VBA Syllabus

Excel VBA (**Visual Basic** for Applications) is the name of the programming language of Excel.

1. **Create a Macro:**

With Excel VBA you can automate tasks in Excel by writing so called macros. In this chapter, learn how to create a simple **macro**.

2. **MsgBox:**

The MsgBox is a dialog box in Excel VBA you can use to inform the users of your program.

3. **Workbook and Worksheet Object:**

Learn more about the Workbook and Worksheet object in Excel VBA.

4. **Range Object:**

The Range object, which is the representation of a cell (or cells) on your worksheet, is the most important object of Excel VBA.

5. **Variables:**

This chapter teaches you how to declare, initialize and display a variable in Excel VBA.

6. **If Then Statement:**

Use the If Then statement in Excel VBA to execute code lines if a specific condition is met.

7. **Loop:**

Looping is one of the most powerful programming techniques. A loop in Excel VBA enables you to loop through a range of cells with just a few codes lines.

8. **Macro Errors:**

This chapter teaches you how to deal with macro errors in Excel.

9. **String Manipulation:**

In this chapter, you'll find the most important functions to manipulate strings in Excel VBA.

10. Date and Time:

Learn how to work with dates and times in Excel VBA.

11. Events:

Events are actions performed by users which trigger Excel VBA to execute code.

12. Array:

An array is a group of variables. In Excel VBA, you can refer to a specific variable (element) of an array by using the array name and the index number.

13. Function and Sub:

In Excel VBA, a function can return a value while a sub cannot.

14. Application Object:

The mother of all objects is Excel itself. We call it the Application object. The application object gives access to a lot of Excel related options.

15. ActiveX Controls:

Learn how to create ActiveX controls such as command buttons, text boxes, list boxes etc.

16. Userform:

This chapter teaches you how to create an Excel VBA Userform.

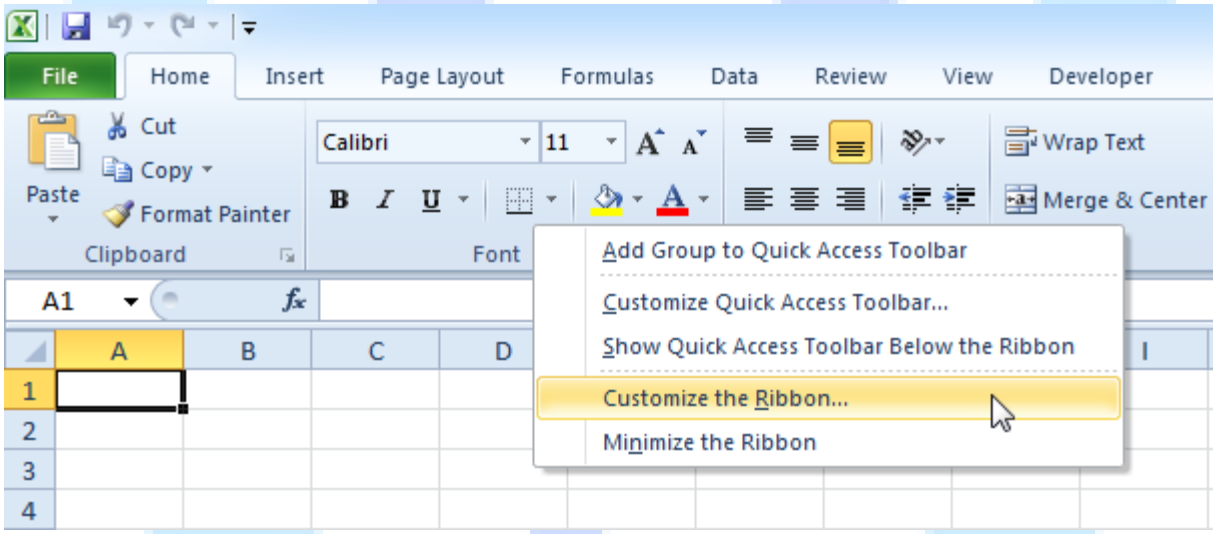
Create a Macro

With Excel VBA you can automate tasks in Excel by writing so called macros. In this chapter, learn how to create a simple macro which will be executed after clicking on a command button. First, turn on the Developer tab.

Developer Tab

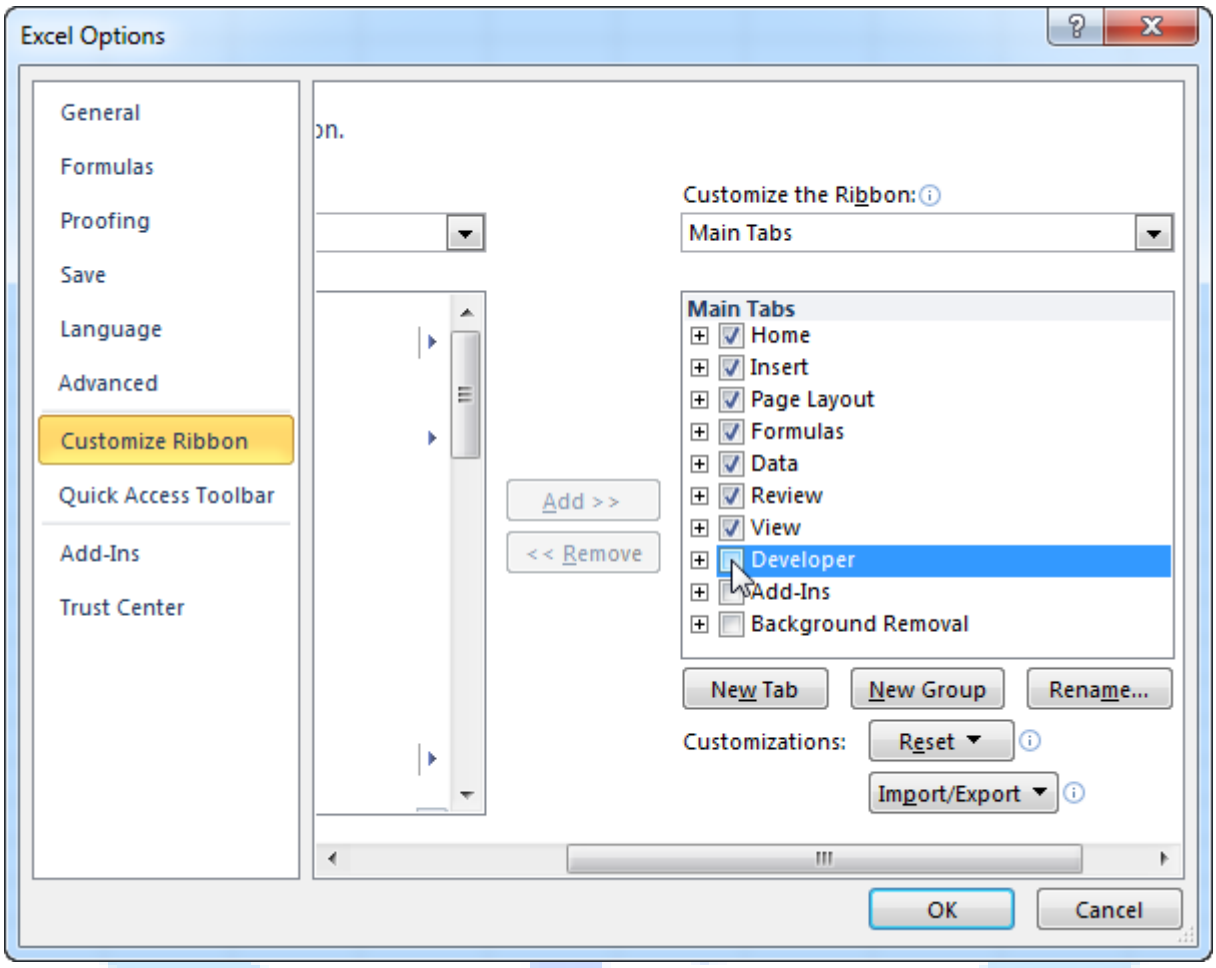
To turn on the Developer tab, execute the following steps.

