SEMESTER – 7

SUBJECT CODE: BAF-701
SUBJECT TITLE: BUSINESS ACCOUNTING

Table of Contents

Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 Basic Concepts

1.1 Microsoft Excel (2013) New Features

1.1.1 New Functions in Excel

When a new version of Microsoft Office is released, sometimes Excel gets lots of new features and other times it gets very few new features. In the case of Office 2013, Excel got quite a few new features.

Here's a quick summary of what's new in Excel 2013, relative to Excel 2010:

Cloud storage: Excel is tightly integrated with Microsoft's SkyDrive web-based storage.

Support for other devices: Excel is available for other devices, including touch-sensitive devices such as Windows RT tablets and Windows phones.

New aesthetics: Excel has a new " flat" look and displays an (optional) graphic in the title bar. The default color scheme is white, but you can choose from two other color schemes (light easy to filter data by dates.

Quick Analysis: Quick Analysis provides single click access to various data analysis tools.

Enhances chart formatting: Modifying charts is significantly easier.

New worksheet functions: Excel 2013 supports dozens of new worksheet functions.

Backstage: The Backstage screen has been recognized and is easier to use.

New add-ins: Three new add-ins are included (for Office Professional Plus only): Power pivot, power View, and Inquire.

New types of assistance: Excel provides recommended pivot tables and recommended charts.

Fill Flash: Fill Flash is a new way to extract (by example) relevant data from text strings. You can also use this features to combine data in multiple columns.

Support for Apps for Office: You can download or purchase apps that can be embedded in a workbook file.

New Slicer option: The slicer feature, introduced in Excel 2010 for use with pivot tables, has been expanded and now works with tables.

1.1.2 Using New Chart Tools

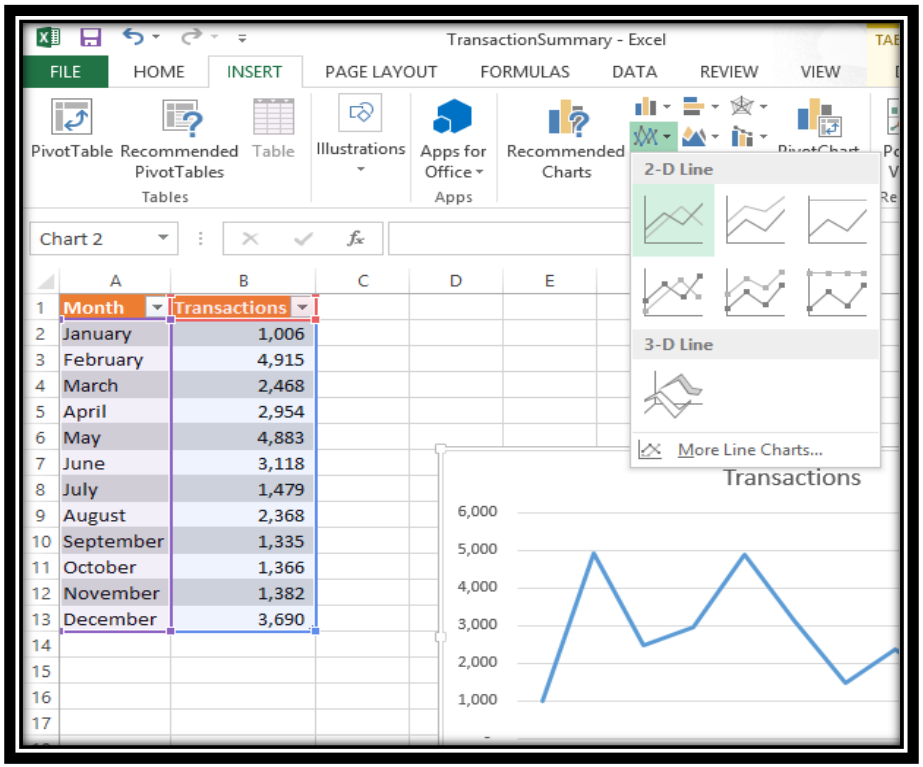
Creating a chart: to present your excel data graphically, select the cells you want to summarize, click the insert tab, and then use the controls in the Chary gallery to select the chart type that's best for your data and the message you want to get across.

The cells with the data to be represented in the chart are part of one or more data series. A series is a collection of related data, such as all sales for a particular product or the sales for each day of a month. A bar chart could contain just one series; a line chart, which might display monthly sales for several years, cam have many series.

You can create a chart manually, or you can create a chart that the program recommends. The recommended Charts gallery, which is new in EXCEL 2013, displays a set of charts that you can create based on you data. All you need to do is click the chart that you want and confirm your choice. In either case, you can then easily change a chart's appearance to suit your preferences.

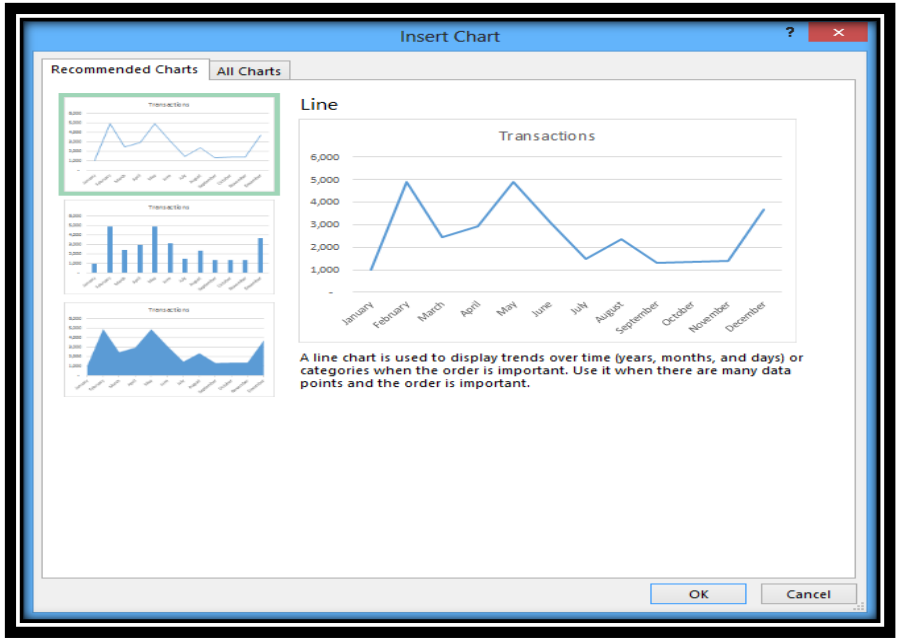
Create a chart

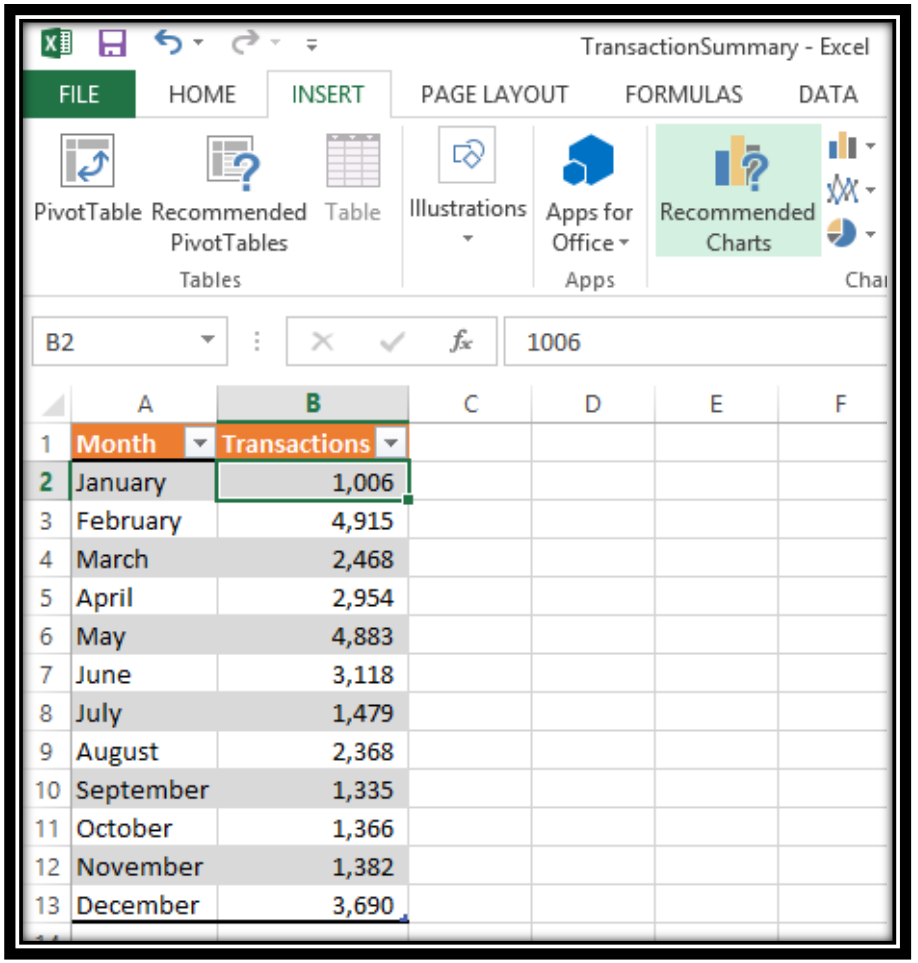
1. Click a cell in the data list that you want to summarize.
2. Click the insert tab.
3. Click the type of chart that you want to create.
4. Click the chart subtype that you want to use.



Create a recommended chart:

1. Click a cell in the data list that you want to summarize.
2. Click the insert tab.
3. Click Recommended Charts.
4. Click the chart that you want to create.
5. Click OK.





Change a chart's layout and style

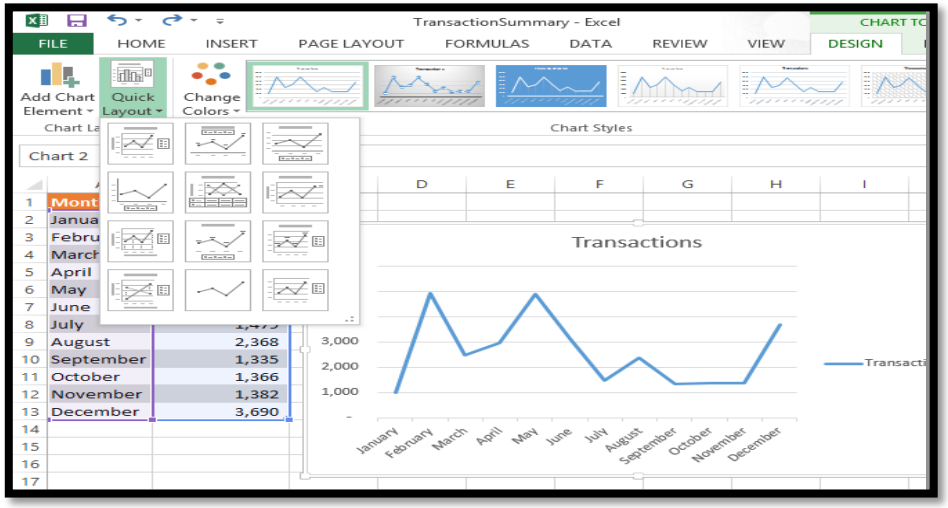
Charts summarize data visually, so every chart has a particular arrangement and presentation of its elements. The overall arrangement of a chart's elements is its *layout*, while the overall appearance of the chart's elements is its style. You can apply

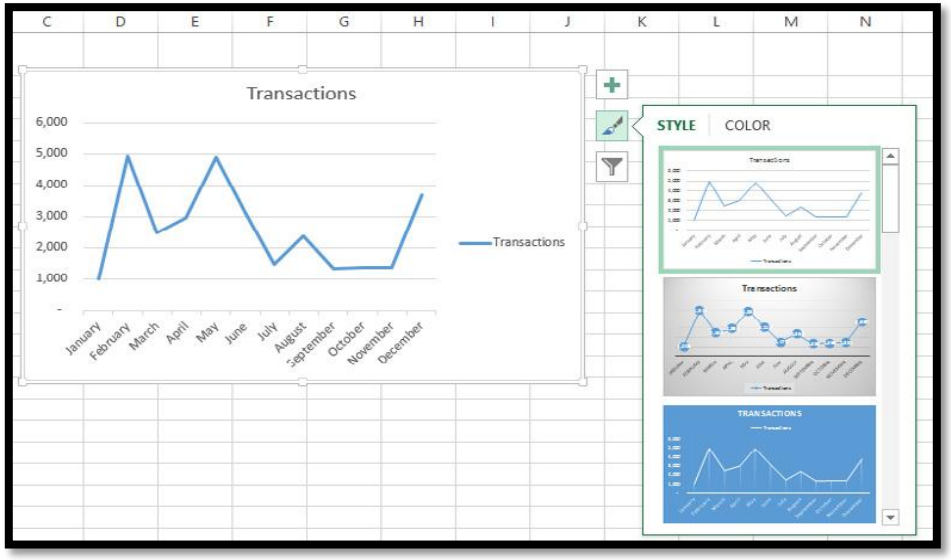
Change a chart's layout

1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Quick Layout.
4. Click the layout that you want to apply.

Change a chart's style

1. Click the chart that you want to change.
2. Click the Chat's Styles button.
3. In the chart styles gallery that appears, click the new style.





2 Advance Formulas

1	ABS	26	DCOUNT	51	HLOOKUP	76	MAX	101	RAND	126	T
2	AND	27	DCOUNTA	52	HOUR	77	MEDIAN	102	RANDBETWEEN	127	TEXT
3	AVERAGE	28	DEC2BIN	53	IF	78	MID	103	RANK	128	TIME
4	BIN2DEC	29	DEC2HEX	54	INDEX	79	MIN	104	REPLACE	129	-Timesheet
5	CEILING	30	DELTA	55	INDIRECT	80	MINUTE	105	REPT	130	TIMEVALUE
6	CELL	31	DGET	56	INFO	81	MINVERSE	106	RIGHT	131	TODAY
7	CHAR	32	DMAX	57	INT	82	MMULT	107	ROMAN	132	TRANSPOSE
8	CHOOSE	33	DMIN	58	ISBLANK	83	MOD	108	ROUND	133	TREND
9	CLEAN	34	DOLLAR	59	ISERR	84	MODE	109	ROUNDDOWN	134	TRIM
10	CODE	35	DSUM	60	ISERROR	85	MONTH	110	ROUNDUP	135	TRUNC
11	COMBIN	36	EDATE	61	ISEVEN	86	MROUND	111	SECOND	136	TYPE
12	CONCATENATE	37	EOMONTH	62	ISLOGICAL	87	N	112	SIGN	137	UPPER
13	CONVERT	38	ERROR.TYPE	63	ISNA	88	NA	113	SLN	138	VALUE
14	CORREL	39	EVEN	64	ISNONTEXT	89	NETWORKDAYS	114	SMALL	139	VAR
15	COUNT	40	EXACT	65	ISNUMBER	90	NOT	115	STDEV	140	VARP
16	COUNTA	41	FACT	66	ISODD	91	NOW	116	STDEVP	141	VLOOKUP
17	COUNTBLANK	42	FIND	67	ISREF	92	ODD	117	SUBSTITUTE	142	WEEKDAY
18	COUNTIF	43	FIXED	68	ISTEXT	93	OR	118	SUBTOTAL	143	WORKDAY
19	DATE	44	FLOOR	69	LARGE	94	PERMUT	119	SUM	144	YEAR
20	DATEDIF	45	FORECAST	70	LCM	95	PI	120	SUM_as_Running_Total	145	YEARFRAC
21	DATEVALUE	46	FREQUENCY	71	LEFT	96	POWER	121	SUM_using_names		
22	DAVERAGE	47	GCD	72	LEN	97	PRODUCT	122	SUM_with_OFFSET		
23	DAY	48	GESTEP	73	LOOKUP (vector)	98	PROPER	123	SUMIF		
24	DAYS360	49	GROWTH	74	LOWER	99	QUARTILE	124	SUMPRODUCT		
25	DB	50	HEX2DEC	75	MATCH	100	QUOTIENT	125	SYD		

3 Working with Lists

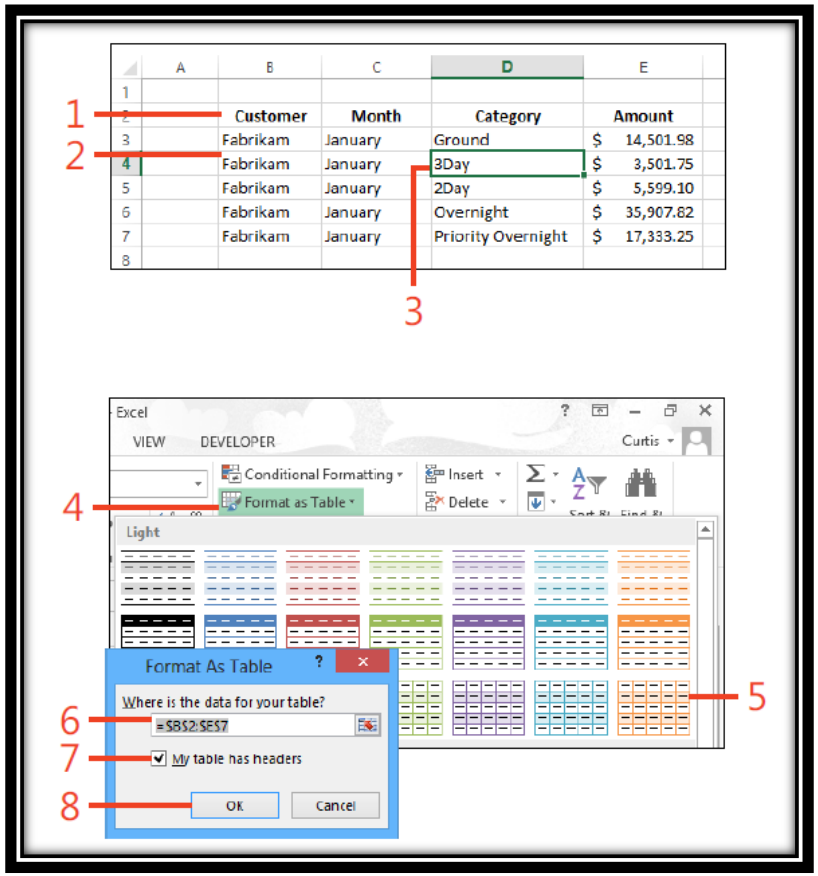
3.1 Converting a List to a Table

One popular way to maintain data in Excel is by creating a data list. A list describes one type of object, such as orders, sales, or contact information. Structurally, a list consists of a header row, which contains labels describing the data in each column, and data rows, which contain data about a particular instance of the list's subject. For example, if you use a list to store your customer's contact information, you could have a separate column for the customer's first name, last name, street address, city, state, postal code, and telephone number. Each row in a list would contain a particular customer's information.

In Excel 2013, you can store your data lists in an Excel table. An Excel table is an object that you can refer to in your formulas and use to summarize your data.

Create an Excel table

1. Type your table headers in a single row.
2. Type your first data row directly below the header row.
3. Click any cell in the range in which you want to create a table.
4. On the Home tab, click Format as Table.
5. Click the table style to use.
6. Verify that Excel identified the data range correctly.
7. If your table has headers, select the My Table Has Headers check box.
8. Click OK.

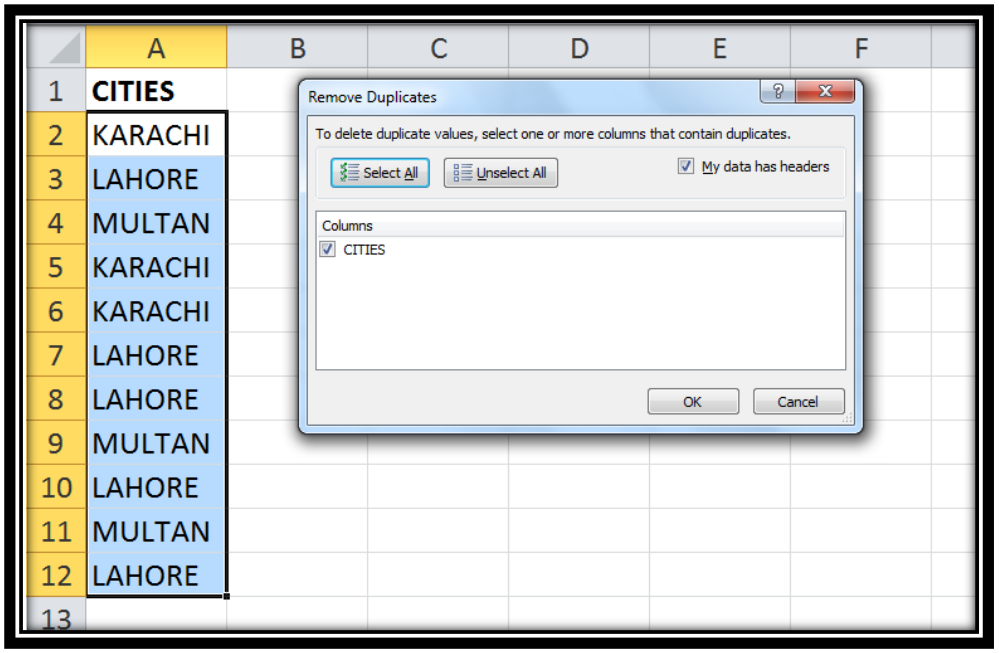


3.2 Removing Duplicates from a List

If data in a table was compiled from multiple sources, the table may contain duplicate items. Most of the time, you want to eliminate the duplicates. In the past, removing duplicate data was essentially a manual task, but it's very easy if the data is in a table.

Removing Duplicates

1. Start by selecting any cell in your table.
2. Then choose Table Tools ➪ Design ➪ Tools ➪Remove Duplicates. Excel responds with Remove Duplicates dialog box shown in Figure
3. The dialog box lists all the columns in your table. Place a check mark next to the columns that you want to be included in the duplicate search. Most of the time, you'll want to select all the columns, which is the default.
4. Click OK, and Excel weeds out the duplicate rows and displays a message that tells you how many duplicates it removed.
5. When you select all columns in the Remove Duplicates dialog box, Excel will delete a row only if the content of every column is duplicated.
6. In some situations, you may not care about matching some columns, so you would deselect those columns in the Remove
7. Duplicates dialog box. When duplicate rows are found, the first row is kept and subsequent duplicate rows are deleted.



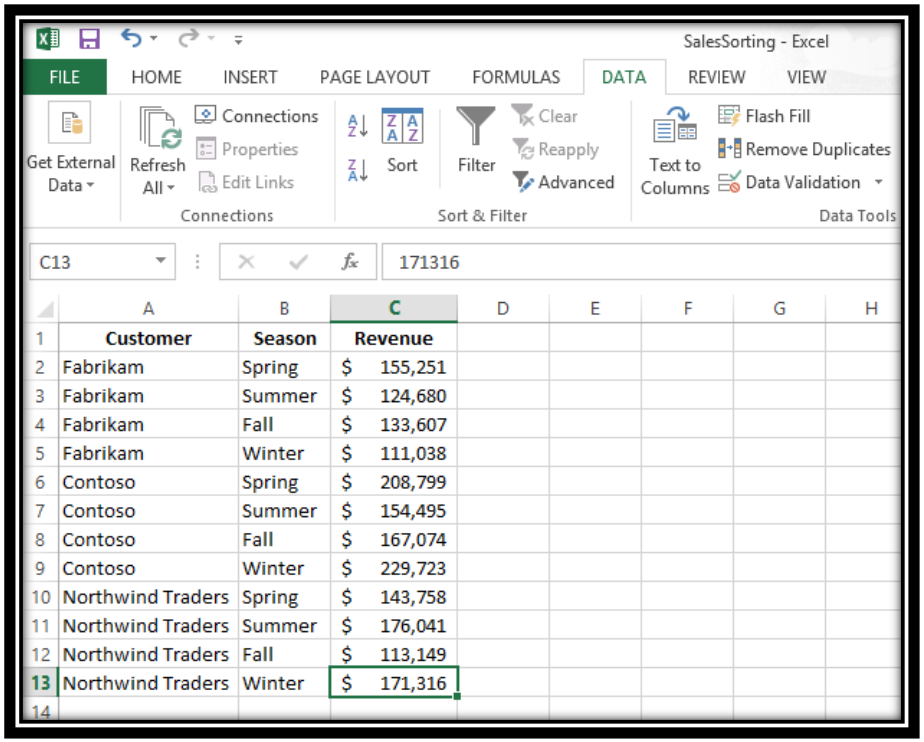
3.3 Sorting Data in a List

Sorting worksheet data

You can sort a group of rows in a worksheet in a number of ways, but the first step is to identify the column that includes the values by which to sort the rows. After you select the column by which you want to sort the worksheet, you can choose whether to display the sorted values in ascending or descending order. For example, if you have a list of products that your company sells with each product's sales in the same row, you could sort the worksheet by the contents of the sales column in descending order to discover which products generated the most revenue for your company. You could sort the worksheet rows in ascending order to put the lowest-revenue products at the top of the list. If you want to sort by the contents of more than one column, you can create a multicolumn sort. One handy use for a multicolumn sort is to sort your products by category and then by total sales.

Sort data in ascending or descending order

1. Click any cell in the column by which you want to sort your data.
2. Click the Data tab.
3. Follow either of these steps: **a** Click the Sort Ascending button in the Sort & Filter group on the ribbon.
4. Click the Sort Descending button in the Sort & Filter group on the ribbon.

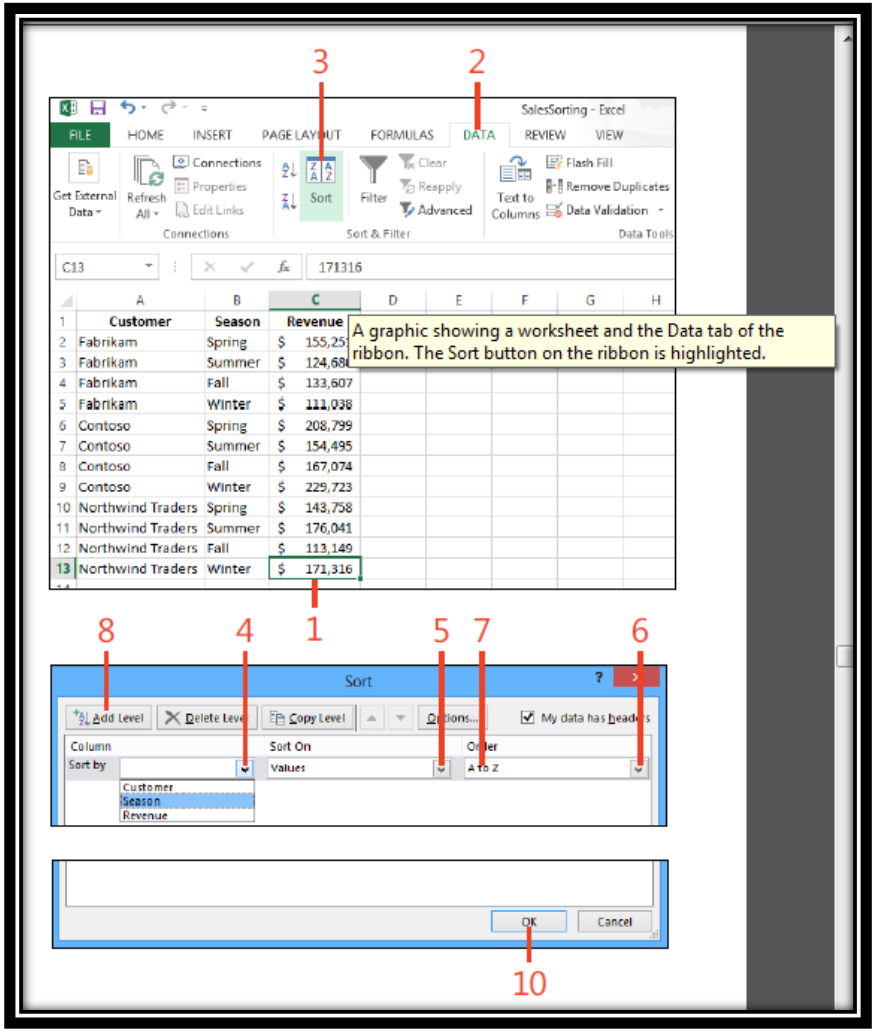


	A	B	C	D	E	F	G	H
1	Customer	Season	Revenue					
2	Fabrikam	Spring	$ 155,251					
3	Fabrikam	Summer	$ 124,680					
4	Fabrikam	Fall	$ 133,607					
5	Fabrikam	Winter	$ 111,038					
6	Contoso	Spring	$ 208,799					
7	Contoso	Summer	$ 154,495					
8	Contoso	Fall	$ 167,074					
9	Contoso	Winter	$ 229,723					
10	Northwind Traders	Spring	$ 143,758					
11	Northwind Traders	Summer	$ 176,041					
12	Northwind Traders	Fall	$ 113,149					
13	Northwind Traders	Winter	$ 171,316					
14								

Working with Lists

Create a multicolumn sort

1. Select a cell in the data list or Excel table you want to sort.
2. Click the Data tab.
3. Click Sort.
4. Click the Sort By down arrow and then click the first column by which you want to sort.
5. Click the Sort On down arrow and then click the criteria by which you want to sort.
6. Click the Order down arrow.
7. Select the A to Z item or the Z to A item to indicate the order in which the column's values should be sorted.
8. Click Add Level.
9. If necessary, repeat steps 4–8 to set the columns and order for additional sorting rules.
10. Click OK.

