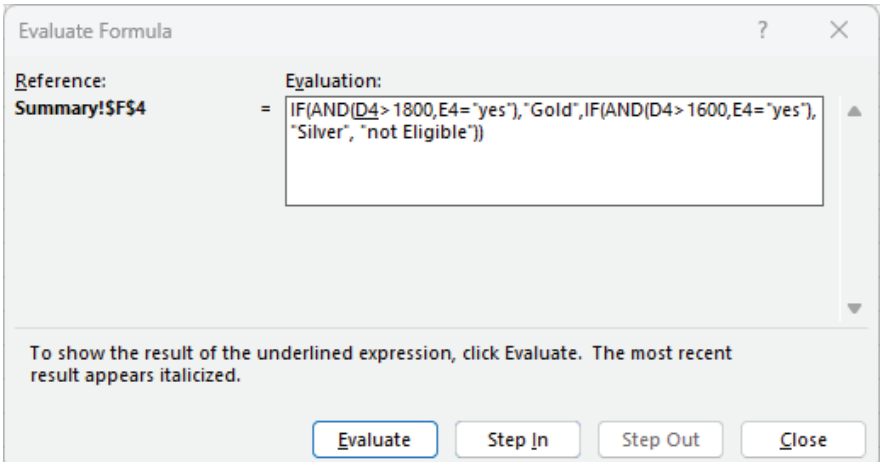
[Ctrl + S] and **[Ctrl + W]**

Evaluating Formulas

Using the Evaluate Formula Tool

The [Evaluate Formula] button will open an Evaluate Formula dialog box, allowing you to see the formula and go through it step-by-step. See the formula's logic by showing each step of the formula's calculation. This tool is also extremely useful in helping to understand where problems exist within a formula.

- Select the cell containing the formula to evaluate.
- Click the **[Evaluate Formula]** button in the **Formula Auditing Group** of the ***Formulas Tab***.
- The *Evaluate Formula* dialog box will open. In the **Evaluation:** field, the formula from the selected cell is displayed.

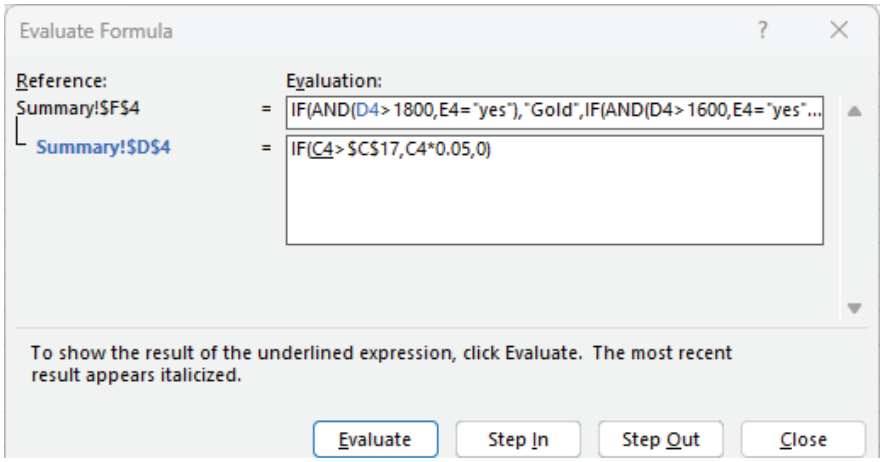


- Clicking the **[Evaluate]** button begins replacing cell addresses with their values and moving through the formula step-by-step. Keep clicking the button until the entire formula is shown and you see the end result.
- When the result is displayed in the **Evaluation:** field, click the **[Restart]** button to go back through the formula again if necessary.

Evaluating Formulas, continued

Using the Evaluate Formula Tool, continued

- Clicking the **[Step In]** button will evaluate each section of the underlined formula. If a cell address is underlined and is, in fact, the end result of a formula, the formula in that cell address is displayed in a new dialog below the main **Evaluation:** field.



- Continue to click the **[Step In]** button to see the string of cells needed for your formula to function. Each connection to cells that use formulas is displayed in a new level field of the **Evaluation:** field.
- To return to the original formula being evaluated, click the [Step Out] button to move back one level at a time until reached.

Action 3 - Evaluating Formulas

Instructions:	Results/ Comments:
1. Open the **Evaluation** file.	
2. Save as **My Evaluation**	
3. Select cell **H2**.	This cell contains a formula to be evaluated.
4. Click the **[Evaluate Formula]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	The *Evaluate Formula* dialog box opens.
5. Examine the formula in the *Evaluate Formula* dialog box.	The formula in cell **H2** is displayed in the **Evaluation** field. The underlined portion of the formula is the active part of the formula being evaluated. As the formula is evaluated step-by-step, the underline changes to show what part of the formula is active .
6. Click the **[Step in]** button.	Since the value is cell **G2** results from a formula, the formula used to determine the value in cell **G2** is displayed below. Notice, it is indented to show it is a secondary formula to the formula being evaluated.
7. Click the **[Step In]** button again.	Each time the **[Step in]** button is clicked, the formula is traced back to the original data the formula is drawing on. Once all levels of the formulas have been displayed, the value returned by the formula is displayed.
8. Click the **[Step out]** buttons twice.	Click this button to step back through the interconnected formula until all indented fields are removed.
9. Click the **[Evaluate]** button.	The first cell reference is replaced by the value in the cell.