1. Right click anywhere on the ribbon, and then click Customize the Ribbon.



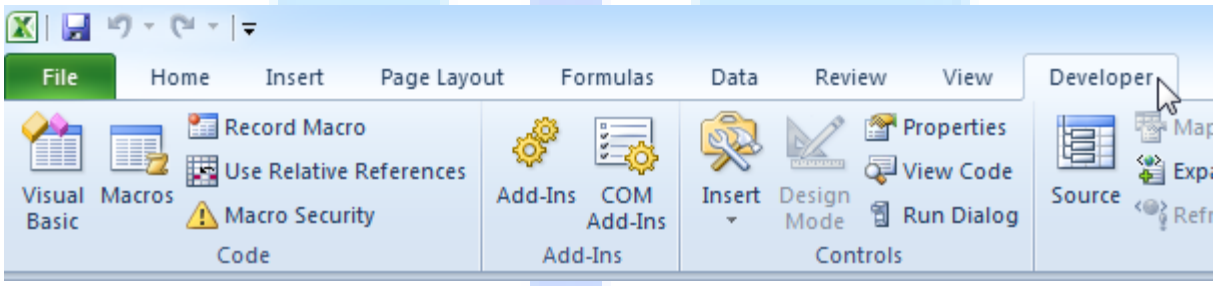
2. Under Customize the Ribbon, on the right side of the dialog box, select Main tabs (if necessary).

3. Check the Developer check box.



4. Click OK.

5. You can find the Developer tab next to the View tab.



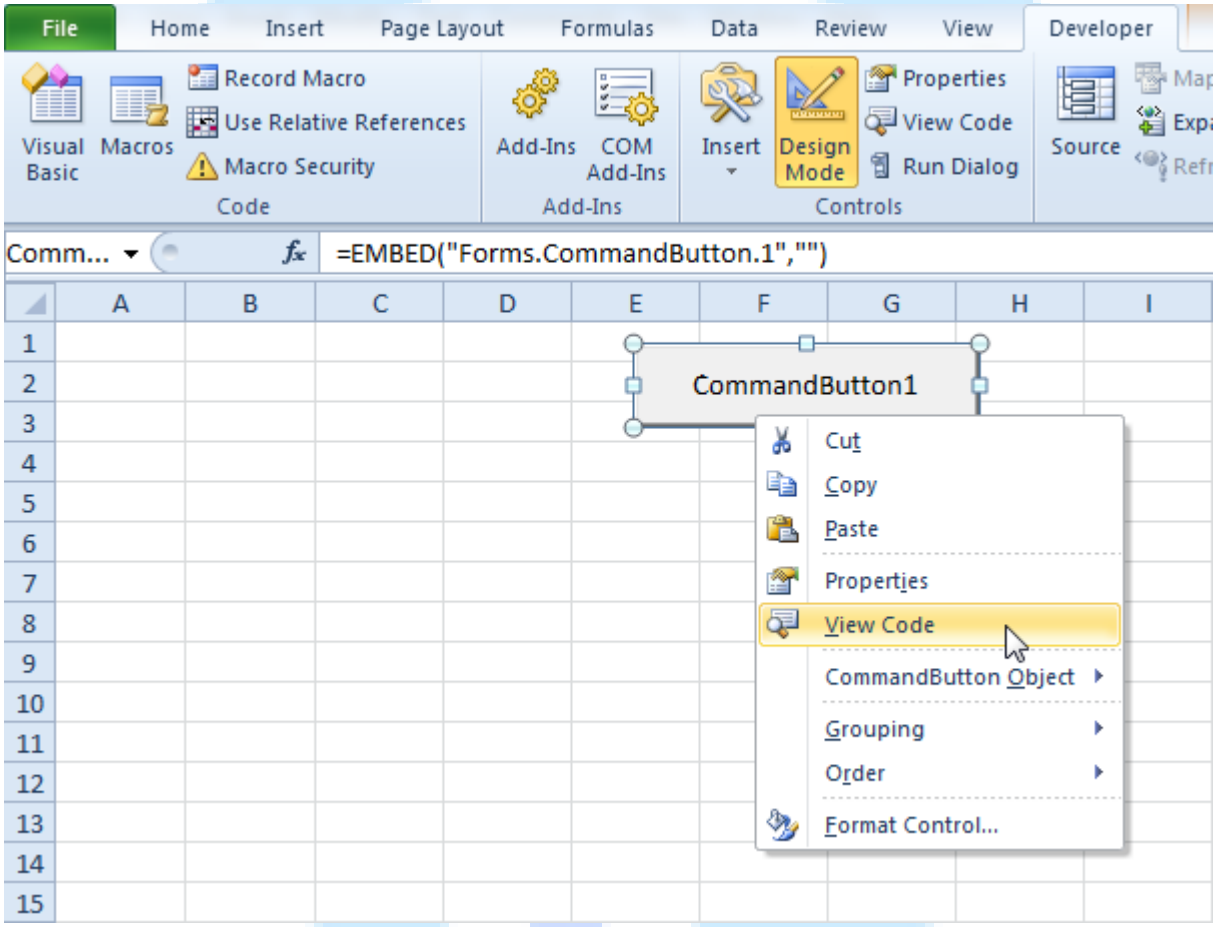
3. Drag a command button on your worksheet.