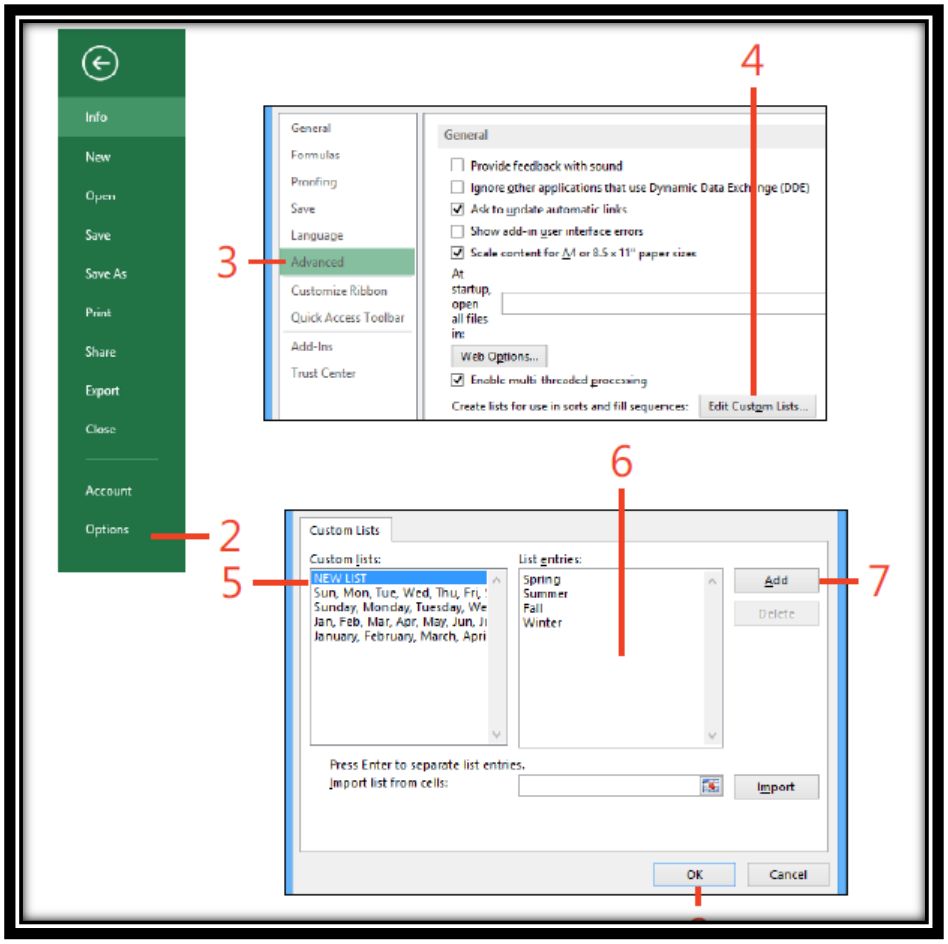


3.4 Creating a custom sort list

When you sort column data, Excel sorts numbers according to their values and words in alphabetical order, but that pattern doesn't work for some sets of values. For example, sorting the months of the year in alphabetical order puts February in front of January. To avoid those errors, Excel comes with four custom lists: days of the week, abbreviations for days of the week, months, and abbreviated month names. You can have Excel sort based on those lists, or if you want to create a custom list of your own, you can do so. Because the day and abbreviated day name lists begin with Sunday, you could create a new list beginning with Monday to reflect your business schedule.

Define a custom list of values

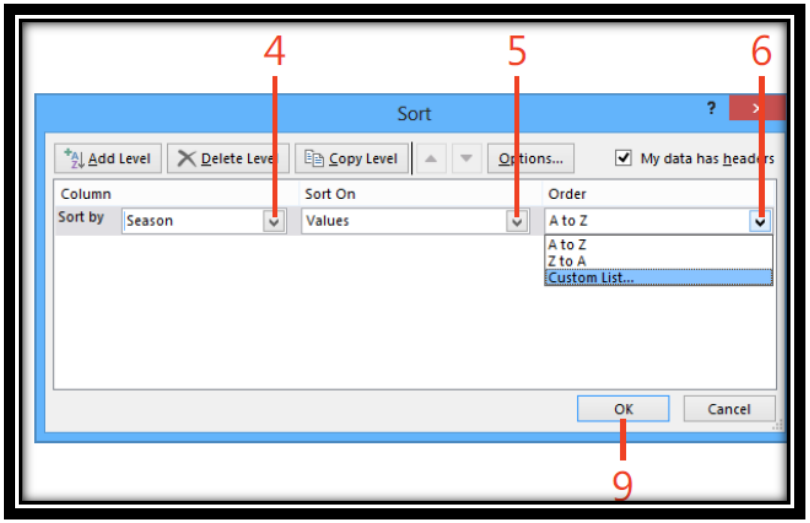
1. Click the File tab.
2. Click Options.
3. Click Advanced.
4. In the General section of the Advanced page, click Edit Custom Lists.
5. Click New List.
6. Type the custom list that you want. Separate each entry by pressing Enter.
7. Click Add.
8. Click OK twice to close the Custom Lists dialog box and the Excel Options dialog box, respectively.

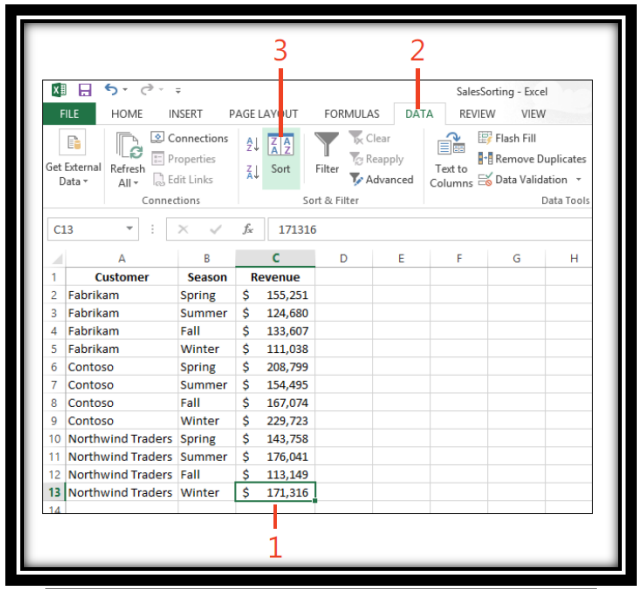


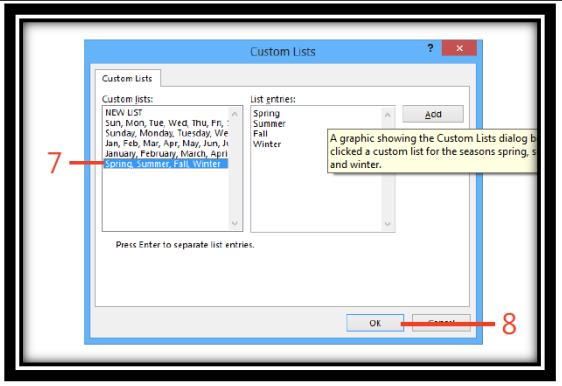
3.5 Sort using a custom sort list

Sort using a custom list

1. Click any cell in the list that you want to sort.
2. Click the Data tab.
3. Click Sort.
4. Click the Sort By down arrow and then click the column that you want to sort by.
5. Click the Sort On down arrow and then click the criteria that you want to sort by.
6. Click the Order down arrow and then click Custom List.
7. Click a custom list.
8. Click OK to close the Custom Lists dialog box.
9. Click OK to sort the data list.





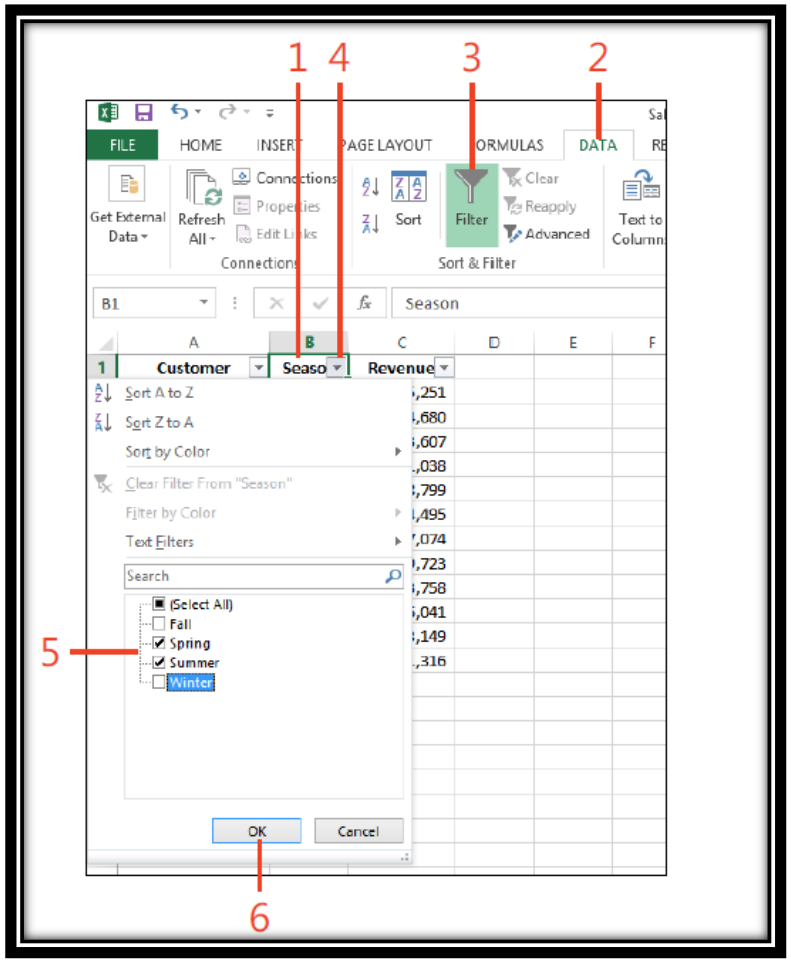


3.6 Filtering Data in a List

An important aspect of working with large amounts of data is the ability focus in on the most important data in a worksheet, whether that data represents the best 10 days of sales in a month or slow-selling product lines that you might need to reevaluate. In Excel 2013, you have a number of powerful, flexible techniques that you can use to limit the data displayed in your worksheet. One of those techniques is to filter the contents of a worksheet. You can filter a data list or Excel table by selecting individual values, creating a rule, or by searching for values within a field.

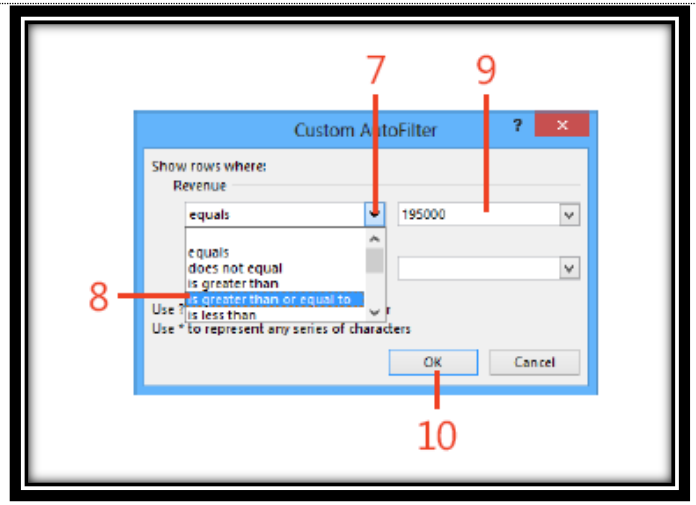
Create a selection filter

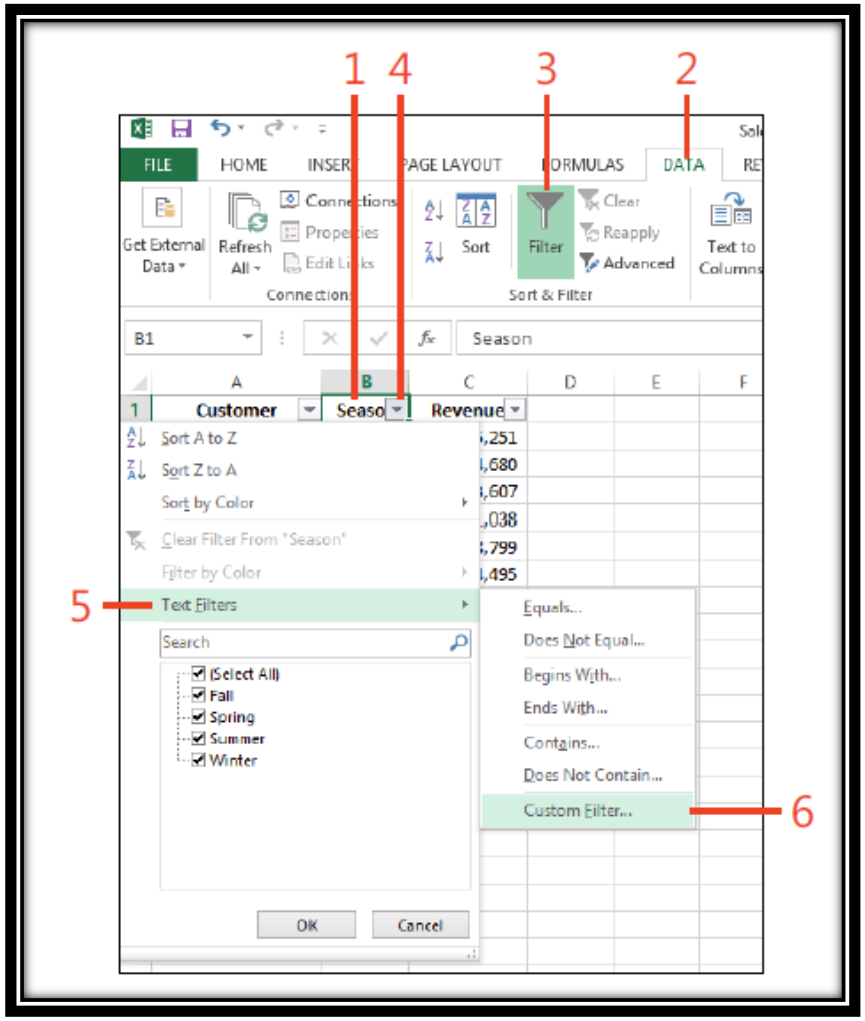
1. Click any cell in the range that you want to filter.
2. Click the Data tab.
3. Click Filter.
4. Click the filter arrow for the column by which you want to filter your worksheet.
5. Select the check boxes next to the values by which you want to filter the list.
6. Click OK.



Create a filtering rule

1. Click any cell in the list that you want to filter.
2. Click the Data tab.
3. If necessary, click Filter to display the filter arrows.
4. Click the filter arrow of the column for which you want to create a custom filter.
5. Point to Text Filters.
6. Click Custom Filter.
7. Click the Comparison Operator down arrow.
8. Click the comparison that you want to use.
9. Type the value with which you want to compare the values in the selected column.
10. Click OK.





3.7 Creating an advanced filter

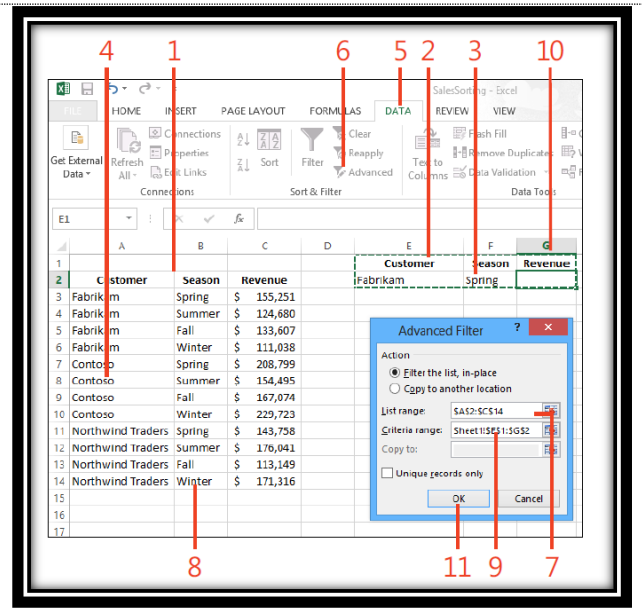
When you create a filter using AutoFilter, you can create complex rules to filter the contents of the worksheet. The limitation is that the rules used to filter the worksheet aren't readily discernible. If you want the rules used to filter a column's values to be displayed in the body of the worksheet, you can write each rule in a cell and identify those cells so that Excel knows how to filter the worksheet. If you ever want to change the rules used to filter your data, all you need to do is change a rule and reapply the filter.

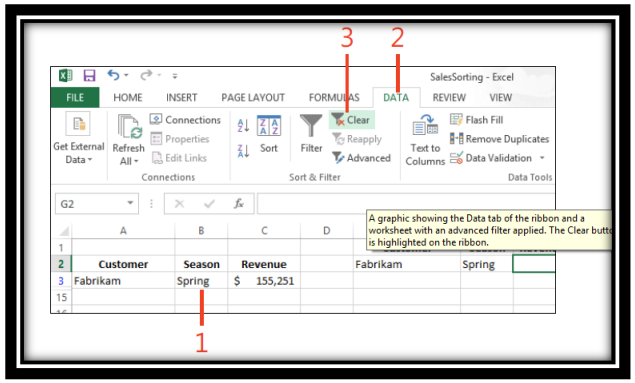
Build an advanced filter

1. Copy the column titles of the list that you want to filter.
2. Paste the titles into another spot on your workbook.
3. Under their respective titles, type the criteria that you want the filter to meet.
4. Select a cell in the list that you want to filter.
5. Click the Data tab.
6. In the Sort & Filter group on the ribbon, click Advanced.
7. Click the List Range box.
8. If necessary, select the entire list that you want to filter, including the column headers.
9. Click the Criteria Range box.
10. Select the cells on which you want to base the filter, including the column headers.
11. Click OK.

Remove an advanced filter

1. Select a cell in the range from which you want to remove the filter.
2. Click the Data tab.
3. In the Sort & Filter group on the ribbon, click Clear.



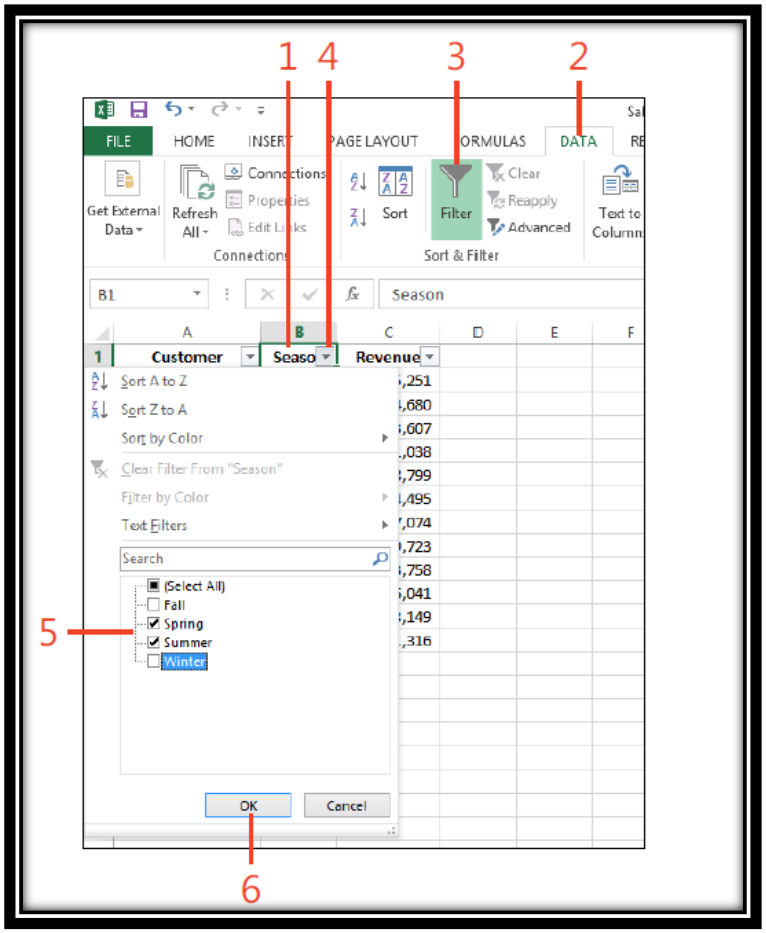


3.8 Filtering Data in a List

An important aspect of working with large amounts of data is the ability focus in on the most important data in a worksheet, whether that data represents the best 10 days of sales in a month or slow-selling product lines that you might need to reevaluate. In Excel 2013, you have a number of powerful, flexible techniques that you can use to limit the data displayed in your worksheet. One of those techniques is to filter the contents of a worksheet. You can filter a data list or Excel table by selecting individual values, creating a rule, or by searching for values within a field.

Create a selection filter

1. Click any cell in the range that you want to filter.
2. Click the Data tab.
3. Click Filter.
4. Click the filter arrow for the column by which you want to filter your worksheet.
5. Select the check boxes next to the values by which you want to filter the list.
6. Click OK.

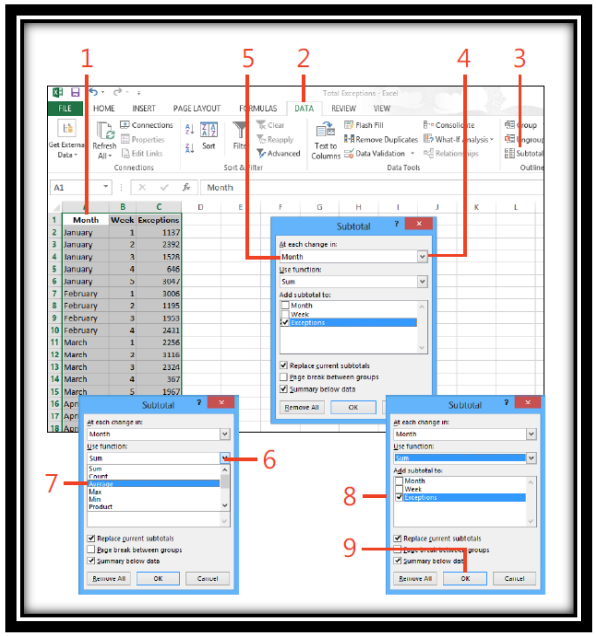


3.9 Adding Subtotals to a List

You frequently need to organize the data in an Excel worksheet by one or more criteria. For example, you might have a worksheet in which you list yearly sales for each product that you offer, with the products broken down by category. If your data is organized this way, you can have Excel calculate a subtotal for each category of products. When you create a subtotal, you identify the cells with the values to be calculated and the cells that identify the change from one category to the next. Excel updates the subtotal and grand total for you if the value of any cell changes.

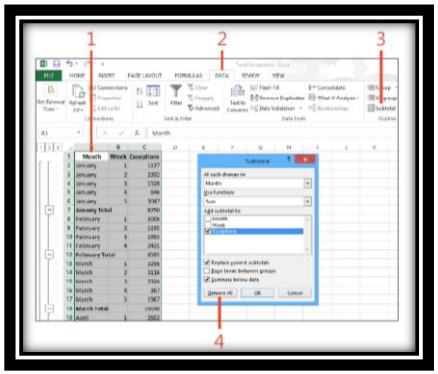
Create a subtotal

1. Click any cell in the range that you want to subtotal.
2. Click the Data tab.
3. In the Outline group, click Subtotal.
4. Click the At Each Change In down arrow.
5. Click the value on which you want to base the subtotals.
6. Click the Use Function down arrow.
7. Click the subtotal function that you want to use.
8. Select which columns should have subtotals calculated.
9. Click OK.



Remove a subtotal

1. Click any cell in the subtotaled range.
2. Click the Data tab.
3. Click Subtotal.
4. Click Remove All.



3.10 Grouping and Ungrouping Data in a List

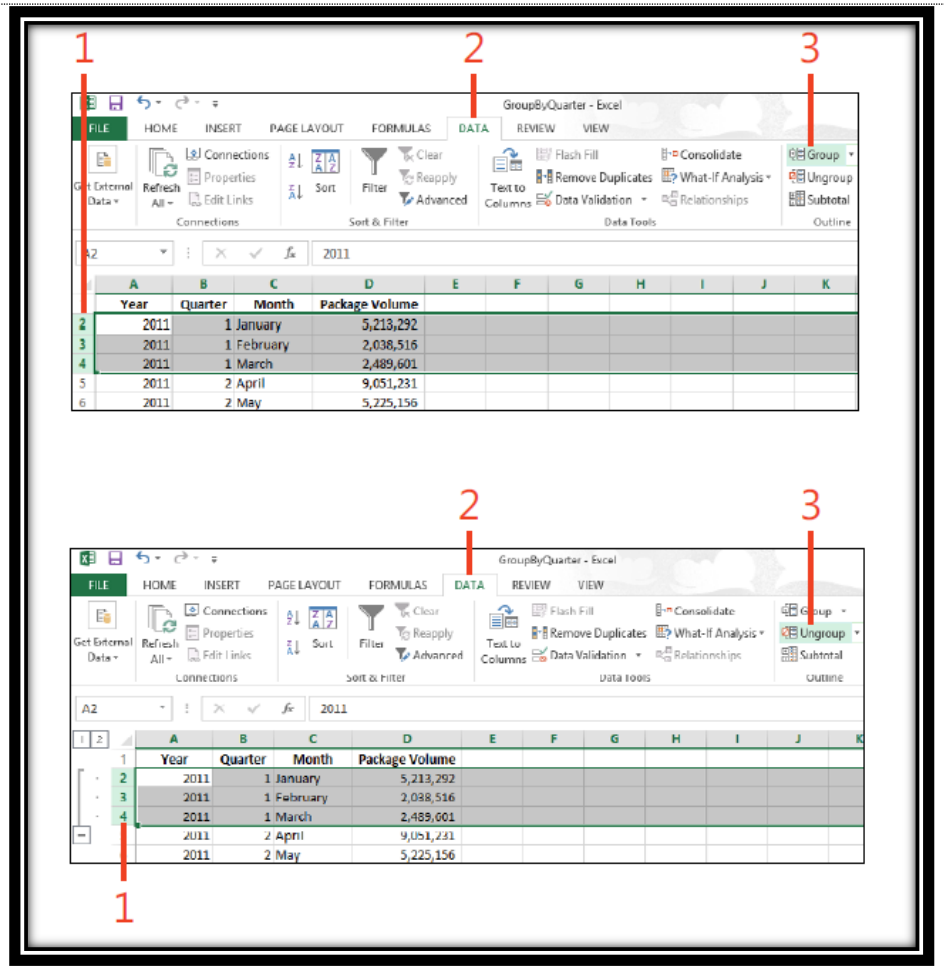
As you develop worksheets, you'll probably try to keep similar entries together. For example, if you create a worksheet that lists all the products you sell, you can group the products by category. In the case of a garden supply store, all the tools can be together, then the furniture, and then a new category for each type of plant. You can always remove a group, or the entire outline, when you no longer need it.

Group worksheet rows

1. Select the rows or columns that you want to group.
2. Click the Data tab.
3. In the Outline group on the ribbon, click Group.

Ungroup worksheet rows

1. Select the rows or columns that you want to ungroup.
2. Click the Data tab.
3. In the Outline group on the ribbon, click Ungroup.



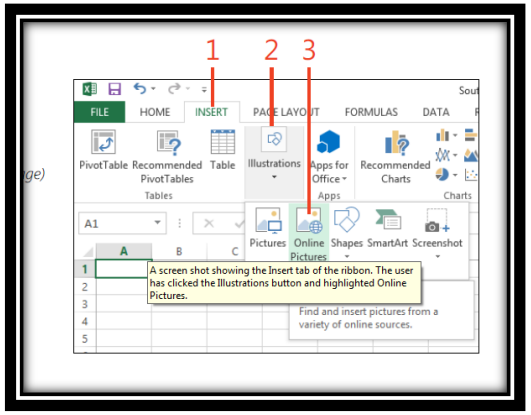
4 Working with Illustrations

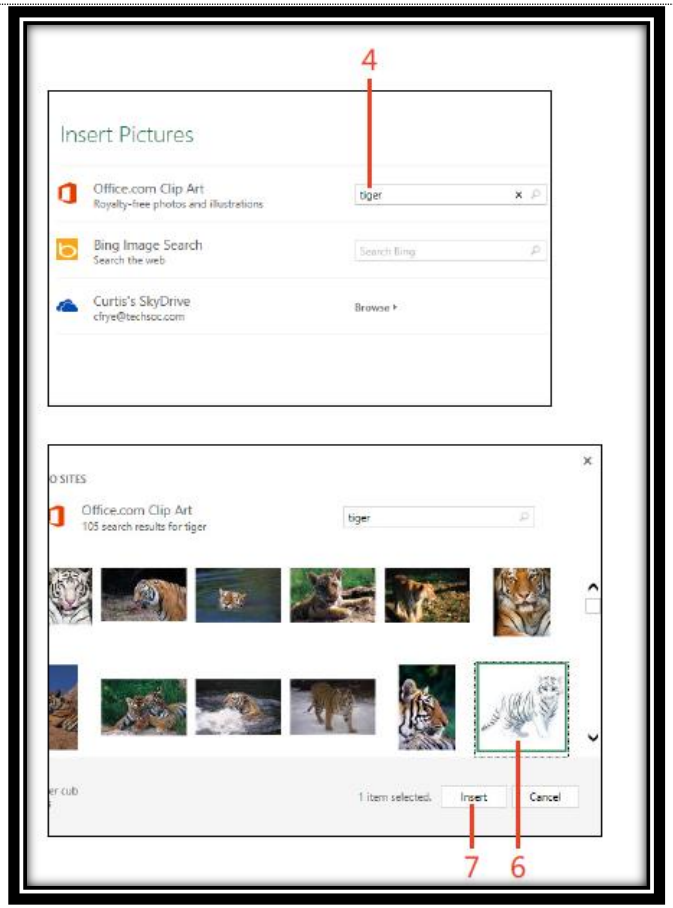
4.1 Working with Clip Art

If you want to include a worksheet in a presentation that you give to your colleagues or in a report that you make available on your company's website, you can add images from the clip art collection that comes with Office to accent your worksheet. You can manage your clip art by using the Clip Art task pane, whether that means searching for clip art on your computer or on the web.

Add clip art

1. Click the Insert tab.
2. Click Illustrations.
3. Click Online Pictures.
4. Type a description of the clip art that you want.
5. Press Enter.
6. Click the clip art that you want to insert.
7. Click Insert.



