

5-5: The Workbook Class

You manipulate each open workbook in Excel through the properties and methods of this class.

5-5-1: ActiveSheet Property; Worksheets Property

These are learnt in section 5-3-1.

5-5-2: FullName Property

This returns the name of the object, including its path on disk, as a string. Read-only String.

5-5-3: Name Property

This returns the name of the object as a string. Read-only String.

5-5-4: Path Property

This returns the complete path to the application, excluding the final separator and name of the application. Read-only String.

Let's try! Corner

Investigate the difference between `FullName`, `Name` and `Path` properties by checking an opened workbook.

5-5-5: Saved Property

True if no changes have been made to the specified workbook since it was last saved. Read/write Boolean.

5-5-6: Activate Method

It has been discussed in section 5-3-1.

5-5-7: Close Method

The `Close` method closes the workbook.

`expression.Close(SaveChanges, Filename, RoutWorkbook)`

- **expression**: Required. An expression that returns a `Workbook` object.
- **SaveChanges**: Optional Variant. If there are no changes to the workbook, this argument is ignored. If there are changes to the workbook and the workbook appears in other open windows, this argument is ignored. If there are changes to the workbook but the workbook doesn't appear in any other open windows, this argument specifies whether changes should be saved, as shown in the following table.

Value	Action
True	Saves the changes to the workbook. If there is not yet a file name associated with the workbook, then <code>FileName</code> is used. If <code>FileName</code> is omitted, the user is asked to supply a file name.
False	Does not save the changes to this file.
Omitted	Displays a dialog box asking the user whether or not to save changes.

- | Value | Action |
|---------|--|
| True | Saves the changes to the workbook. If there is not yet a file name associated with the workbook, then <code>FileName</code> is used. If <code>FileName</code> is omitted, the user is asked to supply a file name. |
| False | Does not save the changes to this file. |
| Omitted | Displays a dialog box asking the user whether or not to save changes. |
- **FileName**: Optional Variant. Save changes under this file name.
 - **RoutWorkbook** (Advanced): Optional Variant. This argument is ignored if the workbook doesn't need to be routed.

5-5-8: FollowHyperlink Method

This method displays a webpage through the hyperlink in a web browser.

*expression.FollowHyperlink(Address, SubAddress, NewWindow, AddHistory, _
ExtraInfo, Method, HeaderInfo)*

- *expression*: Required. An expression that returns a Workbook object.
- *Address*: Required String. The address of the target document.
- *SubAddress* (Advanced): Optional Variant. The location within the target document (i.e. an anchor point). The default value is the empty string.
- *NewWindow*: Optional Variant. True to display the target application in a new window. The default value is False.
- *AddHistory* (Advanced): Optional Variant. Not used. Reserved for future use.
- *ExtraInfo* (Advanced): Optional Variant. A String or byte array that specifies additional information for HTTP to use to resolve the hyperlink (i.e. server-side Get/Post variables).
- *Method* (Advanced): Optional Variant. Specifies the way ExtraInfo is attached. This can be one of the followings: *msoMethodGet*: send ExtraInfo using Get; *msoMethodPost*: send ExtraInfo using Post.
- *HeaderInfo* (Advanced): Optional Variant. A String that specifies header information for the HTTP request. The default value is an empty string.

Let's try! Corner

Make a form and add a command button to a worksheet shown as below, so that when the user clicks the button, a web browser comes up and shows the page entered by the user.

	A	B
1	URL:	http://hk.yahoo.com
2		
3		Connect
4		
5		

5-5-9: PrintOut Method

Prints the whole or part of the workbook.

*expression.PrintOut(From, To, Copies, Preview, ActivePrinter, PrintToFile, _
Collate, PrToFileName)*

- *expression*: Required. An expression that returns a Workbook object.
- *From*: Optional Variant. The number of the page at which to start printing. If this argument is omitted, printing starts at the beginning.
- *To*: Optional Variant. The number of the last page to print. If this argument is omitted, printing ends with the last page.
- *Copies*: Optional Variant. The number of copies to print. If this argument is omitted, one copy is printed.
- *Preview*: Optional Variant. True to have Microsoft Excel invoke print preview before printing the object. False (or omitted) to print the object immediately.

- *ActivePrinter*: Optional Variant. Sets the name of the active printer.
- *PrintToFile*: Optional Variant. True to print to a file. If *PrToFileName* is not specified, Microsoft Excel prompts the user to enter the name of the output file.
- *Collate*: Optional Variant. True to collate multiple copies.
- *PrToFileName*: Optional Variant. If *PrintToFile* is set to True, this argument specifies the name of the file you want to print to.

Remarks:

"Pages" in the descriptions of From and To refers to printed pages — not overall pages in the sheet or workbook.

5-5-10: Save Method

This saves the workbook.

5-5-11: SaveAs Method

Saves changes to the workbook in a different file.

expression.SaveAs(FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local)

- *expression*: Required. An expression that returns a Workbook object.
- *Filename*: Optional Variant. A string that indicates the name of the file to be saved. You can include a full path; if you don't, Microsoft Excel saves the file in the current folder.
- *FileFormat*: Optional Variant. The file format to use when you save the file. For a list of valid choices, see the Excel help. Some common choices are:

Constant	Meaning
<code>xlCSV</code>	Comma separated value
<code>xlCurrentPlatformText</code>	ANSI text format
<code>xlHtml</code>	Web page format
<code>xlUnicodeText</code>	Unicode text format
<code>xlWorkbookNormal</code>	Excel workbook format

For an existing file, the default format is the last file format specified; for a new file, the default is the format of the version of Excel being used.

- *Password*: Optional Variant. A case-sensitive string (no more than 15 characters) that indicates the protection password to be given to the file.
- *WriteResPassword* (Advanced): Optional Variant. A string that indicates the write-reservation password for this file. If a file is saved with the password and the password isn't supplied when the file is opened, the file is opened as read-only.
- *ReadOnlyRecommended* (Advanced): Optional Variant. True to display a message when the file is opened, recommending that the file be opened as read-only.
- *CreateBackup* (Advanced): Optional Variant. True to create a backup file.
- *AccessMode* (Advanced): Optional `XlSaveAsAccessMode`. Sets the access mode of the file.
- *ConflictResolution* (Advanced): Optional `XlSaveConflictResolution`. Sets the action upon conflict.
- *AddToMru* (Advanced): Optional Variant. True to add this workbook to the list of recently used files. The default value is False.
- *TextCodePage* (Advanced): Optional Variant. Not used in U.S. English Microsoft Excel.
- *TextVisualLayout* (Advanced): Optional Variant. Not used in U.S. English Microsoft Excel.
- *Local* (Advanced): Optional Variant. True saves files against the language of Microsoft Excel (including control panel settings). False (default) saves files against the language of Visual Basic for Applications.

Let's try! Corner

Create a Sub to save all the open workbooks to the HTML format, with a name equal to (name of the workbook).html.

