

Welcome to Excel VBA Easy

Our easy to follow tutorial (Macro?, Basics, Programming and Controls) teaches you how to create all sorts of macros in Excel VBA. Below you can find a complete overview. It doesn't matter if you are new to Excel VBA and want to start from scratch or just want to use this tutorial as a reference. We keep it simple!

Jesse - Newport Beach, United States - *"You have the simplest, most comprehensible material I have found for a VBA beginner."* [\[Read More\]](#)

There is really only one way to master Excel VBA, and that is by doing it. Good example programs say more than a lengthy description of the theory. Invest in yourself by ordering our example programs and learn Excel VBA quickly and easily! [**100 Excel VBA examples >>**](#)

Macro?

This section is for Excel users with no knowledge of Excel VBA.

Excel VBA, which stands for Excel Visual Basic for Applications, is the name of the programming language of Microsoft Excel. With Excel VBA you can automate a task in Excel by writing a so called Macro. This can save you a lot of time! More importantly there are certain things you cannot do with Excel alone. Excel VBA allows you to do these things in Excel.

1 [Create a Macro](#): To create a macro in Excel VBA, first activate Excel Visual Basic. Next, you can create a command button and assign a macro to the command button.

2 [Excel Macro Recorder](#): The Excel Macro Recorder is a very useful tool included in Excel VBA. With the Excel Macro Recorder you can record a task you perform with Excel. Next, you can execute the task over and over with the click of a button which can save you a lot of time. This chapter explains the ins and outs of the Excel Macro Recorder.

[back to top](#)

Basics

This section explains the basics of Excel Visual Basic. It is good to know the basic terminology explained in this section before you start programming in Excel Visual Basic.

1 [Macro Security](#): Setting up your macro security settings correctly is essential to protect yourself against potential viruses. Make sure your macro security settings are set up correctly so no harm can be done to your computer.

2 [Visual Basic Editor](#): Learn how to launch the Visual Basic Editor and get the best configuration of the Project Explorer and the Code Window in your Excel Version. The Visual Basic Editor is the starting point for creating macros in Excel VBA, so it is important to get this configuration right.

3 [Macro Comments](#): Add macro comments to your Excel VBA code and your code will be easier to read as program size increases.

4 [MsgBox](#): The Message Box is a dialog box you can have appear to inform the users of your program.

5 [Macro Errors](#): Dealing with VBA-errors can be quite a challenge. This chapter provides you with a simple tip to deal with these errors.

6 [Debug Macros](#): Before you execute your VBA-code you can first debug your macro. This way most of the errors can be corrected before you execute your code.

7 [Objects, Properties and Methods](#): In this chapter you will learn more about Excel VBA objects. An object has properties and methods. Excel Visual Basic is a semi-object oriented programming language. Learn more about the object hierarchy of Excel Visual Basic.

8 [Workbook and Worksheet](#): In this chapter you will learn more about the Excel VBA Workbook and Excel VBA Worksheet object. You will see that the Worksheet and Workbook object have properties and methods as well, such as the count property which counts the number of active workbooks or worksheets. The Workbook and Worksheet object are commonly used in Excel VBA. They are very useful when your macro code has to be executed on different workbooks or worksheets.

9 [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.

[back to top](#)

Programming

This section is for users who want to get the most out of Excel VBA. Excel VBA Programming is not difficult, but you do need to know the keywords used in Excel VBA.

1 [Variables](#): Excel VBA uses variables just like any other programming language. Learn how to declare and initialize an excel vba variable of type Integer, String, Double, Boolean and Date.

2 [String Manipulation](#): There are many functions in Excel VBA we can use to manipulate strings. In this chapter you can find a review of the most important functions.

3 [Calculate](#): Calculate with Excel VBA and add, subtract, multiply and divide values just like you are used to doing in Excel.

4 [If Then Statement](#): In many situations you only want Excel VBA to execute certain code lines when a specific condition is met. The If Then statement allows you to do this. Instead of multiple If Then statements, you can use Select Case.

5 [Cells](#): Instead of the more common Range object we could also use Cells. Using Cells is particularly useful when we want to loop through ranges.

6 [Loop](#): Looping is one of the most powerful programming techniques. A loop (or For Next loop) in Excel VBA enables you to loop through a range of data with just a few lines of code.

7 [Logical Operators](#): Do you want to execute code in Excel Visual Basic when more conditions are met? Or just one? Or none? Logical operators are what you need! Logical operators such as And, Or and Not are often used in Excel VBA.

8 [Range](#): The Range object which is the representation of a cell (or cells) on your worksheet is the most important object of Excel VBA. It has many properties and methods and they are essential to manipulate the content of your Excel worksheet. In this chapter you will discover the most useful properties and methods of the Excel VBA Range object. They enable you to obtain control over your Excel worksheet.

9 [Events](#): This chapter teaches you how to program workbook and worksheet events. Events are actions performed by users which trigger Excel VBA to execute a macro. For example, when you open a workbook or when you change something on an Excel worksheet, Excel VBA can automatically execute a macro.

10 [Array](#): An Excel VBA array is a group of variables. You can refer to a specific variable (element) of an array by using the array name and the index number.

11 [Date and Time](#): Dates and Times in Excel VBA can be manipulated in many ways. Easy examples are given in this chapter.

12 [Function and Sub](#): The difference between a function and a sub in Excel VBA is that a function can return a value and a sub cannot. In this chapter we will look at an easy example of a function and a sub. Functions and subs become very useful as program size increases.

[back to top](#)

Controls

This section is about communicating with users using controls or a Userform. Learn how to use these controls in Excel 2010, Excel 2007 or Excel 2003. You can directly place controls on a sheet or place them on a Userform.

1 [Textbox](#): A textbox is an empty field where the user can fill in a piece of text. Learn how to draw a textbox on your worksheet, how to refer to a textbox in your Excel VBA code, and how to clear a textbox.

2 [Listbox](#): A listbox, is a drop down list from where the user can make a choice. Learn how to draw a listbox on your worksheet and how to add items to a listbox.

3 [Combobox](#): A combobox is the same as a listbox but now the user can also fill in his/her own choice if it is not included in the list. Learn how to draw a combobox on your worksheet and how to add items to a combobox.

4 [Checkbox](#): A checkbox is a field which can be checked to store information. Learn how to draw a checkbox on your worksheet and how to refer to a checkbox in your Excel VBA code.

5 [Option Buttons](#): Option buttons are the same as checkboxes except that option buttons are dependent on each other while checkboxes are not. This means that when you check one option button the other option button will automatically uncheck.

6 [Userform](#): This chapter teaches you how to create an Excel VBA Userform (also known as a dialog box). You can download the Userform on this page as well.

[back to top](#)

Create a Macro

Activating Excel VBA and creating command buttons differs from one Excel version to another. Please choose your version.

[Excel 2010](#) | [Excel 2007](#) | [Excel 2003](#)

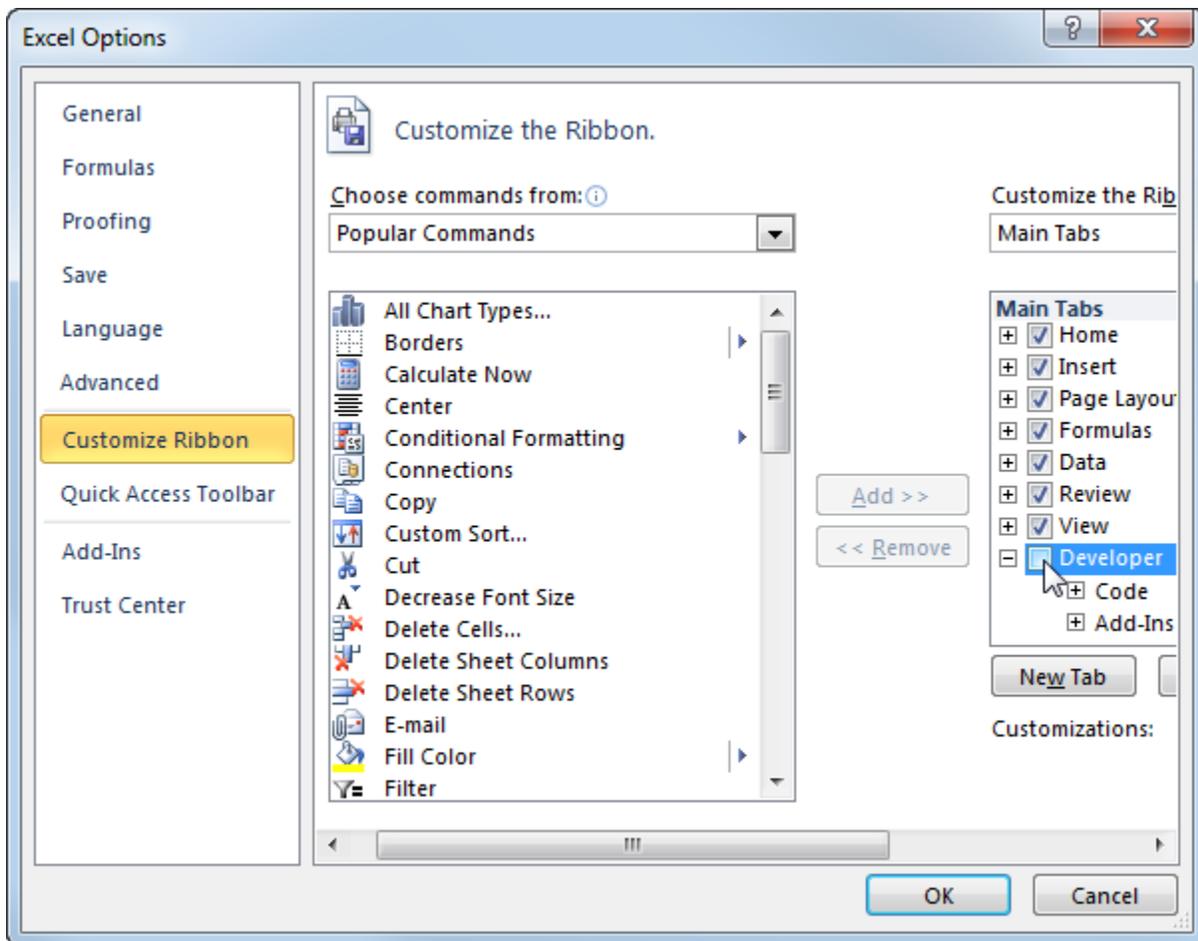
Create a Macro in Excel 2010

[Turn on the Developer Tab](#) | [Create a command button](#) | [Create and Assign the Macro](#)

To create a macro in Excel 2010, you have to turn on the Developer tab. Next, you can create a macro which will be executed after clicking on a command button.