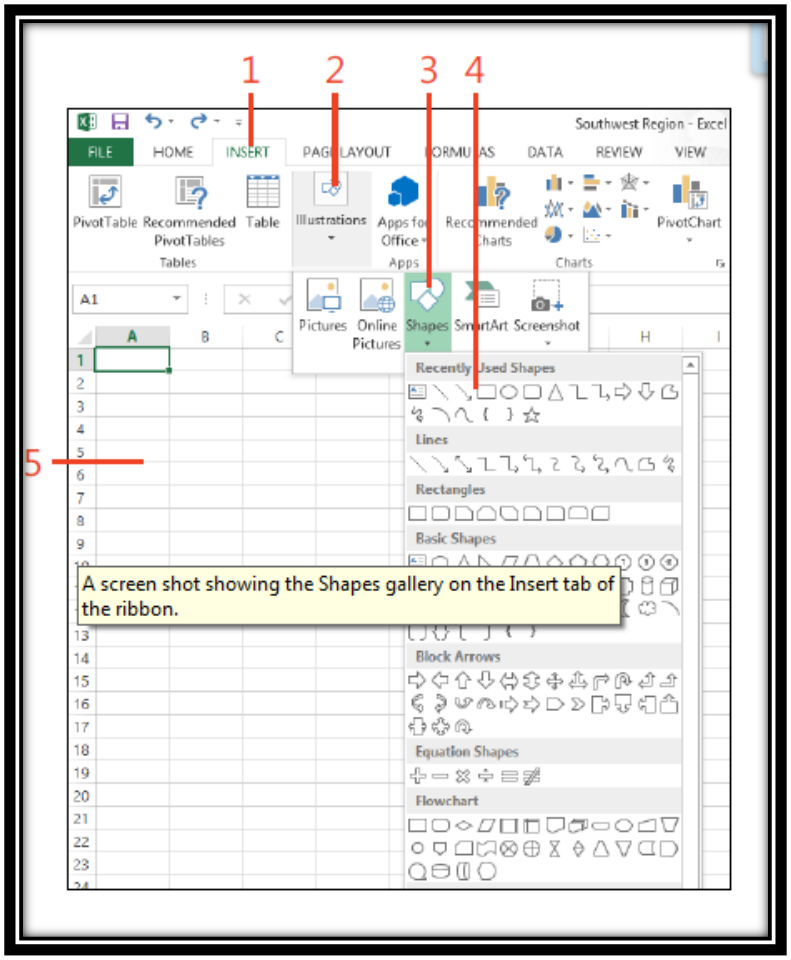


4.2 Adding drawing objects to a worksheet

Adding graphics or pictures of items described in your worksheets makes it easier to understand the data in those worksheets, but you're not limited to adding images from other sources. If you want, you can add drawing objects to your worksheets. For example, you can add shapes such as rectangles and lines to your worksheets, both to add visual interest and to convey additional information about your worksheet's contents.

Add a simple shape

1. Click the Insert tab.
2. Click Illustrations.
3. Click Shapes.
4. Click the button representing the shape that you want to add.
5. Drag to where you want your shape to appear.



4.3 Adding graphics to worksheets

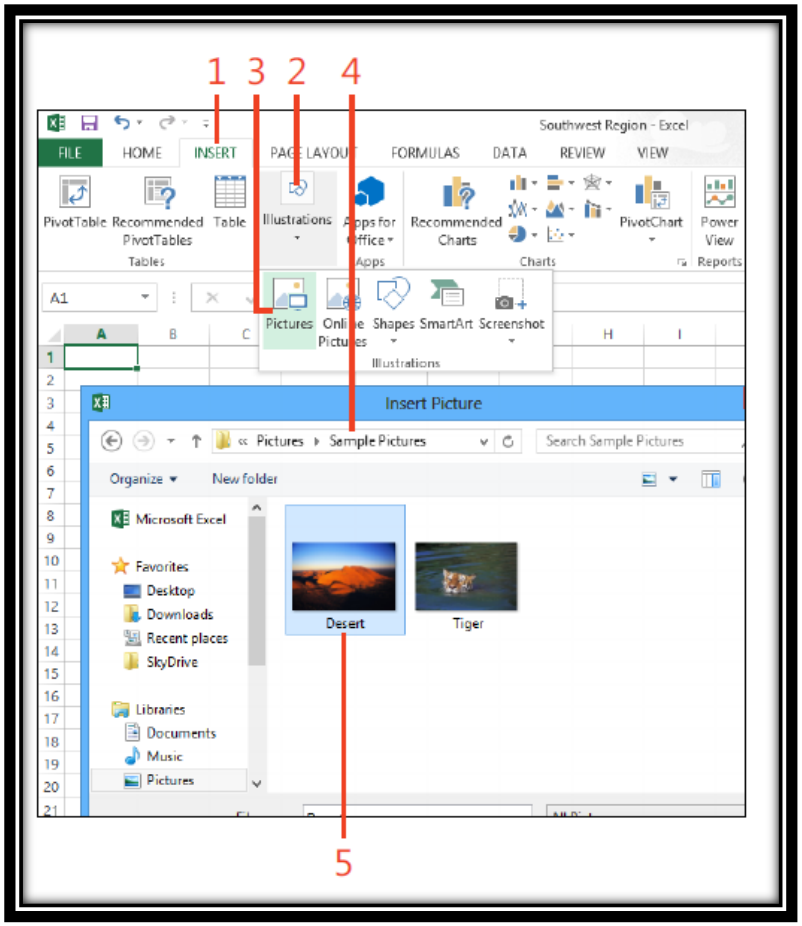
After you create a worksheet, you can add graphics to enhance the data on the worksheet. If your worksheet details the sales of a specific product, you can add a photograph of the product to the worksheet. Adding a visual representation of the item described in the worksheet makes the results you offer much more memorable. Not only do you and your colleagues have numbers to work with, you also have something solid with which to associate the data, enhancing recall.

Add a picture

1. Click the Insert tab.
2. Click Illustrations.
3. Click Pictures.
4. Navigate to the folder with the picture that you want to insert.
5. Double-click the picture that you want to insert.

Delete a picture

1. Click the picture that you want to delete.
2. Press the Delete key.

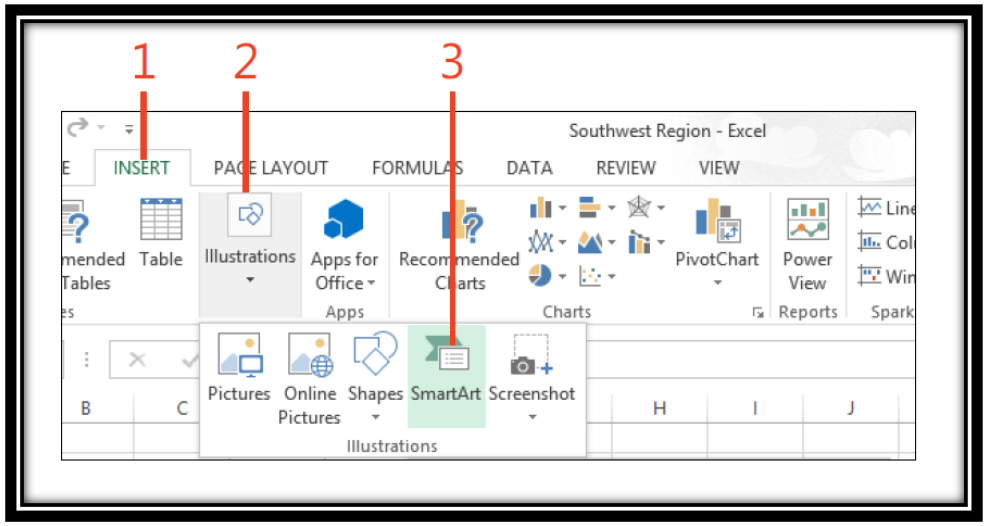


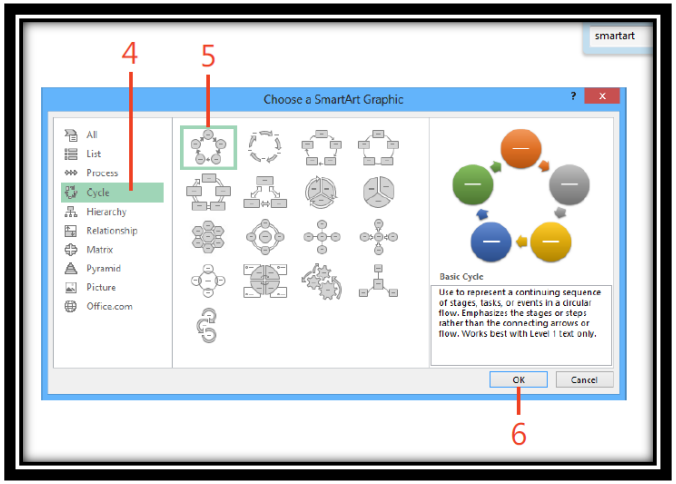
4.4 Inserting and changing a diagram (Working with Smart Art)

A number of diagram types are probably familiar to most business people: the pyramid, which shows a hierarchical relationship; the target, which uses a ring of concentric circles to display the approach to a target; the Venn diagram, which shows the intersection of items in various sets; and the cycle, which displays the steps in a repeating process. Rather than create these diagrams on your own, you can have Excel create the basic diagram, and then you can fill in the details. After you create the diagram, you can apply an AutoFormat to alter its appearance.

Insert a diagram

1. Click the Insert tab.
2. Click Illustrations.
3. Click SmartArt.
4. Click the category of SmartArt that you want to create.
5. Click the specific diagram that you want to add to your worksheet.
6. Click OK.





5 Visualizing Your Data

5.1 Inserting Charts

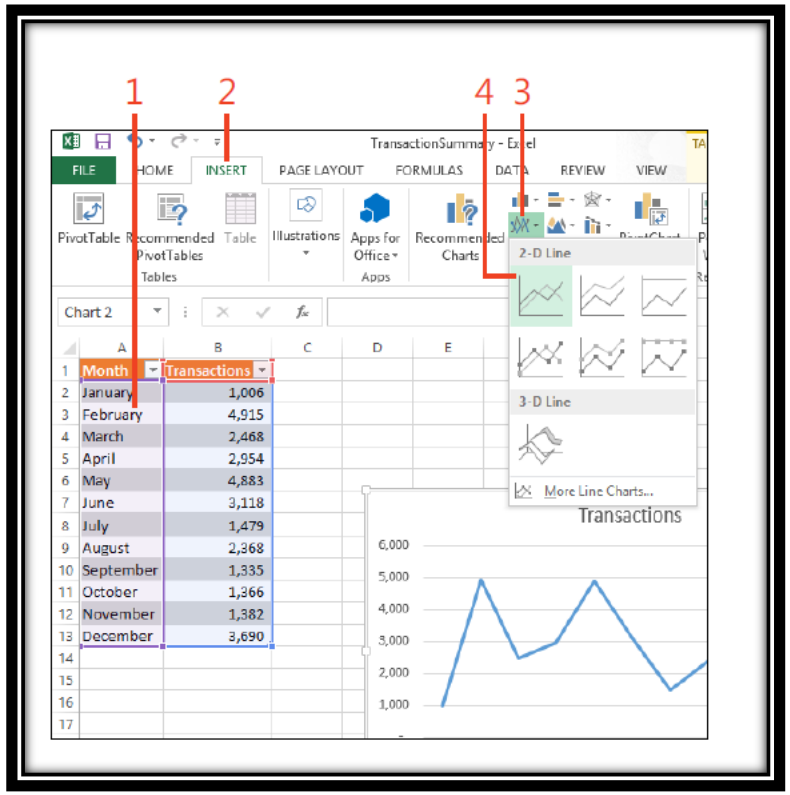
To present your Excel data graphically, select the cells you want to summarize, click the Insert tab, and then use the controls in the Chart gallery to select the chart type that's best for your data and the message you want to get across.

The cells with the data to be represented in the chart are part of one or more data series. A series is a collection of related data, such as all sales for a particular product or the sales for each day of a month. A bar chart could contain just one series; a line chart, which might display monthly sales for several years, can have many series.

You can create a chart manually, or you can create a chart that the program recommends. The Recommended Charts gallery, which is new in Excel 2013, displays a set of charts that you can create based on your data. All you need to do is click the chart that you want and confirm your choice. In either case, you can then easily change a chart's appearance to suit your preferences.

Create a chart

1. Click a cell in the data list that you want to summarize.
2. Click the Insert tab.
3. Click the type of chart that you want to create.
4. Click the chart subtype that you want to use.



5.2 Changing the Layout of a Chart

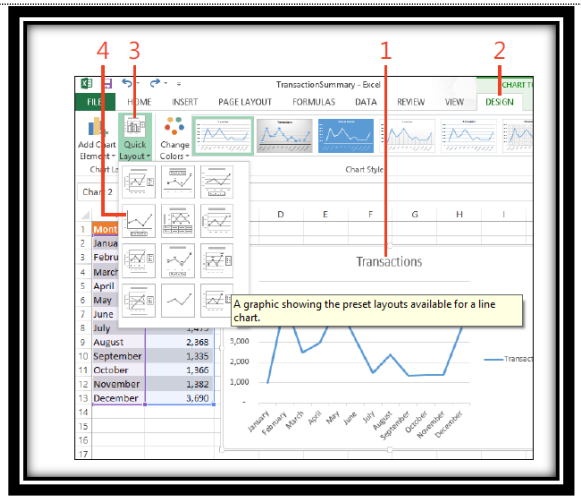
Charts summarize data visually, so every chart has a particular arrangement and presentation of its elements. The overall arrangement of a chart's elements is its *layout*, while the overall appearance of the chart's elements is its *style*. You can apply predefined layouts and styles to your charts. As with any formatting that you apply, you can always fine-tune your choices later.

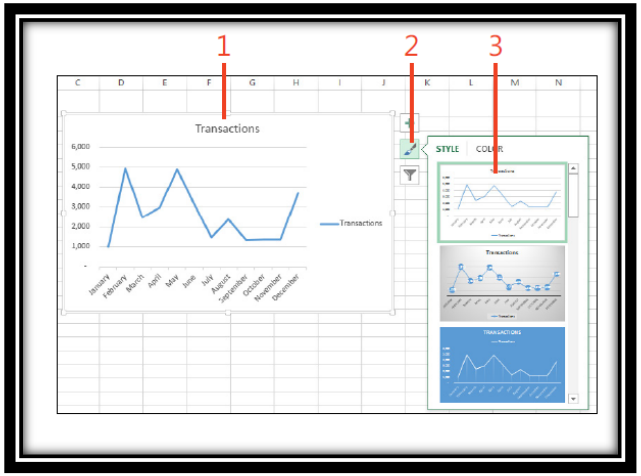
Change a chart's layout

1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Quick Layout.
4. Click the layout that you want to apply.

Change a chart's style

1. Click the chart that you want to change.
2. Click the Chart Styles button.
3. In the Chart Styles gallery that appears, click the new style.



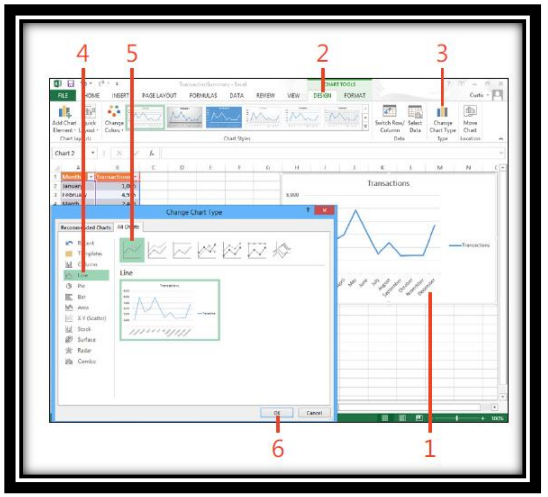


5.3 Changing the Style of a Chart

After you create a chart, you can change any part of its appearance, including the chart's type! If you display monthly sales data as a series of columns and decide that you want to show the data as a line rising and falling as it moves from month to month, you can change the chart's type at will. You can also change the color, font, and other properties of any chart element. If you want your chart's title to be displayed in your company's official font, you can format the title easily.

Change a chart's type

1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Change Chart Type.
4. Click the type of chart that you want.
5. Click the chart subtype that you want.
6. Click OK.



Change the formatting of a chart element

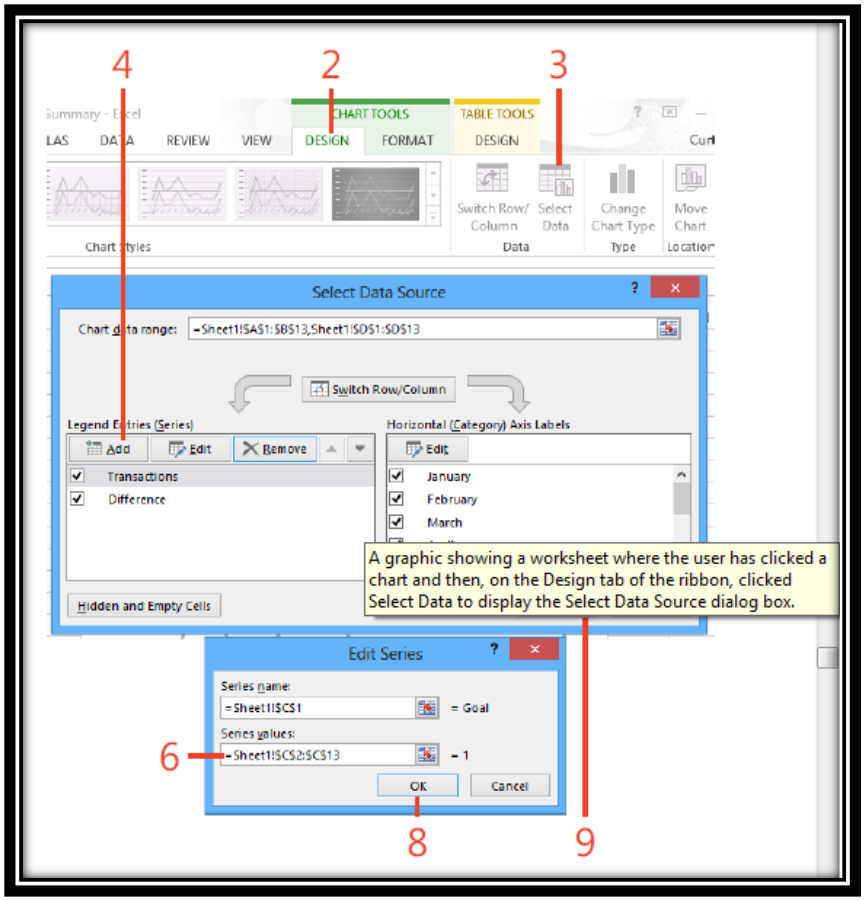
1. Click the chart element that you want to change.
2. Click the Format tab.
3. Use the controls in the Shape Styles group to format the chart element.

5.4 Adding Data Series to a Chart

Charts summarize worksheet data and, in many cases, you will include all of the data in a collection. For example, you could have sales data for each month of a year and create a column chart with a column for each month. In other cases, your data collection might have different types of data. One column might contain monthly sales, the next might contain the sales goal for the month, and the third might display the difference between actual sales and the target. In that case, you might want to limit the data in your chart to just the column summarizing monthly sales. You can add or remove data series from your chart to create exactly the summary you want.

Add a new series

1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Select Data.
4. Click Add.
5. Click the cell that contains the name for the series.
6. Click in the Series Values box.
7. Select the cells that you want to add.
8. Click OK.
9. Click OK.



5.5 Deleting Data Series to a Chart

Remove a series

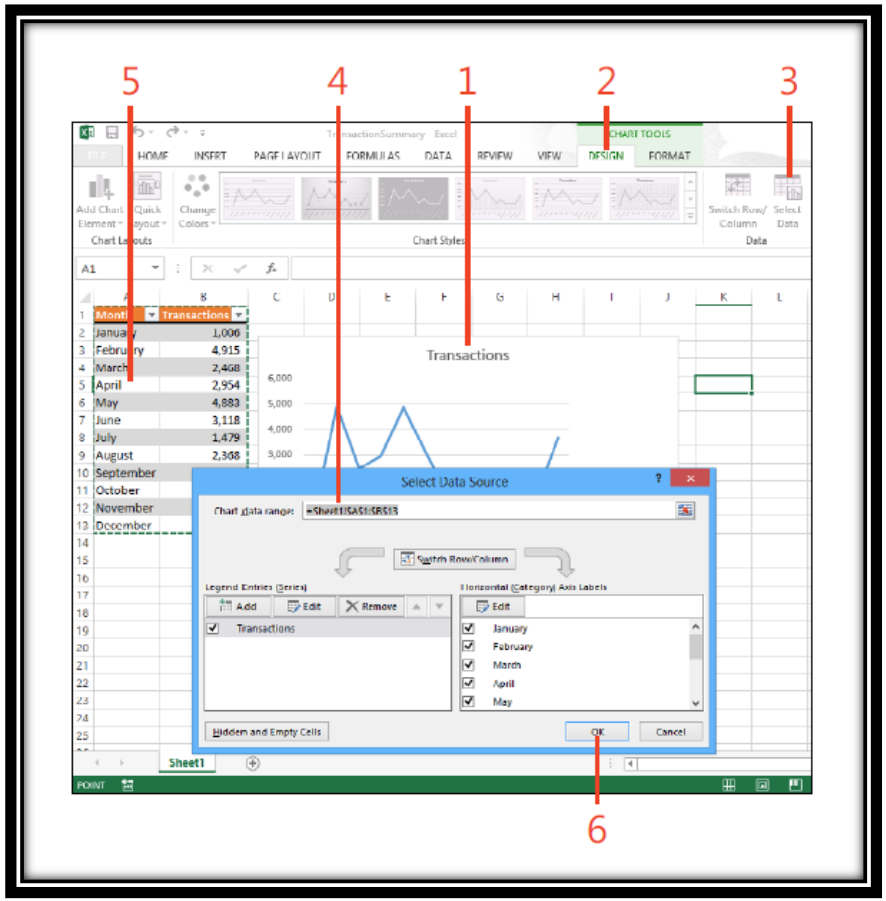
1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Select Data.
4. Click the name of the series that you want to delete.
5. Click Remove.
6. Click OK.

5.6 Switch between Rows and Columns data in a Chart

Whenever you create a chart for a business, there's always the possibility that the data displayed in the chart will change. Whether those changes reflect continuing sales, updated values that account for returns and inventory charges, or investment projections revised to match market conditions, you can update your chart by identifying a new data source. The data can be in any workbook on your computer or network—all you need to do is identify the cells with the data, and Excel does the rest.

Change the source data for your chart

1. Click the chart that you want to change.
2. Click the Design tab.
3. Click Select Data.
4. Click in the Chart Data Range field.
5. Select the cells that you want to provide the new source data.
6. Click OK.

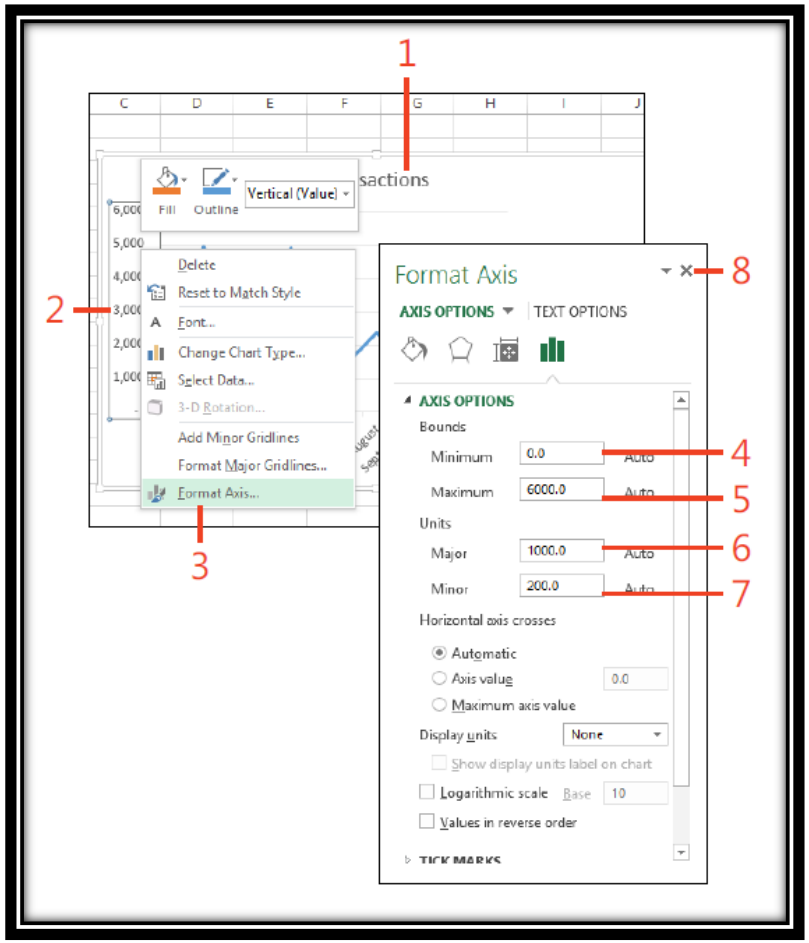


5.7 Formatting Chart Axis

The horizontal and vertical axes help you interpret the values displayed in a chart. For most chart types, including line charts and column charts, the vertical axis summarizes numerical values such as sales or miles driven. The horizontal axis tends to display category data, such as months for which sales data is collected, categories of products, or exhibits at a zoo. You can change the standard formatting of these axes to help viewers understand your data more quickly.

Change the scale on the value axis

1. Click the chart that you want to format.
2. Right-click the vertical axis.
3. Click Format Axis.
4. Under Bounds, in the Minimum box, type a value for the minimum value on the vertical axis.
5. Under Bounds, in the Maximum box, type a value for the maximum value on the vertical axis.
6. Under Units, in the Major box, type a value for the major units to be displayed on the vertical axis.
7. Under Units, in the Minor box, type a value for the minor units to be displayed on the vertical axis.
8. Click Close.



5.8 Modifying Chart and Graph Parameters (Data Labels and Grid Lines)

When you summarize numerical data in a chart, you can easily see the relative values of the data points within the chart. If a column is taller than another, you know the first column represents a larger value than the second. Adding grid lines helps you estimate the value represented by a point on a line or a column, but the chart won't display the exact values unless you add data labels. Data labels, as the name implies, display the exact value represented by each chart data point so that the value it represents is unambiguous.

Add and remove data labels

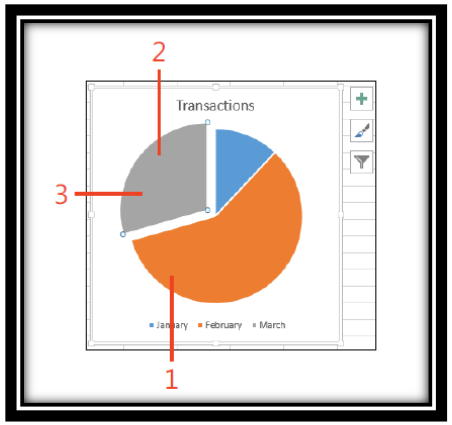
1. Click the chart that you want to format.
2. Click the Chart Elements button.
3. Select or clear the Data Labels check box.
4. If desired, position your mouse pointer over Data Labels and click the triangle that appears.
5. Click an option to apply it, or click More Options for greater control.

Show or hide chart grid lines

1. Click the chart that you want to format.
2. Click the Chart Elements button.
3. Position your mouse pointer over the Gridlines item.
4. Click the set of gridlines that you want to show or hide.
5. If desired, click More Options for greater control.

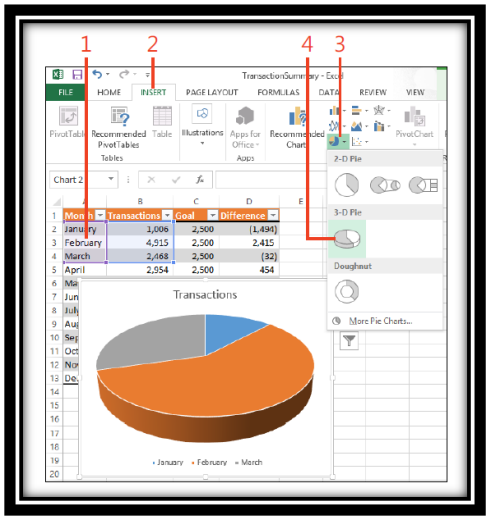
5.9 Changing the Style of Pieces of a Chart

One chart type that you'll probably work with quite frequently is the pie chart, which displays the contribution of a series of values to the total of those values. Each section represents a category of data, such as a month in a year or a category of product. One way that you can extend the capabilities of a basic pie chart and emphasize part of the data is to pull one section of the pie away from the rest of the chart. You can also change how you look at a three-dimensional chart, moving closer or farther away from the chart and changing your perspective.



Pull a slice out of a pie chart

1. Click the body of the pie chart that you want to change.
2. Click the piece of data that you want to separate.
3. Drag the piece away from the pie.



Create a 3-D pie chart

1. Select the cells that you want to summarize in the pie chart.
2. Click the Insert tab.
3. Click the Pie Chart button.
4. Click the 3-D Pie chart.

5.10 Filtering charts

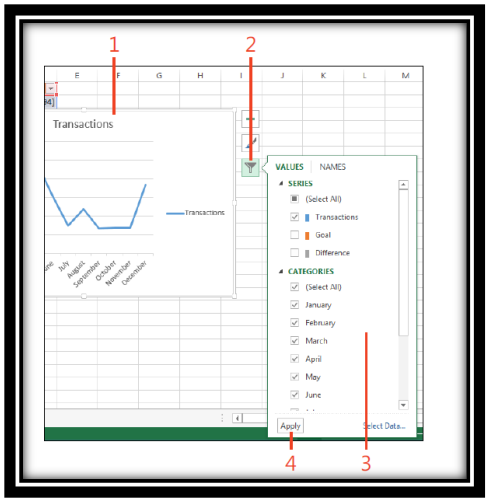
Limiting the data shown in a chart can help you and your colleagues focus on the facts that are most relevant to your situation. Excel 2013 gives you the ability to select which values appear in your chart by using the Chart Filters button, which the program displays next to the chart. Selecting or clearing the check box next to a category (horizontal) axis value displays or hides that value in the chart. When a filter is applied, the Select All check box contains a gray square to indicate that some values are hidden.

