

Lecture7

Using Visual Basic for Applications (VBA)

Introduction: VBA

VBA: Visual Basic for Applications (VBA), the programming language built into Excel (and other applications that make up Microsoft Office).

Workbooks

- The most common Excel object is a **workbook**.
- Everything that you do in Excel takes place in a workbook, which is stored in a file that, by default, has an **.xlsx** extension.
- An Excel workbook can hold any number of sheets (*limited only by memory*).
- **Worksheets** and **Chart sheets** are two basic types of Excel sheets.

Workbooks

- Open or create any number of workbooks (each in its own window), but only one workbook is *active* at any given time.
- Similarly, only **one sheet** in a workbook is the *active sheet*.
- To **activate a sheet**, click its **sheet tab** at the bottom of the screen.
- To **change a sheet's name**, double-click the **tab** and enter the new text name.
- Right-clicking a tab brings up a **shortcut menu** with additional options for the sheet, including changing its **tab color** and **hiding the sheet**.

Worksheets

- The most common type of sheet is a worksheet.
- Worksheets contain **cells**, and **the cells store data and formulas**.
- $(16,384 \times 1,048,576) = 17,179,869,184$ cells.
- A **worksheet cell** can hold a **constant value** — number, date, Boolean value (True or False), or text — or **the result of a formula**.
- Every worksheet also has an *invisible drawing layer*, which enables you to insert graphic objects, such as **charts, shapes, SmartArt, UserForm controls, pictures, and other objects**.

Chart sheets

- Another type of Excel sheets is a **Chart sheet**.
- A chart sheet holds a single chart.
- Many users ignore chart sheets, preferring to store charts on the worksheet's drawing layer.
- Using chart sheets is optional, but they make it a bit easier to print a chart on a page by itself and are especially useful for presentations.
- Figure 1 shows a pie chart on a chart sheet.

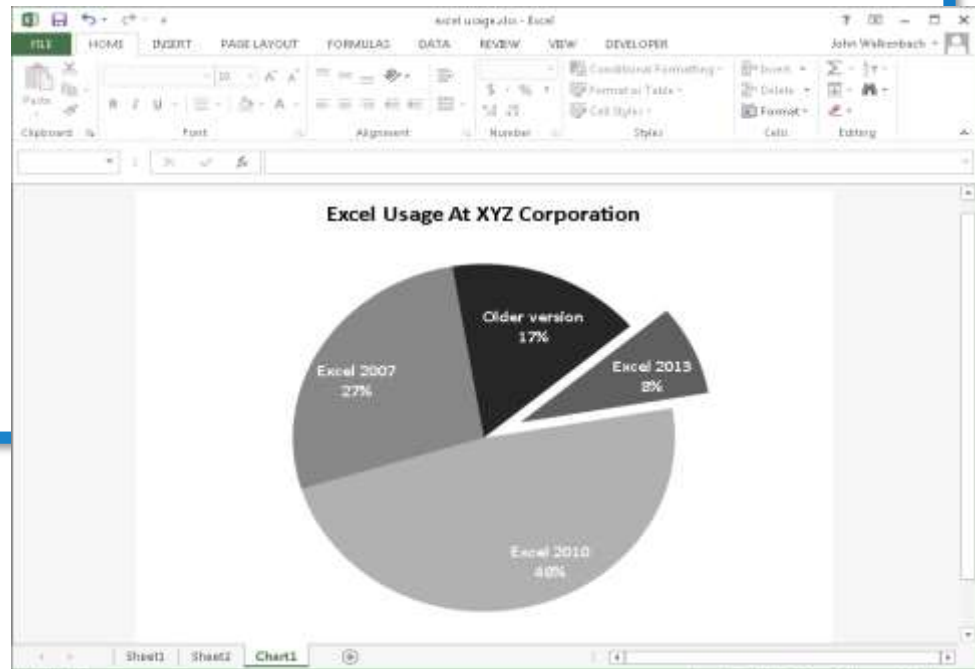


Fig 1

Cell and Range References

Cell references come in four styles:

- **Relative:** The reference is fully relative → When the formula is copied, the cell reference **adjusts to its new location**. **Example:** A1.
- **Absolute:** The reference is fully absolute → When the formula is copied, the cell reference **doesn't change**. **Example:** \$A\$1.
- **Row Absolute:** The reference is *partially absolute*. When the formula is copied, the **column part adjusts**, but the row part doesn't change. **Example:** A\$1.
- **Column Absolute:** The reference is *partially absolute*. When the formula is copied, the row part adjusts, but the column part doesn't change. **Example:** \$A1.