Turn on the Developer Tab

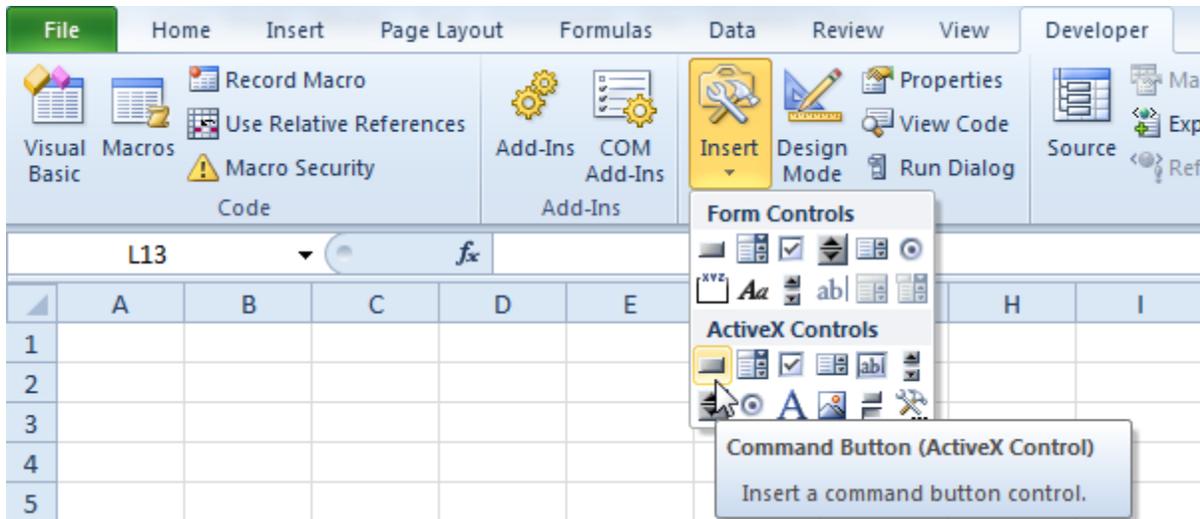
1. Click on the File tab and choose Options. The Excel Options dialog box appears (see picture below).
2. Click Customize Ribbon on the left side of the dialog box.
3. Under Choose commands from on the left side of the dialog box, select Popular Commands (if necessary).
4. Under Customize the ribbon on the right side of the dialog box, select Main tabs (if necessary).
5. Check the Developer check box and click OK.



Create a command button

You can now click on the Developer tab which has been placed next to the View tab.

1. Click on the Developer tab in the ribbon.
2. Click on Command Button in the ActiveX Controls section.

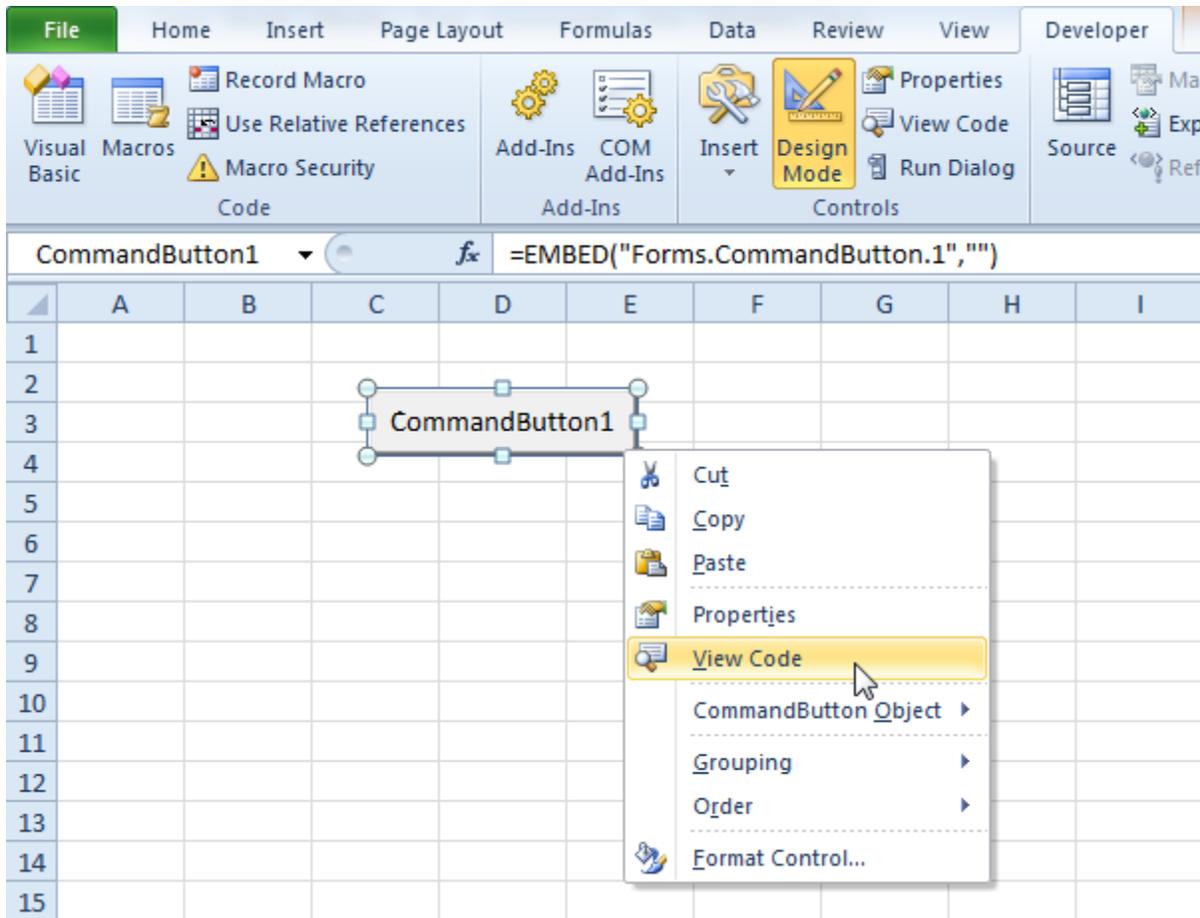


3. Now you can drag a command button on your worksheet.

Create and Assign the Macro

Now it is time to create a macro (a piece of code) and assign it to the command button.

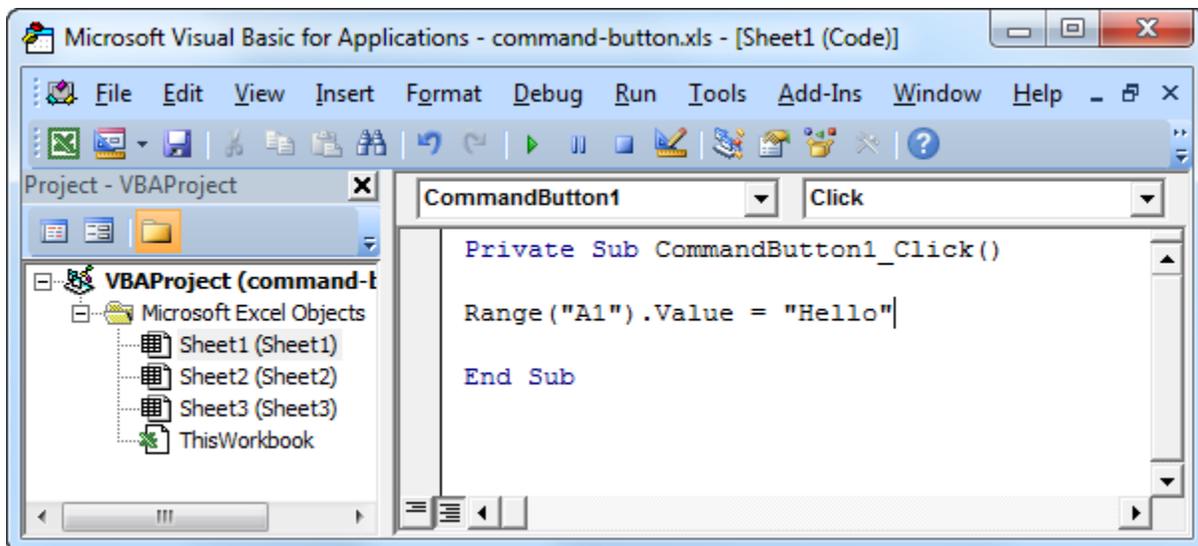
1. Right click on CommandButton1.
2. Click on View Code.



3. The Visual Basic Editor appears. Place your cursor between 'Private Sub CommandButton1_Click()' and 'End Sub'.

4. For example, add the following code line:

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



This macro places the word Hello into cell A1.

5. Close the Visual Basic Editor.

6. Before you click the command button on the sheet, make sure Design Mode is deselected. You can do this by clicking on Design Mode again.

Result when you click the command button on the sheet:

	A	B	C	D	E
1	Hello				
2					
3					
4					

CommandButton1

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

Did you find this information helpful? Show your appreciation, vote for us.

Go to Top: [Create a Macro](#) | Go to Next Topic: [Excel Macro Recorder](#)

[Learn more about excel macros, Login to the right >>](#)

100 easy to follow Excel VBA [examples](#). Limited time-offer: ~~\$39.95~~ but only \$29.00. Ends on 31st December.

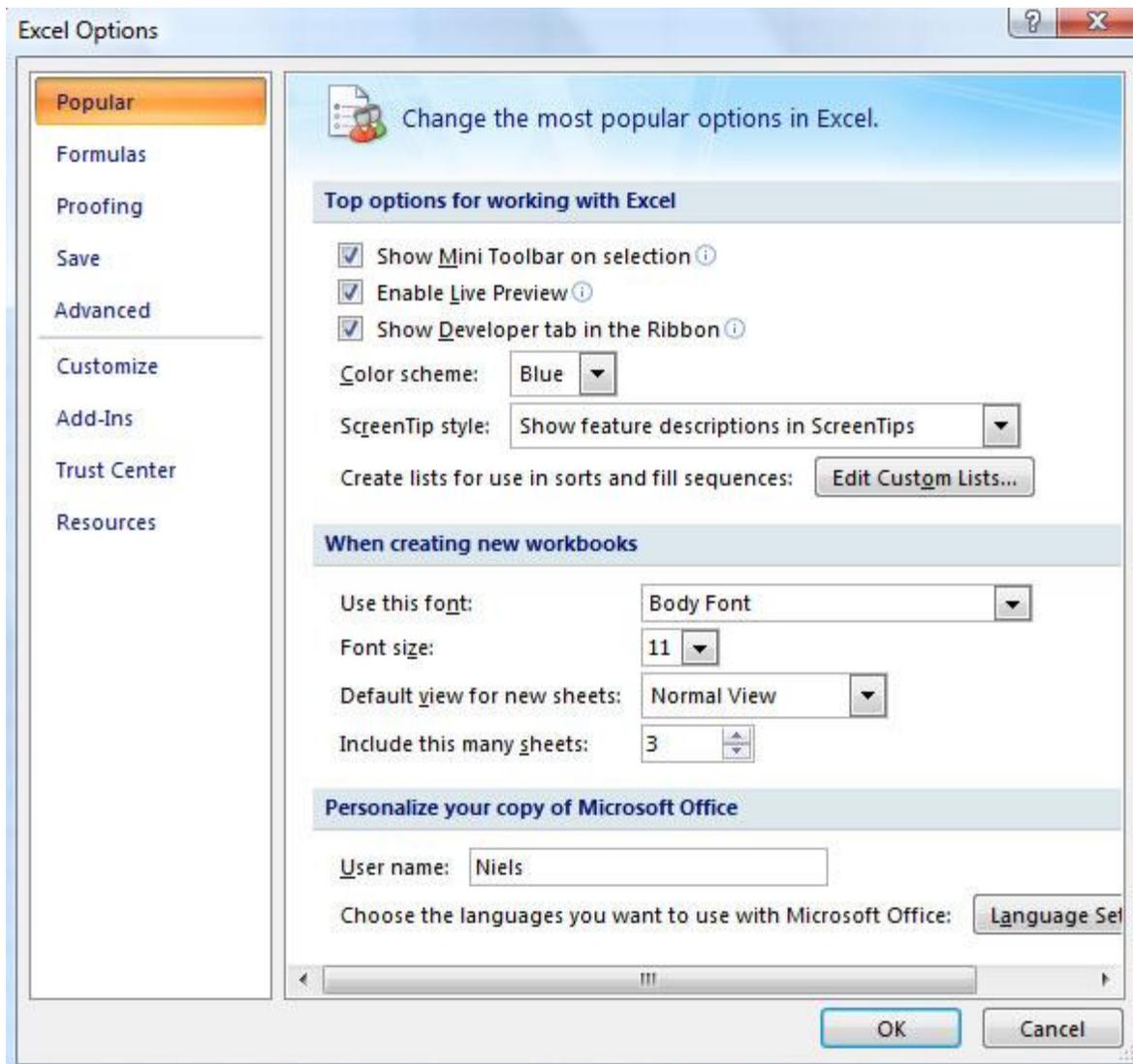
Create a Macro in Excel 2007

[Turn on the Developer Tab](#) | [Create a command button](#) | [Create and Assign the Macro](#)

To create a macro in Excel 2007, you have to turn on the Developer tab. Next, you can create a macro which will be executed after clicking on a command button.