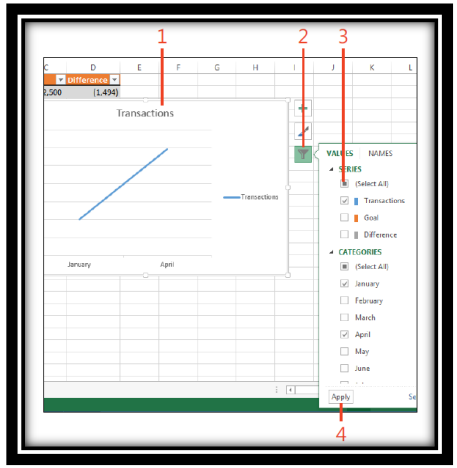
Filter a chart

1. Click the chart that you want to filter.
2. Click the Chart Filters button.
3. Select or clear the check boxes next to items that you want to display or hide.
4. Click Apply.

Remove a chart filter

1. Click the chart that you want to filter.
2. Click the Chart Filters button.
3. Select the Select All check box.
4. Click Apply.



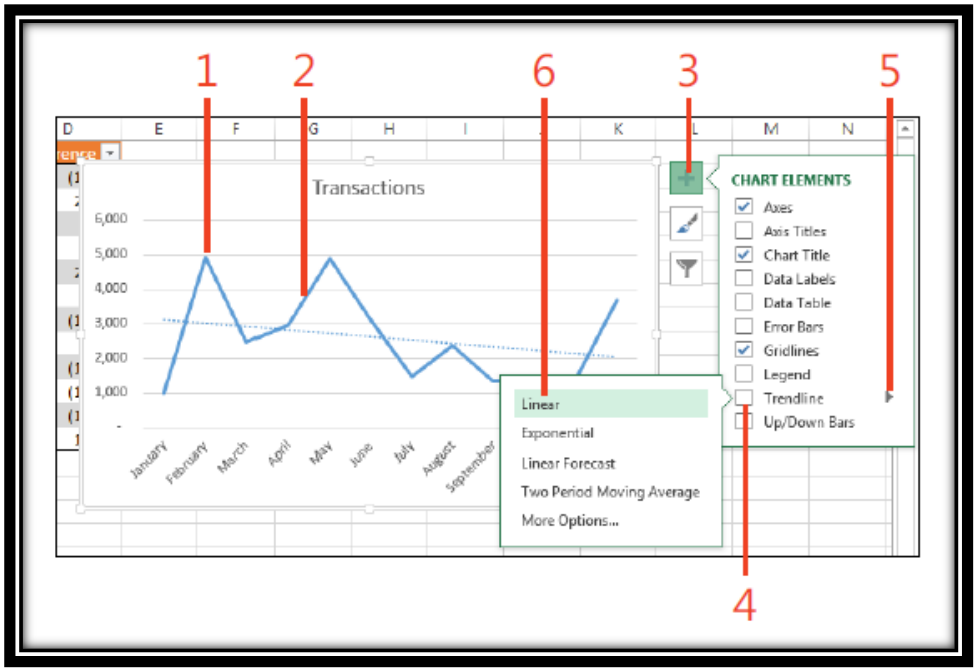


5.11 Adding a trend line to a chart

You can use the data in your Excel workbooks to analyze past performance, but you can also have Excel make its best guess as to future performance if current trends continue. For example, if you create a chart that represents your company's sales for the past five years, you can have Excel analyze the data and add a trendline to the chart to represent how much sales would increase if the current trend holds true for the next year.

Add a trendline to a data series

1. Click the chart to which you want to add a trendline.
2. Click the data series to which you want to add a trendline.
3. Click the Chart Elements button.
4. Position the mouse pointer over Trendline.
5. Click the gray triangle that appears.
6. Click Linear.



5.12 Summarizing data using spark lines

Creating charts in Excel 2013 workbooks enables you to summarize your data visually, using legends, labels, and colors to highlight aspects of your data. It is possible to create very small charts to summarize your data in an overview worksheet, but you can also use sparklines to create compact, informative charts that provide valuable context for your data.

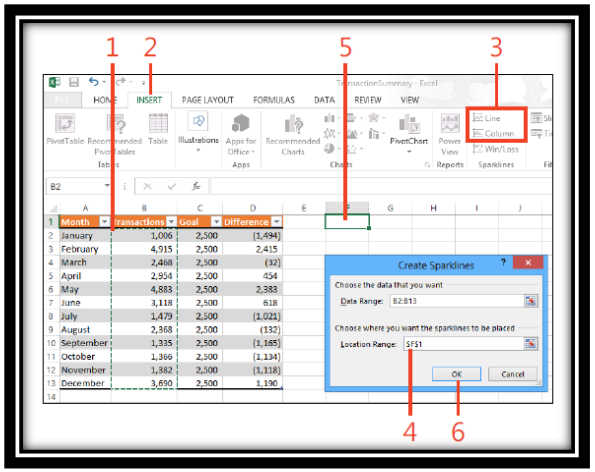
Sparklines has been introduced with the goal of creating charts that imparted information in approximately the same space as a word of printed text. In Excel 2013, a sparkline occupies a single cell, which makes it ideal for use in summary worksheets. You can create three types of sparklines: line, column, and win/loss. The line and column sparklines are compact versions of the standard line and column charts. The win/loss sparkline indicates whether a cell value is positive (a win), negative (a loss), or zero (a tie).

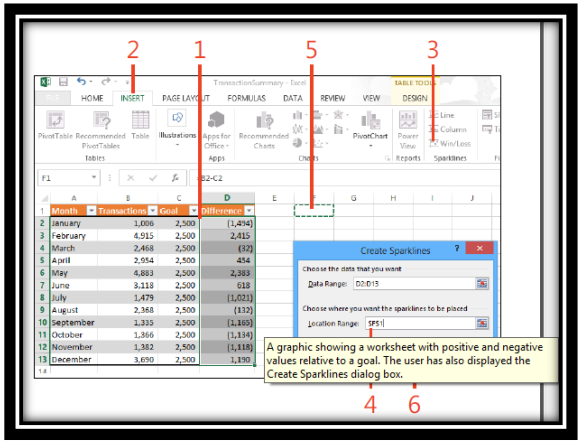
Create a line or column sparkline

1. Select the cells that you want to summarize.
2. Click the Insert tab.
3. In the Sparklines group, click Line or Column.
4. Click the Location Range box.
5. Click the cell where you want the sparkline to appear.
6. Click OK.

Create a win/loss sparkline

1. Select the cells that you want to summarize.
2. Click the Insert tab.
3. In the Sparklines group, click Win/Loss.
4. Click the Location Range box.
5. Click the cell where you want the sparkline to appear.
6. Click OK.





5.13 Formatting and deleting sparklines

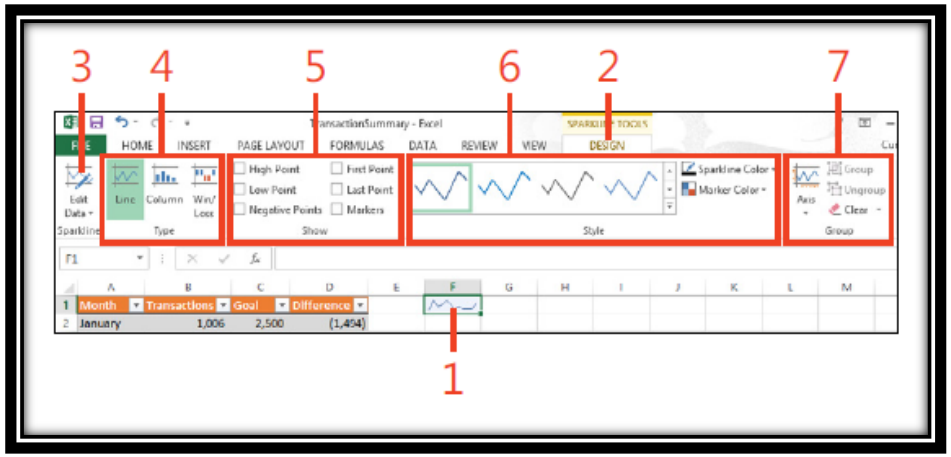
After you create sparkline, you can change its appearance to fit your worksheet's style. When you click a cell that contains a sparkline, you can use the controls on the Design tab of the ribbon to change the sparkline's colors, add markers to line sparklines, and even change the sparkline's type. You can also remove the sparkline entirely if you change your vision for the worksheet and the data it summarizes.

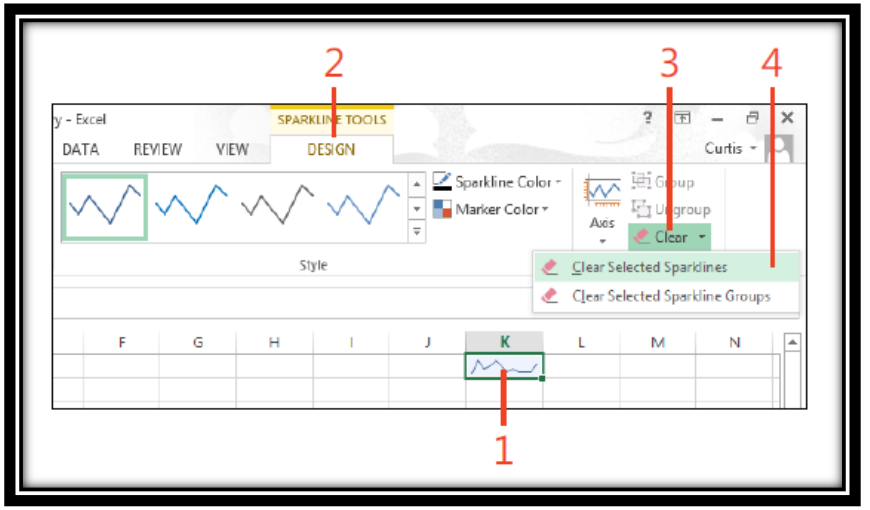
Format a sparkline

1. Select the sparkline that you want to change.
2. Click the Design tab.
3. Use the controls in the Sparkline group to change which data the sparkline summarizes.
4. Use the controls in the Type group to change the sparkline's type.
5. Use the controls in the Show group to determine which sparkline elements to show.
6. Use the controls in the Style group to change the sparkline's style, color, and marker color.
7. Use the

Delete a sparkline

1. Select the cell that contains the sparkline that you want to delete.
2. Click the Design tab.
3. Click the Clear button.
4. Click Clear Selected Sparklines.





6 Working with Tables

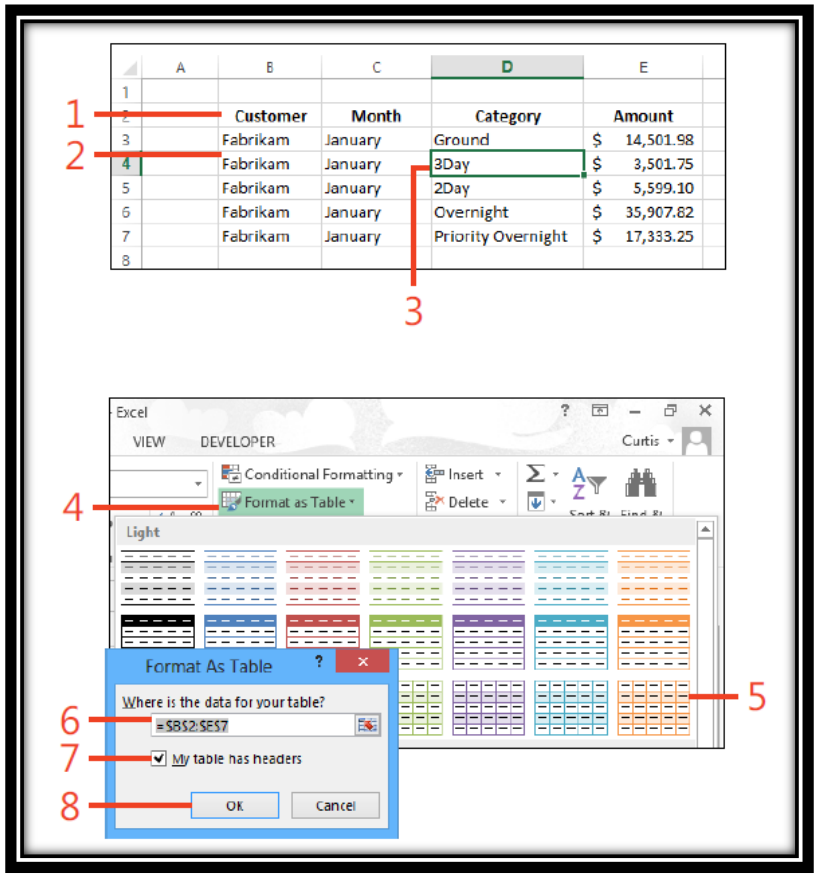
6.1 Create an Excel Table

One popular way to maintain data in Excel is by creating a data list. A list describes one type of object, such as orders, sales, or contact information. Structurally, a list consists of a header row, which contains labels describing the data in each column, and data rows, which contain data about a particular instance of the list's subject. For example, if you use a list to store your customer's contact information, you could have a separate column for the customer's first name, last name, street address, city, state, postal code, and telephone number. Each row in a list would contain a particular customer's information.

In Excel 2013, you can store your data lists in an Excel table. An Excel table is an object that you can refer to in your formulas and use to summarize your data.

Create an Excel table

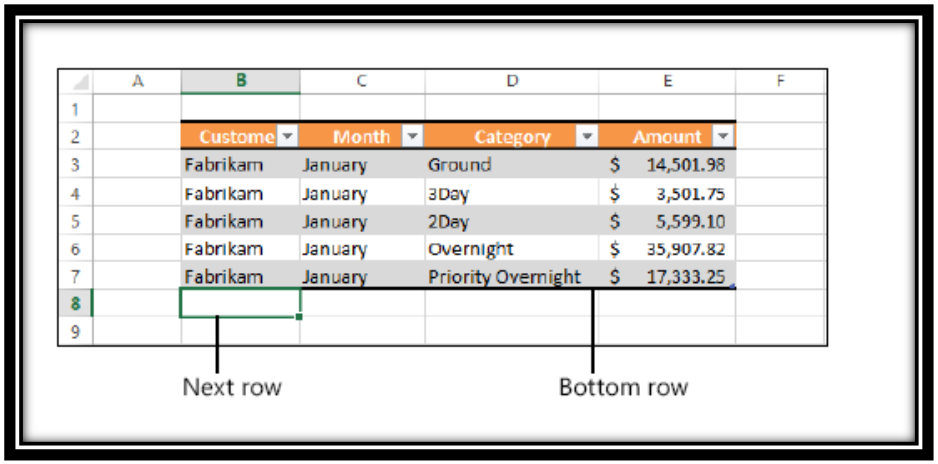
1. Type your table headers in a single row.
2. Type your first data row directly below the header row.
3. Click any cell in the range in which you want to create a table.
4. On the Home tab, click Format As Table.
5. Click the table style to use.
6. Verify that Excel identified the data range correctly.
7. If your table has headers, select the My Table Has Headers check box.
8. Click OK.



6.2 Add data to an Excel table

Add data to an Excel table

1. Click the cell at the lower-right corner of the Excel table, and press Tab to create a new table row.
2. As an alternative to step 1, type data in the cell below the lower-left corner of the Excel table, and press Tab. Excel makes the new row part of the Excel table.



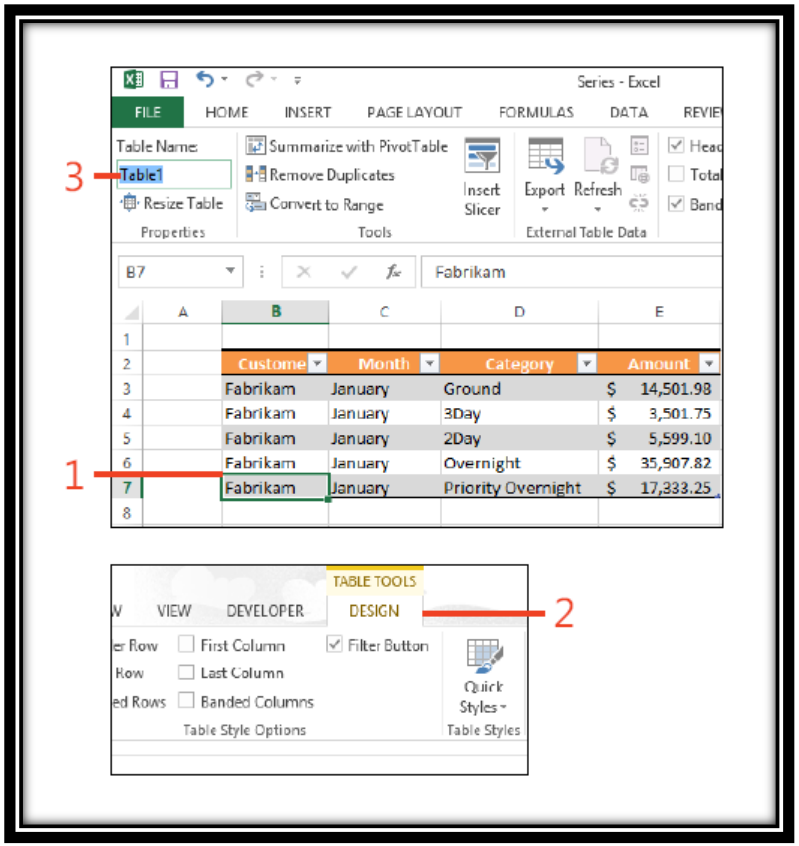
6.3 Modifying Excel Table

When you create an Excel table, you design it with the best structure to store your data. If the data you want to store in your Excel table changes, you can resize the table to fit your new collection.

If you'd like to copy an entire column of data from an Excel table, you can do so by positioning the mouse pointer over the column's header. When the mouse pointer changes to a downward-pointing black arrow, click the left mouse button to select the column. You can then press Ctrl+C to copy the data and paste it elsewhere in your workbook.

Rename an Excel table

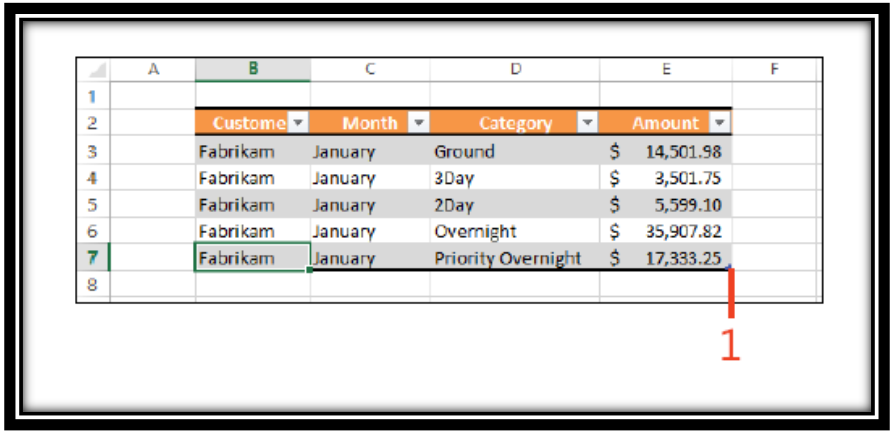
1. Click any cell in the Excel table.
2. On the ribbon, under Table Tools, click the Design tab.
3. In the Properties group, type a new name for your Excel table, and press Enter.



Working with Tables

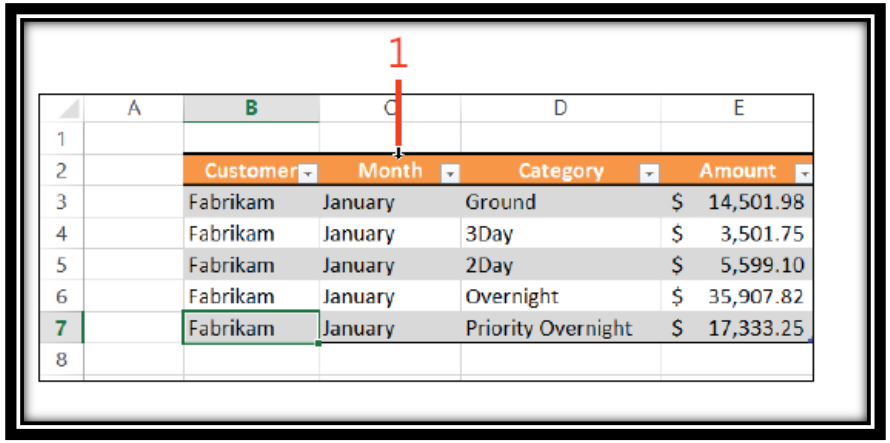
Resize an Excel table

1. Drag the resize handle at the lower-right corner of the Excel table to add or remove table rows or columns.



Select an Excel table column

1. Position the mouse pointer over the header cell of an Excel table column. When the pointer changes to a downward-pointing black arrow, click the header of the column that you want to select.



	A	B	C	D	E	F
1						
2		Customer	Month	Category	Amount	
3		Fabrikam	January	Ground	$ 14,501.98	
4		Fabrikam	January	3Day	$ 3,501.75	
5		Fabrikam	January	2Day	$ 5,599.10	
6		Fabrikam	January	Overnight	$ 35,907.82	
7		Fabrikam	January	Priority Overnight	$ 17,333.25	
8						

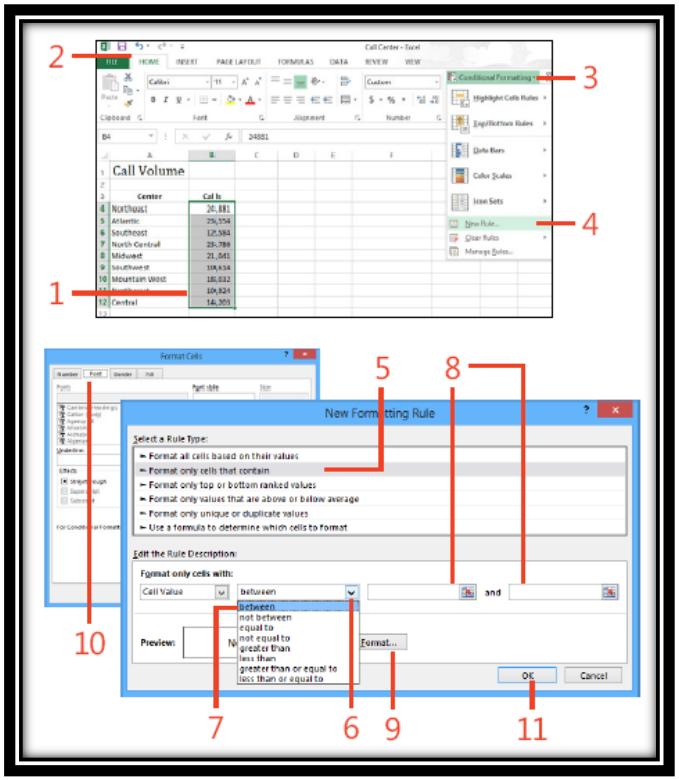
7 Advanced Formatting

7.1 Formatting a cell based on conditions

Another way you can make your data easier to interpret is to change the appearance of your data based on the value in a cell. This kind of formatting is called conditional formatting because the data must meet certain conditions to have a format applied to it. For example, if you use a worksheet to track your customers' credit lines, you could have a customer's outstanding balance appear in red if they are within 10 percent of their credit limit.

Change the format of a cell based on its value

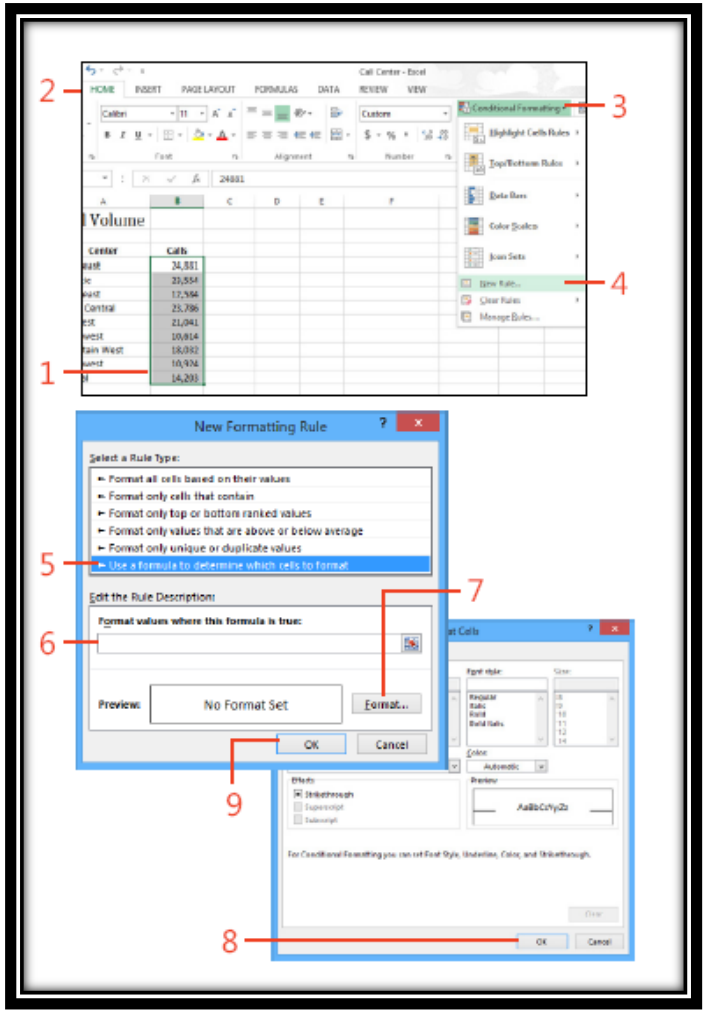
1. Select the cells that you want to change.
2. Click the Home tab.
3. In the Styles group on the ribbon, click Conditional Formatting.
4. Click New Rule.
5. Click Format Only Cells That Contain.
6. Click the Comparison Phrase down arrow.
7. Click the comparison phrase that you want.
8. Type the constant values or formulas that you want evaluated.
9. Click Format.
10. Specify the formatting that you want, and click OK.
11. Click OK.



Advance Formatting

Change the format of a cell based on the results of a formula

1 Select the cells that you want to change.
2 Click the Home tab.
3 In the Styles group on the ribbon, click Conditional Formatting.
4 Click New Rule.
5 Click Use A Formula To Determine Which Cells To Format.
6 Type the formula that you want evaluated.
7 Click Format.
8 Specify the formatting that you want, and click OK.
9 Click OK.

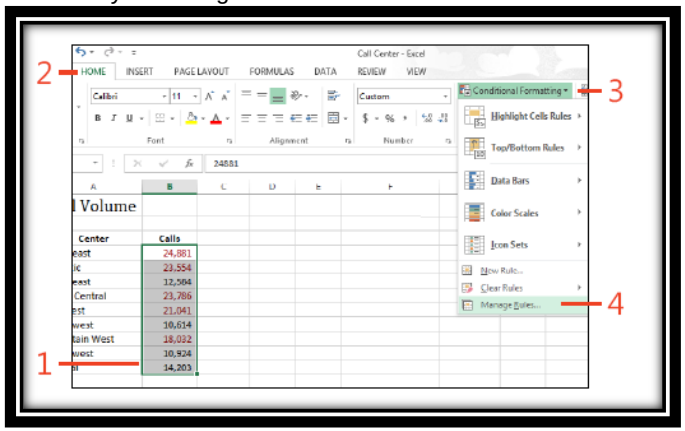


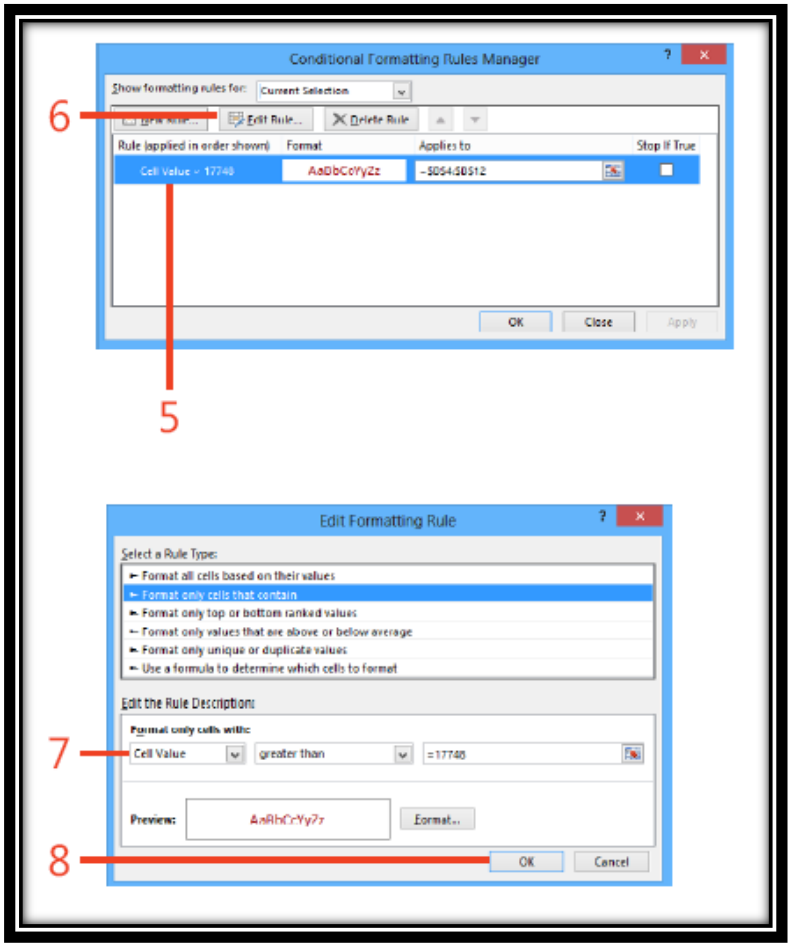
Editing and deleting conditional formats

Excel 2013 makes it easy for you to manage conditional formatting rules that you create in your worksheets. You can change the properties of a rule (such as the formatting applied or the conditions themselves), change the order in which the rules are applied, or delete a rule entirely.