5-6: The Worksheets Collection

The `Worksheets` collection deals with all the worksheets in a workbook. Note that worksheets are spreadsheets, i.e. with cells. Contrastingly, a chart in a separate sheet is a `Chart` object, not a `Worksheet` object. Both `Worksheet` objects and `Chart` objects are contained in the `Sheets` collection. Only `Worksheet` objects are contained in the `Worksheets` collection.

5-6-1: Properties at a Glance

Since the `Worksheets` collection is a `Collection` object, it is very similar to the `Workbooks` collection. So I will not give a detailed explanation to each of the properties and methods.

- Count property
- Item property (default property)

5-6-2: Methods at a Glance

- Add method
 - `expression.Add(Before, After, Count, Type)`
 - `expression`: Required. An expression that returns the `Worksheets` collection.
 - `Before`: Optional Variant. An object that specifies the sheet before which the new sheet is added.
 - `After`: Optional Variant. An object that specifies the sheet after which the new sheet is added.
 - `Count`: Optional Variant. The number of sheets to be added. The default value is one.
 - `Type`: Optional Variant. Specifies the sheet type. Can be one of the following `XlSheetType` constants: `xlWorksheet`, `xlChart`, `xlExcel4MacroSheet`, or `xlExcel4IntlMacroSheet`. If you are inserting a sheet based on an existing template, specify the path to the template. The default value is `xlWorksheet`.
 - Remarks: If `Before` and `After` are both omitted, the new sheet is inserted before the active sheet.
- PrintOut Method
- Select Method
 - Selects all worksheets.

Let's try! Corner

Create a simple form in a worksheet as below:

<input checked="" type="checkbox"/>	Create a new workbook	
Add	3	worksheets
<input type="button" value="Go!"/>		

If “Create a new workbook” is checked, create a new workbook with the number of worksheets equal to the number entered by the user. If “Create a new workbook” is not checked, add the number of worksheets entered to the current workbook.

A checkbox control returns a value True if it is checked and returns a value False if not checked.

(Hints: You may have to read the description of a property of `Application` object called `SheetsInNewWorkbook` to help you specify the number of sheets to be inserted to a new workbook.)

5-7: The Worksheet Class

5-7-1: Cells Property; Columns Property; Range Property; Rows Property

These are explained in 5-3-1.

5-7-2: Name Property

Returns the name of the worksheet.

5-7-3: ScrollArea Property

Returns or sets the range where scrolling is allowed, as an A1-style range reference. Cells outside the scroll area cannot be selected.

Remarks: Set this property to the empty string ("") to enable cell selection for the entire sheet.

Example:

```
Worksheets(1).ScrollArea = "A1:F10"
```

5-7-4: UsedRange Property

Returns a Range object that represents the used range on the specified worksheet. Read-only.

Example:

```
Worksheets("Sheet1").Activate  
ActiveSheet.UsedRange.Select
```

5-7-5: Visible Property

Determines whether the object is visible. Read/write XlSheetVisibility.

XlSheetVisibility Constant	Value	Meaning
xlSheetHidden	False (0)	The sheet is hidden, but user can make the object visible by clicking Format > Sheet > Unhide...
xlSheetVisible	True (-1)	The sheet is visible.
xlSheetVeryHidden	2	Hides the object so that the only way for you to make it visible again is by setting this property to True (the user cannot make the object visible).

Example:

```
For Each sh In Worksheets  
    sh.Visible = True  
Next sh
```

5-7-6: Activate Method

This is explained in 5-3-1.

5-7-7: Calculate Method

This is explained in 5-3-5.

5-7-8: Copy Method

Copies the sheet to another location in the workbook.

expression.Copy(*Before*, *After*)

- *expression*: Required. An expression that returns the Worksheets collection.
- *Before*: Optional Variant. The sheet before which the copied sheet will be placed. You cannot specify *Before* if you specify *After*.
- *After*: Optional Variant. The sheet after which the copied sheet will be placed. You cannot specify *After* if you specify *Before*.

Remarks: If you don't specify either *Before* or *After*, Microsoft Excel creates a new workbook that contains the copied sheet.

5-7-9: Delete Method

Deletes the worksheet.

This example deletes Sheet3 in the active workbook without displaying the confirmation dialog box.

```
Application.DisplayAlerts = False
Worksheets("Sheet3").Delete
Application.DisplayAlerts = True
```

5-7-10: Move Method

Moves the sheet to another location in the workbook. The syntax is very much the same as the Copy method.

```
expression.Move(Before, After)
```

5-7-11: PrintOut Method

Prints the worksheet. Refer to 5-5-9.

5-7-12: SaveAs Method

Saves changes to the worksheet in a different file. The arguments are a bit different from the SaveAs method of a Workbook object. Refer to 5-5-11 for explanations of the arguments.

```
expression.SaveAs(FileName, FileFormat, Password, WriteResPassword, _  
ReadOnlyRecommended, CreateBackup, AddToMru, TextCodepage, TextVisualLayout, _  
Local)
```

5-7-13: Select Method

Selects the worksheet.

5-8: The Range Class

A Range object represents a selection of cells containing one or more contiguous blocks of cells. Recall that we have learnt in 5-3-1 that there are several methods to return a Range object.

```
ActiveCell  
Range("A1")  
Cells(1, 1)  
Columns(1)  
Rows(1)  
Application.Workbooks(1).Worksheets(1).Range("A1")  
Application.Workbooks(1).Worksheets(1).Cells(1, 1)  
Application.Workbooks(1).Worksheets(1).Columns(1)  
Application.Workbooks(1).Worksheets(1).Rows(1)
```

etc.

You can also further manipulate the range of cells by Union and Intersect methods of the Application object.

In addition, you can use a short-form to access a range with a pair of square brackets. For example, to access the cell A1 you can write [A1]. To access the range A1:C5 you can write [A1:C5].

5-8-1: Address Referencing

There are three formats of referencing an address: “A1” format, “R1C1” format and 3D format.

5-8-1-1: A1 Format

This is the referencing method you have been using in Excel. Some examples:

Reference String	Meaning
A1:C2	Refers to the rectangular box bounded by the cell A1 and C2.
A:A	Refers to the whole column A.
1:1	Refers to the whole row 1.
E:H	Refers to the columns from E to H.
\$A1	Refers to the cell A1, with the column being absolute.
\$3:\$4	Refers to the rows 3 and 4, with row 3 and row 4 being absolute.
Marketing!A1:E10	Refers to the rectangular box bounded by the cell A1 and E10 in the sheet Marketing.

5-8-1-2: R1C1 Format

This is the referencing method used when a macro is recorded. Some examples:

Reference String	Meaning
R[1]C	Refers to the cell one row down and in the same column of the current cell.
RC[-2]	Refers to the cell in the same row and two columns to the left of the current cell.
R2C2	Refers to the cell B2 (Absolute reference).
R[-1]	Refers to the whole row one row up of the current cell.
C	Refers to the whole column of the current cell.