By default, all cell and range references are relative.

Cell and Range References

The screenshot shows the Microsoft Excel interface. The ribbon is set to 'Formulas'. The active cell is C2, and the formula bar shows the formula $=A\$2$. The spreadsheet data is as follows:

	A	B	C	D	E	F	G
1							
2	12	66	12				
3	13	55	12				
4	14	34	12				
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							

الأرقام → صفوف
الحروف → أعمدة

The inset screenshot shows a smaller portion of the Excel spreadsheet. The active cell is D2, and the formula bar shows the formula $=\$A2$. The data in this view is:

	A	B	C	D	E
1					
2	12	66	12	12	
3	13	55	12	13	
4	14	34	12	14	
5					
6					

Cell and Range References

- To refer to a cell in a different worksheet

=Sheet2!A1+1

- You can also create link formulas that refer to a cell in a different workbook.

=[Budget.xlsx]Sheet1!A1

- If the workbook name in the reference includes one or more spaces, you must enclose it (and the sheet name) in single quotation marks. For example:

='[Budget For 2013.xlsx]Sheet1'!A1

='C:\Budgeting\Excel Files\[Budget For 2013.xlsx]Sheet1'!A1

Excel Formula error Values

Error Value	Explanation
#DIV/0!	The formula is trying to divide by 0 (zero), an operation that's not allowed on this planet. This error also occurs when the formula attempts to divide by a cell that is empty.
#N/A	The formula is referring (directly or indirectly) to a cell that uses the NA worksheet function to signal the fact that data isn't available. A lookup function that can't locate a value also returns #N/A.
#NAME?	The formula uses a name that Excel doesn't recognize. This can happen if you delete a name that's used in the formula, if you have unmatched quotes when using text, if you omit parentheses for a function that uses no arguments, or if you misspell a function or range name. A formula will also display this error if it uses a function defined in an add-in and that add-in isn't installed.
#NULL!	The formula uses an intersection of two ranges that don't intersect. (This concept is described in the section "Intersecting names," earlier in the chapter.)
#NUM!	A function argument has a problem; for example, the SQRT function is attempting to calculate the square root of a negative number. This error also appears if a calculated value is too large or too small. Excel doesn't support nonzero values less than 1E-307 or greater than 1E+308 in absolute value.
#REF!	The formula refers to a cell that isn't valid. This can happen if a cell used in a formula has been deleted from the worksheet.
#VALUE!	The formula includes an argument or operand of the wrong type. An <i>operand</i> is a value or cell reference that a formula uses to calculate a result. This error also occurs if your formula uses a custom VBA worksheet function that contains an error.
#####	A cell displays a series of hash marks under two conditions: The column isn't wide enough to display the result, or the formula returns a negative date or time value.

VBA

VBA is a specialized version of Microsoft's well-known **Visual Basic language**.

VBA is an Object-Oriented language within Microsoft Office suite:


- Excel, Word, Access, and Power Point.
- We will focus on **Excel**.
- Shortcut **to access the VBA interface** in Excel: **Alt+F11**.

Excel treats everything in a spreadsheet as objects.

So, **VBA** lets us deal with all these objects.

- ✓ Object that we are interested in: **Workbooks, worksheets, charts, cells, rows, ranges.**
- ✓ There are over 200 different class (types of objects) in Excel.

The Basics of VBA

- **Code:** Actions in VBA are performed by executing VBA code, stored in a VBA module.
- **Module:** VBA modules are stored in an Excel workbook file, but you view or edit a module by using Visual Basic Editor (VBE).
 - VBA module consists of procedures.
- **Procedures:** A procedure is basically a unit of computer code that performs some action.
 - VBA supports **two types of procedures:** Sub procedures and Function procedures. 