Assign a Macro

To assign a macro (one or more code lines) to the command button, execute the following steps.

1. Right click CommandButton1 (make sure Design Mode is selected).

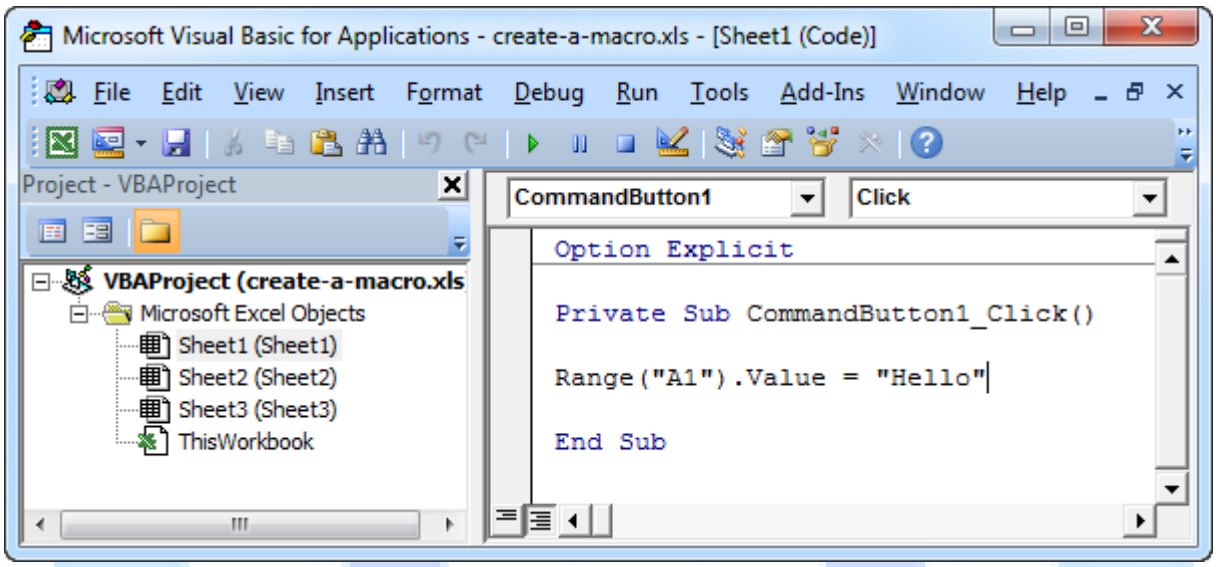
2. Click View Code.



The Visual Basic Editor appears.

3. Place your cursor between Private Sub CommandButton1_Click() and End Sub.

4. Add the code line shown below.

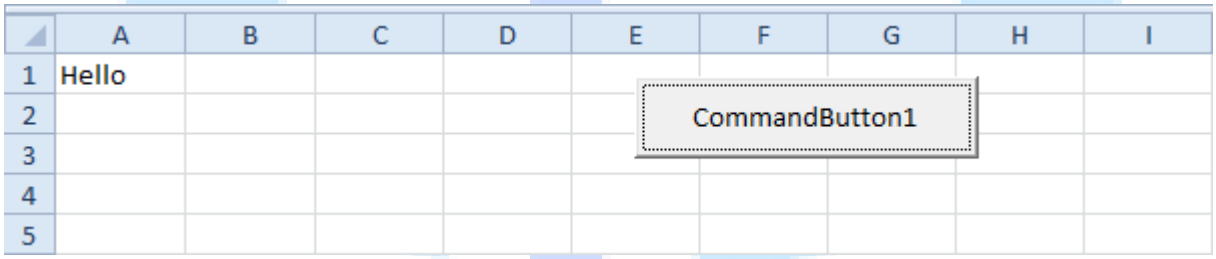


Note: the window on the left with the names Sheet1, Sheet2 and Sheet3 is called the Project Explorer. If the Project Explorer is not visible, click View, Project Explorer. To add the Code window for the first sheet, click Sheet1 (Sheet1).

5. Close the Visual Basic Editor.

6. Click the command button on the sheet (make sure Design Mode is deselected).

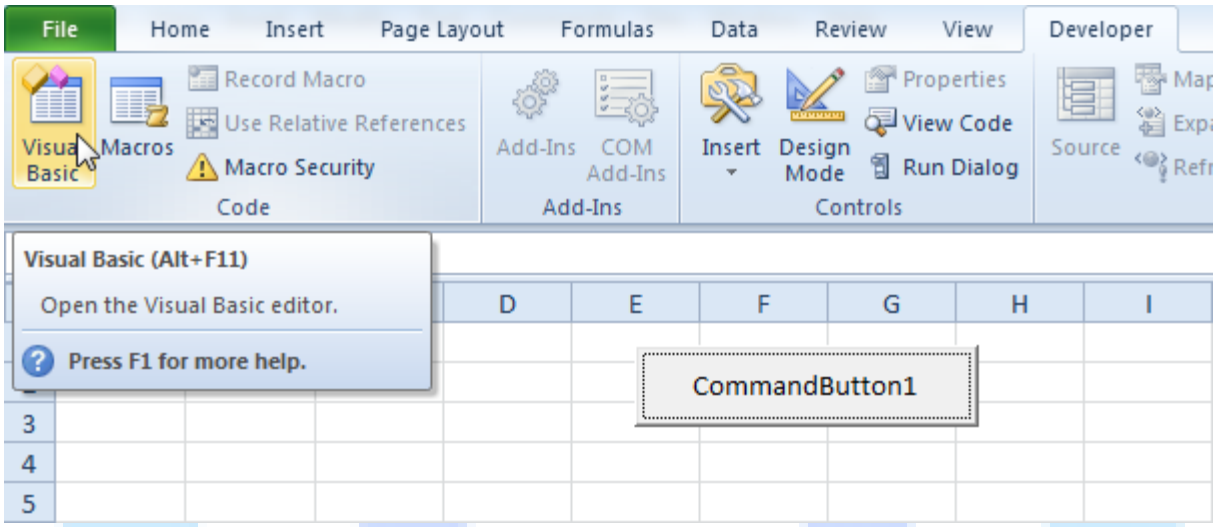
Result:

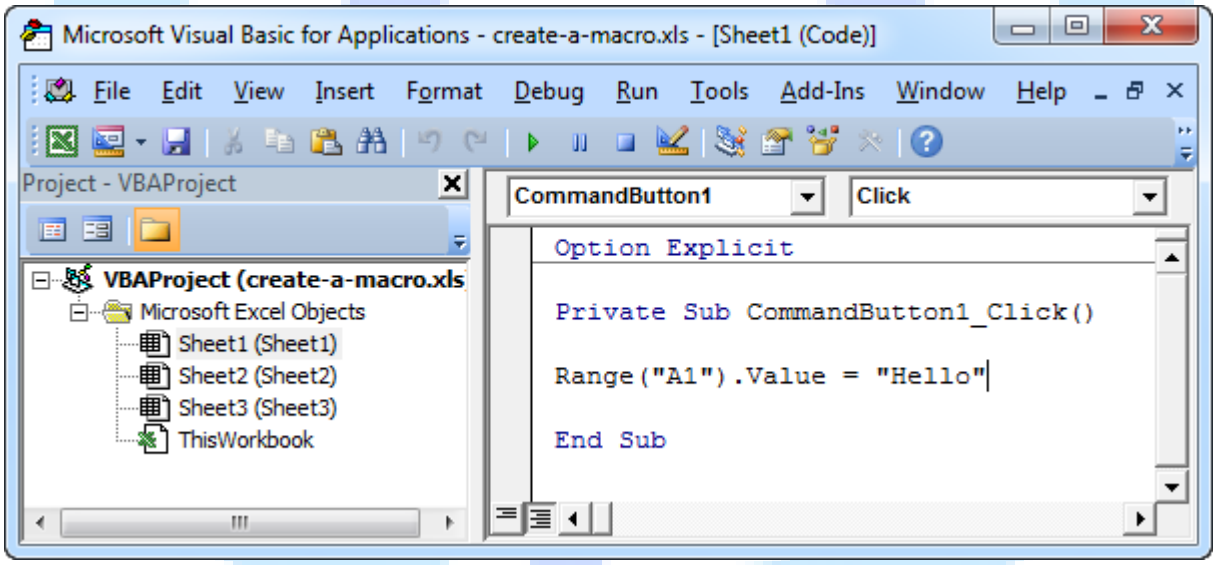


Congratulations. You've just created a macro in Excel!

Visual Basic Editor

To open the Visual Basic Editor, on the Developer tab, click Visual Basic.





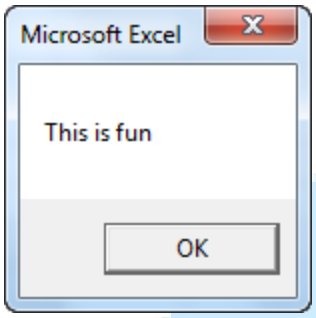
MsgBox

The **MsgBox** is a dialog box in **Excel VBA** you can use to inform the users of your program. Place a command button on your worksheet and add the following code lines:

1. A simple message.

MsgBox "This is fun"

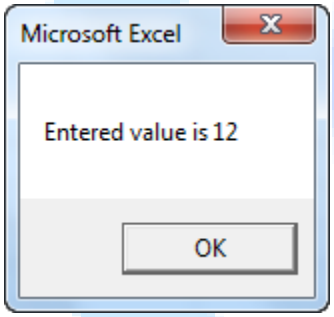
Result when you click the command button on the sheet:



2. A little more advanced message. First, enter a number into cell A1.

MsgBox "Entered value is " & Range("A1").Value

Result when you click the command button on the sheet:

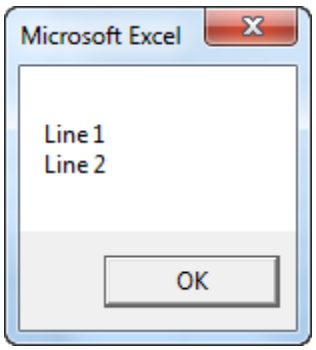


Note: we used the & operator to concatenate (join) two strings. Although Range("A1").value is not a string, it works here.

3. To start a new line in a message, use vbNewLine.

MsgBox "Line 1" & vbNewLine & "Line 2"

Result when you click the command button on the sheet:



Workbook and Worksheet Object

Learn more about the Workbook and Worksheet object in Excel VBA.

Object Hierarchy

In Excel VBA, an object can contain another object, and that object can contain another object, etc. In other words, Excel VBA programming involves working with an object hierarchy. This probably sounds quite confusing, but we will make it clear.

The mother of all objects is Excel itself. We call it the Application object. The application object contains other objects. For example, the Workbook object (Excel file). This can be any workbook you have created. The Workbook object contains other objects, such as the Worksheet object. The Worksheet object contains other objects, such as the Range object.

The Create a Macro chapter illustrates how to run code by clicking on a command button. We used the following code line:

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

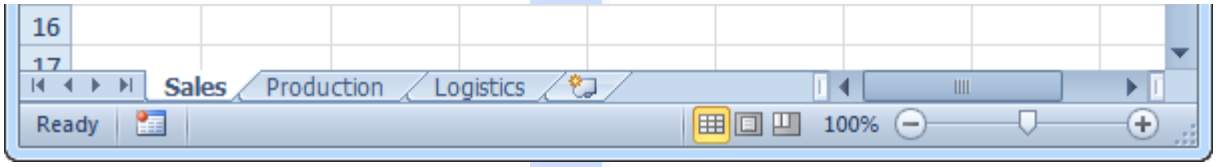
but what we really meant was:

```
Application.Workbooks("create-a-macro").Worksheets(1).Range("A1").Value = "Hello"
```

Note: the objects are connected with a dot. Fortunately, we do not have to add a code line this way. That is because we placed our command button in create-a-macro.xls, on the first worksheet. Be aware that if you want to change things on different worksheets, you have to include the Worksheet object. Read on.