5-8-1-3: 3D Format

This is used when you want to access the same cells in a series of sheets. Some examples:

Reference String	Meaning
Sheet1:Sheet3!A1	Refers to the cell A1 in each of the sheets between Sheet1 and Sheet 3.
Sheet5:Sheet13!B\$3:\$E10	Refers to the cells bounded by B\$3 and \$E10 in each of the sheets between Sheet5 and Sheet 13.

5-8-1-4: XlReferenceStyle Enum

In some cases when you enter or request a return of a formula, you can specify a reference style argument. You need to supply a constant in XlReferenceStyle Enum:

Constant	Meaning
xlA1	Use the A1 style format
xlR1C1	Use the R1C1 style format

5-8-2: Looping through Cells in a Range

A contiguous block of cells is a rectangular block. Although you can include several contiguous blocks of cells in a range (e.g. through the `Union` method), you cannot loop through each cells well in such a range. You need to break down the blocks into a collection of rectangular blocks by means of the `Areas` property of the range. You can then loop through each of the ranges in the `Areas` collection by the `Item` property (the default property of a `Range` object).

Example:

```
Sub ch5_8_2()  
  
    Dim clRange As Range, clAreas As Areas  
    Dim i As Integer, j As Integer  
  
    Set clRange = Union(Union(Range("F10:G11"), Range("G11:H12")), _  
        Range("H12:I13"))  
  
    Set clAreas = clRange.Areas  
  
    For j = 1 To clAreas.count  
        For i = 1 To clAreas(j).count  
            clAreas(j)(i) = clAreas(j)(i).Address  
        Next i  
    Next j  
  
End Sub
```

	E	F	G	H	I
9					
10		\$F\$10	\$G\$10		
11		\$F\$11	\$G\$11	\$H\$11	
12			\$G\$12	\$H\$12	\$I\$12
13				\$H\$13	\$I\$13

In the above example, there are three contiguous ranges (F10:G11, G11:H12, H12:I13). Notice that G11 and H12 are duplicated in the three ranges. However if you invoke `MsgBox clRange.Count` you will get an 12. Therefore do not rely on the `Count` property to get the number of cells of a non-contiguous range. In fact, it is very difficult to get a correct count of the selected cells in this case.

5-8-3: Transferring Contents Between a Range and a Two-dimensional Array

You can transfer contents between a contiguous block of cells and a two-dimension array pretty easily.

5-8-3-1: Transferring from an Array to a Range

When transferring from an array to a range, just declare an array, input data and assign the array to the range

Example:

```
Sub ch5_8_3_1()  
  
    Dim intArray(1 To 2, 1 To 2) As Integer  
  
    intArray(1, 1) = 1  
    intArray(1, 2) = 2  
    intArray(2, 1) = 3  
    intArray(2, 2) = 4  
  
    Range("G8:H9") = intArray  
  
End Sub
```

	F	G	H	
7				
8		1	2	
9		3	4	
10				

5-8-3-2: Transferring from a Range to an Array

When transferring from a range to an array, you should not declare an array; instead, declare it as a Variant. Then, you can successfully transfer the range to the Variant variable, which now holds a two-dimensional array.

Example:

```
Sub ch5_8_3_2()  
  
    Dim varArray, i As Integer, j As Integer  
  
    varArray = Range("G8:H9")  
  
    For i = 1 To UBound(varArray, 1)  
        For j = 1 To UBound(varArray, 2)  
            MsgBox varArray(i, j)  
        Next j  
    Next i  
  
End Sub
```

Notice the difference between the following two statements:

```
varArray = Range("G8:H9")  
Set varArray = Range("G8:H9")
```

5-8-4: Selections

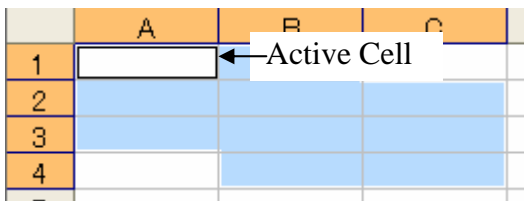
You can manipulating the selection of cells programmatically. A selection of cells is demonstrated in 5-3-11. You can select a range by using the `Select` method of a `Range` object. For example:

```
Range("A1:D4").Select
```

After that, you can use the `Selection` property of the `Application` object to access the selection.

```
Selection = 1  
Selection.Font.Color = RGB(255, 127, 127)
```

Remember that a selection is not the same as an active cell. You can selected several cells, but only one cell within the selection is active.



5-8-4: Address Property

Returns the address of a range. Read-only String.

```
expression.Address(RowAbsolute, ColumnAbsolute, ReferenceStyle, External,  
RelativeTo)
```

- *expression*: Required. An expression that returns a `Range` object.
- *RowAbsolute*: Optional Variant. True to return the row part of the reference as an absolute reference. The default value is True.
- *ColumnAbsolute*: Optional Variant. True to return the column part of the reference as an absolute reference. The default value is True.
- *ReferenceStyle*: Optional `XlReferenceStyle`.
- *External*(Advance): Optional Variant. True to return an external reference. False to return a local reference. The default value is False.
- *RelativeTo*: Optional Variant. If *RowAbsolute* and *ColumnAbsolute* are False, and *ReferenceStyle* is `xlR1C1`, you must include a starting point for the relative reference. This argument is a `Range` object that defines the starting point.

Example:

```
Set mc = Worksheets("Sheet1").Cells(1, 1)  
MsgBox mc.Address() ' $A$1  
MsgBox mc.Address(RowAbsolute:=False) ' $A1  
MsgBox mc.Address(ReferenceStyle:=xlR1C1) ' R1C1  
MsgBox mc.Address(ReferenceStyle:=xlR1C1, _  
    RowAbsolute:=False, _  
    ColumnAbsolute:=False, _  
    RelativeTo:=Worksheets(1).Cells(3, 3)) ' R[-2]C[-2]
```

5-8-5: Borders Property

Returns a Borders collection that represents the borders of a style or a range of cells. Refer to the Excel help files for the usage of Borders collection.

5-8-6: Column Property

Returns the number of columns in the first contiguous block.

5-8-7: ColumnWidth Property

Returns or sets the width of all columns in the specified range.

Remarks:

One unit of column width is equal to the width of one character in the Normal style. For proportional fonts, the width of the character 0 (zero) is used.

If all columns in the range have the same width, the ColumnWidth property returns the width. If columns in the range have different widths, this property returns Null.

5-8-8: CurrentRegion Property

Returns a Range object that represents the current region. The current region is a range bounded by any combination of blank rows and blank columns. Read-only.

Let's try! Corner

Try this example after you have imported the Hang Seng Index history data as stated in 5-4-5.

```
ActiveCell.CurrentRegion.Select
```

Which cells are selected?

5-8-9: End Property

Returns a Range object that represents the cell at the end of the region that contains the source range. Equivalent to pressing END+UP ARROW, END+DOWN ARROW, END+LEFT ARROW, or END+RIGHT ARROW. Read-only Range object.

```
expression.End(Direction)
```

- *expression*: Required. An expression that returns a Range object.
- *Direction*: Required XlDirection. The direction in which to move. XlDirection: xlDown, xlToRight, xlToLeft or xlUp.