Turn on the Developer Tab

1. Click on the Office button in the upper left corner of your screen.
2. Click on Excel Options. The Excel Options dialog box appears (see picture below).
3. Check Show Developer tab in the Ribbon.

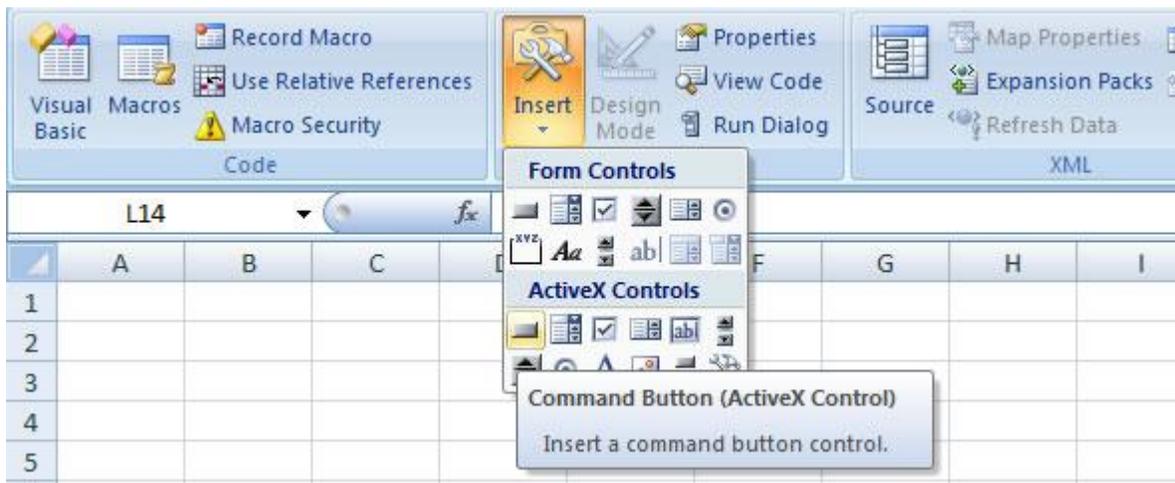


4. Click OK.

Create a command button

You can now click on the Developer tab which has been placed next to the View tab.

1. Click on Insert.
2. Click on Command Button in the ActiveX Controls section.

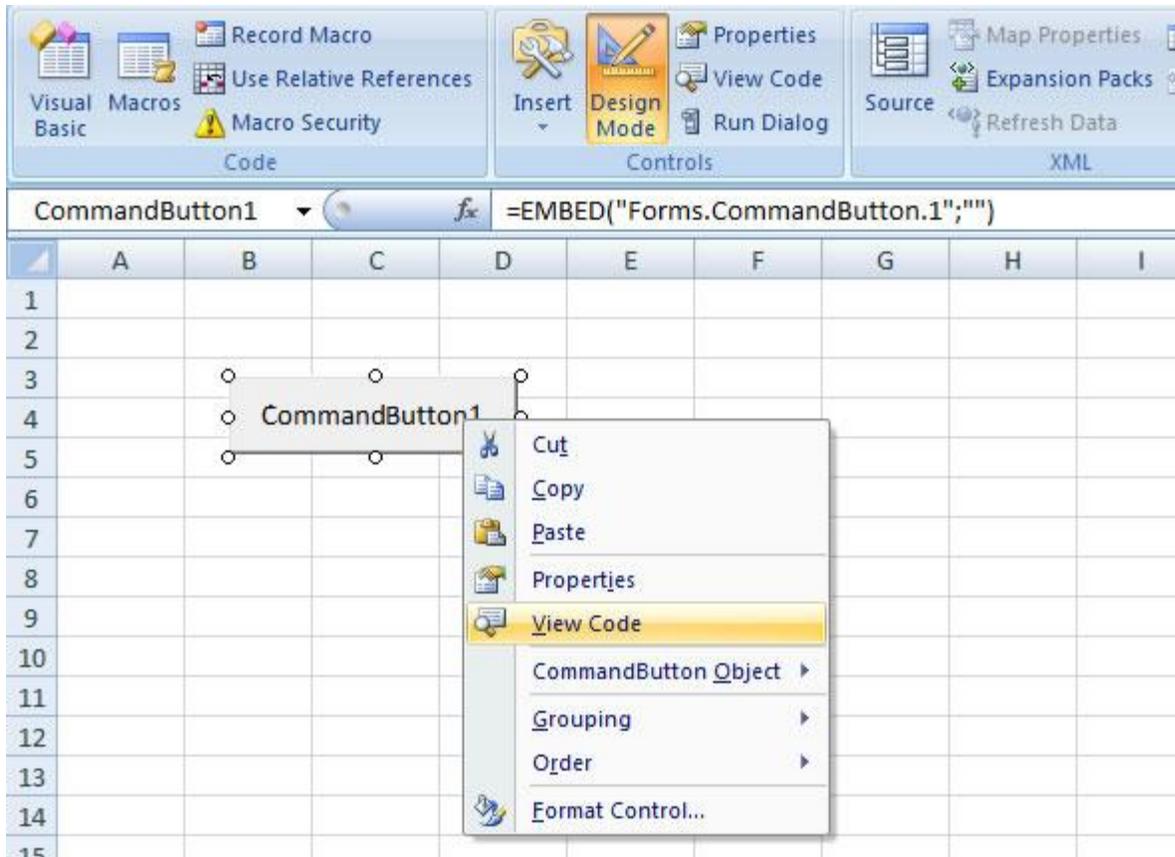


3. Now you can drag a command button on your worksheet.

Create and Assign the Macro

Now it is time to create a macro (a piece of code) and assign it to the command button.

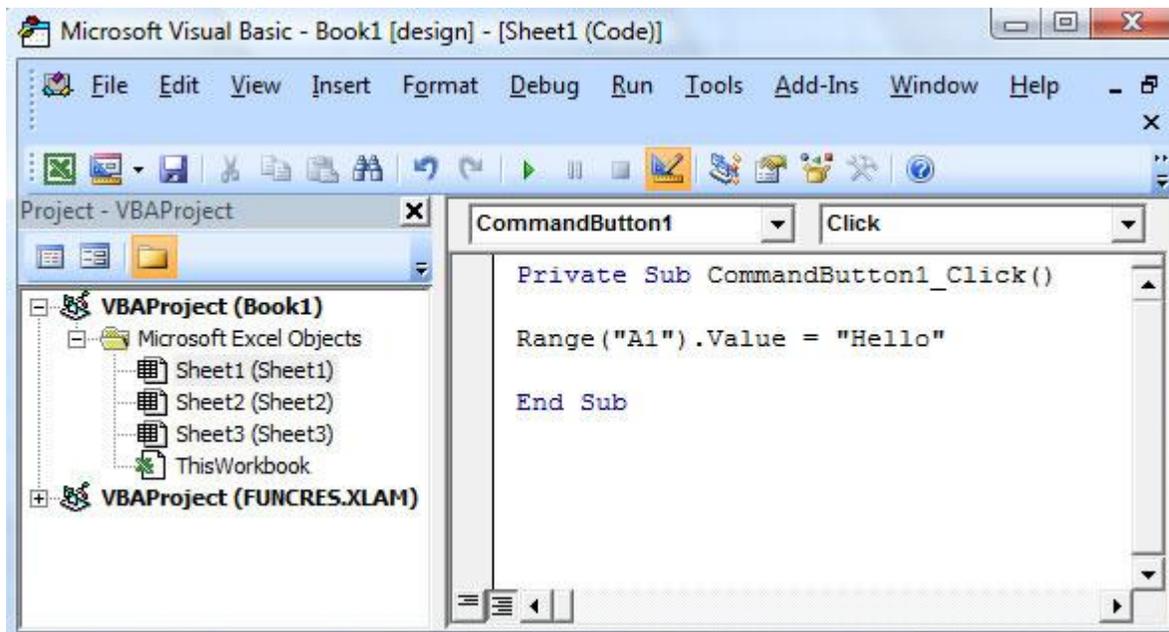
1. Right click on CommandButton1.
2. Click on View Code.



3. The Visual Basic Editor appears. Place your cursor between 'Private Sub CommandButton1_Click()' and 'End Sub'.

4. For example, add the following code line:

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



This macro places the word Hello into cell A1.

5. Close the Visual Basic Editor.

6. Before you click on CommandButton1, make sure Design Mode is deselected. You can do this by clicking on Design Mode again.

7. Click on CommandButton1.

Cell A1 should contain the word Hello now. Congratulations. You've just created a macro in Excel VBA!

Did you find this information helpful? Show your appreciation, vote for us.

Go to Top: [Create a Macro](#) | Go to Next Topic: [Excel Macro Recorder](#)

[Learn more about excel macros, Login to the right >>](#)

100 easy to follow Excel VBA [examples](#). Limited time-offer: ~~\$39.95~~ but only \$29.00. Ends on 31st December.

Create a Macro in Excel 2003

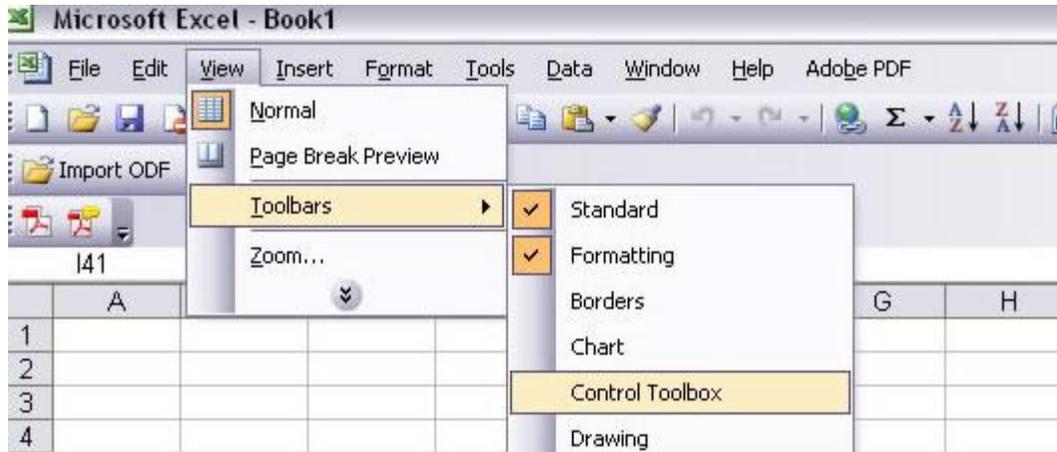
[Activate the Control Toolbox](#) | [Create a Command Button](#) | [Create and Assign the Macro](#)

To create a macro in Excel 2003, you have to activate the Control Toolbox. Next, you can create a macro which will be executed after clicking on a command button.