Collections

You may have noticed that Workbooks and Worksheets are both plural. That is because they are collections. The Workbooks collection contains all the Workbook objects that are currently open. The Worksheets collection contains all the Worksheet objects in a workbook.



You can refer to a member of the collection, for example, a single Worksheet object, in three ways.

1. Using the worksheet name.

Worksheets("Sales").Range("A1").Value = "Hello"

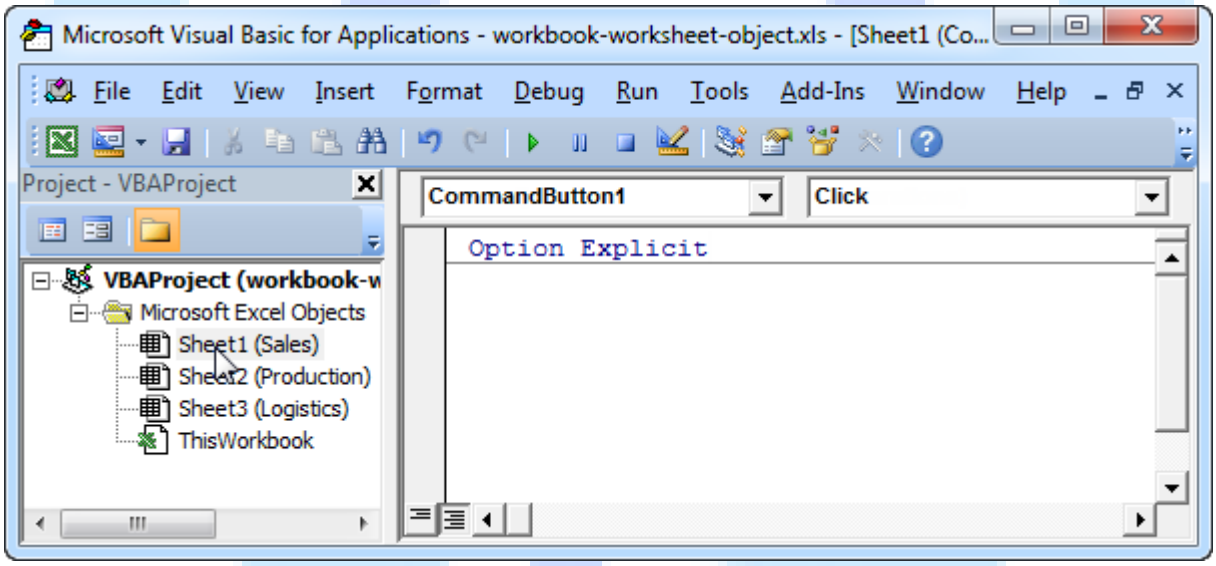
2. Using the index number (1 is the first worksheet starting from the left).

Worksheets(1).Range("A1").Value = "Hello"

3. Using the CodeName.

Sheet1.Range("A1").Value = "Hello"

To see the CodeName of a worksheet, open the Visual Basic Editor. In the Project Explorer, the first name is the CodeName. The second name is the worksheet name (Sales).



Note: the CodeName remains the same if you change the worksheet name or the order of your worksheets so this is the safest way to reference a worksheet. Click View, Properties Window to change the CodeName of a worksheet. There is one disadvantage, you cannot use the CodeName if you reference a worksheet in a different workbook.

Properties and Methods

Now let's take a look at some properties and methods of the Workbooks and Worksheets collection. Properties are something which an collection has (they describe the collection), while methods do something (they perform an action with an collection).

Place a command button on your worksheet and add the code lines:

1. The Add method of the Workbooks collection creates a new workbook.

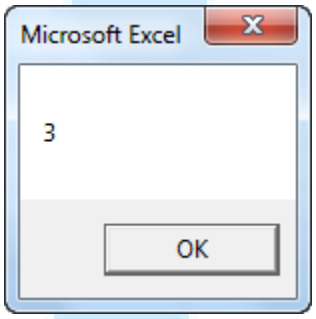
Workbooks.Add

Note: the Add method of the Worksheets collection creates a new worksheet.

2. The Count property of the Worksheets collection counts the number of worksheets in a workbook.

MsgBox Worksheets.Count

Result when you click the command button on the sheet:



Note: the Count property of the Workbooks collection counts the number of active Worksheet.

Range Object

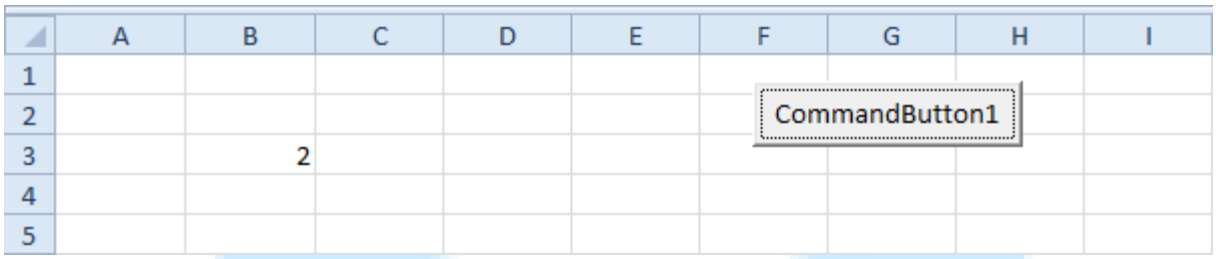
The **Range object**, which is the representation of a cell (or cells) on your worksheet, is the most important object of **Excel VBA**. This chapter gives an overview of the properties and methods of the Range object. Properties are something which an object has (they describe the object), while methods do something (they perform an action with an object).