5-8-10: EntireColumn Property

Returns a Range object that represents the entire column (or columns) that contains the specified range. Read-only.

5-8-11: EntireRow Property

Returns a Range object that represents the entire row (or rows) that contains the specified range. Read-only.

5-8-12: Font Property

Returns a `Font` object that represents the font of the range. For usage of a `Font` object see the Excel help file.

5-8-13: Formula Property

Returns or sets the formula in A1-style notation.

Remarks:

If the cell contains a constant, this property returns the constant. If the cell is empty, this `Formula` property returns an empty string. If the cell contains a formula, the `Formula` property returns the formula as a string in the same format that would be displayed in the formula bar (including the equal sign).

5-8-14: FormulaArray Property

Returns or sets the array formula of a range (e.g. a function returning a matrix). If the specified range doesn't contain an array formula, this property returns `Null`.

Remarks:

If you use this property to enter an array formula, the formula must use the R1C1 reference style, not the A1 reference style.

5-8-15: FormulaR1C1 Property

Returns or sets the formula, using R1C1-style notation.

5-8-16: HasArray Property

True if the specified cell is part of an array formula. Read-only.

5-8-17: HasFormula Property

True if all cells in the range contain formulas; False if none of the cells in the range contains a formula; `Null` otherwise. Read-only.

5-8-18: HorizontalAlignment Property

Returns or sets the horizontal alignment for the specified object. Refer to `XlHAlign` enum for a list of usable constants.

Example:

```
Worksheets("Sheet1").Range("A1:A5").HorizontalAlignment = xlHAlignLeft
```

5-8-19: Item Property

Returns a single cell from a range. Default property of a `Range` object.

5-8-20: MergeArea Property

Returns a **Range** object that represents the merged range containing the specified cell. If the specified cell isn't in a merged range, this property returns the specified cell. Read-only.

Remarks: The MergeArea property only works on a single-cell range.

5-8-21: Name Property

Returns or sets the name of the object. The name of a **Range** object is a **Name** object. See the **Name** object help for more information.

5-8-22: NumberFormat Property

Returns or sets the format code for the object. Returns Null if all cells in the specified range don't have the same number format. Read/write Variant.

Example:

```
Worksheets("Sheet1").Range("A17").NumberFormat = "General"  
Worksheets("Sheet1").Rows(1).NumberFormat = "hh:mm:ss"  
Worksheets("Sheet1").Columns("C"). _  
    NumberFormat = "$#,##0.00_);[Red]($#,##0.00)"
```

Note: It takes a while to learn to write number format codes. Refer to “Create or delete a custom number format” section in the Excel help file for guidelines.

5-8-23: Offset Property

Returns a **Range** object that represents a range which is an offset from the specified range. Read-only.

expression.Offset(RowOffset, ColumnOffset)

- *expression*: Required. An expression that returns a **Range** object.
- *RowOffset*: Optional Variant. The number of rows (positive, negative, or 0 (zero)) by which the range is to be offset. Positive values are offset downward, and negative values are offset upward. The default value is 0.
- *ColumnOffset*: Optional Variant. The number of columns (positive, negative, or 0 (zero)) by which the range is to be offset. Positive values are offset to the right, and negative values are offset to the left. The default value is 0.

Example:

This example activates the cell three columns to the right of and three rows down from the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.Offset(rowOffset:=3, columnOffset:=3).Activate
```

5-8-24: Range Property

In addition to the “one-parameter” syntax of using the **Range** property, you can include two **Range** arguments constructed by the **Cells** property to specify a range which is bounded by the two ranges.

This example sets the font style in cells A1:C5 on Sheet1 to italic.

```
Worksheets("Sheet1").Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```

5-8-25: Resize Property

Resizes the specified range. Returns a `Range` object that represents the resized range.

`expression.Resize(RowSize, ColumnSize)`

- *expression*: Required. An expression that returns a `Range` object to be resized.
- *RowSize*: Optional Variant. The number of rows in the new range. If this argument is omitted, the number of rows in the range remains the same.
- *ColumnSize*: Optional Variant. The number of columns in the new range. If this argument is omitted, the number of columns in the range remains the same.

Note: When using the property, the upper left cell is held fixed.

Example:

This example resizes the selection on Sheet1 to extend it by one row and one column.

```
Worksheets("Sheet1").Activate
numRows = Selection.Rows.Count
numColumns = Selection.Columns.Count
Selection.Resize(numRows + 1, numColumns + 1).Select
```

5-8-26: Row Property

Returns the number of the first row of the first area in the range. Read-only.

5-8-27: RowHeight Property

Returns the height of all the rows in the range specified, measured in points. Returns Null if the rows in the specified range aren't all the same height. Read/write Variant.

Example:

This example sets the row height of every other row on Sheet1 to 4 points.

```
For Each rw In Worksheets("Sheet1").Rows
    If rw.Row Mod 2 = 0 Then
        rw.RowHeight = 4
    End If
Next rw
```

5-8-28: Style Property

Returns a `Style` object that represents the style of the specified range. The `Style` object contains all style attributes (font, number format, alignment, and so on) as properties. Refer to the help file.

5-8-29: Validation Property

Returns the `Validation` object that represents data validation for the specified range. You can easily validate data with this object when the user enters in a cell. This can also be done by using the Data > Validation option in the Excel menu. Check the usage of a `Validation` object in the Excel help.

5-8-30: Value Property

Returns or sets the value of the specified range. This is the default property.

```
Worksheets("Sheet1").Range("A1").Value = 3.14159
```

is the same as

```
Worksheets("Sheet1").Range("A1") = 3.14159
```

5-8-31: VerticalAlignment Property

Returns or sets the vertical alignment of the specified object. Read/write `XlVAlign`: `xlVAlignCenter`, `xlVAlignJustify`, `xlVAlignBottom`, `xlVAlignDistributed`, `xlVAlignTop`.

5-8-32: Worksheet Property

Returns a `Worksheet` object that represents the worksheet containing the specified range. Read-only.

5-8-33: Activate Method; Select Method

See 5-3-1.

5-8-34: BorderAround Method

Adds a border to a range and sets the `Color`, `LineStyle`, and `Weight` properties for the new border.

```
expression.BorderAround(LineStyle, Weight, ColorIndex, Color)
```

- *expression*: Required. An expression that returns a `Range` object.
- *LineStyle*: Optional `XlLineStyle`. The line style for the border. Can be one of these `XlLineStyle` constants: `xlContinuous`(default), `xlDash`, `xlDashDot`, `xlDashDotDot`, `xlDot`, `xlDouble`, `xlLineStyleNone`, `xlSlantDashDot`.
- *Weight*: Optional `XlBorderWeight`. The border weight. Can be one of these `XlBorderWeight` constants: `xlHairline`, `xlMedium`, `xlThick`, `xlThin`(default).
- *ColorIndex*: Optional `XlColorIndex`. The border color, as an index into the current color palette or as a `XlColorIndex` constant. `XlColorIndex` can be one of these `XlColorIndex` constants: `xlColorIndexAutomatic` (default), `xlColorIndexNone`.
- *Color*: Optional Variant. The border color, as an RGB value.