Instructions:	Results/ Comments:
10. Continue clicking the **[Evaluate]** button until the formula results are displayed.	Each click of the **[Evaluate]** button reveals the next step in the formula until all steps are shown and the result of the completed formula is displayed. The **[Step in]** button is only active when the cell being examined has a formula.
11. Click the **[Restart]** button to begin going through the formula again, or click the **[Close]** button to exit the dialog box.	Choose whether to re-evaluate the formula or close the dialog box.
12. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**

Watch Window

Using the Watch Window

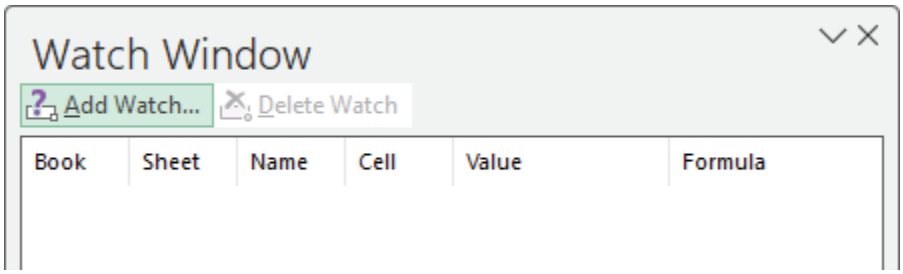
The *Watch Window* allows you to monitor a cell's results when another cell is changed. The Watched cell can be on the same sheet, a different sheet, or even in a different file.

Use the *Watch Window* to add watches to the cells to track changes being made somewhere else in the file without scrolling from one place to another. Once a *Watch* has been added to the dialog, it shows the workbook, worksheet, name of the cell (if the cell has a name), cell address, the cell values, and formulas.

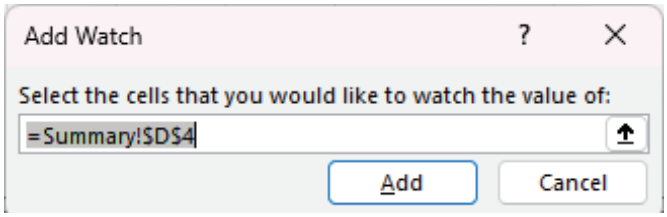
Add as many *Watches* as needed to easily track changes from multiple locations in one dialog. The *Watch Window* can be docked to the top, bottom, or either side of the interface by dragging it to the desired position. It can also be left as a floating dialog, and will remain open until it is closed using the **[Close]** button in the top right corner.

To Add a Cell to the Watch Window

- Click the **[Watch Window]** button in the **Formula Auditing Group** of the ***Formulas Tab*** to open the *Watch Window* dialog box.



- Click the **[Add Watch...]** button in the *Watch Window* dialog box.



- Click on the cell in the worksheet or workbook to watch.
- Click the **[Add]** button.

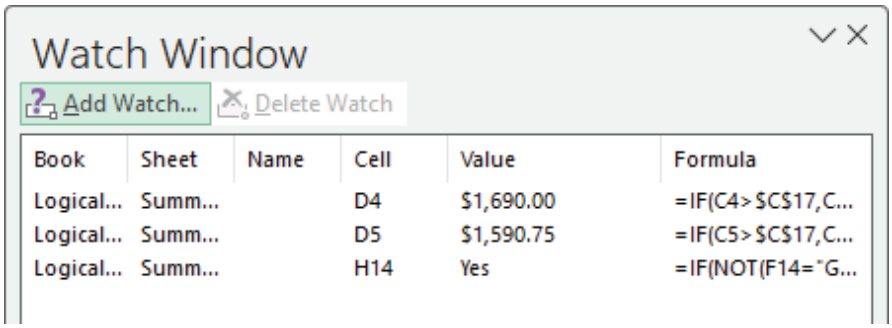
Note

Excel can only have one (1) watch per cell, so each cell will have its own watch when selecting a range.

Watch Window, continued

Using the Watch Window, continued

When the data is altered, the values will update as long as the file and the *Watch Window* dialog box remain open.



To Delete a Cell from the Watch Window

- In the *Watch Window* dialog box, select the ***Watch*** to delete.
- Click the **[Delete Watch]** button.

When the file is closed, the associated cell in the *Watch Window* dialog box will disappear. However, when the file is re-opened, the cell will reappear in the *Watch Window*.

To close the *Watch Window* dialog, click the **[X]** button.