Range Examples

Place a command button on your worksheet and add the following code line:

Range("B3").Value = 2

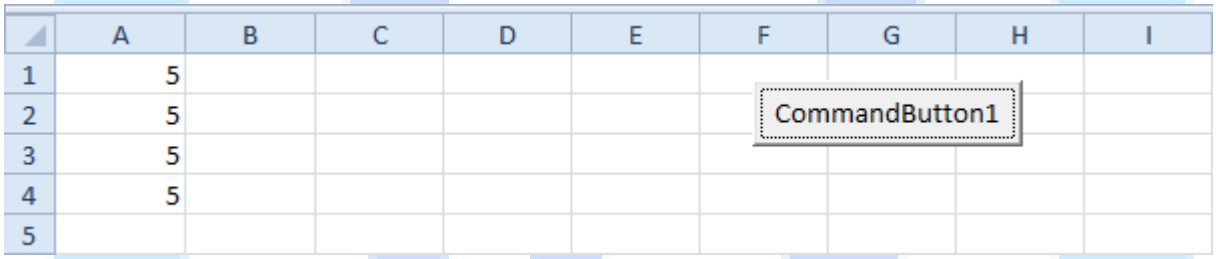
Result when you click the command button on the sheet:



Code:

```
Range("A1:A4").Value = 5
```

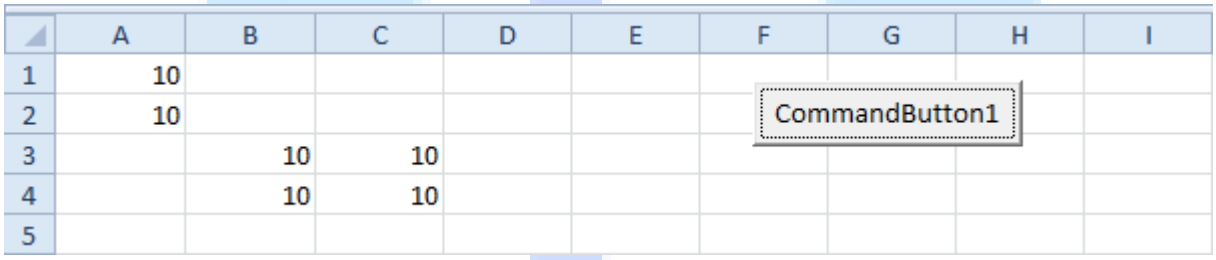
Result:



Code:

```
Range("A1:A2,B3:C4").Value = 10
```

Result:



Note: to refer to a named range in your Excel VBA code, use a code line like this:

```
Range("Prices").Value = 15
```

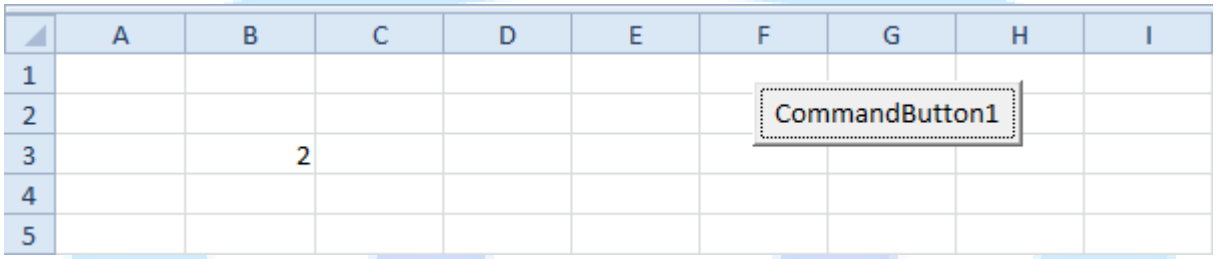
Cells

Instead of Range, you can also use Cells. Using Cells is particularly useful when you want to loop through ranges.

Code:

```
Cells(3, 2).Value = 2
```

Result:

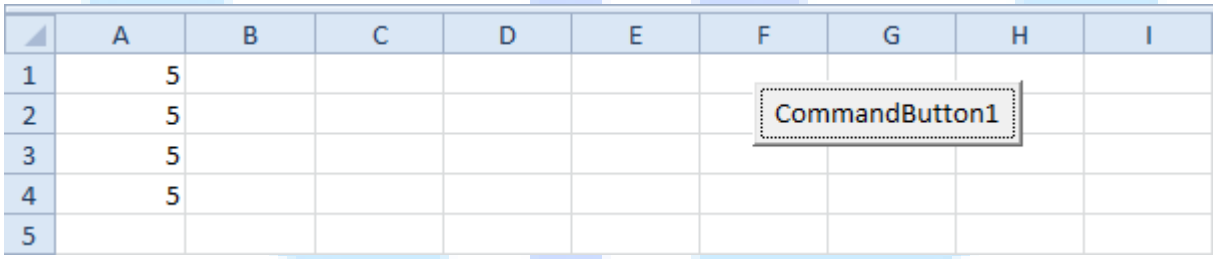


Explanation: Excel VBA enters the value 2 into the cell at the intersection of row 3 and column 2.

Code:

```
Range(Cells(1, 1), Cells(4, 1)).Value = 5
```

Result:



Declare a Range Object

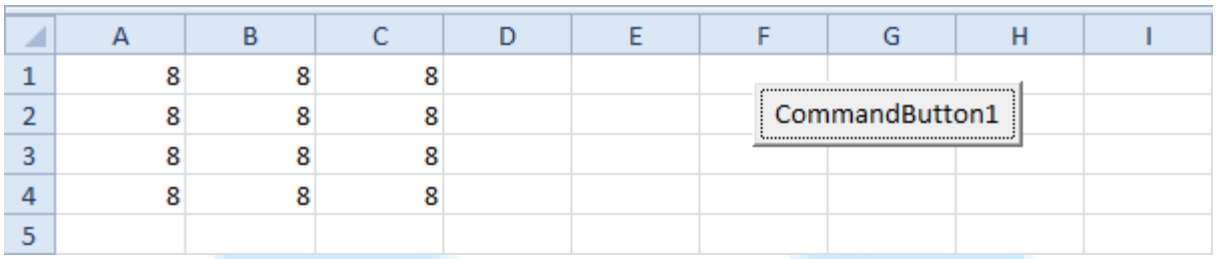
You can declare a Range object by using the keywords Dim and Set.