Remarks:

You must specify either *ColorIndex* or *Color*, but not both. You can use the `RGB` function to return an RGB value.

You can specify either *LineStyle* or *Weight*, but not both. If you don't specify either argument, Microsoft Excel uses the default line style and weight.

This method outlines the entire range without filling it in. To set the borders of all the cells, you must set the *Color*, *LineStyle*, and *Weight* properties for the `Borders` collection. To clear the border, you must set the *LineStyle* property to `xlLineStyleNone` for all the cells in the range.

Example:

This example adds a thick red border around the range A1:D4 on Sheet1.

```
Worksheets("Sheet1").Range("A1:D4").BorderAround ColorIndex:=3, Weight:=xlThick
```

or use the code:

```
Worksheets("Sheet1").Range("A1:D4").BorderAround Color:=RGB(255, 0, 0),  
Weight:=xlThick
```

The following table shows the standard Excel color index, HTML color code and RGB color code for each color (taken from <http://www.mvps.org/dmccritchie/excel/colors.htm>).

Interior	font	<u>HTML</u>	<i>bgcolor=</i>	<i>Red</i>	<i>Green</i>	<i>Blue</i>	<i>Color</i>
Black	[Color 1]	#000000	#000000	0	0	0	[Black]
White		#FFFFFF	#FFFFFF	255	255	255	[White]
Red	[Color 3]	#FF0000	#FF0000	255	0	0	[Red]
Green	[Color 4]	#00FF00	#00FF00	0	255	0	[Green]
Blue	[Color 5]	#0000FF	#0000FF	0	0	255	[Blue]
Yellow	[Color 6]	#FFFF00	#FFFF00	255	255	0	[Yellow]
Magenta	[Color 7]	#FF00FF	#FF00FF	255	0	255	[Magenta]
Cyan	[Color 8]	#00FFFF	#00FFFF	0	255	255	[Cyan]
[Color 9]	[Color 9]	#800000	#800000	128	0	0	[Color 9]
[Color 10]	[Color 10]	#008000	#008000	0	128	0	[Color 10]
[Color 11]	[Color 11]	#000080	#000080	0	0	128	[Color 11]
[Color 12]	[Color 12]	#808000	#808000	128	128	0	[Color 12]
[Color 13]	[Color 13]	#800080	#800080	128	0	128	[Color 13]
[Color 14]	[Color 14]	#008080	#008080	0	128	128	[Color 14]
[Color 15]	[Color 15]	#C0C0C0	#C0C0C0	192	192	192	[Color 15]
[Color 16]	[Color 16]	#808080	#808080	128	128	128	[Color 16]
[Color 17]	[Color 17]	#9999FF	#9999FF	153	153	255	[Color 17]
[Color 18]	[Color 18]	#993366	#993366	153	51	102	[Color 18]
[Color 19]	[Color 19]	#FFF0CC	#FFF0CC	255	255	204	[Color 19]
[Color 20]	[Color 20]	#CCFFFF	#CCFFFF	204	255	255	[Color 20]
[Color 21]	[Color 21]	#660066	#660066	102	0	102	[Color 21]
[Color 22]	[Color 22]	#FF8080	#FF8080	255	128	128	[Color 22]
[Color 23]	[Color 23]	#0066CC	#0066CC	0	102	204	[Color 23]
[Color 24]	[Color 24]	#CCCCFF	#CCCCFF	204	204	255	[Color 24]
[Color 25]	[Color 25]	#000080	#000080	0	0	128	[Color 25]
[Color 26]	[Color 26]	#FF00FF	#FF00FF	255	0	255	[Color 26]
[Color 27]	[Color 27]	#FFFF00	#FFFF00	255	255	0	[Color 27]
[Color 28]	[Color 28]	#00FFFF	#00FFFF	0	255	255	[Color 28]
[Color 29]	[Color 29]	#800080	#800080	128	0	128	[Color 29]
[Color 30]	[Color 30]	#800000	#800000	128	0	0	[Color 30]
[Color 31]	[Color 31]	#008080	#008080	0	128	128	[Color 31]
[Color 32]	[Color 32]	#0000FF	#0000FF	0	0	255	[Color 32]
[Color 33]	[Color 33]	#00CCFF	#00CCFF	0	204	255	[Color 33]
[Color 34]	[Color 34]	#CCFFFF	#CCFFFF	204	255	255	[Color 34]
[Color 35]	[Color 35]	#CCFFCC	#CCFFCC	204	255	204	[Color 35]
[Color 36]	[Color 36]	#FFFF99	#FFFF99	255	255	153	[Color 36]
[Color 37]	[Color 37]	#99CCFF	#99CCFF	153	204	255	[Color 37]
[Color 38]	[Color 38]	#FF99CC	#FF99CC	255	153	204	[Color 38]
[Color 39]	[Color 39]	#CC99FF	#CC99FF	204	153	255	[Color 39]
[Color 40]	[Color 40]	#FFCC99	#FFCC99	255	204	153	[Color 40]
[Color 41]	[Color 41]	#3366FF	#3366FF	51	102	255	[Color 41]
[Color 42]	[Color 42]	#33CCCC	#33CCCC	51	204	204	[Color 42]
[Color 43]	[Color 43]	#99CC00	#99CC00	153	204	0	[Color 43]
[Color 44]	[Color 44]	#FFCC00	#FFCC00	255	204	0	[Color 44]

[Color 45]	[Color 45]	#FF9900	#FF9900	255	153	0	[Color 45]
[Color 46]	[Color 46]	#FF6600	#FF6600	255	102	0	[Color 46]
[Color 47]	[Color 47]	#666699	#666699	102	102	153	[Color 47]
[Color 48]	[Color 48]	#969696	#969696	150	150	150	[Color 48]
[Color 49]	[Color 49]	#003366	#003366	0	51	102	[Color 49]
[Color 50]	[Color 50]	#339966	#339966	51	153	102	[Color 50]
[Color 51]	[Color 51]	#003300	#003300	0	51	0	[Color 51]
[Color 52]	[Color 52]	#333300	#333300	51	51	0	[Color 52]
[Color 53]	[Color 53]	#993300	#993300	153	51	0	[Color 53]
[Color 54]	[Color 54]	#993366	#993366	153	51	102	[Color 54]
[Color 55]	[Color 55]	#333399	#333399	51	51	153	[Color 55]
[Color 56]	[Color 56]	#333333	#333333	51	51	51	[Color 56]

The following color pairs are the same color

11 & 25, 5 & 32, 14 & 31, 8 & 28, 9 & 30, 13 & 29, 18 & 54, 20 & 34, 7 & 26, and 6 & 27.