Edit a conditional formatting rule

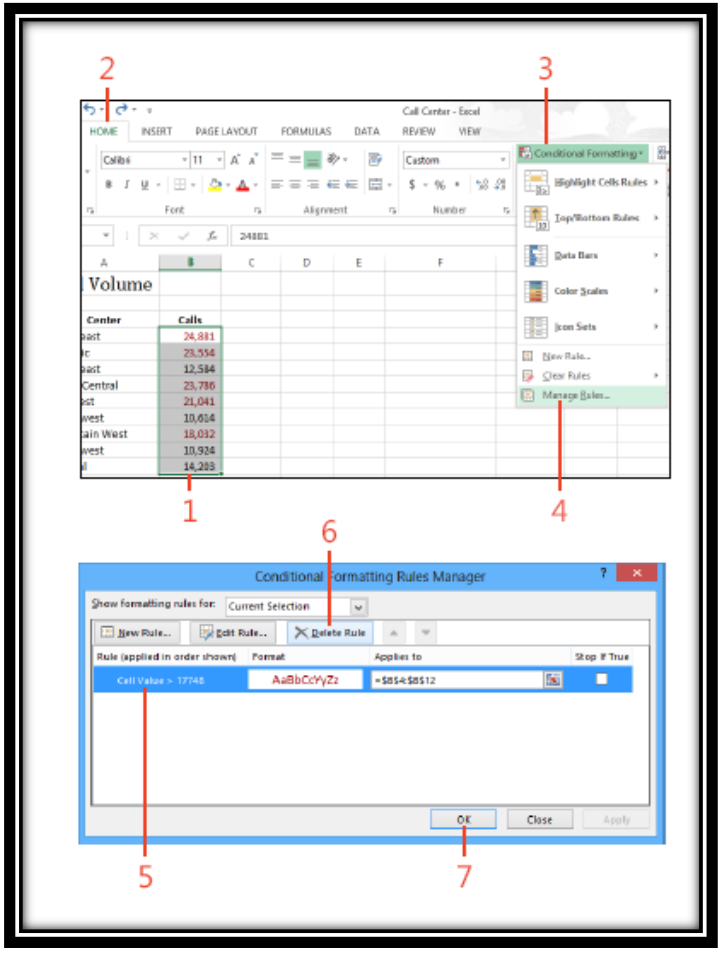
1 Select the cells that contain the rule you that want to edit.
2 Click the Home tab.
3 In the Styles group, click Conditional Formatting.
4 Click Manage Rules.
5 Click the rule that you want to change.
6 Click Edit Rule.
7 Use the controls to make your changes.
8 Click OK twice to save your changes.





Delete a conditional formatting rule

1. Select the cells that contain the rule that you want to delete.
2. Click the Home tab.
3. In the Styles group, click Conditional Formatting.
4. Click Manage Rules.
5. Click the rule that you want to delete.
6. Click Delete Rule.
7. Click OK

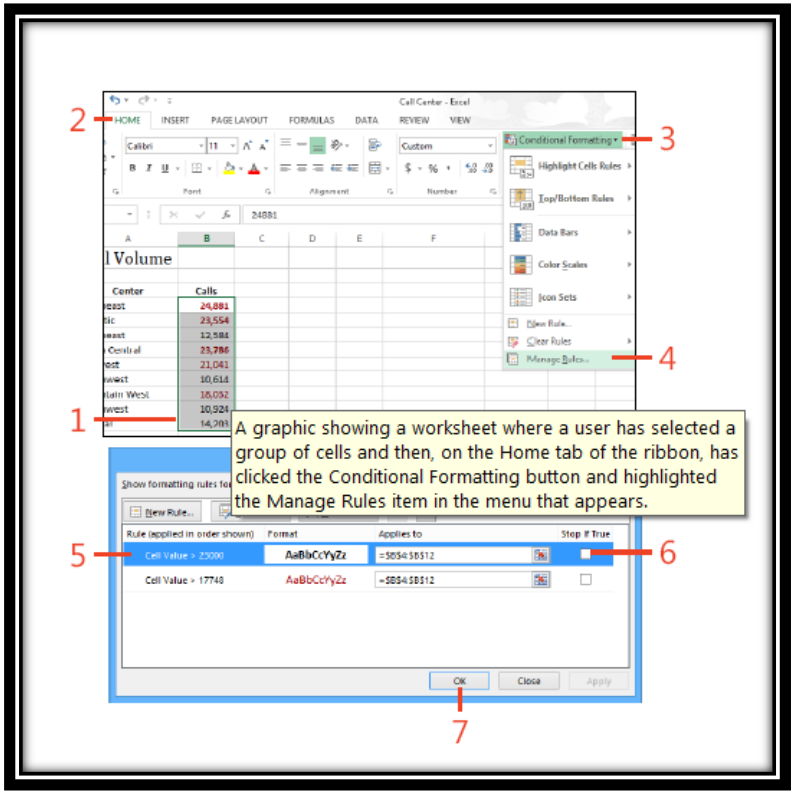


7.3 Changing how conditional formatting rules are applied

Excel 2013 enables you to create powerful conditional formatting rules and to control how those rules are applied. For example, you can have Excel apply more than one conditional format to a cell, so if you want to display a value of more than 1,000 in bold text and values that exceed a sales target with a yellow background, you can define those conditions separately and have Excel apply them both.
You can also decide whether Excel should stop after it finds a rule that applies to your data or continue to check other rules. Finally, you can change the order in which Excel checks your conditions to control how Excel applies the rules. Rule order matters only if you choose to have Excel stop after it applies a rule.

Stop when a condition is met

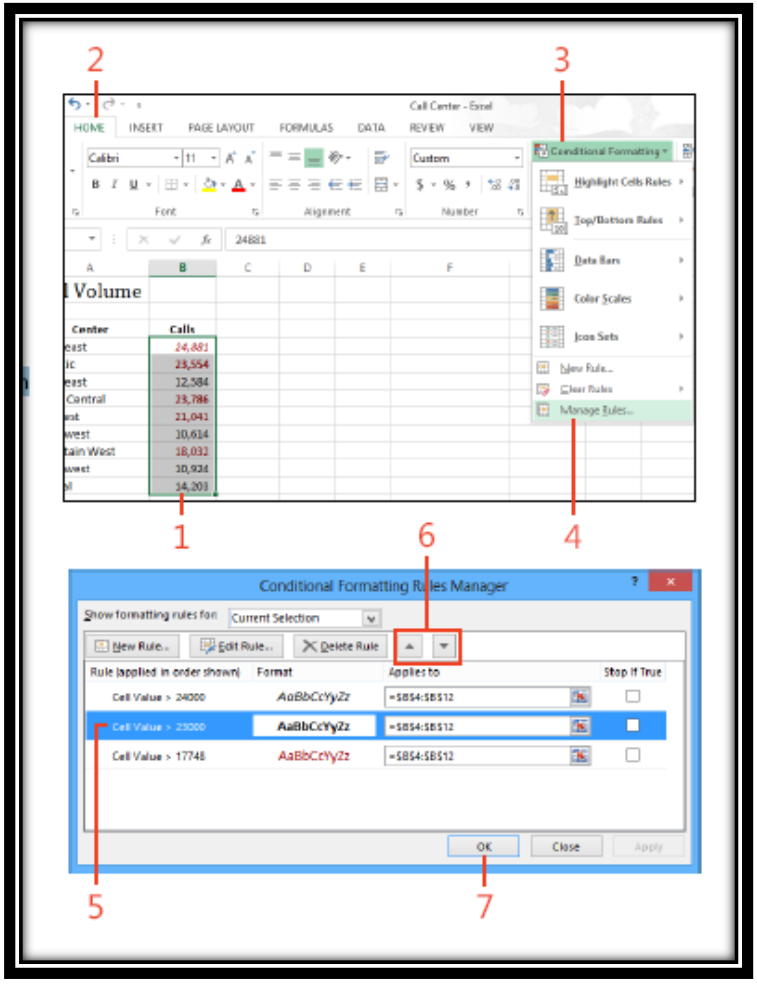
1 Select the cells that contain the rule that you want to edit.
2 Click the Home tab.
3 In the Styles group, click Conditional Formatting.
4 Click Manage Rules.
5 Click the rule that you want to change.
6 Select the Stop If True check box.
7 Click OK.



Advance Formatting

Change the order of conditions

1. Select the cells that contain the rules that you want to edit.
2. Click the Home tab.
3. In the Styles group, click Conditional Formatting.
4. Click Manage Rules.
5. Click the rule that you want to change.
6. Follow either of these steps:
    a. Click the Move Up button to move the rule one place higher in the order.
    b. Click the Move Down button to move the rule one place lower in the order.
7. Click OK.

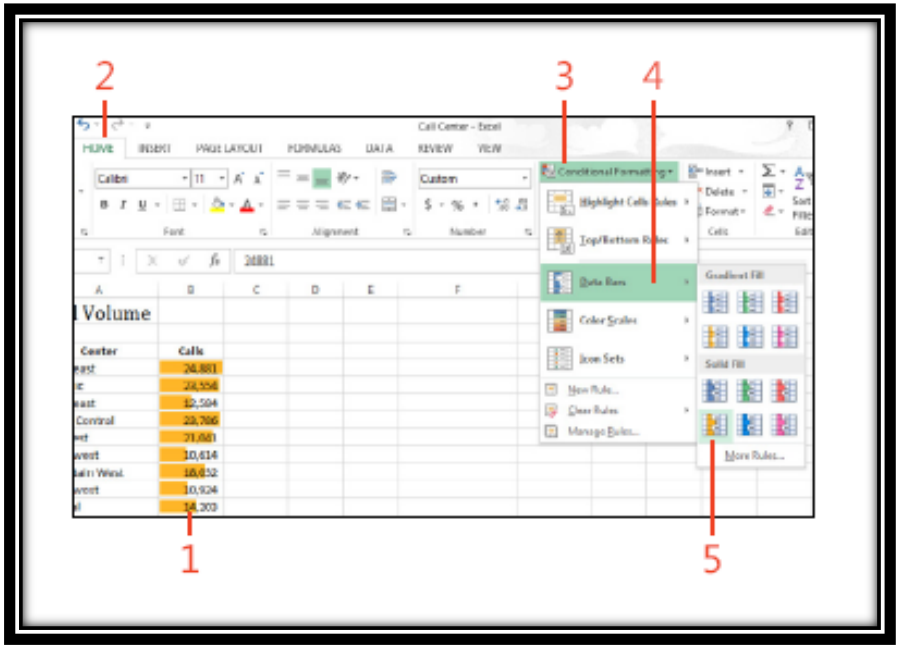


7.4 Displaying data bar and icon set formats

Organizations of all kinds track their performance data in Microsoft Excel. The specific numbers are important, of course, but it's also important that you be able to tell how the numbers relate to each other and whether your department is meeting its performance goals. Data bars show how the data in a cell compares to other data in a selected range. The higher a value is in relation to the other values in the range, the longer the data bar. Icon sets, by contrast, test a value to see whether it is in the top, middle, or bottom third of the values in the range. You can change the tests to determine which icon or color is shown when.

Display data bars

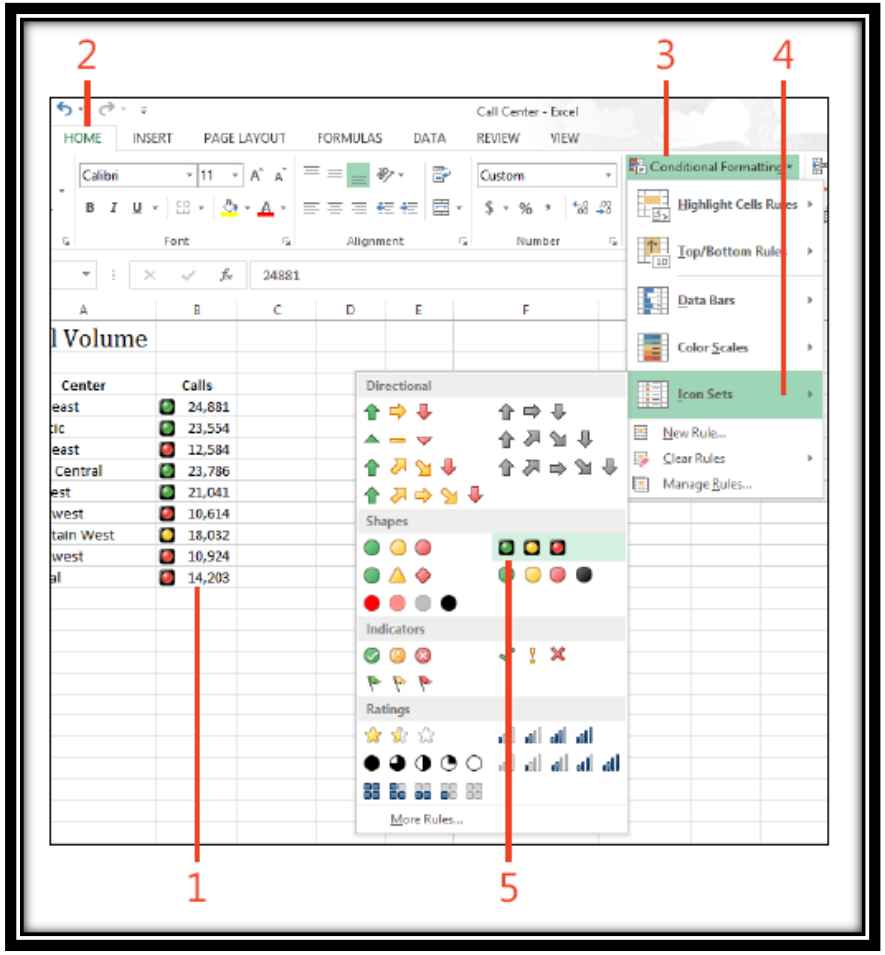
1 Select the cells that contain your data.
2 Click the Home tab.
3 In the Styles group on the ribbon, click Conditional Formatting.
4 Point to Data Bars.
5 Click the data bar option that you want to apply.



Advance Formatting

Display icon sets

1 Select the cells that contain your data.
2 Click the Home tab.
3 In the Styles group on the ribbon, click Conditional Formatting.
4 Point to Icon Sets.
5 Click the icon set that you want to apply.

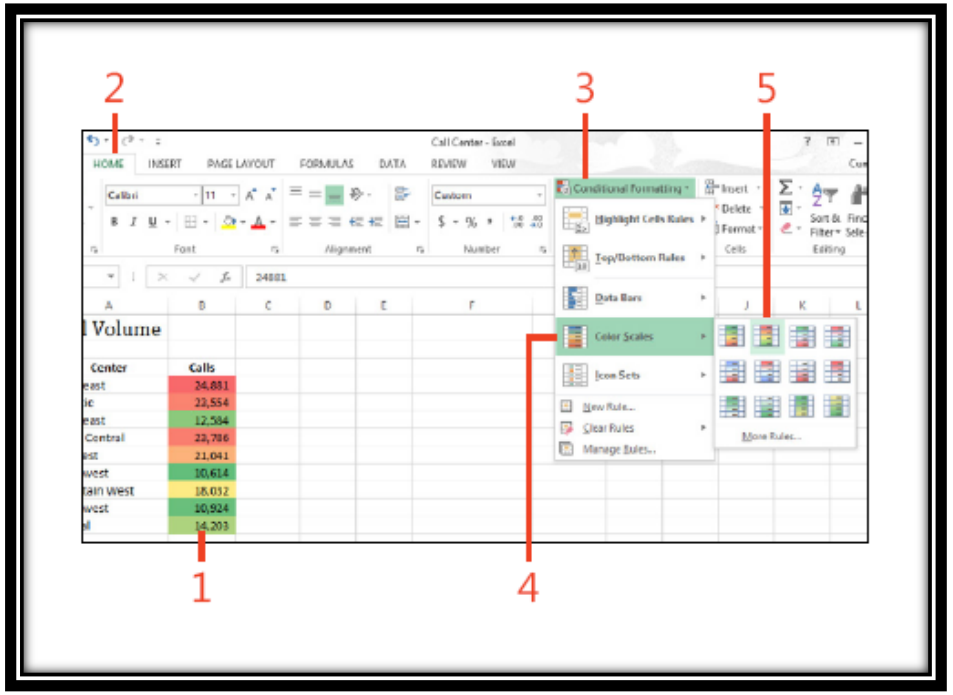


7.5 Displaying color scales based on cell values

Color scales compare the relative magnitude of values in a cell range by applying colors from a two-color or three-color set to your cells. The intensity of a cell's color reflects the value's tendency toward the top or bottom of the values in the range. Applying two-color color scales, using colors such as white and red, to a cell range is a terrific way to create an easy-to-interpret visual map of your data.

Display color scales

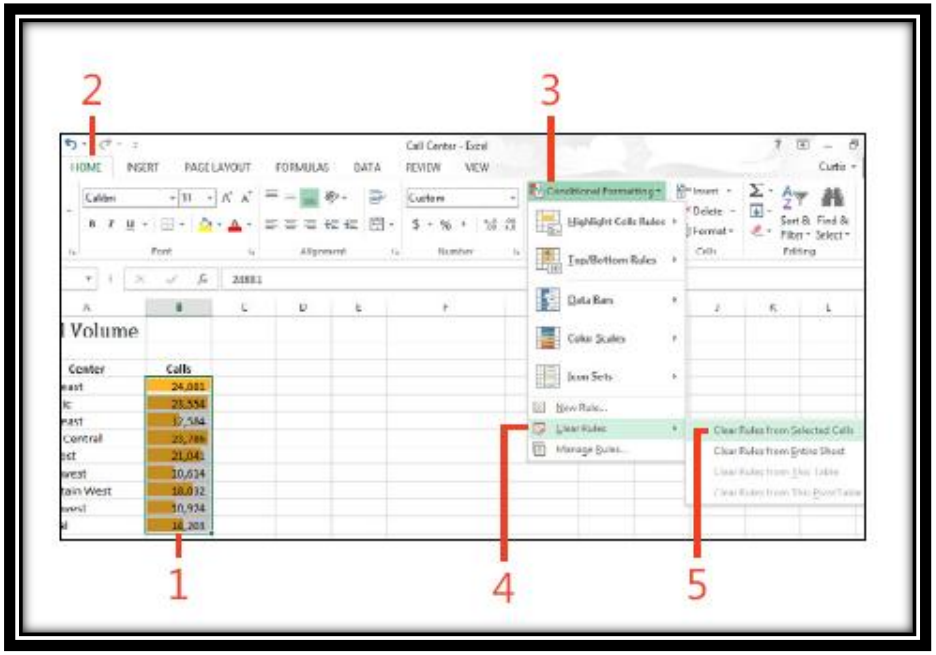
1 Select the cells that contain your data.
2 Click the Home tab.
3 In the Styles group on the ribbon, click Conditional Formatting.
4 Point to Color Scales.
5 Click the color scale pattern that you want to apply.



7.6 Deleting conditional formats

If you no longer want to use conditional formats to summarize the data in your workbook, you can delete the formats without affecting the underlying data. When you delete a conditional format, you can select whether to do so for the selected cells, the entire sheet, the active PivotTable, or the active Excel table.

1 Select the cells with the conditional formatting that you want to delete.
2 Click the Home tab.
3 In the Styles group on the ribbon, click Conditional Formatting.
4 Point to Clear Rules.
5 Click the set of rules that you want to delete.

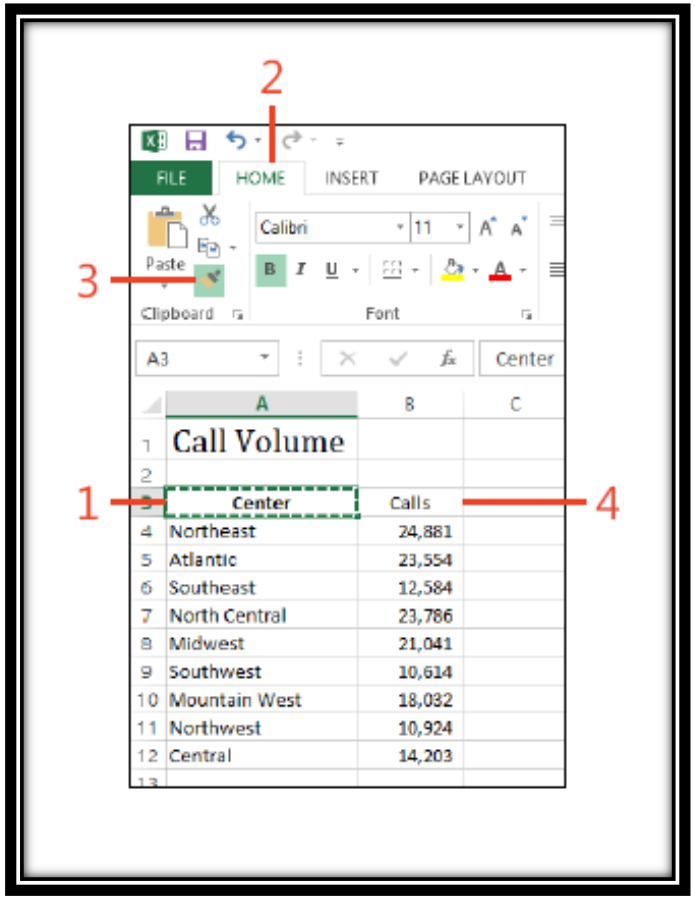


7.7 Copying formats with Format Painter

After you create a format for a cell or group of cells, you can copy the format to another group of cells using Format Painter. Format Painter lets you copy the format quickly, saving you the time and effort it takes to copy the contents of another cell with the formats that you want and then changing the data or formula.

Copy styles with Format Painter

1 Select the cells with the formatting that you want to copy.
2 Click the Home tab.
3 In the Clipboard group, click the Format Painter button.
4 Select the cells where you want the formatting applied.



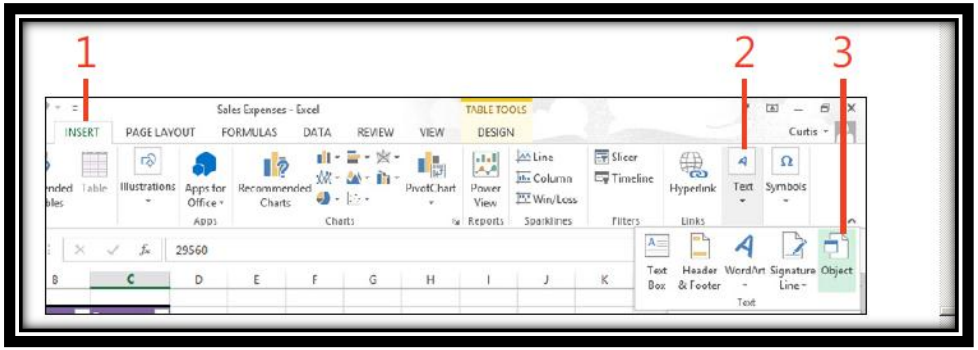
8 Excel and Other Applications

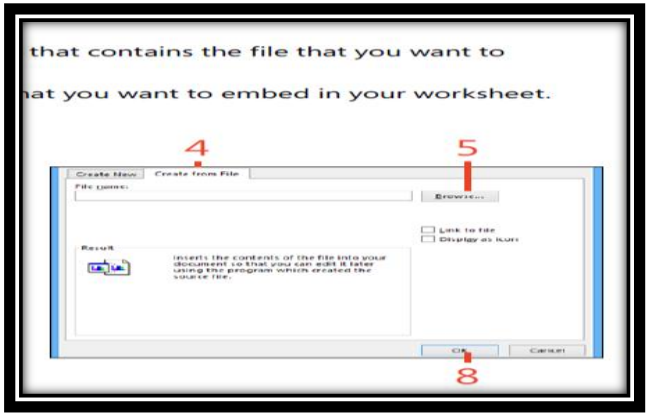
8.1 Linking and embedding other files

Excel workbooks can hold a lot of data, and you can make your data even more meaningful when you include files created with other programs. For example, you might have a Word document with important background information or a PowerPoint presentation that puts your data into context for your colleagues. You can include those files in your worksheets by linking or embedding the files as objects.

Embed a file in a worksheet

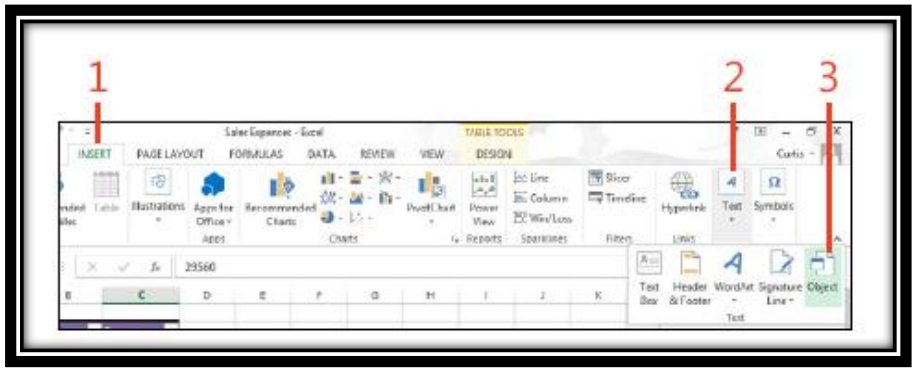
1. Click the Insert tab.
2. Click Text.
3. Click Object.
4. Click Create From File.
5. Click Browse.
6. Navigate to the folder that contains the file that you want to embed.
7. Double-click the file that you want to embed in your worksheet.
8. Click OK.

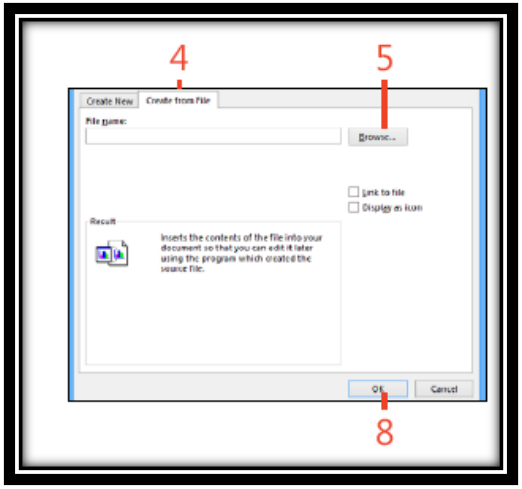


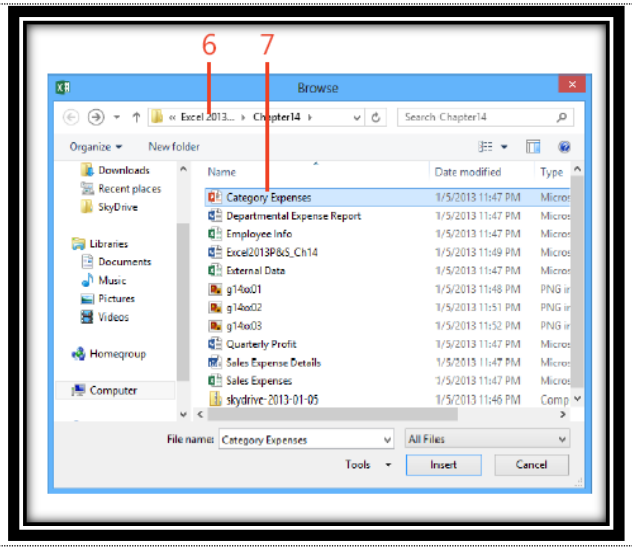


Embed a file in a worksheet

1. Click the Insert tab.
2. Click Text.
3. Click Object.
4. Click Create From File.
5. Click Browse.
6. Navigate to the folder that contains the file that you want to embed.
7. Double-click the file that you want to embed in your worksheet.
8. Click OK.

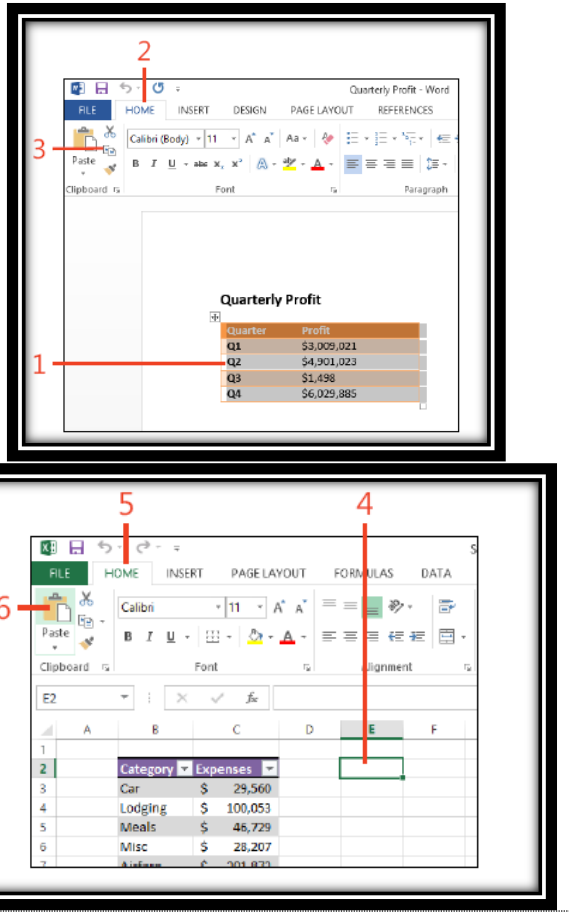






8.2 Exchanging table data between Excel and Word

Just as you can create workbooks to store and manipulate financial and other data in Excel, you can use Word to create reports and other text documents to interpret and provide valuable context for your worksheet data. Word documents can also present data in tables, which are arranged in rows and columns like a worksheet. For example, if you receive a report from a traveling colleague in which she created a table listing the prices of popular products at a competitor's store, you can copy the data from the Word document to an Excel worksheet for direct comparison. You can also go in the opposite direction, copying Excel data to a table in Word.