Code:

```
Dim example As Range
Set example = Range("A1:C4")

example.Value = 8
```

Result:



Select

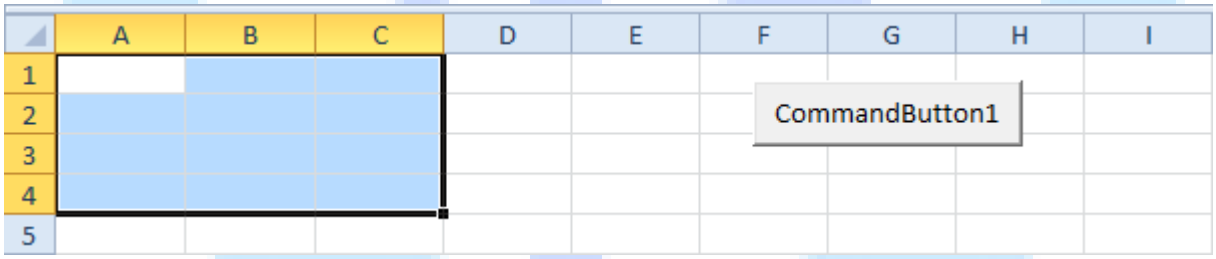
An important method of the Range object is the Select method. The Select method simply selects a range.

Code:

```
Dim example As Range
Set example = Range("A1:C4")

example.Select
```

Result:



Rows

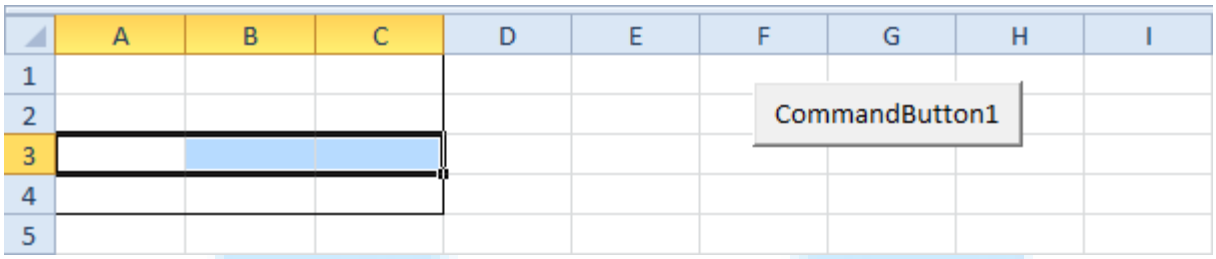
The Rows property gives access to a specific row of a range.

Code:

```
Dim example As Range
Set example = Range("A1:C4")

example.Rows(3).Select
```

Result:



Note: border for illustration only.

Columns

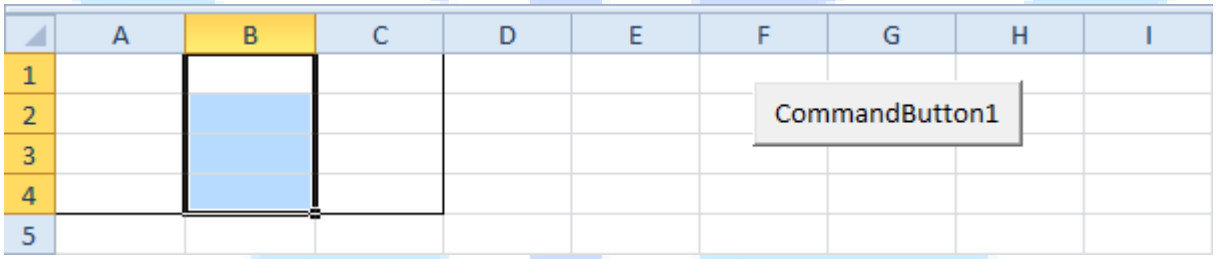
The Columns property gives access to a specific column of a range.

Code:

```
Dim example As Range
Set example = Range("A1:C4")

example.Columns(2).Select
```

Result:



Note: border for illustration only.

Copy/Paste

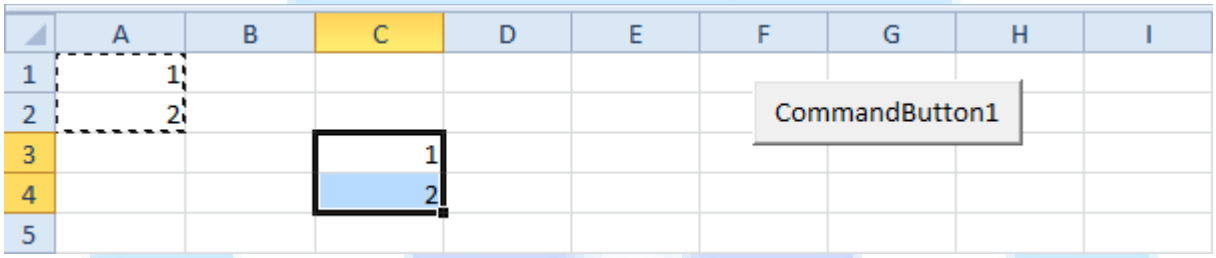
The Copy and Paste method are used to copy a range and to paste it somewhere else on the worksheet.

Code:

```
Range("A1:A2").Select
Selection.Copy
```

```
Range("C3").Select
ActiveSheet.Paste
```

Result:



Although this is allowed in Excel VBA, it is much better to use the code line below which does exactly the same.

```
Range("C3:C4").Value = Range("A1:A2").Value
```

Clear

To clear the content of an Excel range, you can use the ClearContents method.

```
Range("A1").ClearContents
```

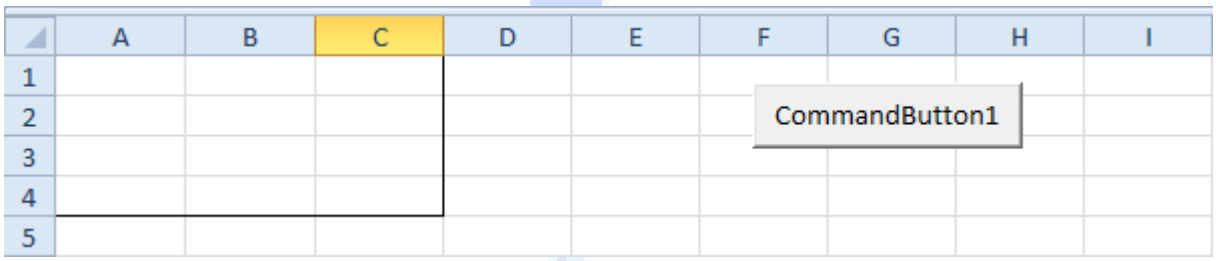
or simply use:

```
Range("A1").Value = ""
```

Note: use the Clear method to clear the content and format of a range. Use the ClearFormats method to clear the format only.