Let's try! Corner

Use the Hang Seng Index example, draw a thick blue border around the whole table. (Use the code in 5-8-8 to select the whole table.)

5-8-35: Calculate Method

Refer to 5-3-5.

5-8-36: Clear Method

Clears the formulas and formatting in the cells specified by the range.

Example:

```
Worksheets("Sheet1").Range("A1:G37").Clear
```

5-8-37: ClearFormats Method

Clears all formatting in the cells specified by the range.

Example:

```
Worksheets("Sheet1").Range("A1:G37").ClearFormats
```

5-8-38: Delete Method

Deletes the cells specified by the range.

`expression.Delete(Shift)`

- **expression:** Required. An expression that returns a Range object.
- **Shift:** Optional Variant. Specifies how to shift cells to replace deleted cells. Can be one of the following `XlDeleteShiftDirection` constants: `xlShiftToLeft` or `xlShiftUp`. If this argument is omitted, Microsoft Excel decides based on the shape of the range.

This example deletes cells A1:D10 on Sheet1 and shifts the remaining cells to the left.

```
Worksheets("Sheet1").Range("A1:D10").Delete Shift:=xlShiftToLeft
```

5-8-39: Dirty Method

Designates a range to be recalculated when the next recalculation occurs.

The `Calculate` method forces the specified range to be recalculated. If the application is in manual calculation mode, using the `Dirty` method instructs Excel to identify the specified cell to be recalculated. If the application is in automatic calculation mode, using the `Dirty` method instructs Excel to perform a recalculation.

5-8-40: FillDown Method; FillLeft Method; FillRight Method; FillUp Method

Fills down, left, right or up from the top, rightmost, leftmost or bottom cell or cells in the specified range to the bottom, leftmost, rightmost or top of the range respectively. The contents and formatting of the original cell or cells are copied into the rest of the rows or columns in the range.

Example:

This example fills the range A1:M1 on Sheet1, based on the contents of cell M1.

```
Worksheets("Sheet1").Range("A1:M1").FillLeft
```

5-8-41: Find Method

Finds specific information in a range, and returns a Range object that represents the first cell where that information is found, equal to using the Find dialog box. Returns Nothing if no match is found. Doesn't affect the selection or the active cell.

`expression.Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection, _ MatchCase, MatchByte, SearchFormat)`

- **expression:** Required. An expression that returns a Range object.
- **What:** Required Variant. The data to search for. Can be a string or any Microsoft Excel data type.
- **After:** Optional Variant. The cell after which you want the search to begin. This corresponds to the position of the active cell when a search is done from the user interface. Note that *After* must be a single cell in the range. Remember that the search begins after this cell; the specified cell isn't searched until the method wraps back around to this cell. If you don't specify this argument, the search starts after the cell in the upper-left corner of the range.

- *LookIn*: Optional Variant. The type of information. Can be one of the following *XlFindLookIn* constants: *xlComments*, *xlFormulas*, *xlValues*.
- *LookAt*: Optional Variant. Can be one of the following *XlLookAt* constants: *xlWhole* or *xlPart*.
- *SearchOrder*: Optional Variant. Can be one of the following *XlSearchOrder* constants: *xlByRows* or *xlByColumns*.
- *SearchDirection*: Optional *XlSearchDirection*. The search direction. Can be one of these *XlSearchDirection* constants: *xlNext*(default), *xlPrevious*.
- *MatchCase*: Optional Variant. True to make the search case sensitive. The default value is *False*.
- *MatchByte*(advanced): Optional Variant. Used only if you've selected or installed double-byte language support. True to have double-byte characters match only double-byte characters. False to have double-byte characters match their single-byte equivalents.
- *SearchFormat*(advanced): Optional Variant. True if the search format is used, *False* (default) if the search format is not used. You can set the search format by accessing *Application.FindFormat* property. Refer to the help file for information about the *Application.FindFormat* property or the *CellFormat* class.

Remarks:

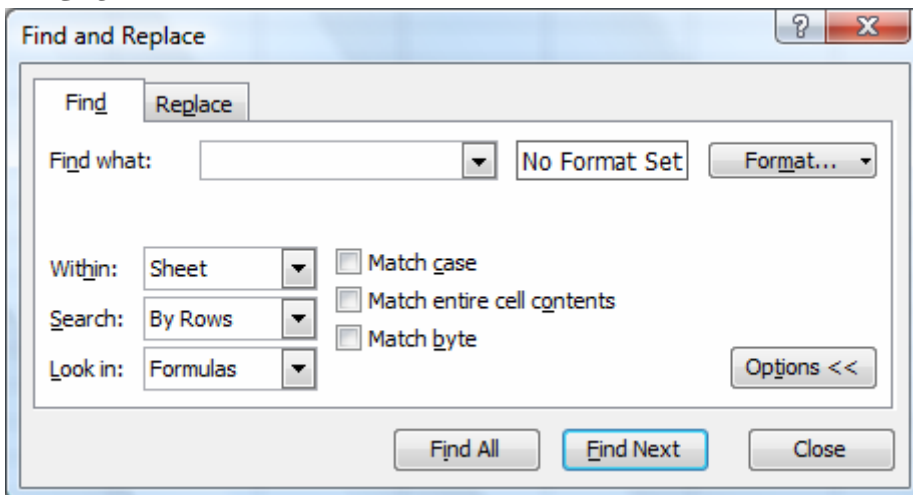
The settings for *LookIn*, *LookAt*, *SearchOrder*, and *MatchByte* are saved each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Setting these arguments changes the settings in the Find dialog box, and changing the settings in the Find dialog box changes the saved values that are used if you omit the arguments. To avoid problems, set these arguments explicitly each time you use this method.

You can use the *FindNext* and *FindPrevious* methods to repeat the search.

When the search reaches the end of the specified search range, it wraps around to the beginning of the range. To stop a search when this wraparound occurs, save the address of the first found cell, and then test each successive found-cell address against this saved address.

To find cells that match more complicated patterns, use a *For Each...Next* statement with the *Like* operator (to be discussed in later sections). For example, the following code searches for all cells in the range *A1:C5* that use a font whose name starts with the letters *Cour*. When Microsoft Excel finds a match, it changes the font to *Times New Roman*.

```
For Each c In [A1:C5]
    If c.Font.Name Like "Cour*" Then
        c.Font.Name = "Times New Roman"
    End If
Next
```



5-8-42: FindNext Method; FindPreviousMethod

The FindNext method continues a search that was begun with the Find method. Finds the next cell that matches those same conditions and returns a Range object that represents that cell. Doesn't affect the selection or the active cell.

The FindPrevious method works in a similar manner, with a difference that it finds the previous cell instead of the next cell.

```
expression.FindNext(After)  
expression.FindPrevious(After)
```

- *expression*: Required. An expression that returns a Range object.
- *After*: Optional Variant. The cell before which you want to search. See the argument description of the Find method (5-8-41).