The Basics of VBA

● **Sub:** A **Sub procedure** consists of a series of statements and can be executed in a number of ways.

- Here's an example of a simple Sub procedure called **Test**: This procedure calculates a simple sum and then displays the result in a message box.

```
Sub Test()  
    Sum = 1 + 1  
    MsgBox "The answer is " & Sum  
End Sub
```



The Basics of VBA

● **FUNCTION:**

A Function procedure **returns a single value** (or possibly an array).

A Function can be **called from another VBA procedure or used in a worksheet formula.**

Here's an example of a Function named **AddTwo:**

```
Function AddTwo(arg1, arg2)
```

```
    AddTwo = arg1 + arg2
```

```
End Function
```



The Basics of VBA

- **Objects:** VBA manipulates objects contained in its host application. (In this case, Excel is the host application.)
 - **Examples of Excel objects** include a **workbook**, a **worksheet**, a **range** on a worksheet, a **chart**, and a **shape**.



The Basics of VBA

- **Objects** also can act as containers for other objects.
- **For example**, **Excel** is an object called **Application**, and it contains other objects, such as **Workbook objects**.
- The **Workbook** object contains other objects, such as **Worksheet objects** and **Chart objects**.
- A **Worksheet** object contains objects such as **Range** objects, **PivotTable** objects, and so on.
- The arrangement of these objects is referred to as Excel's ***object model***.



The Basics of VBA

➤ **Collections:** Like objects form a *collection*. **For example**, the Worksheets collection consists of all the worksheets in a particular workbook.

Collections are objects in themselves.

➤ **Object hierarchy:** When you refer to an object, you specify its position in the object hierarchy by using a **dot as a separator between the container and the member**.

For example, you can refer to a workbook named **Book1.xlsx** as

`Application.Workbooks("Book1.xlsx")`

This code refers to the **Book1.xlsx** workbook in the **Workbooks** collection.



The Basics of VBA

The **Workbooks** collection is contained in the Excel Application object.

You can refer to **Sheet1** in **Book1** as

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1")
```

and refer to a specific **cell** as follows:

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1")  
.Range("A1")
```



The Basics of VBA

➤ **Active objects:** If you omit a specific reference to an object, Excel uses the active objects.

- If **Book1** is the active workbook, the preceding reference can be simplified as

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

- If you know that **Sheet1** is the active sheet, you can simplify the reference even more:

`Range("A1")`



The Basics of VBA

➤ Objects properties:

Objects have *properties*.

For example,

A **range** object has properties such as **Value** and **Address**.

A **chart** object has properties such as **HasTitle** and **Type**.

A **Shape** object has properties such as **Width** and **Height**.

You can use VBA to determine object properties and also to change them.

Some properties are **read-only** properties and can't be changed by using VBA.

You refer to properties by combining the object with the property, separated by a **dot**.

For example, you can refer to **the value in cell A1 on Sheet1** as

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



The Basics of VBA

➤ VBA variables:

- You can assign values to VBA variables.
- Think of a variable as **a name that you can use to store a particular value.**
- **To assign the value in cell A1 on Sheet1 to a variable called Interest, use the following VBA statement:**

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



The Basics of VBA

➤ Object methods:

- Objects have **methods**.
- A **method** is an action that is performed with the object.
- For example, one of the methods for a **Range** object is **ClearContents**. (This method clears the contents of the range).
- You specify methods by combining the object with the method, separated by a **dot**.
- **For example**, to clear the contents of cell A1 on the active worksheet, use
`Range("A1").ClearContents`



The Basics of VBA

➤ **Standard programming constructs:** VBA also includes many constructs found in modern programming languages, including **arrays**, **conditional statements**, and **loops**.

➤ **Events:** Some objects recognize specific **events**, and you can write **VBA** code that is executed when the event occurs.

For example, opening a workbook triggers a `Workbook_Open` event.

Changing a cell in a worksheet triggers a `Worksheet_Change` event.