Count

With the Count property, you can count the number of cells, rows and columns of a range.



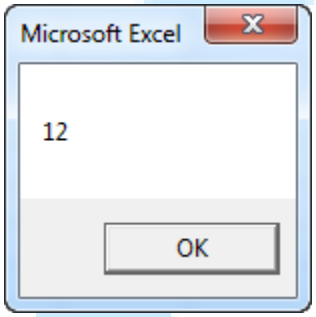
Note: border for illustration only.

Code:

```
Dim example As Range
Set example = Range("A1:C4")

MsgBox example.Count
```

Result:

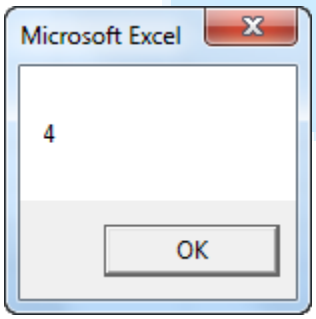


Code:

```
Dim example As Range
Set example = Range("A1:C4")

MsgBox example.Rows.Count
```

Result:



Variables

This chapter teaches you how to declare, initialize and display a **variable** in **Excel VBA**. Letting Excel VBA know you are using a variable is called declaring a variable. Initializing simply means assigning a beginning (initial) value to a variable.

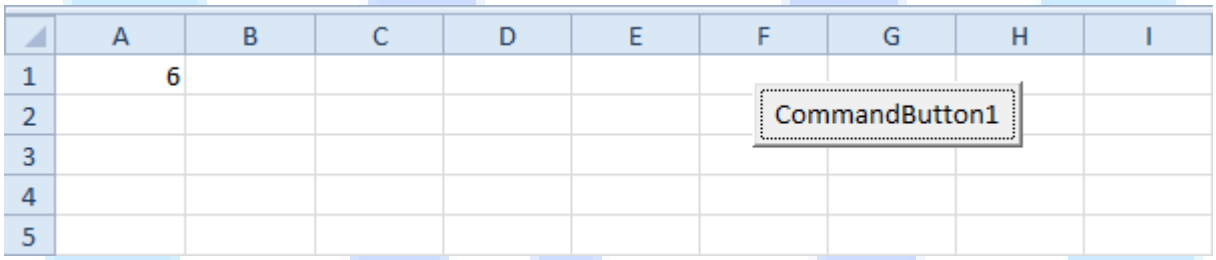
Place a command button on your worksheet and add the code lines below. To execute the code lines, click the command button on the sheet.

Integer

Integer variables are used to store whole numbers.

```
Dim x As Integer
x = 6
Range("A1").Value = x
```

Result:



Explanation: the first code line declares a variable with name x of type Integer. Next, we initialize x with value 6. Finally, we write the value of x to cell A1.

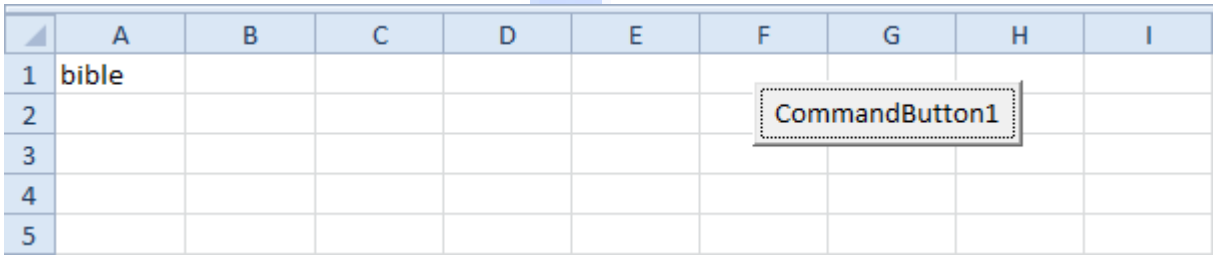
String

String variables are used to store text.

Code:

```
Dim book As String
book = "bible"
Range("A1").Value = book
```

Result:



Explanation: the first code line declares a variable with name book of type String. Next, we initialize book with the text bible. Always use apostrophes to initialize String variables. Finally, we write the text of the variable book to cell A1.

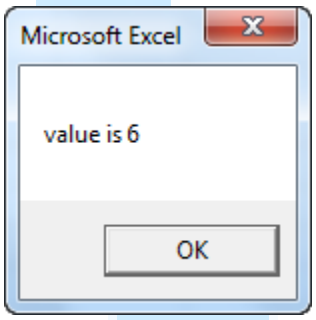
Double

A variable of type Double is more accurate than a variable of type Integer and can also store numbers after the comma.

Code:

```
Dim x As Integer
x = 5.5
MsgBox "value is " & x
```

Result:

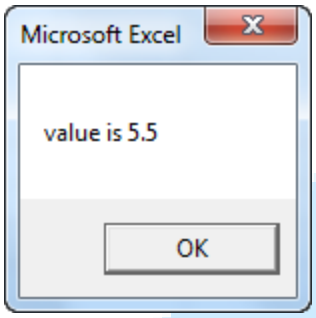


But that is not the right value! We initialized the variable with value 5.5 and we get the value 6. What we need is a variable of type Double.

Code:

```
Dim x As Double
x = 5.5
MsgBox "value is " & x
```

Result:



Note: Long variables have even larger capacity. Always use variables of the right type. As a result, errors are easier to find and your code will run faster.

Boolean

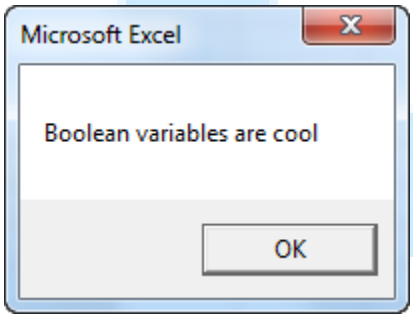
Use a Boolean variable to hold the value True or False.

Code:

```
Dim continue As Boolean
continue = True

If continue = True Then MsgBox "Boolean variables are cool"
```

Result:



Explanation: the first code line declares a variable with name continue of type Boolean. Next, we initialize continue with the value True. Finally, we use the Boolean variable to only display a MsgBox if the variable holds the value True.