Remarks

When the search reaches the end/beginning of the specified search range, it wraps around to the beginning/end of the range. To stop a search when this wraparound occurs, save the address of the first found cell, and then test each successive found-cell address against this saved address.

Example

This example finds all cells in the range A1:A500 on worksheet one that contain the value 2 and changes it to 5.

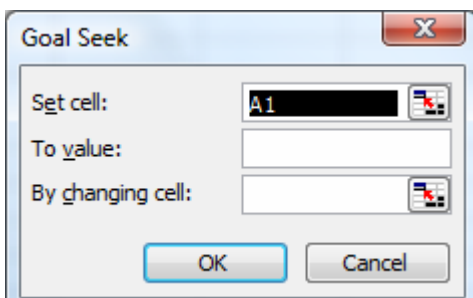
```
With Worksheets(1).Range("A1:A500")  
    Set c = .Find(2, lookin:=xlValues)  
    If Not c Is Nothing Then  
        firstAddress = c.Address  
        Do  
            c.Value = 5  
            Set c = .FindNext(c)  
        Loop While Not c Is Nothing And c.Address <> firstAddress  
    End If  
End With
```

5-8-43: GoalSeek Method

It is the same as the Goal Seek option available in Excel.

```
expression.GoalSeek(Goal, ChangingCell)
```

- *expression*: Required. An expression that returns a Range object. Must be a single cell.
- *Goal*: Required Variant. The value you want returned in this cell.
- *ChangingCell*: Required Range. Specifies which cell should be changed to achieve the target value.



5-8-44: Insert Method

Inserts a cell or a range of cells into the worksheet and shifts other cells away to make space.

expression.Insert(Shift, CopyOrigin)

- *expression*: Required. An expression that returns a Range object.
- *Shift*: Optional Variant. Specifies which way to shift the cells. Can be one of the following XlInsertShiftDirection constants: xlShiftToRight or xlShiftDown. If this argument is omitted, Microsoft Excel decides based on the shape of the range.
- *CopyOrigin*: Optional Variant. The copy origin. Can be one of the following XlInsertFormatOrigin constants: xlFormatFromLeftOrAbove, xlFormatFromRightOrBelow.

5-8-45: Merge Method

Creates a merged cell from the specified Range object.

expression.Merge(Across)

- *expression*: Required. An expression that returns the Range object.
- *Across*: Optional Variant. True to merge cells in each row of the specified range as separate merged cells. The default value is False.

5-8-46: PrintOut Method

Prints the specific range. Similar to the PrintOut method of the Workbook class. Refer to 5-5-9.

5-8-47: Replace Method

Replaces all occurrences of a value with another value in a specific range.

*expression.Replace(What, Replacement, LookAt, SearchOrder, MatchCase, _
MatchByte, SearchFormat, ReplaceFormat)*

- *expression*: Required. An expression that returns a Range object.
- *What*: Required Variant. The string you want Microsoft Excel to search for.
- *Replacement*: Required Variant. The replacement string.
- *LookAt*: Optional Variant. Can be one of the following XlLookAt constants: xlWhole or xlPart.
- *SearchOrder*: Optional Variant. Can be one of the following XlSearchOrder constants: xlByRows or xlByColumns.
- *MatchCase*: Optional Variant. True to make the search case sensitive.
- *MatchByte*(Advanced): Optional Variant. You can use this argument only if you've selected or installed double-byte language support in Microsoft Excel. True to have double-byte characters match only double-byte characters. False to have double-byte characters match their single-byte equivalents.
- *SearchFormat*(Advanced): Optional Variant. True if the search format is used. Refer to the same argument description in 5-8-41.
- *ReplaceFormat*(Advanced): Optional Variant. True if the replace format is used. Similar to the search format, you can access the replace format with Application.ReplaceFormat.

Remarks:

The settings for *LookIn*, *LookAt*, *SearchOrder*, and *MatchByte* are saved each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Setting these arguments changes the settings in the Find dialog box, and changing the settings in the Find dialog box changes the saved values that are used if you omit the arguments. To avoid problems, set these arguments explicitly each time you use this method.

Example:

This example replaces every occurrence of the trigonometric function SIN with the function COS. The replacement range is column A on Sheet1.

```
Worksheets("Sheet1").Columns("A").Replace What:="SIN", Replacement:="COS", _
    SearchOrder:=xlByColumns, MatchCase:=True
```

Let's try! Corner

Use the 5-3-1 example and the `Replace` method, replace all the 1's to 0's and 0's to 1's. (Note: You can record a macro to simplify your code writing.)

5-8-48: Show Method

Scrolls through the contents of the active window to move the range into view. The range must consist of a single cell in the active document.

5-8-49: Sort Method

Sorts a PivotTable report, a range, or the active region if the specified range contains only one cell.

```
expression.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, _
    OrderCustom, MatchCase, Orientation, SortMethod, DataOption1, DataOption2, _
    DataOption3)
```

- *expression*: Required. An expression that returns a `Range` object.
- *Key1*: Optional Variant. The first sort field, as either text (a PivotTable field or range name) or a `Range` object ("Dept" or Cells(1, 1), for example).
- *Order1*: Optional `XlSortOrder`. The sort order for the field or range specified in *Key1*. Can be one of these `XlSortOrder` constants: `xlDescending`, `xlAscending`(default).
- *Key2*: Optional Variant. The second sort field, as either text (a PivotTable field or range name) or a `Range` object. If you omit this argument, there's no second sort field. Cannot be used when sorting PivotTable reports.
- *Type*(Advanced): Optional Variant. Specifies which elements are to be sorted. Use this argument only when sorting PivotTable reports. Can be one of these `XlSortType` constants `xlSortLabels`, sorts the PivotTable report by labels; `xlSortValues`, sorts the PivotTable report by values.
- *Order2*: Optional `XlSortOrder`. The sort order for the field or range specified in *Key2*. Cannot be used when sorting PivotTable reports.
- *Key3*: Optional Variant. The third sort field.
- *Order3*: Optional `XlSortOrder`. The sort order for the field or range specified in *Key3*.
- *Header*: Optional `XlYesNoGuess`. Specifies whether or not the first row contains headers. Cannot be used when sorting PivotTable reports. Can be one of these `XlYesNoGuess` constants:

Constants	Meanings
<code>xlGuess</code>	Let Microsoft Excel determine whether there's a header, and to determine where it is, if there is one.
<code>xlNo</code> (default)	The entire range should be sorted.
<code>xlYes</code>	The first row should not be sorted.