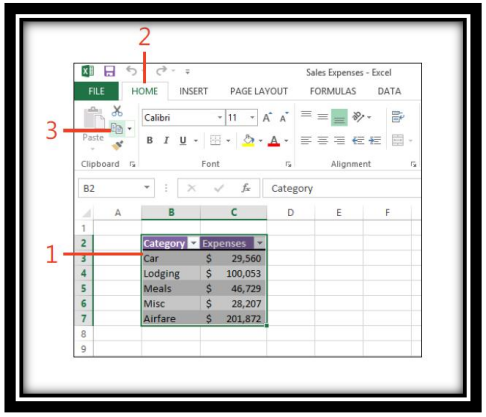
Paste Word data into Excel

1. In Word, select the table that you want to import into Excel.
2. Click the Home tab.
3. Click the Copy button.
4. In Excel, click the cell in which you want the upper-left table cell to appear.
5. Click the Home tab.
6. Click Paste.

8.3 Copying Excel charts and data into PowerPoint

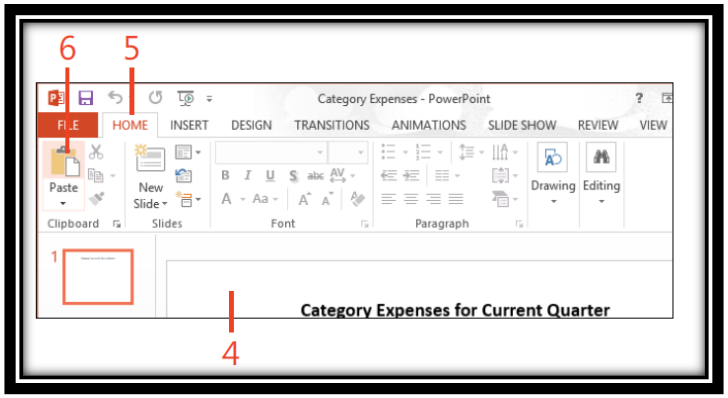
It's possible to present Excel data to your colleagues by using Excel exclusively, but if you're developing a formal presentation, you might want to use PowerPoint. Because presentations often contain graphical representations of numerical data, it's easy to include your Excel data in a PowerPoint presentation.

When you copy an Excel chart into a PowerPoint presentation, you can select how you want PowerPoint to manage the chart and its data. If you don't make any changes, PowerPoint creates the chart and maintains a link to the workbook from which it came. You can also choose to paste the entire workbook into PowerPoint or to paste a picture of the chart's current state.



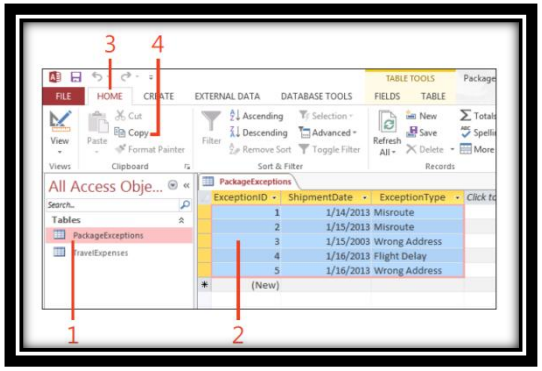
Move Excel data to PowerPoint

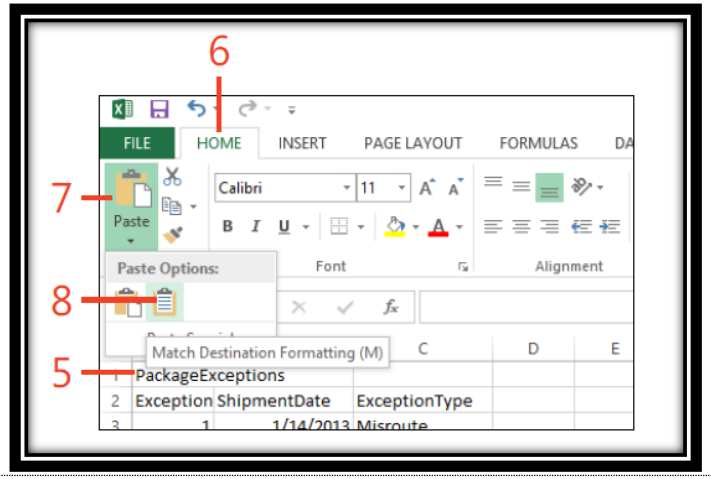
1. In Excel, select the cells that you want to export.
2. Click the Home tab.
3. Click the Copy button.
4. In PowerPoint, click the location where you want the pasted cells to appear.
5. Click the Home tab.
6. Click Paste.



8.4 Exchanging data between Access and Excel

No other Microsoft Office 2013 programs have as much in common as Access and Excel, but each program retains its unique strengths. Where Excel offers a wide range of data analysis and presentation tools that you can use to summarize your data, Access is designed to let you store, manipulate, and ask questions about large data collections. You can also use Access queries to locate and summarize table data. Although it is possible to look up data in an Excel worksheet, it's much easier to do in Access.





Paste Access table data into an Excel worksheet

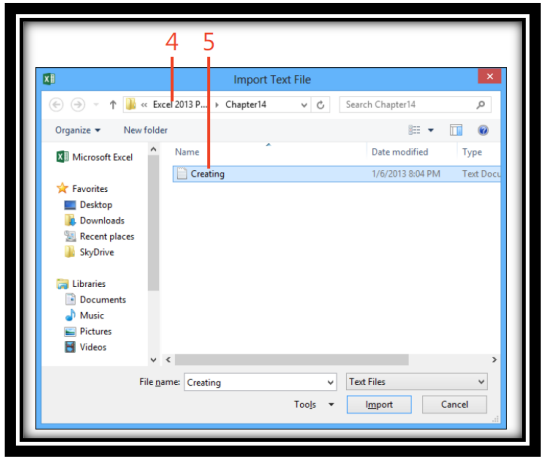
1. In Access, display the table from which you want to copy the data.
2. Select the table cells that you want to copy.
3. Click the Home tab.
4. Click Copy.
5. In Excel, click the cell where you want the first table value to appear.
6. Click the Home tab.
7. Click the Paste button's down arrow.
8. Click the Match Destination Formatting icon.

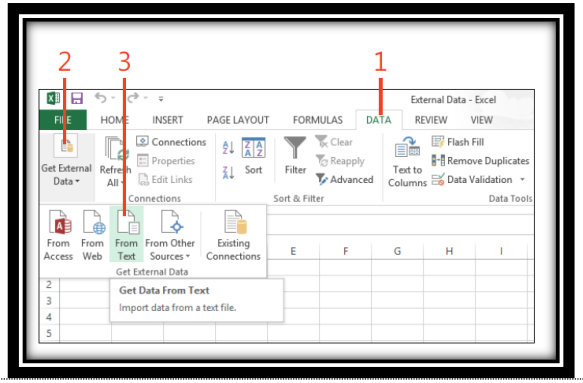
8.5 Importing a text file

Excel can read data from quite a few other spreadsheet and database programs, but you might have a colleague who uses a spreadsheet or database program that creates files that you can't read with Excel. If that's the case, you can ask your colleague to save the file as a text file, using a comma, tab, or other character (called a delimiter) to mark the end of each cell's data. Even if you can't transfer data any other way, you can always read spreadsheet data if it's presented to you in a text file. Any formatting and formulas are lost, but the data will be there for you to analyze.

Paste text into Excel

1. Click the Data tab.
2. Click Get External Data.
3. Click From Text.
4. Navigate to the folder that contains the text file that you want to import.
5. Double-click the text file.





9 The power of Excel – Visual Basic for Applications (VBA)

9.1 Visual Basic for Applications

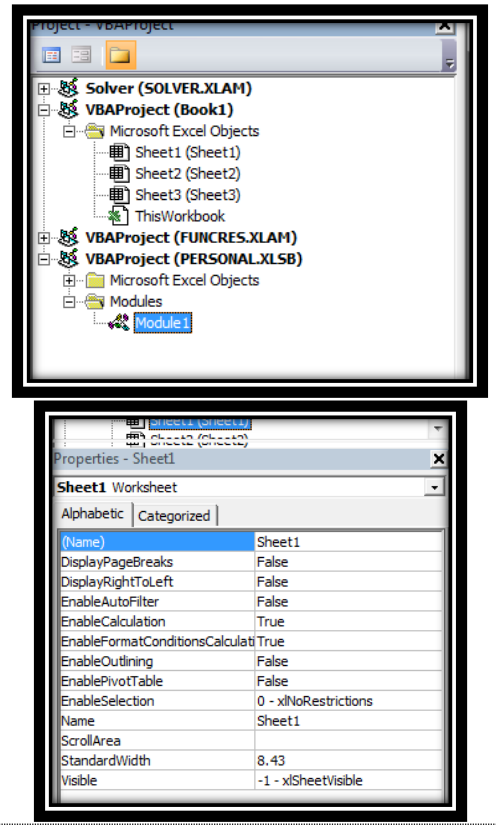
9.1.1 Third headings

The first step in becoming an Excel VBA programmer is to become familiar with the environment in which Excel VBA programming is done. Each of the main Office applications has a programming environment referred to as its *Integrated Development Environment* (IDE). Microsoft also refers to this programming environment as the *Visual Basic Editor*.

The Project Window

The window in the upper-left corner of the client area (below the toolbar) is called the *Project Explorer*.

Note that the Project Explorer has a treelike structure, similar to the Windows Explorer's folders pane (the left-hand pane). Each entry in the Project Explorer is called a *node*. The view of each project can be expanded or contracted by clicking on the small boxes (just as with Windows Explorer). Note that there is one project opened out of four currently open Excel workbook.

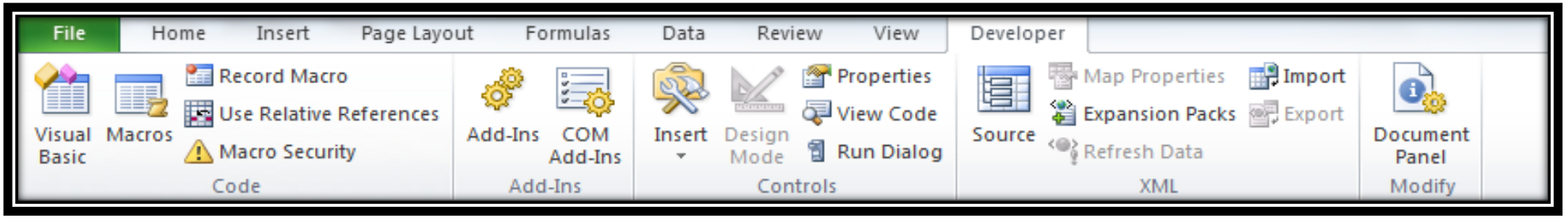


Visual Basic Editor

9.2 The Visual Basic Editor

Many tasks you perform in Microsoft Excel 2013 are done once or can be repeated quickly by using tools in Excel. However, you probably have one or two tasks you perform frequently that require a lot of steps to accomplish. For example, you might have several cells in a worksheet that contain important data you use quite often in presentations to your colleagues. Instead of going through a lengthy series of steps to highlight the cells that have the important information, you can create a macro, which is a recorded series of actions, to perform the steps for you. After you have created a macro, you can run, edit, or delete it as needed. In Excel, you run and edit macros by using the items available in the Macros group on the View tab. You can make your macros easier to access by creating new buttons on the Quick Access Toolbar, to which you can assign your macros. If you run a macro to highlight specific cells in a worksheet every time you show that worksheet to a colleague, you can save time by adding a Quick Access Toolbar button that runs the macro to highlight the cells for you.
Another handy feature of Excel macros is that you can create macros that run when a workbook is opened. For example, you might want to ensure that no cells in a worksheet are highlighted when the worksheet opens. You can create a macro that removes any special formatting from your worksheet cells when its workbook opens, which enables you to emphasize the data you want as you present the information to your colleagues.

You can also use form controls and macros to create custom solutions for your business. By adding controls such as text boxes, spin controls, and list boxes, you can design a user friendly interface for you and your colleagues to enter data quickly while minimizing errors. In this chapter, you'll open, run, create, and modify macros; create Quick Access Toolbar buttons and shapes that you can use to run macros with a single mouse click; run a macro when a workbook is opened; and add controls and set form properties for a UserForm.

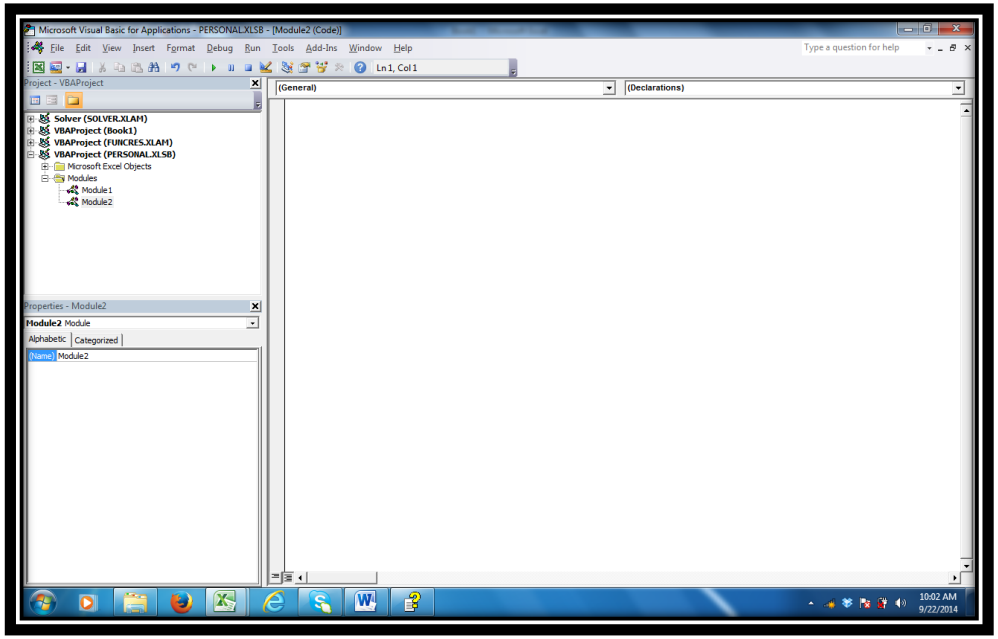


Visual Basic Editor

9.3 Exploring the Visual Basic Editor in Excel

Most users are unaware that, in addition to the spreadsheet application of Excel, there is an extremely powerful programming language built into Excel that you can use to design your own applications. You can use VBA code to write macro applications in VBA that do some very powerful things. **A macro** is a procedure written in VBA code that performs certain tasks. This could be something like sorting all worksheets within a workbook into alphabetical order or adding menu structures to the Excel menu. Whatever you decide to do with them, macros automate tasks and make life easier for you and the users of your workbook.

However, before you can start programming in Excel, you need to know where the macros are stored. This is not as obvious as it used to be when text-based macros were entered into a special macro spreadsheet. In the old macro language, you simply inserted a macro sheet and entered your commands anywhere. Now that the macro language has grown and become a full object-oriented language, the method of storing it has also changed. Macros are now kept inside hidden VBA Projects that are stored and saved within the workbook. These VBA Projects can be accessed through a companion program called the Visual Basic Editor (VBE). Press Alt-F11 to see the window shown.



At first glance, this window with its new menu bar, containing menus for File, Edit, View, Insert, Format, Debug, Run, Tools, Window, and Help, might be confusing. It opens up as a separate application window, but it is still very much a part of the Excel application. In fact, this window opens up a whole new ball game in terms of what you can do with Excel. In the next section, I'll explain the windows in more detail.

.

9.4 VBA Project Explorer and Code Windows

The Project Explorer, which shows a Project tree, is in the top-left corner of the screen, just below the menu and toolbar. It shows the VBA project for the active workbook as it stands, displaying the details in tree form so that you can easily navigate between them. If you click a branch of the tree, you'll enter that particular workbook or worksheet from the Visual Basic Editor. The VBA project is the root of the tree and the workbook, and worksheet objects are the branches coming off the tree. As you add and delete worksheets or workbooks, the branches of the tree change to reflect the new worksheets or workbooks being created. You can also add in other objects such as UserForms and modules. Effectively, what you are seeing is a list of currently loaded workbooks and the worksheets within them in an Explorer-like interface.

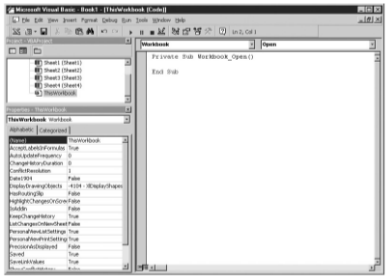
Remember, VBA is an object-oriented language. The first branch on the tree coming from the root of the VBA project says Microsoft Excel Objects. Coming off this branch are objects for the workbook that contain worksheets. Other objects can be inserted into the Project tree, such as UserForms, modules, and class modules.

This is a very important concept to understand because the workbook is an object that can be referred to, and each sheet is also an object that can be referred to. These are not the only objects within Excel, but looking at the Project Explorer simplistically, these are the objects shown there.

Double-click ThisWorkBook, and the code window for the **Workbook** object will open. Initially, it does not show a great deal, and you may wonder what to do with it. If you type something at random such as **What do I do now?** and press Enter, you will get a compile error. This is because there are disciplines and rules about entering code. Everything that you enter here goes through a Visual Basic compiler that interprets what you have written and converts it into instructions that your computer understands. Unfortunately, it does not understand plain English, which is why you get a compile error.

Click OK in the Compile Error message box and delete your statement. Notice the statement line turned red when the compile error appeared to draw your attention to the problem. Even if you do nothing about it, it will remain red as a danger signal to show that there is a problem in your code.

The drop-down list in the top-left corner of the window shows (General). Click the drop- down and to see another choice, Workbook. Click Workbook to see the code for the workbook event of **Open**. Your screen should now look like this



10 Programming Basics: Decisions and Looping

10.1 Overview

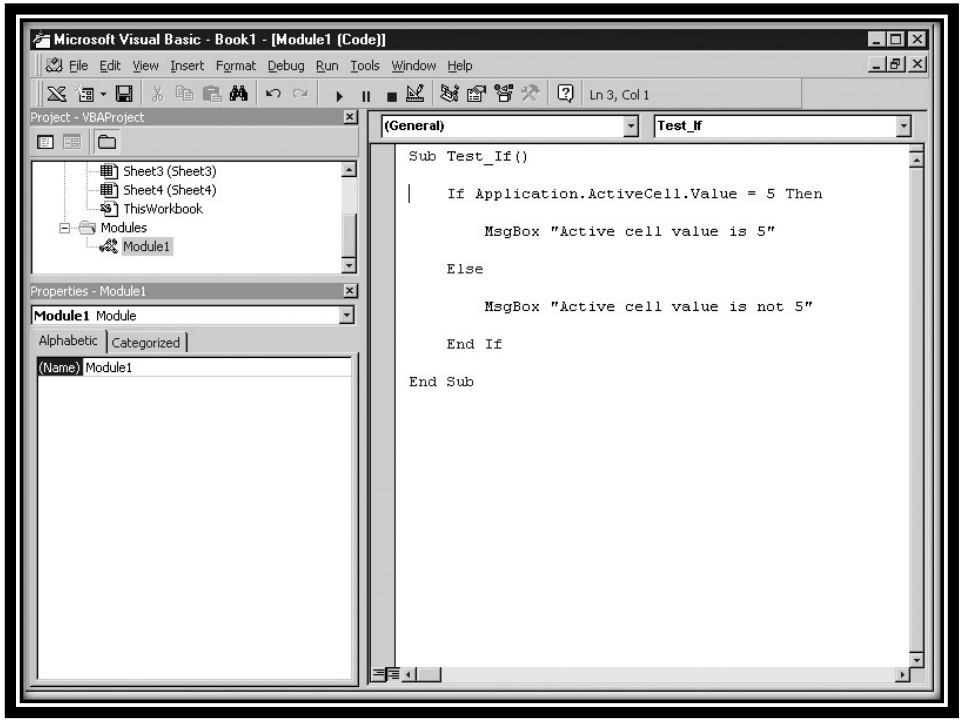
When writing programs, it's important to understand how programs make decisions and how they perform looping. Looping is the process of carrying out the same set of instructions until certain conditions are met.

Everyone is familiar with decisions. After all, you have to make them every day. For example when you wake up, which shirt do you decide to put on? You make this decision based on various facts such as what the weather is and what you are doing today. Your life would be very dull if you never had to make any decisions—think what it would be like if it was already decided what shirt you wore each day!

Programs also have to make decisions based on parameters that the program has access to. The computer program would also be very dull if it never made a decision. For example, if a program tries to load a workbook file and the file is not found, a decision needs to be made as to what to do next. Should the program simply display an error message and crash, or should it show some intelligence and alert the user that the file is missing and offer an alternative action?

Artificial intelligence is something that is frequently discussed in computing circles. By making your programs make decisions, you are introducing some artificial intelligence into your program. Admittedly, it is your intelligence that goes into the code, but it tells the program what to do in the event of different circumstances happening. You are effectively writing a set of rules to deal with various situations.

Looping is also something you all do daily without thinking about it. When you eat a meal, you perform the same routine of taking food from a plate and putting it into your mouth. Computer programs frequently loop around the same piece of code a number of times until a certain condition is met



10.2 Decisions

Programs, unless they are extremely simple, usually have to make decisions according to data retrieved or input by the user. Decision making is one of the most important areas of programming, because it specifies what will happen when different events occur.

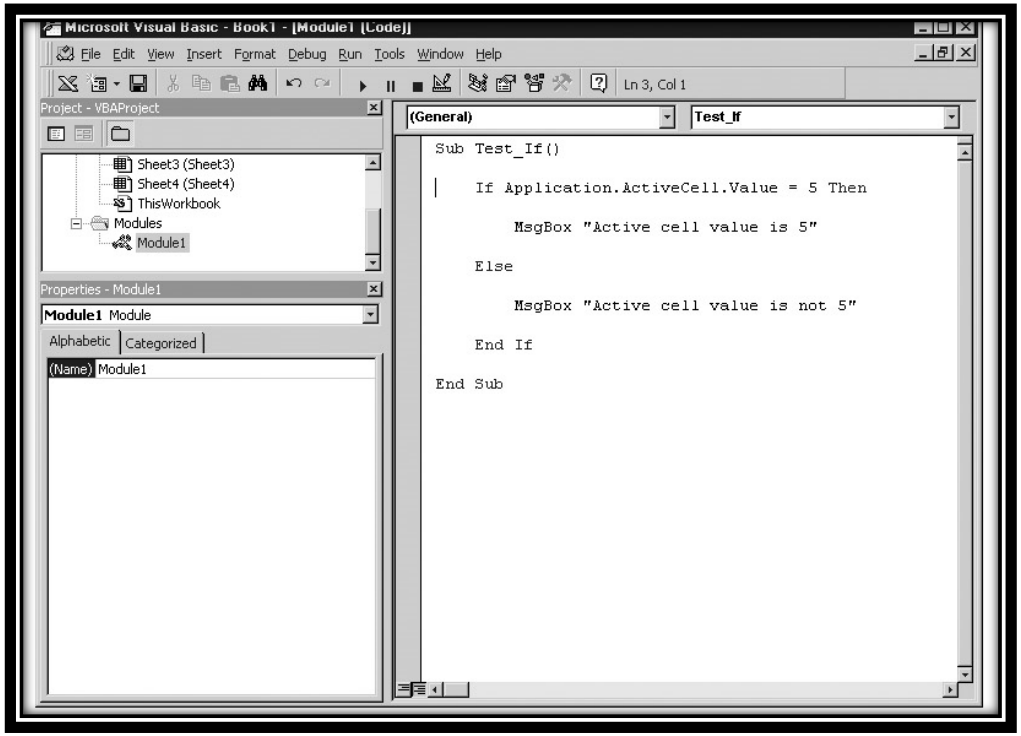
A good example of a common programming decision is IF something is true, THEN do *action1*, or ELSE do *action2*. In everyday life, this might be the statement "IF it is raining, THEN carry an umbrella, or ELSE (if the condition is not met and it is not raining) carry sunglasses."

Here is some sample code to show how the conditional If..Then..Else statement works and how it produces different results. Enter this in the module you created. See for an example of what your code window should look like.

```
Sub test_if()
If Application.ActiveCell = 5 Then

    MsgBox "Cell is 5"
Else

    MsgBox "Cell is not 5"
End If
End Sub
```



This example refers to the active cell within the Excel application object with the line **Application.ActiveCell**, where the cursor is currently positioned. Click the worksheet and enter **5** in the current cell. Make sure that the cursor remains on that cell. Go back to your code window and press F5. You will get a message box saying that the cell is 5. Now go back to the spreadsheet and change the cell to another value or delete the value. Run the macro again and you will get the message box saying the cell is not 5.

The End..If statement shows where the conditional statements finish, or you can put the entire If statement onto one line, which then would not require an End..If, as shown here:

If Application.ActiveCell = 5 Then MsgBox "Cell is 5"

If you have multiple instructions to be executed, you can place the statements on a single line if you separate each statement with a colon. However, this can become very difficult to read and debug and there are often several instructions to be carried out that preclude putting everything on one line.

Conditional operators that can be used are as follows:

Operator	**Meaning**
=	Both numbers or values are equal. This condition will also work for values, such as “dog” and “cat.”
<	First value is less than second value.
>	First value is greater than second value.
<=	First value is less than or equal to second value.
>=	First value is greater than or equal to second value.
<>	First value is unequal to second value.

An expression such as **x=1** is evaluated as a Boolean value, that is True or False or Non-zero or Zero. This means that you do not always have to use an operator—if you are only interested in whether a cell has a non-zero value in it then you can use

If Application.ActiveCell Then MsgBox "Cell has a value"

10.2.1 Multiple Conditional Statements

You can also use multiple conditional statements using a logical operator. Multiple conditional statements are straightforward and work almost like plain English. They use the operators **And** and **Or** and, for the purposes of this example, mean exactly what they mean in English.

If you have two conditions that you want to test, you write the If statement in the following form:

```
If x = 1 And y > 5 Then
    MsgBox "x=1 and y>5"
Endif
```

The message box will be displayed only if both conditions (x = 1 and y < 5) are met. If, for instance, x > 1 but y has a value of 4, the message box will not be displayed. Similarly, you could use the following statement:

```
If x = 1 Or y > 5 Then
    MsgBox "x=1 or y>5"
End If
```

In the case of the preceding **Or**, the message box will be displayed if either one of the conditions is met. For example, if x = 1 or y > 5, the message box will be displayed. Therefore, x could be 0 and y could be 6, or x could be 1 and y could be 4, and the message box would still be displayed in either case.

You can put in several **And**s or **Or**s within the condition, although it gets complicated with more than three. It all depends on what you are trying to achieve in your decision statement and what the procedure is trying to do. You may be writing something very simple such as **If x=1 Then**, or you may be working on a more complicated conditional statement.

10.2.2 Select Case Statements

Another statement available in VBA for conditional processing is the **Select Case** statement. If you have a variable and you want different actions to occur based on the value of that variable, you can use a series of If statements as follows:

```
If x=1 then MsgBox "x=1"
If x=2 then MsgBox "x=2"
If x=3 then MsgBox "x=3"
```

However, this is a good example of where a **Select Case** statement makes the code much cleaner:

```
x = 23
Select Case (x)
    Case (1)
      MsgBox "x=1"
   Case (23)
      MsgBox "x=23"
 End Select
```

The **Select Case** statement provides a simple means to interrogate a specified variable and take action accordingly. The statement **Select Case (x)** defines the variable to be interrogated as **x** and is the start of the block of code for this procedure. **Case (1)** gives the action for if the value is 1—show a message box showing "x=1." **Case (23)** gives the action for if the value is 23—show a message box showing "x=23." Because **x** has been set to **23** at the start of the code, the message box will show "x=23."

You can also include the statements **To** and **Is** in the **Case** statement:

```
Function Test_Case (Grade)
    Select Case Grade
        Case 1
            Msgbox "Grade 1"
        Case 2, 3
            Msgbox "Grade 2 or 3"
        Case 4 To 6
            Msgbox "Grade 4, 5 or 6"
        Case Is > 8
            MsgBox "Grade is above 8"
        Case Else
            Msgbox "Grade not in conditional statements"
    End Select
End Function
```

10.3 Looping

Without looping facilities, programs would be extremely tedious and difficult to maintain. Looping allows a block of code to be repeated until a condition or a specified value is met. Suppose, for example, you wanted to display the numbers from 1 to 5. You could write the program as follows:

```
MsgBox "1"
MsgBox "2"
Msgbox "3"
Msgbox "4"
MsgBox "5"
```

This would work, but it is very inefficient and does not make use of the functionality of VBA. If you wanted to display more numbers, you would have to write more code. If you wanted to display all the numbers up to 1,000, it would require you to add an additional 995 lines of code!

10.3.1 For..Next Loops

This code can be reduced and made easier to maintain by using the For..Next looping statement as follows:

```
For n = 1 to 5
    MsgBox n
Next n
```

The message box will appear five times showing the values of **n** from 1 to 5.

The variable used can be anything—although I used **n** here, it could be a word such as **num**, but it must be consistent throughout the looping process. You could not use **For n = 1 to 5** and then try to use an index called **m**. Also, you must not use a reserved word for the variable name. You can put as many instructions as necessary between For and Next and even call subroutines or functions. The start and end values in the For..Next loop can also be different—they do not have to start at 1 or end at 5.

Step gives extra functionality. You may have noticed that the variable **n** is incremented by 1 each time in the loop—this is the default. You can change this behavior by using the **Step** option. **Step** allows you to specify the size of the increment and also the direction by using the following code:

```
For n = 3 to 12 Step 3
```

```
    MsgBox n
Next n
```

You will get the results 3, 6, 9, and 12, because it works in increments of 3.

To see how **Step** works backward, try this example:

```
For n= 10 to 1 Step -1
    MsgBox n
Next n
```

You will get the results 10, 9, 8, 7, 6, 5, 4, 3, 2, and 1.

For..Next loops are ideal for reading across column numbers or row numbers. You can use a For..Next loop to automatically increment a row number in a cell address.