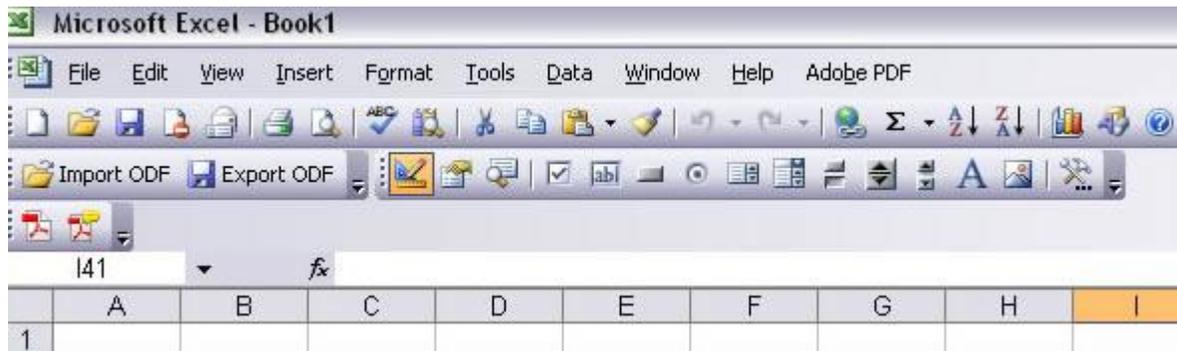
Activate the Control Toolbox

In order to use Excel VBA in Excel 2003, you have to activate the Control Toolbox.

1. Click on View, Toolbars, Control Toolbox.

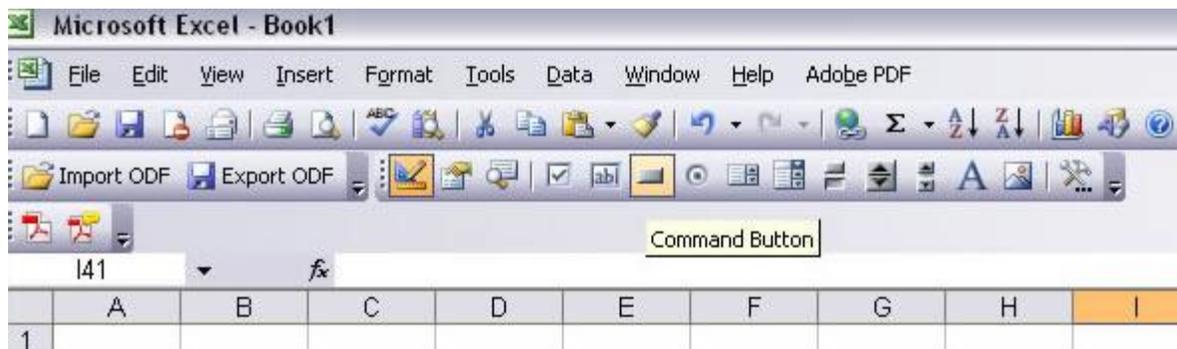


2. You can integrate the Control Toolbox if you want by simply dragging it into your menu.



Create a Command Button

1. Click on Command Button from the Control Toolbox.



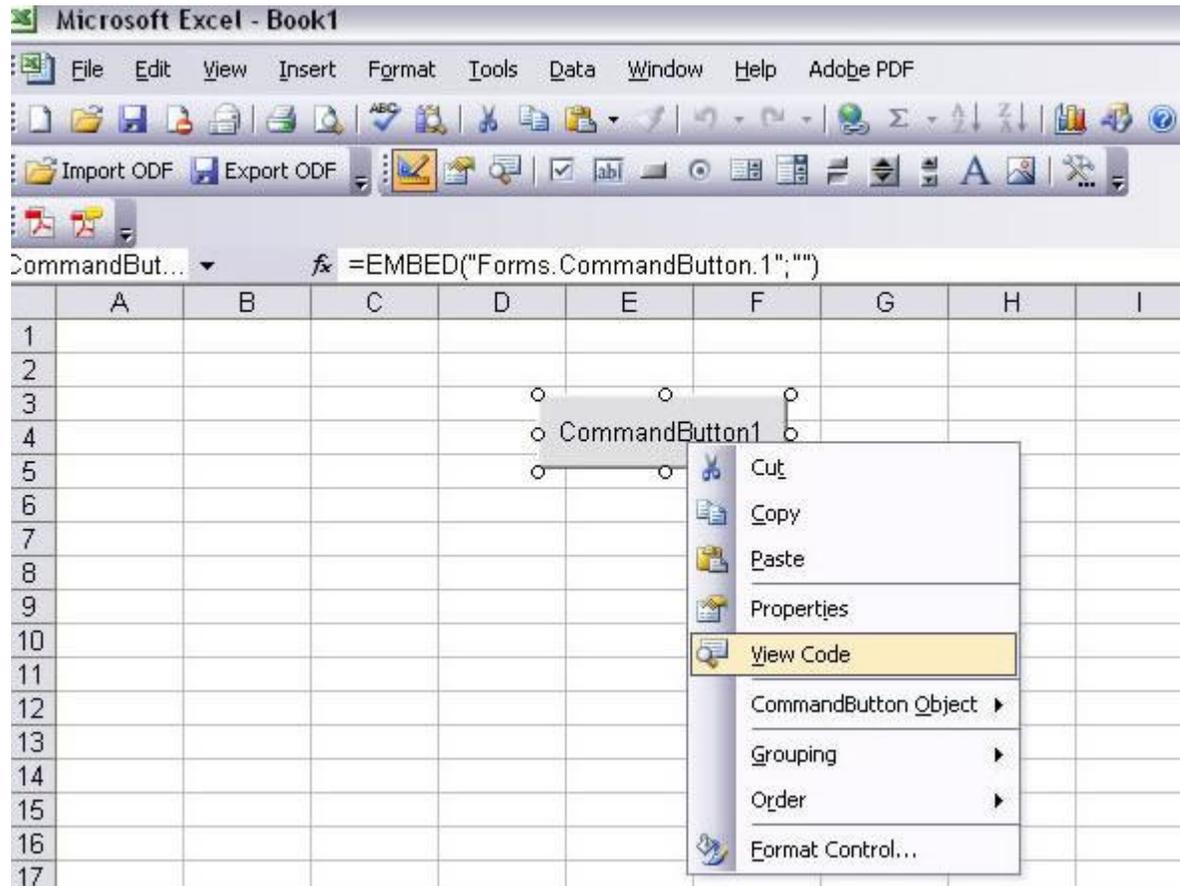
2. Now you can drag a command button on your worksheet.

Create and Assign the Macro

Now it is time to create a macro and assign it to the command button.

1. Right click on CommandButton1.

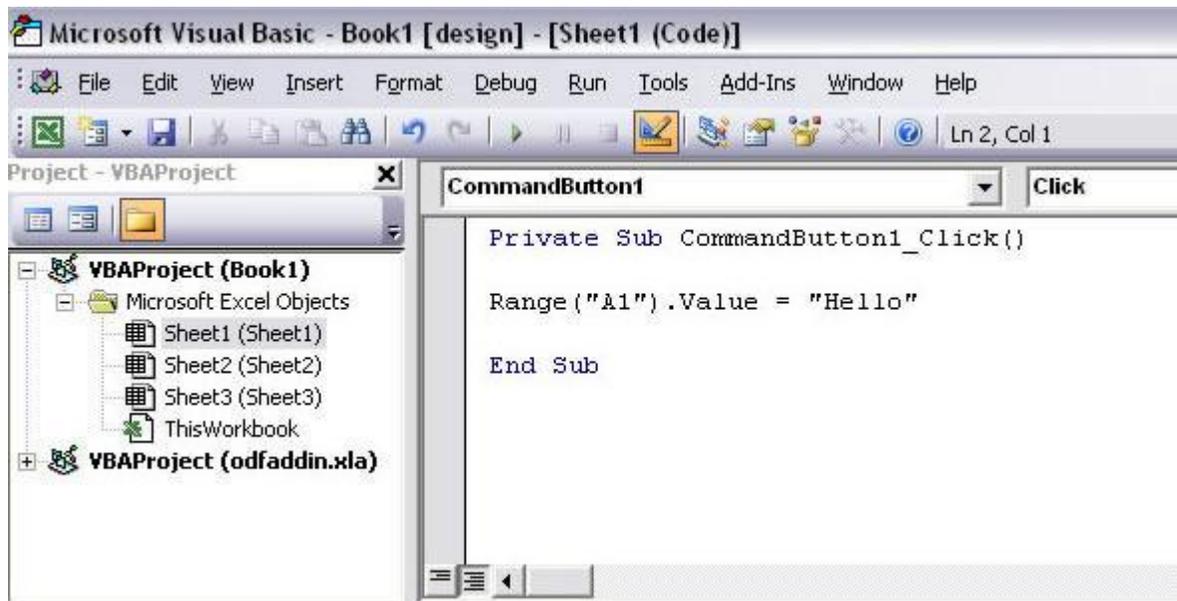
2. Click on View Code.



3. The Visual Basic Editor appears. Place your cursor between 'Private Sub CommandButton1_Click()' and 'End Sub'.

4. Add the line:

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

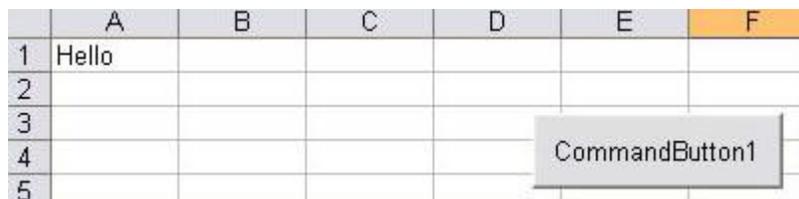


5. Close the Visual Basic Editor.

6. Before you click on the command button, make sure Design Mode (most left field of the Control Toolbox) is deselected. You can do this by clicking on Design Mode again.

7. Click on CommandButton1.

Result:



Well done! You've just created a macro in Excel VBA!

Excel Macro Recorder

Recording macros with the Excel Macro Recorder differs from one Excel version to another. Please choose your version.

[Excel 2010 / 2007](#) | [Excel 2003](#)

Excel Macro Recorder in Excel 2010 / 2007

[Why not use the Excel Macro Recorder for everything?](#) | [Record a Macro](#) | [Run a Recorded Macro](#) | [Edit the Macro](#)

The Macro Recorder, a very useful tool included in Excel VBA, records every task you perform with Excel. This is good news if you want to automate repetitive tasks. All you have to do is record a specific task once. Next, you can execute the task over and over with the click of a button. This can save you a lot of time! The Macro Recorder is also a great help when you don't know how to program a specific task in Excel VBA. Simply open the Visual Basic Editor after recording the task to see how it can be programmed.