- *OrderCustom*(Advanced): Optional Variant. This argument is a one-based integer offset to the list of custom sort orders. If you omit *OrderCustom*, a normal sort is used.
- *MatchCase*: Optional Variant. True to do a case-sensitive sort; False to do a sort that's not case sensitive. Cannot be used when sorting PivotTable reports.
- *Orientation*: Optional *XlSortOrientation*. The sort orientation. Can be one of these *XlSortOrientation* constants: *xlSortRows*(default), *xlSortColumns*.
- *SortMethod*(Advanced): Optional *XlSortMethod*. The type of sort. Can be one of these *XlSortMethod* constants: *xlStroke*, sorting by the quantity of strokes in each character; *xlPinYin*(default), phonetic Chinese sort order for characters.
- *DataOption1*: Optional *XlSortDataOption*. Specifies how to sort text in key 1. Cannot be used when sorting PivotTable reports. Can be one of these *XlSortDataOption* constants: *xlSortTextAsNumbers*, treat text as numeric data for the sort; *xlSortNormal*(default), sorts numeric and text data separately.
- *DataOption2*: Optional *XlSortDataOption*. Specifies how to sort text in key 2.
- *DataOption3*: Optional *XlSortDataOption*. Specifies how to sort text in key 3.

Remarks:

The settings for *Header*, *Order1*, *Order2*, *Order3*, *OrderCustom*, and *Orientation* are saved, for the particular worksheet, each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Set these arguments explicitly each time you use *Sort* method, if you choose not to use the saved values.

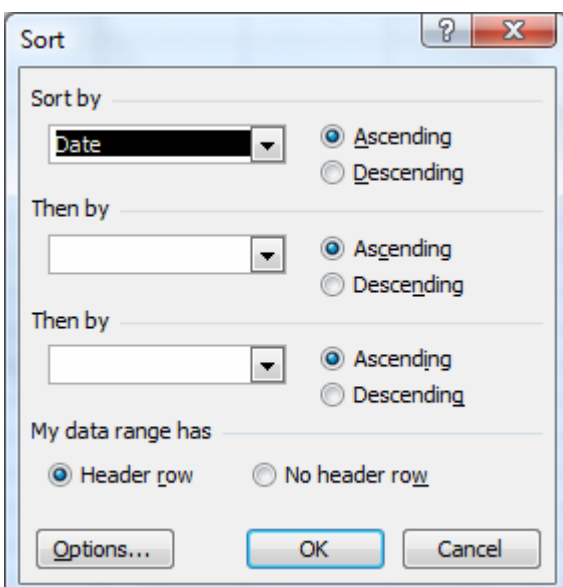
Text strings which are not convertible to numeric data are sorted normally.

If no arguments are defined with the *Sort* method, Microsoft Excel will sort the selection in ascending order.

Example:

This example sorts the range A1:C20 on Sheet1, using cell A1 as the first sort key and cell B1 as the second sort key. The sort is done in ascending order by row, and there are no headers. This example assumes there is data in the range A1:C20.

```
Sub SortRange1()  
    Worksheets("Sheet1").Range("A1:C20").Sort _  
        Key1:=Worksheets("Sheet1").Range("A1"), _  
        Key2:=Worksheets("Sheet1").Range("B1")  
End Sub
```



Let's try! Corner

Use the Hang Seng Index example, write codes to calculate the daily return (in percentage) and add to the sheet. Use the Sort method to sort the table based on the daily return in descending order. (You may insert the code by macro recording.)

5-8-50: SpecialCells Method

Returns a Range object that represents all the cells that match the specified type and value.

expression.SpecialCells(*Type*, *Value*)

- *expression*: Required. An expression that returns a Range object.
- *Type*: Required XlCellType. The cells to include. Can be one of these XlCellType constants.

Constant	Meaning
xlCellTypeAllFormatConditions	Cells of any format
xlCellTypeAllValidation	Cells having validation criteria
xlCellTypeBlanks	Empty cells
xlCellTypeComments	Cells containing notes
xlCellTypeConstants	Cells containing constants
xlCellTypeFormulas	Cells containing formulas
xlCellTypeLastCell	The last cell in the used range
xlCellTypeSameFormatConditions	Cells having the same format
xlCellTypeSameValidation	Cells having the same validation criteria
xlCellTypeVisible	All visible cells

- *Value*: Optional Variant. If *Type* is either *xlCellTypeConstants* or *xlCellTypeFormulas*, this argument is used to determine which types of cells to include in the result. These values can be added together to return more than one type. The default is to select all constants or formulas, no matter what the type is. Can be one of the following XlSpecialCellsValue constants:

Constant	Meaning
xlErrors	Returns cells that contain errors
xlLogical	Returns cells that contain Boolean values
xlNumbers	Returns cells that contain numerical values
xlTextValues	Returns cells that contain text

Example:

This example selects the empty cells in the range A1:H16.

```
Range("A1:H16").SpecialCells(xlCellTypeBlanks).Select
```

5-8-51: TextToColumns Method

Parses a column of cells that contain text into several columns. This is useful when working with file input to parse a delimited file easily.

`expression.TextToColumns(Destination, DataType, TextQualifier, _
ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, _
FieldInfo, DecimalSeparator, ThousandsSeparator, TrailingMinusNumbers)`

- *expression*: Required. An expression that returns a Range object.
- *Destination*: Optional Variant. A Range object that specifies where Microsoft Excel will place the results. If the range is larger than a single cell, the top left cell is used.
- *DataType*: Optional XlTextParsingType. The format of the text to be split into columns. Can be one of these XlTextParsingType constants: xlDelimited(default), xlFixedWidth.
- *TextQualifier*: Optional XlTextQualifier. Specifies how text is represented in the delimited string. Can be one of these XlTextQualifier constants. xlTextQualifierDoubleQuote(default), double quotes are used to specify a text value; xlTextQualifierNone, no quote is used for a text value; xlTextQualifierSingleQuote, a pair of single quotes are used for a text value.
- *ConsecutiveDelimiter*: Optional Variant. True to have Microsoft Excel consider consecutive delimiters as one delimiter. The default value is False.
- *Tab*: Optional Variant. True to have *DataType* be xlDelimited and to have the tab character be a delimiter. The default value is False.
- *Semicolon*: Optional Variant. True to have *DataType* be xlDelimited and to have the semicolon be a delimiter. The default value is False.
- *Comma*: Optional Variant. True to have *DataType* be xlDelimited and to have the comma be a delimiter. The default value is False.
- *Space*: Optional Variant. True to have *DataType* be xlDelimited and to have the space character be a delimiter. The default value is False.
- *Other*: Optional Variant. True to have *DataType* be xlDelimited and to have the character specified by the *OtherChar* argument be a delimiter. The default value is False.
- *OtherChar*: Optional Variant (required if *Other* is True). The delimiter character when *Other* is True. If more than one character is specified, only the first character of the string is used; the remaining characters are ignored.
- *FieldInfo*(Advanced): Optional Variant. An array containing parse information for the individual columns of data. See the help file for this function for more information.
- *DecimalSeparator*: Optional String. The decimal separator that Microsoft Excel uses when recognizing numbers. The default setting is the system setting.
- *ThousandsSeparator*: Optional String. The thousands separator that Excel uses when recognizing numbers. The default setting is the system setting.
- *TrailingMinusNumbers*: Optional Variant. Numbers that begin with a minus character.

5-8-52: UnMerge Method

Separates a merged area into individual cells.