For..Next loops can also be nested inside each other. For example, if you want to look at each value in a spreadsheet, you can use one For..Next to go across the columns and a second For..Next to go down the rows.

Following is an example that loops through values for **n** and **m**. Notice the indentation of the code; it makes the nesting of the For..Next clear. The **m** loop has been nested inside of the **n** loop so that it will perform the first **n** value, then all values of **m**, then the next **n** value, then all values of **m** again. Indenting helps prevent you from getting lost in your code when you look at it in a month's time.

```
Sub test_loop()
  For n = 1 To 4

    For m = 1 To 5

      MsgBox "n= " & n
      MsgBox "m= " & m
    Next m
  Next n
End Sub
```

10.3.2 For Each Loops

The For Each loop is very similar to a For..Next loop, but it is specifically for use on collections or arrays. For Each allows you to step through each item within the collection or array. You do not use an index (such as **n** in the previous example) because it automatically moves through each item within the collection. This is very useful if you need to search through a collection for a certain object and then delete it because the position in the collection after deletion is maintained in your loop. If you use a For..Next loop with an index and delete the object, the index will be moved up one and your routine will go through one loop too many, causing an error message.
The following example displays worksheet names using a For Each loop:

```
Sub ShowName()
    Dim oWSheet As Worksheet
    For Each oWSheet In Worksheets
        MsgBox oWSheet.Name
    Next oWSheet
End Sub
```

10.3.3 Do Until Loops

The **Do Until** loop keeps looping until a specified condition is met. Often this means waiting for a variable to contain a particular value. When the condition is met, the loop stops, and the program continues executing on the next instruction after the loop. You can also use a While statement so that while a certain condition is met the code will carry on looping. Here is a simple example:

```
Sub test_do()
x = 0
Do Until x = 100
   x = x + 1
Loop
MsgBox x
End Sub
```

First a variable **x** is set to the value 0. The condition of x=100 is then supplied as the criteria for when the **Do** loop should stop. The variable (**x**) is then incremented by 1 each time through the loop, so it loops 100 times until x=100. At this point, it displays a message box giving the value of **x** that is 100.

10.3.4 While..Wend Loops

Finally, there is the While..Wend loop. This continues to loop while a specified condition is true. It stops as soon as the condition is false. Here is a simple example that is very similar to the previous Do Until loop:

```
Sub test_do()
x = 0
While x < 50
   x = x + 1

Wend
MsgBox x
End Sub
```

Again, a variable, **x**, is set to 0. The condition that **x** must be less than 50 is supplied, and **x** is incremented by 1 each time the loop is run. When x=50, it is no longer less than 50, so a message box is displayed showing the value of **x** at 50.

10.3.5 Early Exit of Loops

Under some circumstances, you may want your procedure to exit a loop early before it has worked all the way through and satisfied its criteria. An example might be where you are searching for a particular string of characters within an array. You may have 25 instances of that string to look through, but once the procedure has found what it is looking for, there is no point in further looping until the final condition is met. You could have an array of several thousand records that you are searching through and a lot of time could be wasted in carrying on to the bitter end when the instance has already been found. In the case of a For..Next loop, the value of the index is also preserved, which means that you can use it to locate where your condition was correct. Here is an example:

```
Sub test_exit()

   For x = 1 To 100
      If x = 50 Then
         Exit For
      End If
  Next x
   MsgBox x
End Sub
```

You exit a loop by using an Exit For statement in a For..Next loop or a For Each loop. You use an Exit Do within a Do Until loop. In the case of a For..Next loop, the value of the index is preserved. If the loops are nested, your code will only exit from the loop it is actually in. It will not exit from the outer loop unless you put another Exit statement in. The statement Exit Do and Exit For will stop execution of the loop and go onto the next instruction after the end of that loop.

11 Control Structure (Variables, Arrays, Constants, and Data Types)

11.1 Variables

A variable can have its value changed by the program when running, which is why it is called a variable. The same rules apply from the filing cabinet example, in that you do not mix the data types between the variables. If a variable has been defined as a certain type, it will not accept data specified as another type. For example, if you have defined a variable as an integer (whole) number, you cannot put text into it, or if you put a floating point number (with decimal places) into an integer, you will lose the decimal places.

As your program runs, you often need somewhere to store data temporarily. In the past, macro programmers often stored this data on the spreadsheet itself. With VBA, you could do the same thing and write data to cells on the spreadsheet itself, but this would be inefficient and you, as the coder, would need a very good memory to organize where each piece of data was on the spreadsheet was stored. Also, people do tend to change spreadsheets and someone could easily delete or overwrite your variables, causing your program to crash or give incorrect results.

Instead, you can now use variables to store values while your code is executing. Within a procedure, you declare a variable using the **Dim** statement, supplying a name for the variable:

Dim *variablename* [As type]

Variable names must follow these rules:

- They must begin with a letter.
- They must contain only letters, numbers, or the underscore character—no spaces!
- They must not exceed 40 characters.
- They must not be a reserved word (see the section "Reserved Words" at the end of this chapter).

The optional **As type** clause allows you to define the data type of the variable you are declaring. If you omit the type, it defaults to the Variant data type discussed in the next section.

Dim MyInteger as Integer

11.1.1 Implicit Declaration

You do not have to declare a variable before using it. You can just include the statement

TempVal=6

A variable will automatically be created for **TempVal** as a variant (default type) and it will have the value of 6.

However, a problem with doing this is that it can lead to subtle errors in your code if you misspell the name of the variable in a later statement. For example, if you refer to it as **temval** instead of **tempval**, you know what you mean but VBA does not. It assumes that **temval** is a new variable and assigns it as such. The old variable, **tempval**, is still there but is no longer being used. You now have two different variables, although you think you only have one. This can lead to enormous problems that can take some time to straighten out in your code.

11.1.2 Explicit Declaration

To avoid the problem of misnaming variables, you can stipulate that VBA always generates an error message whenever it encounters a variable not declared. To do this, you'll need to go to the declarations section of the code module. If you look at a module within the VB Editor window, you will see a heading called (General) in the top left of the module window and a heading called (Declarations) in the top right of the module window. Click (Declarations), and you will go straight to the declarations section. Do not worry if it appears that you are not typing into a defined section. Type the following statement. As soon as you type a declaration, a line will automatically appear underneath to show that it is within the declarations section.

Option Explicit

This prevents implicit declarations from being used. Now you have to define **TempVal**:

Dim TempVal

If you refer to **temval** during execution, an error message will be displayed stating that the variable has not been defined.

Note Option Explicit works on a per-module basis—it must be placed in the declarations section of every code module you want it to apply to unless you define the variable as a global variable.

Which method you use (implicit or explicit) depends on personal preference. Coding is often much faster using implicit because you do not have to initially define your variables before you use them. You can simply make variable statements and VBA will take care of the rest. However, as discussed, this can lead to errors unless you have a good memory for variables you are using and have the experience to know exactly what you are doing. Implicit can also make it more difficult for someone else to understand your code. Using Option Explicit is the best practice and helps stop runtime errors.

11.1.3 Scope and Lifetime of Variables

If you declare a variable within a procedure, only code within that procedure can access that variable. The scope is local to that procedure. You will often need variables that can be used by several procedures or even the whole application. For these reasons, you can declare a variable at local, module, or global level.

11.1.4 Local Variables

A local variable uses **Dim**, **Static**, or **ReDim** (arrays only) to declare the variable within a procedure. Several procedures can have a variable called **temp**, but because every variable is local to its procedure they all act independently of each other and can hold different values. Local variables declared with the **Dim** statement remain in existence only as long as the procedure is executing. Local variables declared with **Static** remain in existence for the lifetime of the application

```
Dim TempVal
Static TempVal
```

You can also dimension a variable as an array of several elements, and even several dimensions. An array is almost exactly like a spreadsheet in concept. You can define an array with 10 elements so that it has 10 pigeonholes or cells to store information. You can also give it another dimension so that it is a 10 by 10 array and has 100 pigeonholes or cells to store your information. An array gives you tremendous flexibility over storing data—it is like poking the data into individual spreadsheet cells. For example, if you recursively searched a disk drive for all subdirectories on it, the way Windows or NT Explorer does, then you would need an array to store all the pathnames as they were found so that you could easily find and refer to them within your program.

```
Dim A(3)
ReDim A(10)
ReDim Preserve A(12)
```

To use **ReDim**, you must define the variable initially as an array. **Dim A(3)** creates a small array with 4 elements (0–3), so there are effectively 4 **A** variables. **ReDim A(10)** then makes it an 11-element array but loses all the data in it. **ReDim A(13) Preserve** makes a 13-element array but keeps all existing data. Note all subscripts start at 0 by default.

ReDim is useful when you need an array to store data but you do not know how many elements you will need. For example, if you are recursively searching directories, you have no idea how many will be on a disk device so you start by specifying a small array of 10 elements. As this fills up it can be resized using **ReDim** and **Preserve** to keep the data already in there.

11.1.5 Module-Level Variables

A module-level variable is declared for a particular module. It is available to all procedures within that module but not to the rest of the application. Module-level variables remain in existence for the lifetime of the application and preserve their values.

Dim TempVal

This would be placed in the declarations section of the module instead of an actual procedure on that module.

11.1.6 Global Variables

Global variables are declared in the declarations part of a module with the **Global** statement, but they can be accessed by any code within the application. Global variables exist and retain their values for the lifetime of the application.
Global TempVal
Again, this would be placed in the declarations section of any module. Because you have specified that it is global, it can be accessed for any part of your code.

11.1.7 Name Conflicts and Shadowing

A variable cannot change scope while your code is running. However, you can have a variable with the same name in a different scope or module. You can have a global variable called **temp** and also a local variable in a procedure called **temp**. References to **temp** within the procedure would access the local variable **temp**, and references outside the procedure would access the global variable **temp**. In this case, the local variable *shadows* (that is, is accessed in preference to) less local variables. The only way to use the global variable over the local variable is to give it a different name. Shadowing can be confusing and can produce subtle errors which are difficult to debug. The best way is to use unique names for all variables.

The names of module level and global variables can also cause conflicts with procedure names. A procedure (a subroutine) has global scope unless it is declared privately,. A global variable cannot have the same name as any public procedure in any code module.

11.1.8 Static Variables

Variables also have a lifetime based on their scope. Module and global variables are preserved for the lifetime of the application, which means they hold their values while the application is executing until the user closes the application. Local variables declared with **Dim** exist only when the procedure is executing. When it stops, the values are not preserved and the memory is released. The next execution reinitializes the variables for the lifetime of the procedure. You should only use local variables to hold values that are being used during that procedure. If you expect to access them from other modules, they need to be global. However, you can use the **Static** keyword to declare and preserve a local variable:

Static Temp
You can make all local variables static by placing the **Static** keyword at the beginning of a procedure heading:
Static Sub Test_Static()

12 Modules, Functions, and Subroutines

Modules are where you write your code. Functions and subroutines are the two different ways of creating a piece of working code.

12.1 Modules

Modules are code sheets that are specific to your application. They are not fired off directly by events on the spreadsheet, such as a new worksheet being added or a workbook being closed, but have to be called directly. They are a means of creating procedures in a general manner, rather than specifically running in an object like a workbook or worksheet. You can call them in a number of ways:

- Use a custom menu command or a custom toolbar command.
- Insert a VBA control from the Control toolbox into the spreadsheet directly and attach your code to this; for example, you might enter code for a user's actions on a command button or a combo box.
- Choose Tools | Macro | Macros from the Excel menu, select the macro name from the list and click Run. However, for a professional application this is not recommended. A user will not want to have to select a macro from a list box where they have to know the macro name. Most users want things to happen in the most straightforward way possible, usually through clicking something once.
- Run the code from a UserForm. When the user clicks the OK button on the form, your macro runs and picks up the user preferences.
- Call your code from another macro. Code can form subroutines or functions that can then be used within other macros written on the same spreadsheet. For example, say you have to search a string of text for a particular character and you write a subroutine to do this, using a parameter to pass the text string to the subroutine. You can use this subroutine as a building block by calling it from anywhere else within other procedures in exactly the same way as you would a normal VBA keyword.
- Write your code as a function and call it directly by inserting the function into a cell, just as you would with the built-in functions in Excel.
- Click directly on your code and press f5. This is for development work only. For example, if you are working on a subroutine in isolation, you may wish to run it only to see how it works.

A VBA project normally uses at least one module to store the necessary functions and subroutines known as procedures. To insert a new module, simply select Insert | Module from the VBE menu, and the new module will appear. Note that this contains only a general area initially. There are no events on it, as there were on the workbook and worksheet code sheets.

You can enter subroutines or functions here and make them public or private. The distinction between public and private is to decide whether other modules within the same workbook can access the procedure. If the code is private, it can only be used in the current workbook where it resides. If it is public, it can be used by any other procedure in any other module in the workbook. If you have a subroutine that you do not want to be used elsewhere in the code, make the subroutine private. The default is always public.

12.2 The Difference between Subroutines and Functions

There are two types of code procedures: subroutines and functions, and on casual inspection they both appear to look the same. However, they actually are different.

A subroutine is a piece of code that performs a set of actions or calculations or a combination of the two. It can form a "building block" within a program and may sometimes need to be repeated. It can be called by several different routines. The programmer has to write a subroutine only once, and it can be called from anywhere within the program as many times as needed. However, it does not return a value; if it performs a calculation, there is no direct way of finding the result. It is called by inserting a **Call** instruction into the code, as shown here:

```
Sub Main()
    Call MySub      'Calls another macro/procedure called MySub
End Sub
```

You do not have to use the word **Call** to use the subroutine **MySub**. The following example also works:

```
Sub Main()
    MySub      'Calls another macro/procedure called MySub
End Sub
```

A function is exactly like a subroutine except that it returns a value. Functions start with **Function** (instead of **Sub**) and end with **End Function** (instead of **End Sub**). This means that, generally speaking, functions should be called by using a variable, to accept the return value:

```
x=Now()
```

The variable **x** will contain the value of today's date. This is a very simple example of calling a built-in function.

There are many other built-in functions that can also be used in this way. You cannot use the Excel formula functions, but many of them are already built into VBA as part of the language. Functions that you write can be used within spreadsheet formulas.

Both subroutines and functions can have parameters or values passed to them. These are passed inside parentheses.

12.3 Writing a Simple Subroutine

A subroutine is different from a function in that it does not return anything directly and so cannot be used directly in the spreadsheet the way a function can. A subroutine is usually a building block that forms a piece of code that is going to be called many times, possibly from different points from within your program. This is one of the great flexibilities of a subroutine. When it is called, the return address (from where the subroutine was called) is stored. When the subroutine finishes running, control is passed back to the return address. You can still pass parameters to it, but these are used internally within the code itself.

Click back to Module1 and add the following code:

```
Sub Display(target)
    MsgBox target
End Sub
```

Note that this subroutine has an argument parameter for a variable called **target**. This is because you are going to call the subroutine from another procedure and pass a variable across.

A line is drawn to separate off the new subroutine, and the subroutine that you have written is automatically added to the pull-down in the top-left corner. Click the **This Workbook** object and return to the initial Hello World example and on the **Workbook_Newsheet** event, add the following code:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
'MsgBox "Hello World"
x = Multiply(3, 5)
MsgBox x
Call Display("my subroutine")
End Sub
```

Now, click the Excel worksheet and choose Insert | Worksheet from the menu. You will see the message box showing 15 followed by a message box showing “my subroutine.”
The **Call** command calls your subroutine and passes any required parameters to it. It then executes the code in the subroutine and returns to the next instruction following the **Call** statement. In this particular case, it passes the string “my subroutine” into the variable called **target**.
If the subroutine that you have written does not use parameters (arguments), you can run it from the code page by selecting Run | Run Sub | UserForm from the VBE (Visual Basic Editor) menu, pressing F5, or clicking the Run symbol on the toolbar. The cursor must be on the subroutine you intend to run. This is a useful way of testing the code you have written and seeing if there are any bugs in it.
Subroutines are a useful way of breaking large projects down into manageable pieces so that you do not end up with enormous, cumbersome routines. It is far easier to break a problem into constituent parts and work separately on each section, making sure you get that section working properly before moving onto other parts. The alternative is to write a large chunk of code, which inevitably leads to unnecessary duplication.

12.4 Writing a Simple Function

The object of this exercise is to create a function to accept two numbers and multiply them together and return the result. The function will have the name **Multiply**. The following table cites the four main mathematical operators that you will use when writing functions and subroutines in VBA.

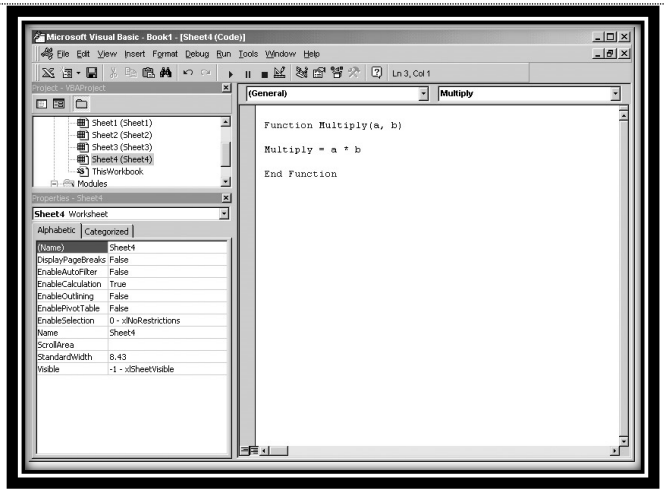
Add	+
Subtract	–
Multiply	*
Divide	/

The code for this function is as follows:

```
Function Multiply(a, b)

    Multiply = a * b

End Function
```



The header introduces two parameters, **a** and **b**, by showing them in parentheses after the title of the function. A comma separates the two arguments. These arguments represent the two numbers to be multiplied—they could be called anything, as long as the variable name is consistent throughout the function.

The name of the function is **Multiply**, and this is used as a variable to return the answer. This is the only way to return the answer back to the routine that called the function. Note that the name of the function now appears in the pull-down on the top-right of the code window. This is because it is now an official function within both your VBA code and Excel.

You can now use this function in two ways: by calling it directly on the spreadsheet as a function or by using it within your VBA code. To call it directly on the spreadsheet, you do so in the same way that you would use any other function—for example, **SUM**. That's right, you have just written your first practical extension to Excel, and it's all your own work!

Click the spreadsheet and enter **=multiply(3,4)** into a cell. The answer 12 will appear in the cell. You can see how easy it is to write your own formula into Excel. The difficult calculation that you put together last week can now be turned into a custom function and used like any other Excel function.

Now, for the second way to use the function: calling it from within your VBA code. For the sake of simplicity, next you will call your new function from the same event that you called the Hello World example from.

Click the **ThisWorkbook** object and return to the initial Hello World example. Turn the “Hello World” statement into a comment by putting a single quote (') character before it and enter the following code so that it looks like this:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    'MsgBox "Hello World"
    x = Multiply(3, 5)
    MsgBox x
End Sub
```

Note that when you type the word **Multiply** and open the brackets, VBA automatically displays the parameters it is expecting by name. By inserting the function into the code, you are forcing a call to that function using the parameters **3** and **5** to replace **a** and **b** in your function. The result is returned in the variable **x**.

Now click the worksheet and choose Insert | Worksheet from the menu. A message box will appear with the answer of 15.

12.5 Public and Private Functions and Subroutines

VBA allows you to define your functions or subroutines as public or private using the keyword **Public** or **Private**. For example:

```
Private Sub PrivateSub()
End Sub
```

Any subroutines or functions that you create are public by default. This means that they can be used throughout the modules within your application, and spreadsheet users will find the subroutines available as macros by choosing Tools | Macro | Macros from the spreadsheet menu. They will also be able to access public functions in your code and will see them listed in the custom section if they click the formula icon on the spreadsheet toolbar.

There is one exception to this: UserForms. UserForms represent dialog forms and have their own modules. A public subroutine or function on a UserForm can be called from other modules within your code by referencing the form object—for example, **UserForm1.MysubRoutine**—but it will not appear in the macro or function list on the spreadsheet, and any function written on a UserForm cannot be used on the spreadsheet itself.

Also, if you create an Excel add-in, public procedures within that add-in can still be accessed by other modules by referencing the add-in object. This happens even if the code for the add-in is password protected. This can have advantages if you wish to write an add-in of your own procedures for use by other VBA programmers without letting them see how you did it! The disadvantage is that others can access your public procedures when you do not want them to. If the add-in is loaded, the public procedures within it are available to all modules.

Using private declarations, you can have procedures that have the same names but are in different modules. That procedure is private to that module and cannot be seen by other modules and, more importantly, cannot be seen and run by the spreadsheet user. This can cause confusion both for the programmer and for VBA. Which one does VBA choose to invoke if you call that procedure? Fortunately, VBA has a set of rules it uses for this. VBA first looks in the current module where the code is executing. If it cannot find a procedure of that name there, it then scans all modules for the procedure. Calls within the module where the private procedure is defined will go to that procedure. Calls outside that module will go to the public procedure.

12.6 Argument Data Types

When you specify arguments for a procedure, they always default to Variant, which is the default variable type in VBA. You can also declare your parameters as other data types based on the data types.

The advantage of declaring with data types is that they introduce a discipline to your code in terms of what information the procedure is looking for. If you do not specify a type and use the default Variant, then your procedure will accept anything, be it a number or a string. This could have unfortunate consequences within that procedure if you are expecting a string and a number gets passed across or vice versa. If you specify the parameter as a string, an error will occur if you do not pass a string:

Function (Target as String)

This can also be useful if you are writing a custom spreadsheet function. When users enter your function into a cell, they must give the parameters according to the data type specified. If a string is specified, then they must put the value in quotes, or it must refer to a cell holding a text value.

12.7 Optional Arguments

You can make specific arguments optional by using the **Optional** keyword:

Function Myfunction (Target as String, Optional Flag as Integer)

In this example, the user has to specify the parameter **Target** as a string, but the parameter **Flag** will appear with square brackets around it and need not be specified. All optional parameters must come after the required ones.

12.8 Passing Arguments by Value

You can also use the **ByVal** or **ByRef** keyword to define how parameters are passed to your procedure:

```
Function MyFunction (ByVal Target as String)
```

The **ByVal** keyword ensures that parameters are passed by value rather than by a reference to a value from a variable. Passing a value by reference can easily allow bugs to creep into your code. To demonstrate this, if you pass by reference using a variable name, the value of that variable can be changed by the procedure that the variable is passed to—for example,

```
x = 100
z = Adjust(x)
Function Adjust(ByRef Target as Integer)
```

The variable **x** will be modified by what is going on within the function **Adjust**. However, if you pass by value, as shown here, only a copy of the variable is passed to the procedure:

```
x = 100
z = Adjust(x)
Function Adjust(ByVal Target as Integer)
```

If the procedure changes this value, the change only affects the copy and not the variable itself. The variable **x** will always keep its value of 100.

Normally, you would not expect a function to modify an argument, but it can happen and can lead to hard-to-find bugs within the code. To avoid these effects, use the declaration **ByVal**.

13 Debugging & Error Handling

13.1 Debugging

Whenever you write code, there is a chance that bugs will exist that can cause a program failure, the program to hang, or simply unexpected results. In my opinion, this is the area that sorts out the true analytical programmers from people who just type in code and hope it will work.
There have been many reports written on "computer rage," where users of an application get upset because the results are different from what they expected. All I can say to those people is wait until you work on code for an application like Excel! You will then realize how straightforward and well put together this application is. Once you have tried your hand at fixing a few bugs in what appears to be a simple program, you will appreciate what goes on behind the scenes when you make a menu selection or press an OK button.

13.2 Types of Errors

Errors occur very easily when you are writing code. This section gives you examples of the types of errors you can expect to see.

13.2.1 Compile Errors

Compile errors result from incorrectly constructed code. You may have used a property or method that does not exist on an object, or put in a For without a Next or an If without an Endif. When you run code, the compiler goes through the code first and checks for these types of errors. If any are found, the code will not be run and an error message will be displayed referring to the first error found. Note that there could be several compile errors in a procedure, but only the first one will be flagged. You might correct the error and think, "I've fixed it now," and then rerun the procedure, and up comes another one! This can be very frustrating to fix, but the real answer is to obey the rules of coding in the first place.
These are the types of errors that appear when the code is compiled, and they are often referred to as design-time or compile-time errors.

13.2.2 Runtime Errors

These are errors that occur when your program is running. You could, for example, try to open a file that does not exist, or attempt a division by zero. These would create an error message and halt execution of the program. They would not show up at compile time because they are not breaking any programming rules, but they will cause the code not to run.

13.2.3 Logic Errors

Logic errors occur when your application does not perform the way you intended. The code can be valid and run without producing any errors, but what happens is incorrect. For example, the user could have two workbooks open, each with a Sheet1 worksheet in them. You write code in a module referring to Sheet1 but not specifying which workbook collection it resides in.

These are by far the most difficult errors to locate. They can require a lot of painstaking searching to find, even using all the debugging tools at your disposal. It is very easy to keep looking at a few lines of code and thinking, "There is nothing wrong with this, it should give the right answer." It is only by looking at each line in turn that you can see what is going wrong, and usually, when the problem hits you in the face, you will suddenly realize what a simple mistake it was. You may also find a situation that you did not envisage when you designed your code.

13.3 Errors and the Error Function

Runtime errors can creep into code very easily, through no fault of the programmer. The user does something outside the scope of the code and causes an error message to occur that stops all execution of code. This may occur where you are accessing an external data source such as a database. The user may also take an action that the programmer never envisaged.

No matter how carefully you try to provide for all conditions, there is always a user who does something that you never thought of and effectively breaks the set of rules that you have written. It may be as simple as including a name with an apostrophe in such as O'Brien. If you use this name in a SQL query string, it will cause problems in the way it is handled.

Another example is reading a file in from a disk device. The programmer may allow the user to select the drive letter for the file to be read. You assume that the files will mainly come from network drives, and an A: floppy drive option is just nice to have, but it will probably never be used. However, if the user selects A: (floppy disk drive), then there is a possibility that there will not be a disk in the drive. This will create an error that will stop execution of your program. The user will be very unhappy and will lose a great deal of faith in your application. The error needs to be trapped, and an appropriate action needs to be taken from within the VBA code.

Error handling is also used on common dialog forms to show that the Cancel button has been clicked. In order to test whether the user clicked the Cancel button, you had to set the **CancelError** property to True and then put in an **On Error** statement to direct the code where to go for an error.

Try this simple example without a disk in drive A. Place the code into a code module and then run it by pressing F5:

```
Sub Test_Error()
    temp = Dir("a:\*.*")
End Sub
```

This will produce an error message saying that the A drive is not ready. In normal terms, your program has crashed and will not work any further until this error is resolved. Not very good from the user's point of view!

You can place a simple error-trapping routine as follows:

```
Sub Test_Error()
    On Error GoTo err_handler
        temp = Dir("a:\*.*")
    Exit Sub
    err_handler:
    MsgBox "The A drive is not ready" & "    " & Err.Description
End Sub
```

The first line sets up a routine to jump to when an error occurs using the **On Error** statement. It points to **err_handler**, which is a label just below the **Exit Sub** line further down that will deal with any error condition. The purpose of a label is to define a section of your code you can jump to by using a **GoTo** statement.

The line to read the A drive is the same as before, and then there is an **Exit Sub** line, because if all is well you do not want the code continuing into the **err_handler** routine.

If an error happens at any point after the **On Error** line, the code execution jumps to **err_handler** and displays a message box that says drive A is not ready. However, you may have noticed that the code execution jumps to **err_handler** when *any* error occurs, not just the drive not ready error. An error could occur because you made a mistake in typing this code in. This could have unfortunate consequences in your code.

Fortunately, you can interrogate the error to find out what went wrong. You can also use the **Err** object to give the description of the error and to concatenate it into your message so that it also says "Drive not ready." You do this using the **Err** function. This will return the number associated with the runtime error that happened. You can add this into the previous example as follows:

```
Sub Test_Error()
    On Error GoTo err_handler
    temp = Dir("a:\*.*")
    Exit Sub
    err_handler:
    If Err.Number = 71 Then
       MsgBox "The A drive is not ready"
    Else
       MsgBox "An error occurred"

    End If
End Sub
```

You saw from the first example that the number for "Drive not ready" came up in the error message box as 71. The program looks at **Err** (a system variable that holds the last error number) and checks to see if it is 71. If it is, it displays the message box, "The A drive is not ready"; if it is not, it displays the message box, "An error occurred."

14 Spreadsheet Modeling (Introduction & Techniques)

14.1 Introduction

What is a financial model? What is the difference between a financial model and the spreadsheet solutions you create or VBA programs you write all the time to answer financial questions or solve financial problems? A simple, practical answer is that a financial model is designed to represent in mathematical terms the relationships among the variables of a financial problem so that it can be used to answer "what if" questions or make projections. Some of the spreadsheet solutions that people create capture some of these relationships as well and, therefore, can answer "what if" questions to some extent. But because they are not primarily designed with these objectives in mind, they do not try to capture as many of these interdependencies as possible, and their structures often make it cumbersome to answer "what if" questions or make projections with them.

"**P**art of corporate modeling"

- concerned with the construction and use of computer - based models
- to carry out the calculations necessary to produce financial models
- based on given sets of assumptions and predictions.

Financial modelling covers a wide area from simple sheets to add up expenses to sophisticated risk modelling for projects. While there are aspects of design to be considered, the financial aspects could cover:

- developing specialist programs which answer specific business problems,
- e.g. cash flow cover and variability;
- analysing and processing data;
- modelling the future or a considered view of the future;
- processing data quickly and accurately into management information;
- testing assumptions in a 'safe' environment, e.g. project scenarios;
- supporting management decision making through a structured approach;
- understanding more precisely the variables or rules in a problem;
- learning more about processes and behaviour of variables;
- discovering the key variables and their sensitivity.

14.2 Tips for basic structure and design

Design is personal and you develop a style over time. It is important to be consistent and follow a clear methodology. The stages discussed in this chapter are not exhaustive and include the following: Follow the design process and method for all models.