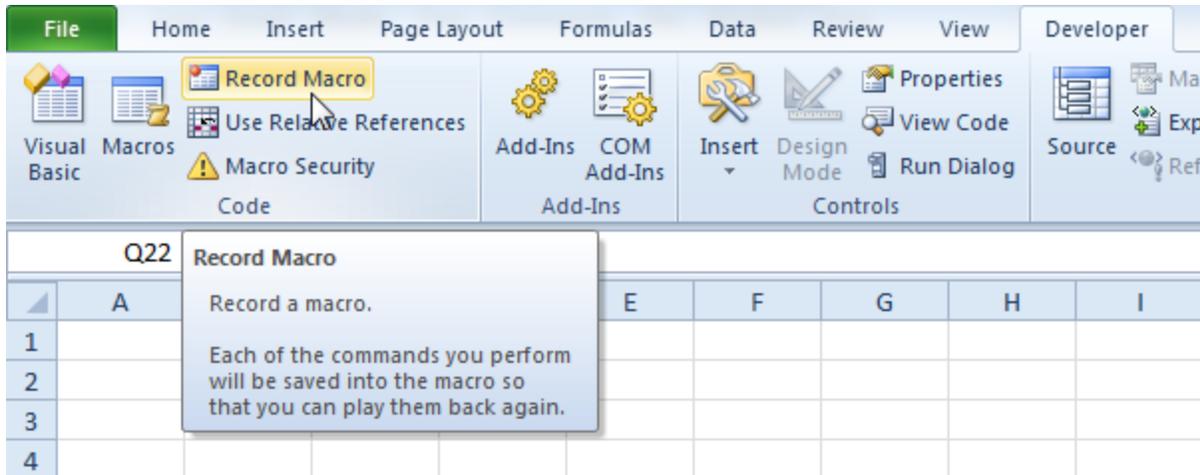
Why not use the Excel Macro Recorder for everything?

Unfortunately there are a lot of things you cannot do with the Excel Macro Recorder. For example, we cannot loop through a range of data with the Macro Recorder. Moreover the Macro Recorder uses a lot more code than is required, which can slow your process down.

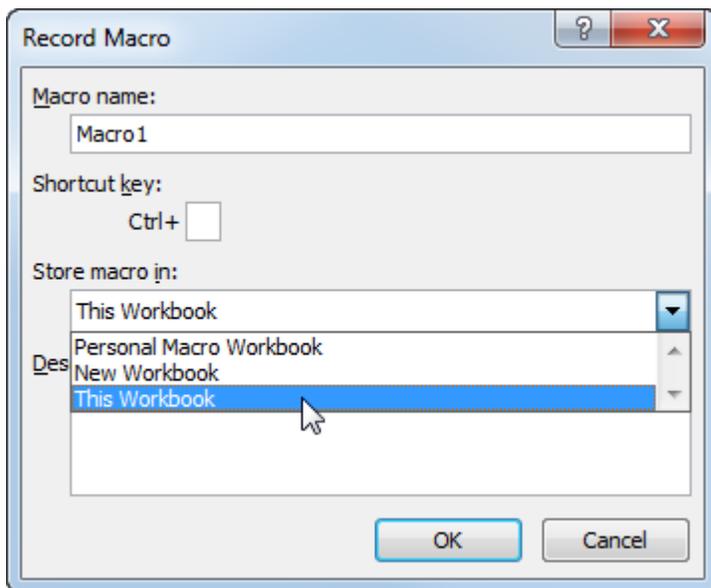
To record, run and edit a recorded macro, execute the following steps.

Record a Macro

1. Click on the Developer tab. Don't know where to find the Developer tab? Go to the [Create a Macro](#) chapter.
2. Click on Record Macro. See the picture below. Each of the commands you perform will be saved into the macro!



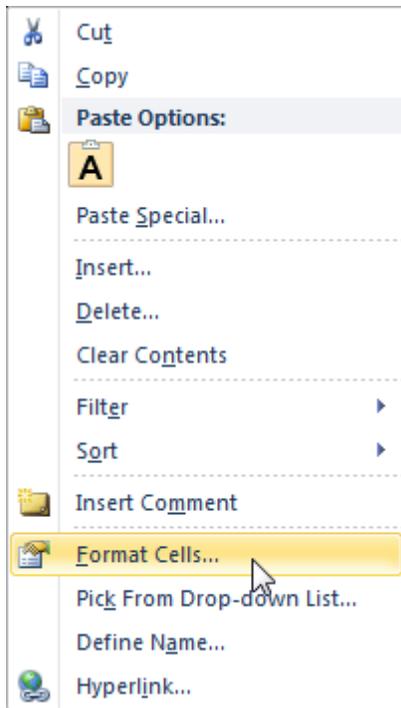
3. Here you can give your macro a name and you can enter a shortcut for your macro (both optional). You can store your macro in three workbooks. If you choose to store your macro in Personal Macro Workbook, the macro will be available to all your workbooks (Excel Files). This is because Excel stores your macro in a hidden workbook that opens automatically when Excel starts. If you choose to store your macro in New Workbook, the macro will only be available in an automatically new opened workbook. If you choose to store your macro in This Workbook, the macro will only be available in the current workbook.



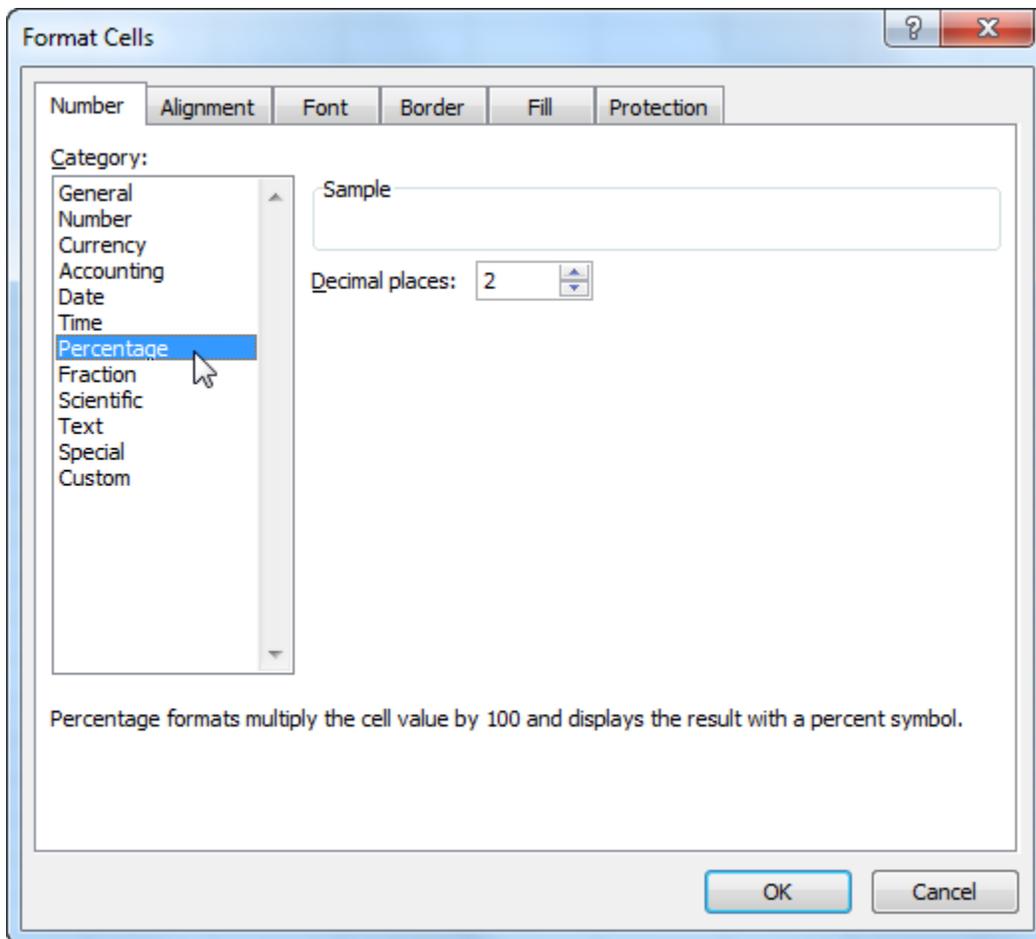
We will now record a macro that changes the format of Cells to Percentage.

4. Click on OK.

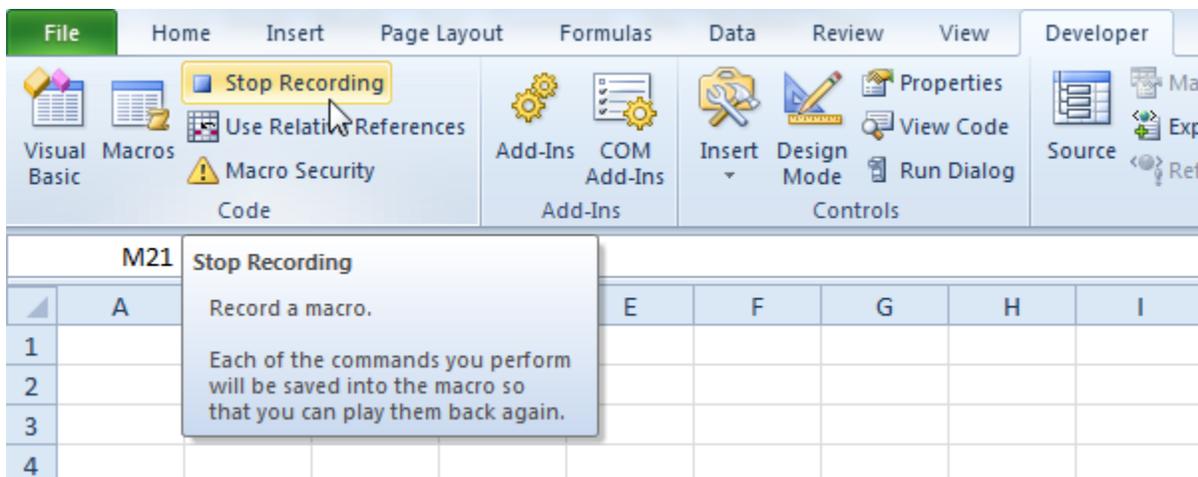
5. Right mouse click on the active cell (selected cell). Be sure not to select any other cell! Then click on Format Cells...



6. Choose Percentage and click on OK.



7. Finally, Click on Stop Recording.



Congratulations! You've just-recorded a macro with the Excel Macro Recorder!

Run a Recorded Macro

Now you can test the macro to see if it can change the format of cells to Percentage Format.

1. Enter some numbers between 0 and 1 in Excel. Select the numbers.

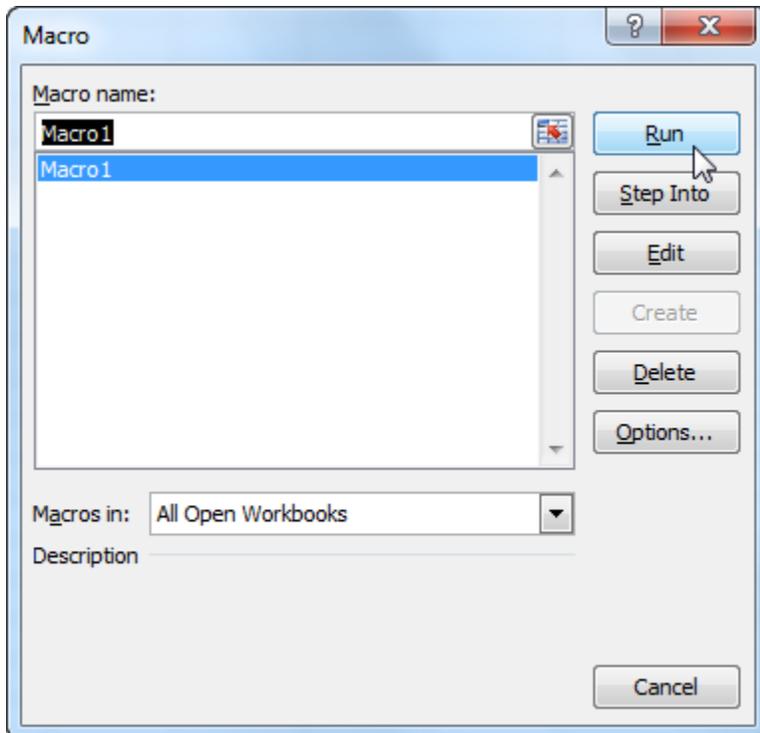
	A	B
1	percentages	
2	0.2	
3	0.6	
4	0.8	
5	0.6	
6	1	
7	0.4	
8		

2. Click on the Developer tab.

3. Click on Macros.

The screenshot shows the Microsoft Excel ribbon with the 'Developer' tab selected. The 'Macros' button is highlighted, and a tooltip is displayed over it. The tooltip text reads: 'View Macros (Alt+F8) View the list of macros, from which you can run, create, or delete a macro. Press F1 for more help.' The background shows the same spreadsheet as in the previous image.