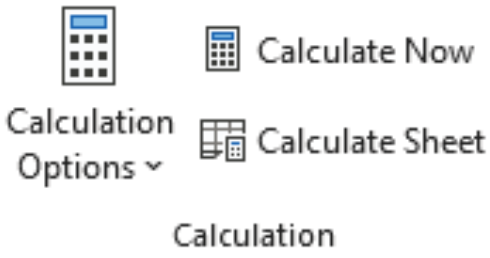
Action 4 - Using the Watch Window

Instructions:	Results/ Comments:
1. Open the **WatchWindow** file.	
2. Save as **My WatchWindow**	
3. Select cell **A67** and type: **Totals**	A new header is added.
4. Select cell **G67** and enter the following formula: **=SUM(G2:G65)** then, press the **[Ctrl + Enter]**.	A total of units sold is added.
5. Use **AutoFill** to populate cell **H67** with a relative copy of the formula.	A total of all sales is added.
6. Select cell **A1**.	**[Ctrl + Home]** key combination takes you back to cell **A1** from anywhere in the spreadsheet.
7. Click the **[Watch Window]** button in the **Formula Auditing Group** on the ***Formulas Tab***.	The *Watch Window* opens.
8. Click the **[Add Watch...]** button.	The *Add Watch* dialog box opens.
9. Highlight the field and select cell **G67** and click the **[Add]** button.	The cell is added to the *Watch Window.* Any relevant information regarding the cell is displayed for tracking proposes.
10. Add another watch for cell **H67**.	Repeat the Add Watch process for cell **H67**.
11. Dock the *Watch Window* above the Formula bar.	Drag the dialog by the title-bar up toward the formula bar. It will snap into position.
12. Select cell **G5** and type: **25** then, press the **[Enter]** key.	Editing a cell's contents will impact the results of the formulas in cells **G67** and **H67**. These changes are shown in the *Watch Window*.
13. Save the file and leave it open.	**[Ctrl + S]**

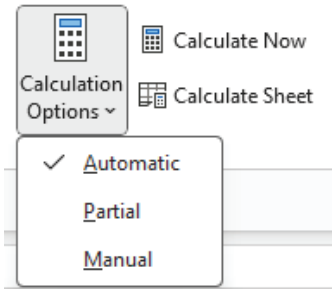
Calculations

By default, *Excel's* calculation setting updates the formula results as the user makes changes. However, having **Excel** automatically update the formula as new data is entered or changed can sometimes be confusing.

Excel can update the calculation once all changes are entered instead of doing it in real time. To change the default calculation setting, click the **[Calculation Options]** button in the **Calculation Group** on the ***Formulas tab.***



Clicking the **[Calculation Options]** button offers three choices:



- ***Automatic:*** This is the default choice. *Excel* recalculates the formula any time data is updated.
- ***Automatic Except for Data Tables:*** If data tables are used in the file, they are not recalculated as data is updated.
- ***Manual:*** The cell will not recalculate until the user runs the calculation.

Changing the Calculation Option

- Click the **[Calculation Options]** button in the **Calculations Group** on the ***Formulas Tab*** and choose *Manual*.
- Change data in the cells that the formula needs to calculate.

Calculations, continued

Changing the Calculation Option, continued

- Click the **[Calculate Now]** button in the **Calculation Group** on the ***Formulas Tab*** to recalculate the formulas on all open sheets and workbooks.

-OR-

- Click the **[Calculate Sheet]** button in the **Calculation Group** on the ***Formulas Tab*** to recalculate all the formulas on the *active sheet only*, not on other open sheets or open workbooks.

The ***Manual*** option and the **[Calculate Now]** button operate independently of the selected cells. Both commands apply to *all* cells, *all* sheets, and *all* files.

Both commands affect the entire *Excel* program and neither command is limited to any specific cell(s), sheet(s), or workbook(s).

Changing the ***Manual*** option on the ribbon is the same as changing it by going to the ***File Tab***, selecting **Options**, clicking the ***Formulas Tab*** on the left, and choosing *Workbook Calculation* under the ***Calculation options*** section.

Instructions:	Results/ Comments:
1. The **My WatchWindow** file should still be open. If not, reopen it.	
2. Close the *Watch Window*.	Click the **[X]** button at the upper right corner of the dialog.
3. Select cell **G65** and type: **25** then, press the **[Enter]** key.	The values in cells **G67:H67** change to reflect the changes to the data.
4. Select cells **G67:H67**.	These cells currently are set to calculate automatically.
5. Click the **[Calculation Options]** button and choose *Manual*.	This turns off the automatic calculation for these cells.
6. Select cell **G65** and type: **50** then, press the **[Enter]** key.	Notice the values in cells **G67:H67** do not update to reflect the changes to the data.
7. Click the **[Calculate Sheet]** button.	The values in cells **G67:H67** are updated.
8. Try changing the value in cell **G65** again.	The values in cells **G67:H67** are not updated.
9. Recalculate the sheet.	The values in cells **G67:H67** are updated.
10. Save and close the file.	**[Ctrl + S]** and **[Ctrl + W]**