- Set aims and objectives.
- Examine user needs and required user interface.
- Set out key variables and rules.
- Break down the calculations into manageable groups.
- Produce the individual modules.
- Menu structure.
- Management reports and summaries.
- Development, e.g. sensitivity.
- Testing and auditing.
- Protection as an application.
- Documentation.
- Ask for peer group comments.

14.3 Basics of Design

Design is personal and you develop a style that you approve of, like and can repeat easily. This may sound simplistic, but a sound methodology cuts development time and error correction. While there are degrees of planning needed depending on the complexity of the application, you have to have a plan and a method for different sorts of spreadsheets.
How many times do you insert and delete columns or rows or at a later stage wonder how a particular cell formula works? It is easy to start keying formulas without thinking too carefully. The objective is to set out a tick list of considerations for superior design. Follow a design process and method on all models and make the sheets follow a pattern. The examples with this book unashamedly follow exactly the same layout and design. While simple spreadsheets may be sufficient for one person, models should conform to simple rules, especially when used by others or are incorporated into decision making. In its basic form, this means splitting the functions in the model between inputs, calculations and outputs.

14.4 Objectives

Many people do not think through the aims and objectives. Although it sounds simplistic, it is a good idea to write them down somewhere in the documentation and refer to them during development to make sure that you do not deviate from the original aims. In many examples, it is difficult to work out where the answer is as it is hidden in the calculations. Models are often capable of providing more information. For example, a simple cash flow budget could also use further sheets for recording the actual profit and loss and balance sheet. With both budget and actual figures, variance reports based on absolute and percentage differences are possible together with management reports and graphs for divisional reporting.

14.5 User Interface

This needs to be reviewed critically since this is what you and your users will work with. There may be a number of different audiences for the same model with different requirements for inputs, detail and information. Older models sometimes put the variables on the left between the label and the figures, for example the tax rate could be placed here.
However, users may like to see all the inputs in one place and be directed as to what and where to input data. It is very frustrating for somebody who has received a copy of a new application to have to spend time understanding how it works and where to enter data. Visual Basic programming works by designing the interface of forms first and then attaching code to buttons and controls to make it work. This is not a bad analogy for Excel, since many authors have not placed themselves in the user's shoes and critically examined the user's perceptions. The interface should be:

- intuitive
- clear
- guide the user through a logical flow of information.

The use of borders, colours and formats assists this process as in the calculator shown in Figure 2.2 (Calculator.xls). The user is directed to enter variables and press buttons to calculate an answer as with a hand-held financial calculator such as an HP17BII. The answer is updated at the bottom based on the button the user pushes so that the flow of information is from top to bottom.
User interface

14.6 Key Variables And Rules

Variables and rules should be broken down and variables must be placed together as in the calculator above. It is essential that variables are not hard coded. For example, the frequency must be a user input. Otherwise what would the user change if the payments were monthly as opposed to quarterly?
Distilling out the rules means that the author becomes better organized and may understand the process of solving the business problem more succinctly. The process may also uncover new variables which need to be modelled. Rules are also important: corporate taxes are complex in most jurisdictions and models need to reflect exactly tax shields and tax settlement dates. The corporate tax payment method is changing in the UK to a four-quarter payment system from once a year and this presents the modeller with a fresh set of challenges to understand both the transitional and final arrangements. Using names for the main variables and a modular approach assists with simplifying the maintenance of existing models.

14.7 Layout

Breaking down the calculations into manageable groups shows workings and results clearly. Modern Excel allows separate sheets in one two-dimensional workbook rather than attempting to link a series of separate files as was the norm with the original Lotus 1-2-3 and Excel. Rather than placing a profit and loss, balance sheet and cash flow on the same sheet, it is surely more logical to put these three facets on separate sheets in a single file.
The example in Figure 2.3 breaks up the layout into:

- user inputs;
- management summary – visible on updating inputs. This saves the user from scrolling to the answer;
- calculations area using variables from above inputs area;
- answer;
- area for sensitivity, graphics or other detail;
- workings area outside the printing area.

The flow of information through the model follows a logical pattern with the inputs in the top left where a user would expect to find them. Models that are more complex would place these areas on different worksheets, but again inputs and calculations should not be mixed and development should be split into logical sections.
As in Figure 2.3, the use of colours, typefaces, patterns and borders for different data and information in a consistent manner can assist to show the logical framework. The models in this book follow this format.
Design introduction

14.8 Individual Modules

Individual modules can then be produced within a planned framework and calculations broken up into separate areas or sheets. The layout is important for user and author understanding and it is critical for ease of further development. Calculation areas must contain only formulas and they must not be mixed with numbers. This is to ensure the integrity of the calculations. For Fexample, multiplying by 0.3 for corporation tax will only cause problems if the tax rate were to change since you would have to search and replace
through all the sheets in a file and in the Visual Basic macro code. Using an input cell as a range or a named cell means that you can be confident of only changing one cell for the whole file to accurately update itself.

14.9 Menu Structure And Macros

A menu structure is useful in complex models, since it:

- forces a structure to the model;
- makes it easier for a user to understand;
- facilitates easier navigation using buttons rather than tabbing along sheets.

The model in Figure 2.4 (Menu_Structure.xls) uses buttons or a combo box to access two other sheets called Inputs and Reports. The Inputs and Reports sheets include buttons to take the user back to the Menu. These features are discussed in more detail in the next chapter. A user can see immediately what sheets are available and can be guided to where data is required.

14.10 Management Reporting

Management reports and summaries are normally required for larger models as they would be in a full management report. Not everybody needs all the detail and calculations, and summaries assist the user in understanding the results and the important outcomes. For example, a project management application could demonstrate the coverage ratios and the degree of security in the model.

14.11 Future Development

Development within a model is important: a budget model may need further variables in the next year and a structured model aids future development. The test is: see how new variables could be added and check the disruption in the design. Alternatively sensitivity tables and scenarios allow a user to produce multiple answers within the same model and test the variance based on changing inputs. A single net present value is not enough for informed decision making and development should include some further testing of how variables 'flex' the eventual results.

Risk may also be a decisive factor and therefore the design of a model may need to allow for risk or simulation techniques. Simulation involves developing models to allow for a range of inputs rather than single point figures, which produce a range of outputs. Similarly, graphs can be useful in demonstrating the answer to management or other audiences. People often grasp complex ideas more easily through pictures. For example, a cash flow model could include a coverage graph of the cash coverage above a minimum limit.

14.12 Testing

Testing is required to ensure that there are no mathematical errors and that the information flow through the model is correct. The calculator in Figure 2.2 can be tested against discount rate tables or the output from another financial calculator. Test data is required, which makes uses of all the buttons, inputs, frequencies and payment types. A later chapter outlines a number of techniques for reviewing the accuracy of model output.

14.13 Protection

Protection is beneficial if a model is given to others. This is simple if the author clusters all the inputs together and colour codes them. Entire sheets can be protected and then the input cells can be unprotected in blocks. Protecting sheets and workbooks preserves the author's work and ensures that the application is used as intended. For example, if a budget model were given out to a user who then overwrote cell formulas with numbers, then the integrity of the model is threatened and one would have to begin by checking every cell for possible changes.

14.14 Documentation

Many authors do not bother to write notes about a spreadsheet and its construction. This is risky since either they or their colleagues may have difficulty at some point in the future in maintaining the code. Many models may start as 'pet projects' and, as with any other computer program need background information. Ideally, notes should be in the model rather than on scraps of paper in a file and show:

- reasons for adopting a particular design or template;
- outline key formulas and calculations;
- rules and methodology.

14.15 Peer Group Comments

Users or colleagues can often make constructive suggestions and although this process is often painful after you have spent time on producing a masterpiece, potential users need to attempt to enter data and be comfortable with how a model operates. Users involved in the design process and asked for their opinions may be more enthusiastic users. The main factors are:

- ease of use with a clear interface;
- user guidance from inputs through calculation to answers and reports;
- complexity reduced to a minimum for audit and checking purposes;
- answers shown clearly.

The above 13 points will help you produce more organized work. Review some of your own models and see how many of these points you include regularly in your applications. Obviously, the degree of complexity affects how much you need to do. However, this represents good practice which the author has developed over a number of years.

The next chapter discusses a number of features to make your models more powerful and the following chapter applies the design methodology to the original example in Chapter 1. The objective is to show how applied Excel leads to more powerful and error-free models.

Design introduction

14.16 Summary

Design is personal and you develop a style over time. It is important to be consistent and follow a clear methodology. The stages discussed in this chapter are not exhaustive and include the following:

- Follow the design process and method for all models.
- Set aims and objectives.
- Examine user needs and required user interface.
- Set out key variables and rules.
- Break down the calculations into manageable groups.
- Produce the individual modules.
- Menu structure.
- Management reports and summaries.
- Development, e.g. sensitivity.
- Testing and auditing.
- Protection as an application.
- Documentation.
- Ask for peer group comments.

15 Microsoft Excel Features and Tools

The basics of design revolve around planning and logic whereas this chapter concentrates on a list of features that can be included to make models more user-friendly. This is not an exhaustive list, but aims to show the difference between the original and the finished model. The features in this chapter are:

The basics of design revolve around planning and logic whereas this chapter concentrates on a list of features that can be included to make models more user-friendly. This is not an exhaustive list, but aims to show the difference between the original and the finished model. The features in this chapter are:

- formats;
- number formats;
- lines and borders;
- colour and patterns;
- specific colour for inputs and results;
- data validation to control inputs;
- controls – combo boxes and buttons;
- conditional formatting to illustrate changes in data;
- use of functions and types of function;
- add-ins for more financial functions;
- text and updated labels;
- recording a version number, author, development date and other information;
- using names to make formulas easier to understand;
- pasting a names table as part of documentation;
- comment cells;
- graphics and charts;
- dynamic graphs to plot individual lines;
- data tables for sensitivity;
- scenarios for _what-if' analysis;
- goal seek for simple targeting;
- solver for optimization and targeting;
- use of templates to speed up development.

16 Automating your model with macros

16.1 What is a Macro, Anyway?

A macro is a collection of commands that are performed in a set order. A macro enables you to repeat operations that you would normally do by hand, but it is much faster, and when written correctly, much more reliable. Often, a macro will do in seconds what takes hours or days by hand. It can also perform tasks that are physically impossible manually. If you find you are performing the same commands or actions over and over again, in exactly the same sequence, you can create a macro to record all those actions for you. You can then assign the macro to a button and then run the macro using a single click and even assign the macro to a keyboard command.

16.2 What can a Macro do for me?

There are loads of analytical situations where macros can save time and increase accuracy. They might not necessarily be built and saved within financial models, but they could be used in the collection of the data that goes into model. Here is a small sample of the kinds of situations I've worked on recently where macros come in handy for financial modelling and analysis:

- Fifty identical budget templates have been created, and you discover a formula needs to be changed. Instead of making the change fifty times, record a macro – it will be much quicker, and far less prone to error.
- A non-profit organisation sets their pricing so that all costs are fully recovered to break even. The costs keep changing though and the user is a senior account manager who doesn't know how to do a goal seek in Excel. The modeller creates a simple macro using a goal seek so that all the user needs to do is change the costs and press the button to find out how much the pricing needs to change to under the new costing.
- A dump of information containing several thousand rows is exported from a database into Excel every day. The data needs to be formatted and manipulated manually to be used in a daily takings report. Automating this with a macro can literally save hours of manual data manipulation.
- A reporting model is built using a pivot table, but when new data is entered, the user sometimes forgets to refresh the pivot table. The modeller builds a button to refresh a pivot table every time it's pressed.

Automating Your Models with Macros

16.3 Macros security level

Macros have the potential to cause serious damage to your computer, such as erasing files or installing malware. Consequently, Microsoft has added macro-security features to help prevent macro-related problems.
The Macro Settings section of the Trust Center dialog box. By default, Excel uses the Disable All Macros with Notification option. With this setting in effect, if you open a workbook that contains macros (and the file is not digitally ―signed‖), the macros will be disabled, and Excel displays a Security Warning above the Formula bar (see Figure 38.3). If you're certain that the workbook comes from a trusted source, click the Enable Content button in the security warning area, and the macros will be enabled. Excel remembers your decision; if you enable the macros, you won't see the Security Warning the next time you open that file.
Excel displays a Security Warning if a workbook contains macros. Rather than deal with individual workbooks, you may prefer to designate one or more folders as ―trusted locations.‖ All the workbooks in a trusted location are opened without a macro warning. You designate trusted folders in the Trusted Locations section of the Trust Center dialog box.
Saving Workbooks That Contain Macros If you store one or more VBA macros in a workbook, you must save the file with an XLSM extension.
The first time you save a workbook that contains macros (or even an empty VBA module), the file format defaults to XLSX — and this format can't contain macros. Unless you change the file format to XLSM, Excel displays the warning shown in Figure 38.4. You need to click No, and then choose Excel Macro-Enabled Workbook (*.xlsm) from the Save As Type drop-down list in the Save As dialog box. Excel warns you if your workbook contains macros and you attempt to save it in a nonmacro file format.

16.4 Recording and running simple macros

Excel 2013 enables you to add an optional Developer tab to the Ribbon that contains its own Record Macro command button (among other command buttons that are very useful when doing more advanced work with macros). To add the Developer tab to the Excel 2013 Ribbon, follow these two steps:

1. Choose File→Options or press Alt+FT to open the Excel Options dialog box.
2. **Click the Customize Ribbon tab, select the Developer check box under Main Tabs in the Customize the Ribbon list box on the right side of the dialog box, and then click OK.**

Even if you don't add the Developer tab to the Ribbon, the Excel Status bar at the bottom of the Excel 2013 program window contains a Record Macro button. You click this button to turn on the macro recorder. Also, the View tab contains a Macros command button with a drop-down menu containing a Record Macro option.

When you turn on the macro recorder either by clicking the Record Macro button on the Status bar, clicking the Record Macro option on the Macros button's drop-down menu (Alt+WMR), or clicking the Record Macro button on the Developer tab (Alt+LR), the macro recorder records all your actions in the active worksheet or chart sheet when you make them.

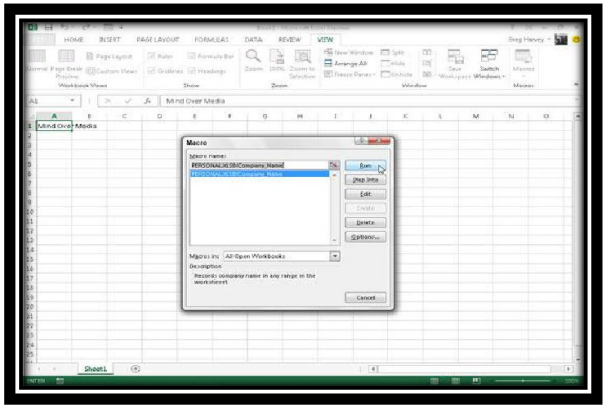
The macro recorder doesn't record the keystrokes or mouse actions that you take to accomplish an action — only the VBA code required to perform the action itself.

This means that mistakes that you make while taking an action that you rectify won't be recorded as part of the macro; for example, if you make a typing error and then edit it while the macro recorder is on, only the corrected entry shows up in the macro without the original mistakes and steps taken to remedy them.
The macros that you create with the macro recorder can be stored as part of the current workbook, in a new workbook, or in a special, globally available Personal Macro Workbook named PERSONAL.XLSB that's stored in a folder called XLSTART on your hard drive.
When you record a macro as part of your Personal Macro Workbook, you can run that macro from any workbook that you have open. When you record macros as part of the current workbook or a new workbook, you can run those macros only when the workbook in which they were recorded is open in Excel.
When you create a macro with the macro recorder, you decide not only the workbook in which to store the macro but also what name and shortcut keystrokes to assign to the macro that you are creating. When assigning a shortcut keystroke to run the macro, you can assign

- The Ctrl key plus a letter from A to Z, as in Ctrl+Q
- Ctrl+Shift and a letter from A to Z, as in Ctrl+Shift+Q

You can't, however, assign the Ctrl key plus a punctuation or number key (such as Ctrl+1 or Ctrl+/) to your macro.

After you record a macro in Excel 2013, you can run it by clicking the View Macros option on the Macros button's drop-down menu on the View tab, the Macros button on the Developer tab of the Ribbon, or by pressing Alt+F8 to open the Macro dialog box.
As you can see, Excel lists the names of all the macros in the current workbook and in your Personal Macro Workbook (provided you've created one) in the Macro Name list box.
Simply click the name of the macro that you want to run and then click the Run button or press Enter to play back all its commands.



If you assigned a shortcut keystroke to the macro, you don't have to bother opening the Macro dialog box to run the macro: Simply press Ctrl plus the letter key or Ctrl+Shift plus the letter key that you assigned and Excel immediately plays back all the commands that you recorded.
The reason that macros you record in the Personal Macro Workbook are always available in any Excel workbook is because the PERSONAL.XLSB workbook is also open — you just don't know it because Excel hides this workbook immediately after opening it each time you launch the program.

As a result, if you try to edit or delete a macro in the Macro dialog box saved in the Personal Macro Workbook, Excel displays an alert dialog box telling you that you can‘t edit a hidden workbook.

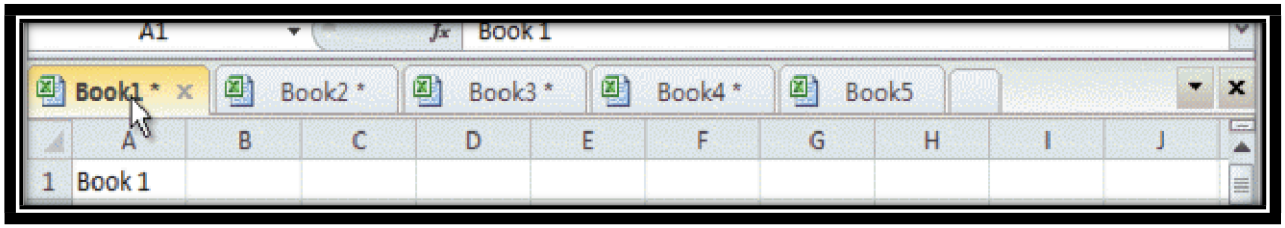
To unhide the Personal Macro Workbook, first clear the alert dialog box and close the Macro dialog box; then click the Unhide button on the View tab (Alt+WU) and click the OK button in the Unhide dialog box while PERSONAL.XLSB is selected.

Excel then makes the Personal Macro Workbook active, and you can open the Macro dialog box and edit or delete any macros you‘ve saved in it. After you finish, close the Macro dialog box and then click the Hide button on the View tab (or press Alt+WH) to hide the Personal Macro Workbook once more

17 Protecting your model from undesired changes

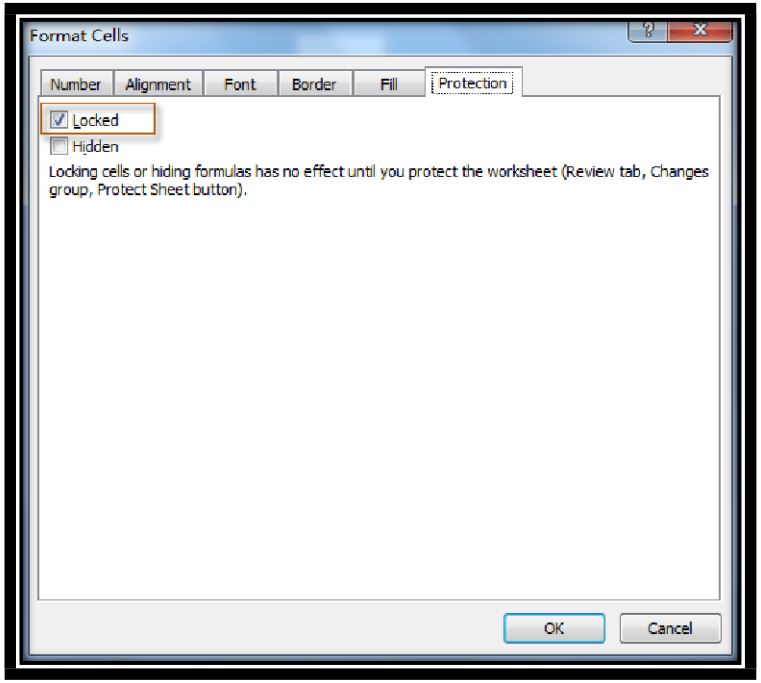
17.1 Model structure and locking cells

It's easy to lock and protect the whole worksheet or workbook with clicking the Protect Sheet button or Protect Workbook button under Review tab. However, sometimes you may need to lock and protect only specific cells or selections in a sheet. How would you like to do? This article will guide you to lock and protest selected cells or ranges in Excel with following steps:



Step 1: Select all cells in current worksheet with pressing the Ctrl key and A key together.
Step 2: Right click, and select the Format Cell item from the context menu.
Step 3: In the Format Cells dialog box, uncheck the Locked option under Protection tab, and click OK button. See the following screen shot:

Protecting Model from Undesired Changes

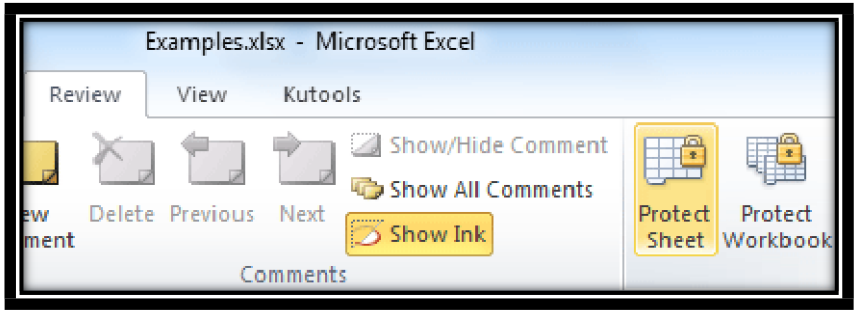


Step 4: Select cells and ranges that you want to lock.

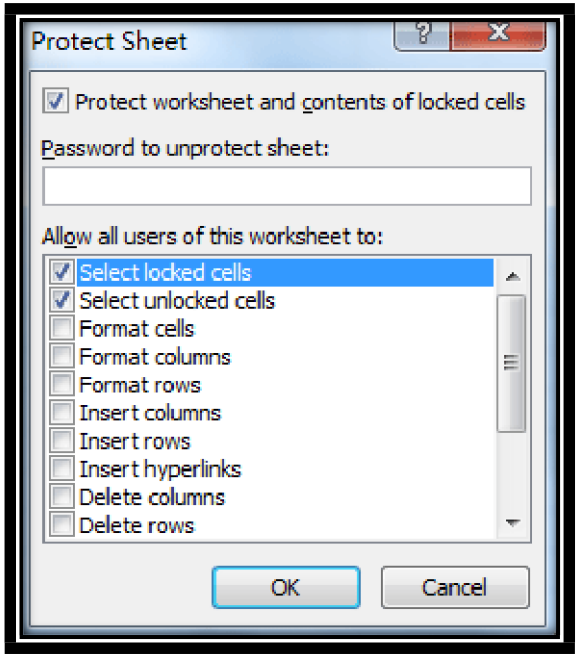
Step 5: Right click selected ranges, and select the Format Cell item from the context menu.

Step 6: In the Format Cells dialog box, check the Lock option under Protection tab, and click OK. Step 7: Click the Protect Sheet button in the Changes group under Review tab.

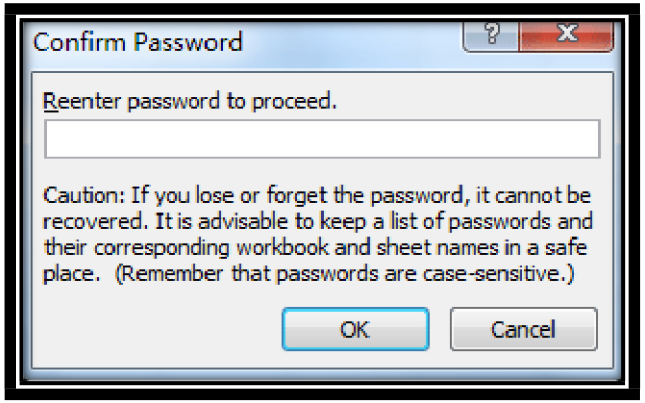
Protecting Model from Undesired Changes



Step 8: In the Protect Sheet dialog box, enter a password in the blank box under Password to unprotect sheet:. See the following screen shot:



Step 9: Confirm Password dialog box pops up, please reenter the password again.



Step 10: Click OK.

Then it locks and protects only selected cells and ranges in current worksheet, while unselect ranges are editable.

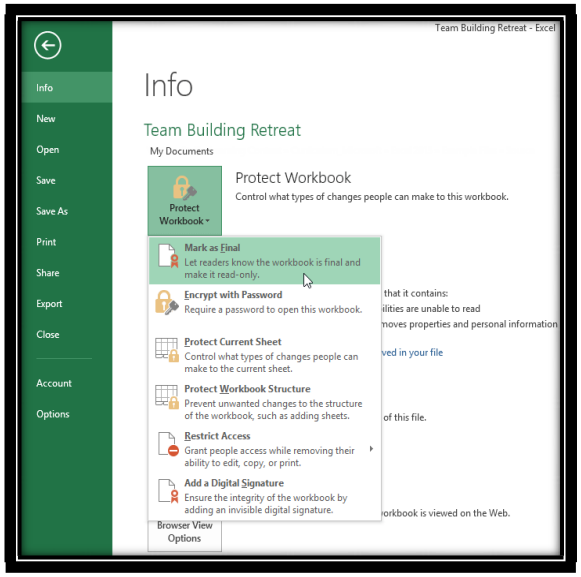
17.2 Worksheet and workbook protection

By default, anyone with access to your workbook will be able to open, copy, and edit its content unless you protect it. There are many different ways to protect a workbook, depending on your needs.

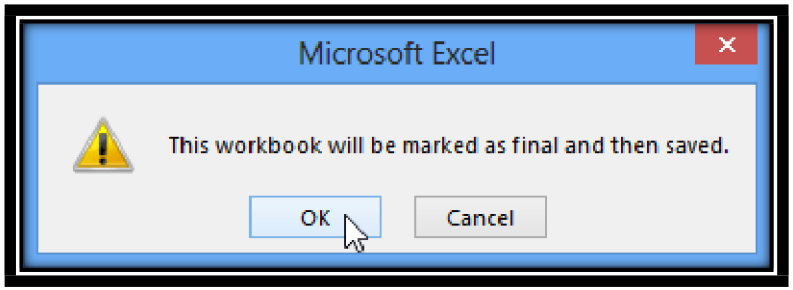
To protect your workbook:

1. Click the **File** tab to access **Backstage view**.
2. From the **Info** pane, click the **Protect Workbook** command.

3. In the drop-down menu, choose the option that best suits your needs. In our example, we'll select **Mark as Final**. Marking your workbook as final is a good way to discourage others from editing the workbook, while the other options give you even more control if needed.

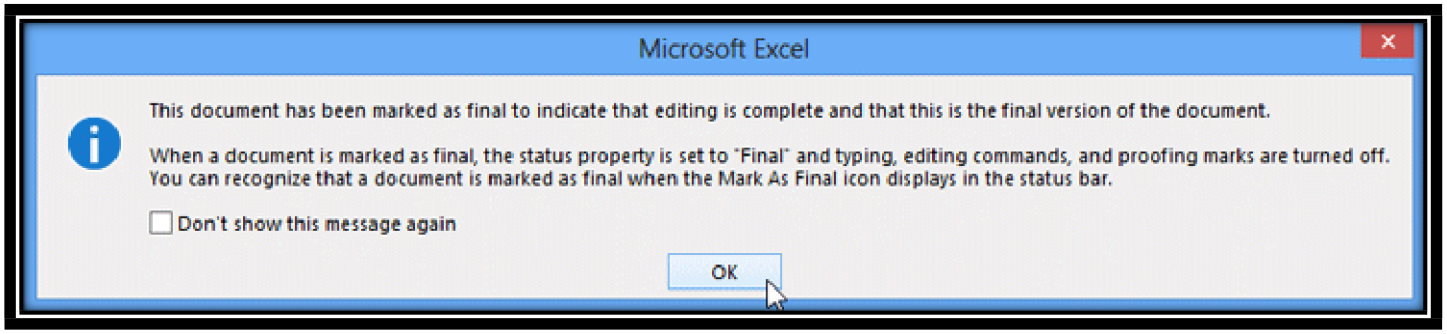


4. A dialog box will appear, prompting you to save. Click **OK**.

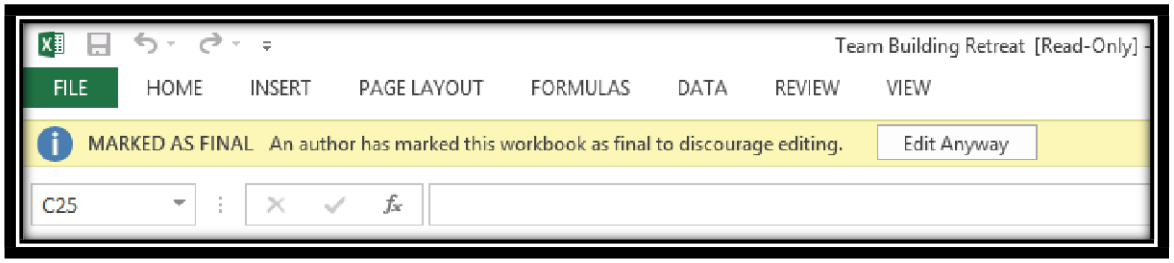


Protecting Model from Undesired Changes

5. Another dialog box will appear. Click **OK**.



6. The workbook will be marked as final.



Marking a workbook as final will not prevent someone from editing it. If you want to prevent people from editing it, you can use the **Restrict Access** option instead.

17.3 Audit Sheet

Financial model is normally a combination of many sheets consisting of details and summary data. In order to make the authenticity of linked data all over the financial model, audit sheet are prepared. The purpose of audit sheet is:

- To make sure that the data in detail sheet are linked or transferred correctly on summarized sheets
- To make sure that opening balances plus any addition, less any deduction is providing the correct closing balances. This is specially done with balance sheet accounts In case of any cross checking differences, the relevant sections are highlighted with red or any other color.