3. Click on Run.

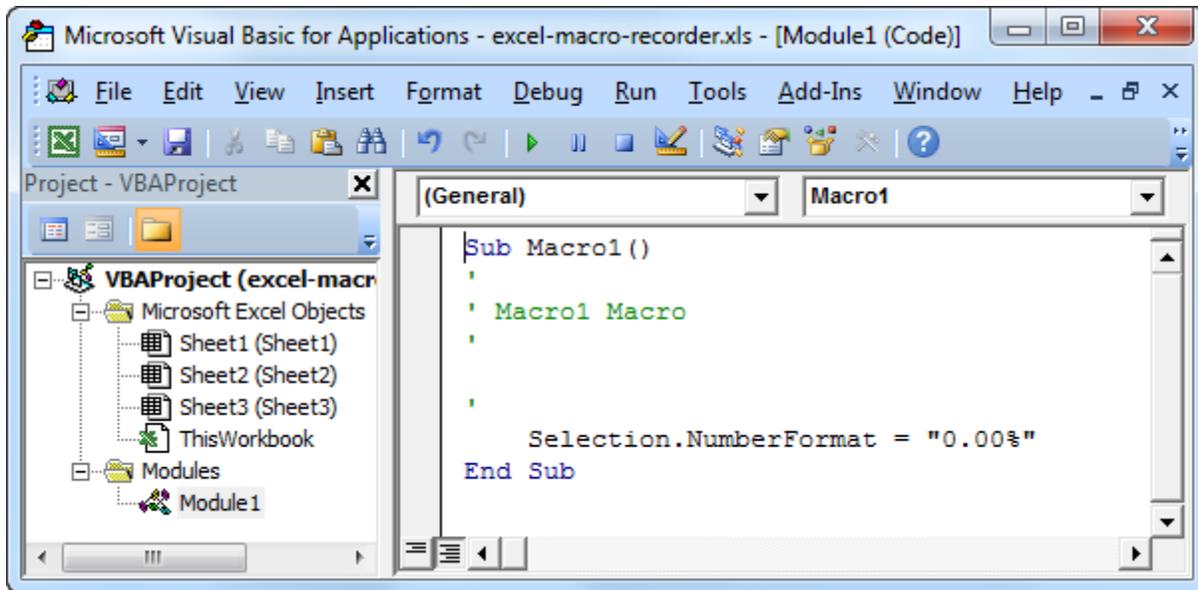


The format of the cells should change to Percentage Format. Result:

	A	B
1	percentages	
2	20.00%	
3	60.00%	
4	80.00%	
5	60.00%	
6	100.00%	
7	40.00%	
8		

Edit the Macro

There are two ways to take a look at the just-recorded macro. You can click on Macros from the Developer tab and then click on Edit. The Visual Basic Editor will appear. You can also directly open the Visual Basic Editor by clicking on Visual Basic from the Developer tab (or press Alt+F11).



Our macro has been placed into a module called Module1. Earlier we placed our code created without the Excel Macro Recorder directly on Sheet1. Code placed into a module is available to the whole workbook, while code placed on a sheet is only available for that particular sheet.

Did you find this information helpful? Show your appreciation, vote for us.

Go to Top: [Excel Macro Recorder](#) | Go to Next Section: [Basics](#)

[More about the excel macro recorder, Login to the right >>](#)

100 easy to follow Excel VBA [examples](#). Limited time-offer: ~~\$39.95~~ but only \$29.00. Ends on 31st December.

Excel Macro Recorder in Excel 2003

[Why not use the Excel Macro Recorder for everything?](#) | [Record a Macro](#) | [Run a Recorded Macro](#) | [Edit the Macro](#)

The Macro Recorder, a very useful tool included in Excel VBA, records every task you perform with Excel. This is good news if you want to automate repetitive tasks. All you have to do is record a specific task once. Next, you can execute the task over and over with the click of a button. This can save you a lot of time! The Macro Recorder is also a great help when you don't know how to program a specific task in Excel VBA. Simply open the Visual Basic Editor after recording the task to see how it can be programmed.

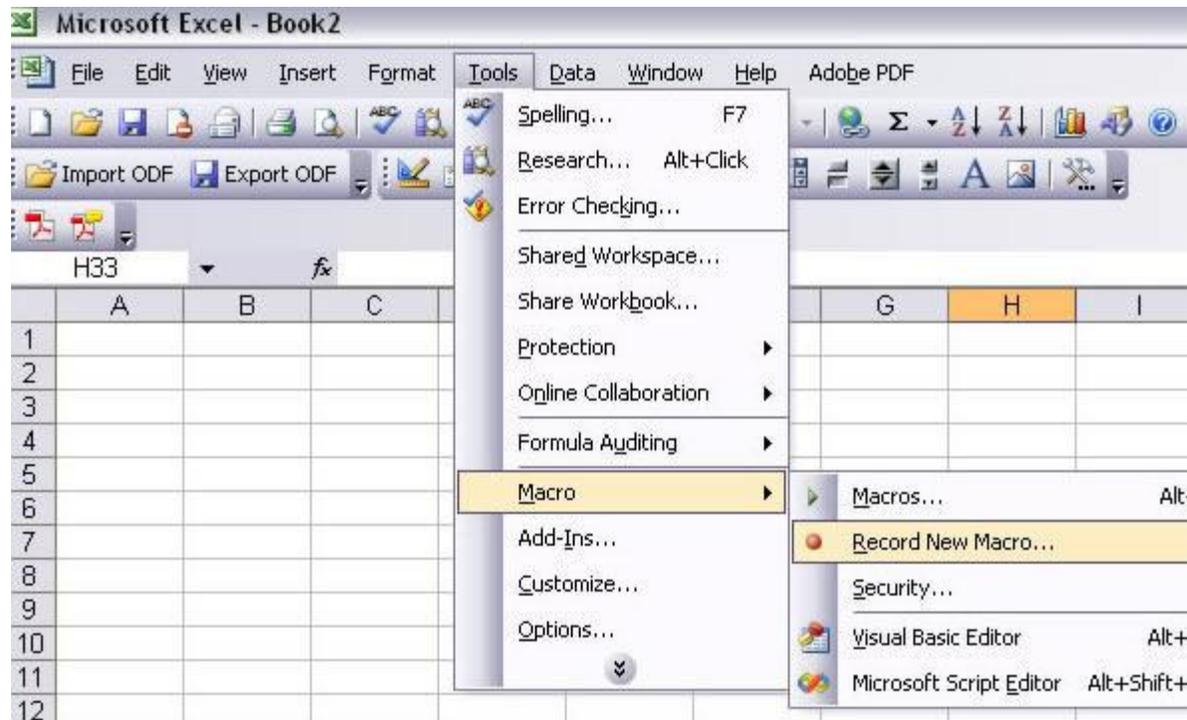
Why not use the Excel Macro Recorder for everything?

Unfortunately there are a lot of things you cannot do with the Excel Macro Recorder. For example, we cannot loop through a range of data with the Macro Recorder. Moreover the Macro Recorder uses a lot more code than is required, which can slow your process down.

The following example illustrates the use of the Excel Macro Recorder.

Record a Macro

1. Click on Tools, Macro, Record New Macro.



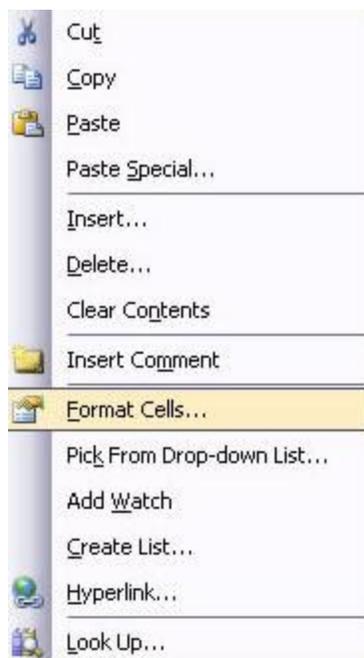
2. Here you can give your macro a name and you can enter a shortcut for your macro (both optional). You can store your macro in three workbooks. If you choose to store your macro in Personal Macro Workbook, the macro will be available to all your workbooks (Excel Files). This is because Excel stores your macro in a hidden workbook that opens automatically when Excel starts. If you choose to store your macro in New Workbook, the macro will only be available in an automatically new opened workbook. If you choose to store your macro in This Workbook, the macro will only be available in the current workbook.



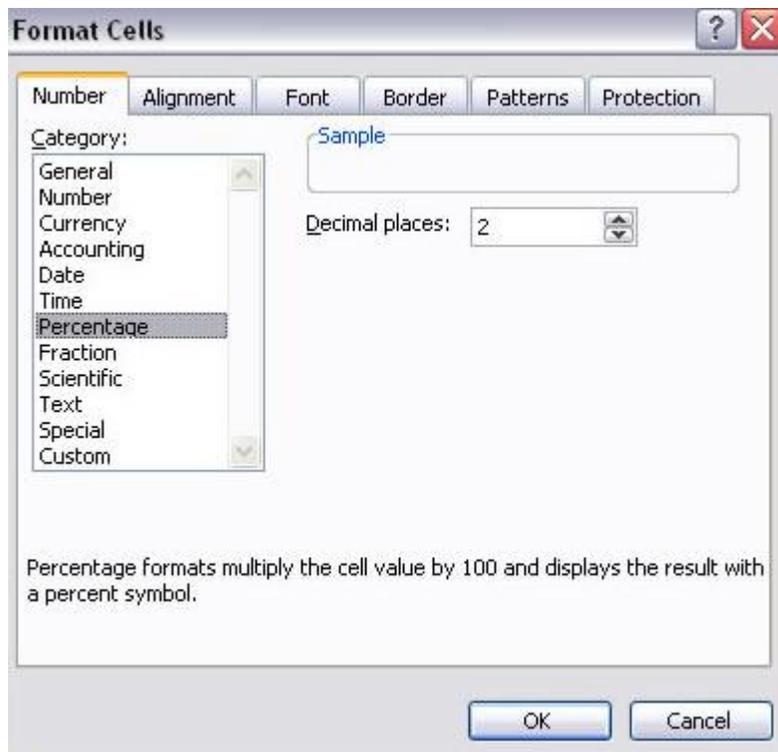
We will now record a macro that changes the format of Cells to Percentage.

3. Click on OK.

4. Right mouse click on the active cell (selected cell). Be sure not to select any other cell! Then click on Format Cells...



5. Choose Percentage and click on OK.



6. Finally, Click on Stop Recording.



Well done! You have just-recorded a macro with the Excel Macro Recorder!

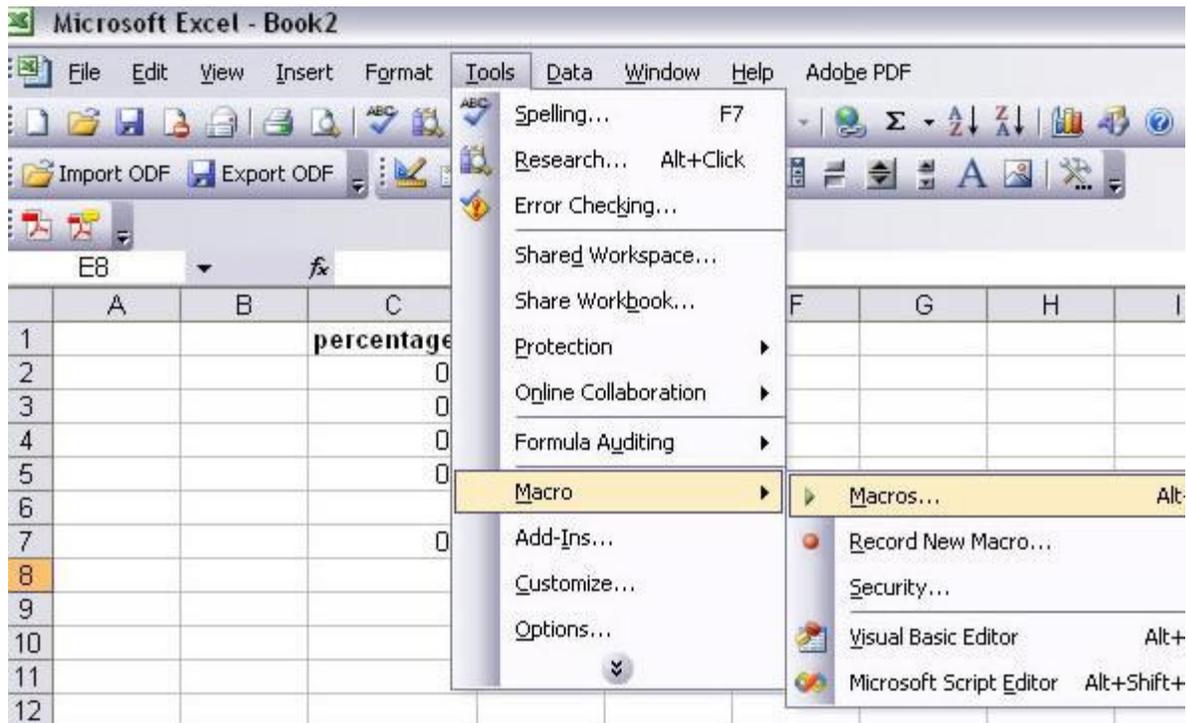
Note: if you accidentally close the Stop Recording box, go to View, Toolbars, Stop Recording to turn it back on while recording a macro.

Run a Recorded Macro

1. Enter some numbers between 0 and 1 in Excel. Select the numbers.

C
percentages
0.2
0.6
0.8
0.6
1
0.4

2. Click on the Tools, Macro, Macros.



3. Click on Run.



The format of the cells should change to Percentage Format. Result:

C
percentages
20.00%
60.00%
80.00%
60.00%
100.00%
40.00%

Edit the Macro

There are two ways to take a look at the just-recorded macro. You can click on Macros from the Developer tab and then click on Edit. The Visual Basic Editor will appear. You can also directly open the Visual Basic Editor by clicking on View Code from the Control Toolbox (or press Alt+F11). If you don't know where to find the Control Toolbox, go to the [Create a Macro](#) chapter.

Macro Security

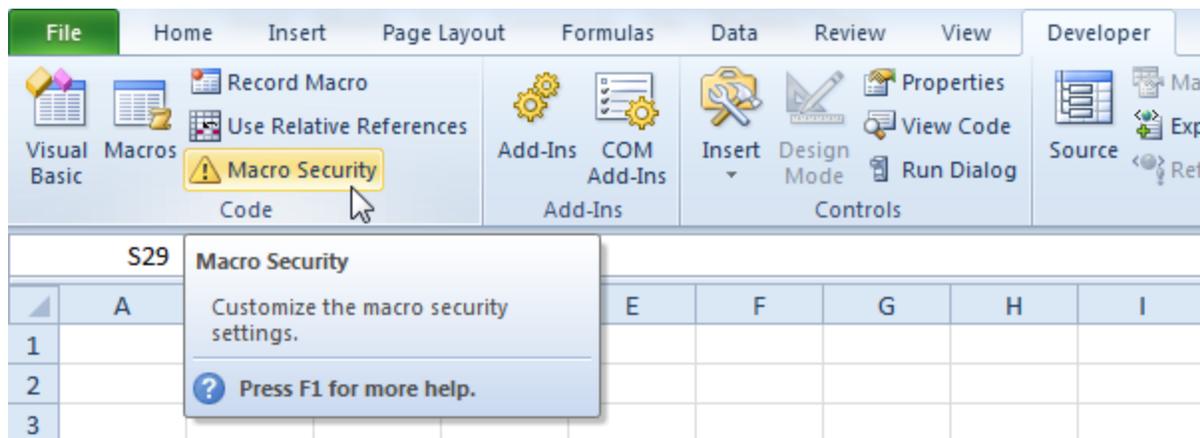
Setting up your macro security settings differs from one Excel version to another. Please choose your version.

[Excel 2010 / 2007](#) | [Excel 2003](#)

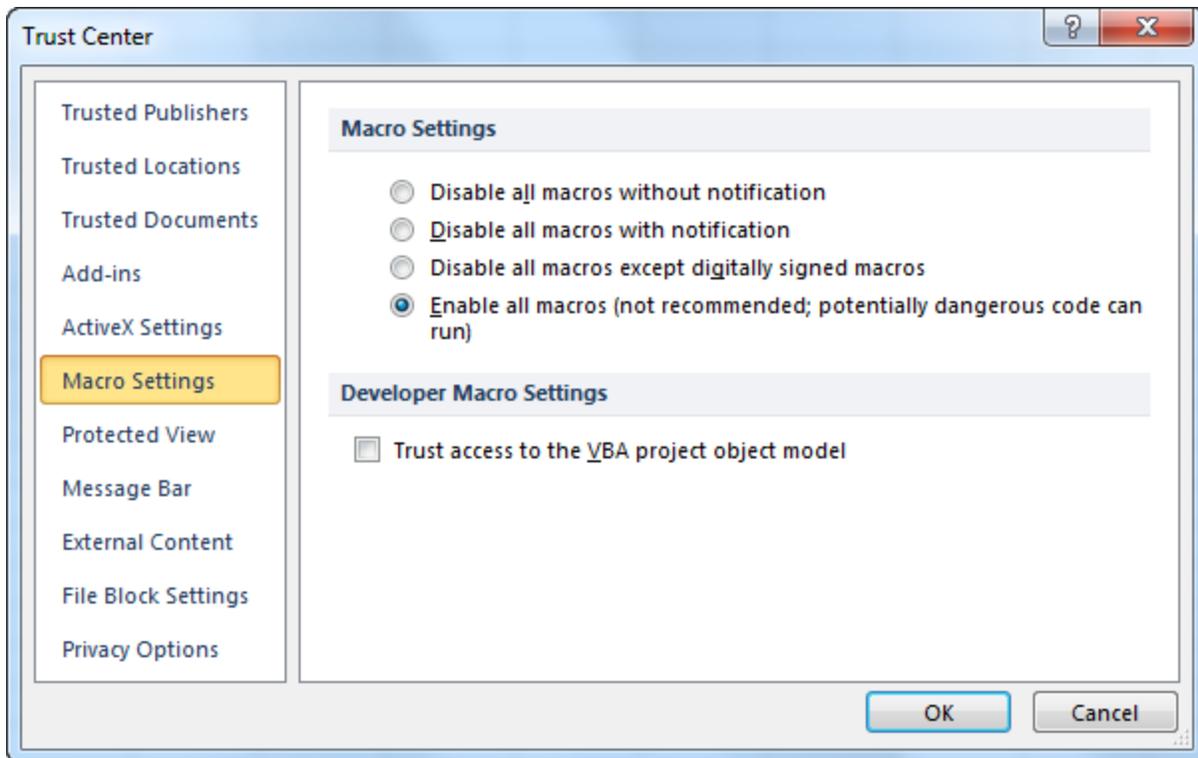
Macro Security in Excel 2010 / 2007

To set up the macro security settings in Excel 2010 or Excel 2007, execute the following steps.

1. Click on Macro Security.



2. Here you have four options. The first option will disable all macros. The second option will always ask you to enable a macro. The third option will only allow macros with a digital signature to run, and ask you to enable others. The fourth option will enable all macros.



Our advice is to use the second security level if you are downloading a lot of Excel files from the internet. With this security level you can always disable the macro if you don't trust the owner of the Excel file. Use the fourth security level if you are a beginner and only typing your own macros at the moment. With this security level you don't have to enable macros all the time.

Did you find this information helpful? Show your appreciation, vote for us.

Go to Top: [Macro Security](#) | Go to Next Topic: [Visual Basic Editor](#)

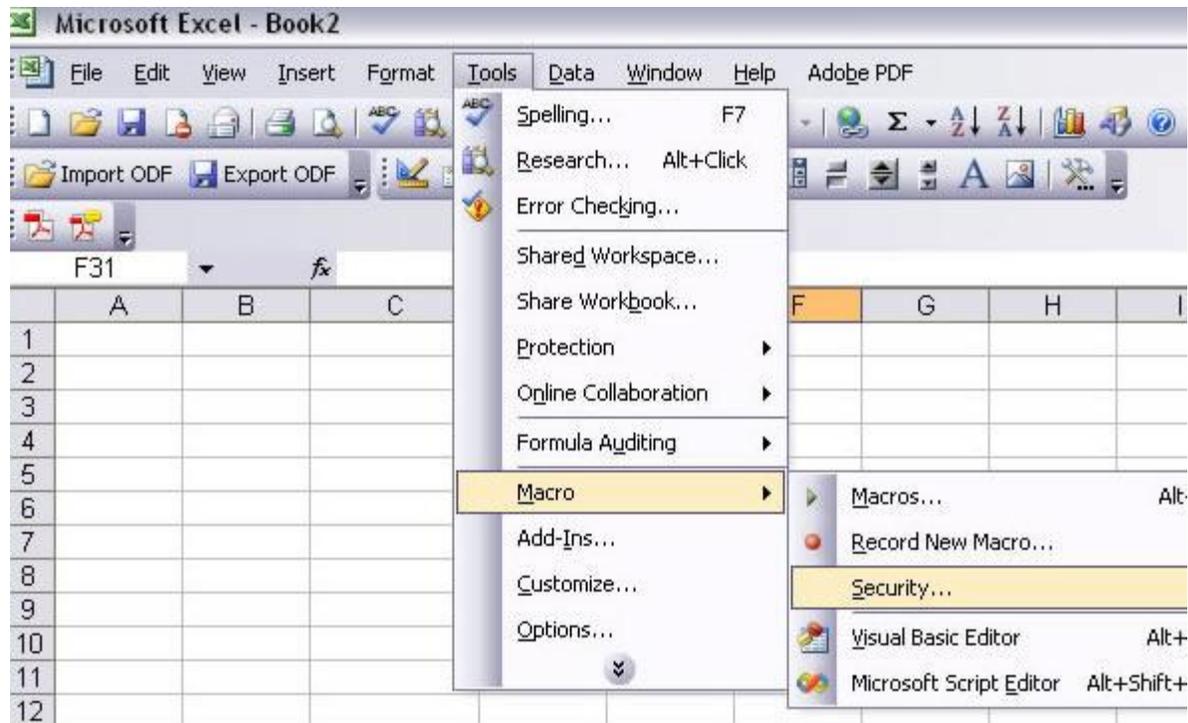
[More about password protection, Login to the right >>](#)

100 easy to follow Excel VBA [examples](#). Limited time-offer: ~~\$39.95~~ but only \$29.00. Ends on 31st December.

Macro Security in Excel 2003

Before you start, make sure that your macro security settings are set up correctly. It is not likely that nasty things will happen to your system, but there is a chance that a macro contains a virus.

1. Click on Tools, Macro, Security.



2. Here you have four options. Very High will disable all macros, unless you've stored your macro in a specific trusted folder. High will only allow macros which are signed by an acknowledged trusted source. Medium will allow you to decide whether or not to run a macro. Low will enable all macros.



Our advice is to use the medium level if you are downloading a lot of Excel files from the internet. This security level allows you to disable the macro if you don't trust the owner of the Excel file. You can use the low level if you're only creating your own macros at the moment. With this security level you don't have to enable macros all the time.

Basics

This section explains the basics of Excel Visual Basic. It is good to know the basic terminology explained in this section before you start programming in Excel Visual Basic.

1 [Macro Security](#): Setting up your macro security settings correctly is essential to protect yourself against potential viruses. Make sure your macro security settings are set up correctly so no harm can be done to your computer.

2 [Visual Basic Editor](#): Learn how to launch the Visual Basic Editor and get the best configuration of the Project Explorer and the Code Window in your Excel Version. The Visual Basic Editor is the starting point for creating macros in Excel VBA, so it is important to get this configuration right.

3 [Macro Comments](#): Add macro comments to your Excel VBA code and your code will be easier to read as program size increases.

4 [MsgBox](#): The Message Box is a dialog box you can have appear to inform the users of your program.

5 [Macro Errors](#): Dealing with VBA-errors can be quite a challenge. This chapter provides you with a simple tip to deal with these errors.

6 [Debug Macros](#): Before you execute your VBA-code you can first debug your macro. This way most of the errors can be corrected before you execute your code.

7 [Objects, Properties and Methods](#): In this chapter you will learn more about Excel VBA objects. An object has properties and methods. Excel Visual Basic is a semi-object oriented programming language. Learn more about the object hierarchy of Excel Visual Basic.

8 [Workbook and Worksheet](#): In this chapter you will learn more about the Excel VBA Workbook and Excel VBA Worksheet object. You will see that the Worksheet and Workbook object have properties and methods as well, such as the count property which counts the number of active workbooks or worksheets. The Workbook and Worksheet object are commonly used in Excel VBA. They are very useful when your macro code has to be executed on different workbooks or worksheets.

9 [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.

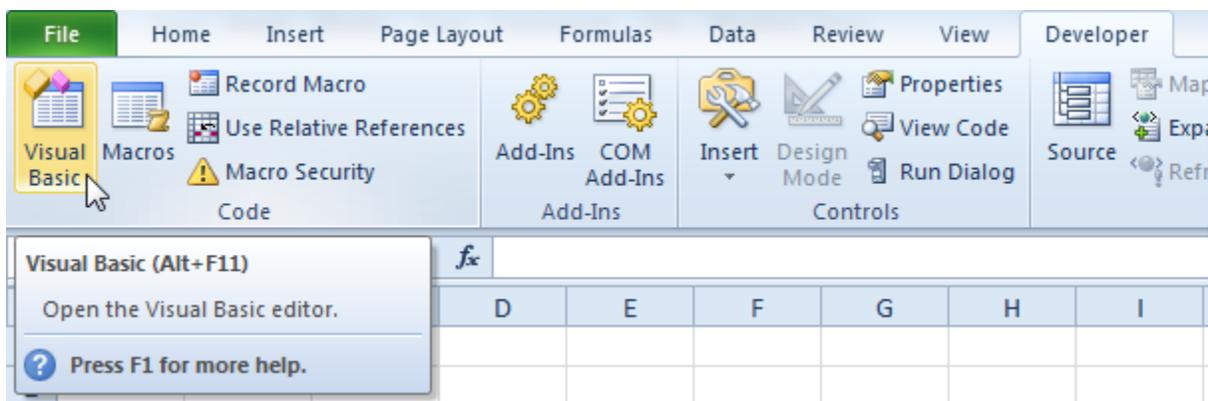
Visual Basic Editor

Launching the Visual Basic Editor differs from one Excel version to another. Please choose your version.

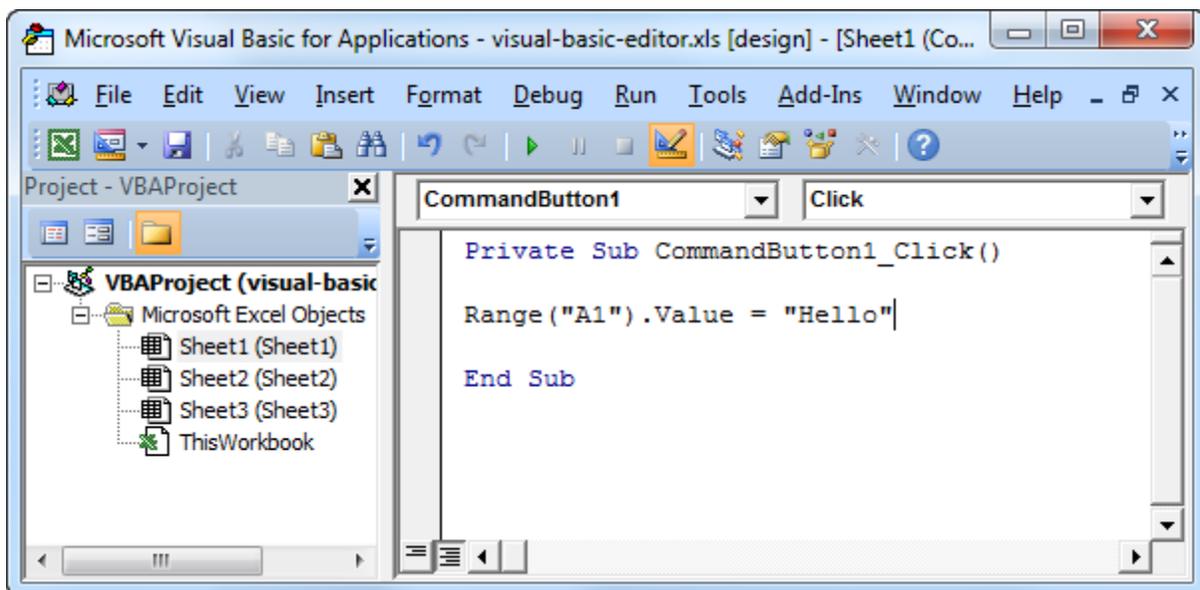
[Excel 2010 / 2007](#) | [Excel 2003](#)

Visual Basic Editor in Excel 2010 / 2007

To launch the Visual Basic Editor in Excel 2010 or Excel 2007, click on Visual Basic (or press Alt+F11).



The Visual Basic Editor appears:



The left window with the sheet names in it is called the Project Explorer. If you can't see the Project Explorer, click on View and then Project Explorer. Most probably the Project Explorer will already appear as a column on the left side of the screen. If not, execute the following three steps to achieve this.

1. Right click on the Project Explorer.
2. Check Dockable (if necessary).
3. Click on Project - VBAProject and drag the Project Explorer to the left side of the screen.

The Code window can be added by clicking on one of the sheet names. To cover the whole screen, you can maximize the Code Window. We think this is the best configuration of the Visual Basic Editor.

Did you find this information helpful? Show your appreciation, vote for us.

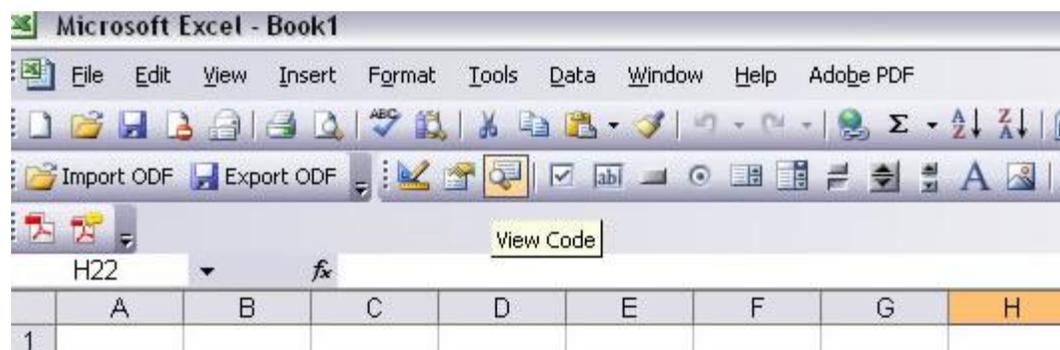
Go to Top: [Visual Basic Editor](#) | Go to Next Topic: [Macro Comments](#)

[More about the visual basic editor, Login to the right >>](#)

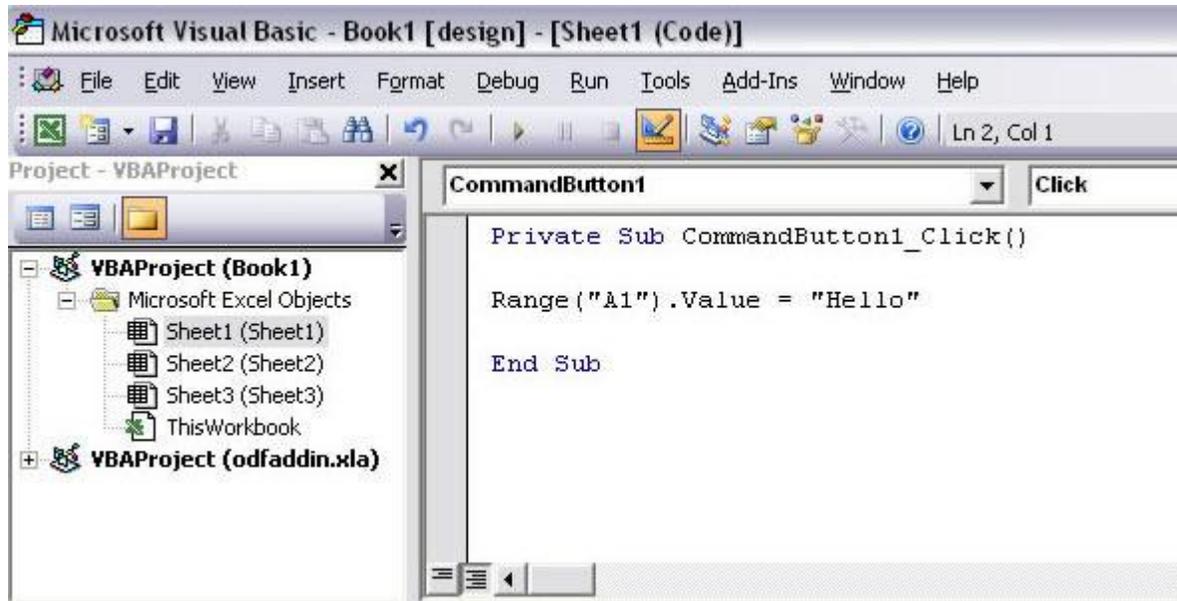
100 easy to follow Excel VBA [examples](#). Limited time-offer: ~~\$39.95~~ but only \$29.00. Ends on 31st December.

Visual Basic Editor in Excel 2003

To launch the Visual Basic Editor in Excel 2003, click on View Code from the Control Toolbox (or press Alt+F11).



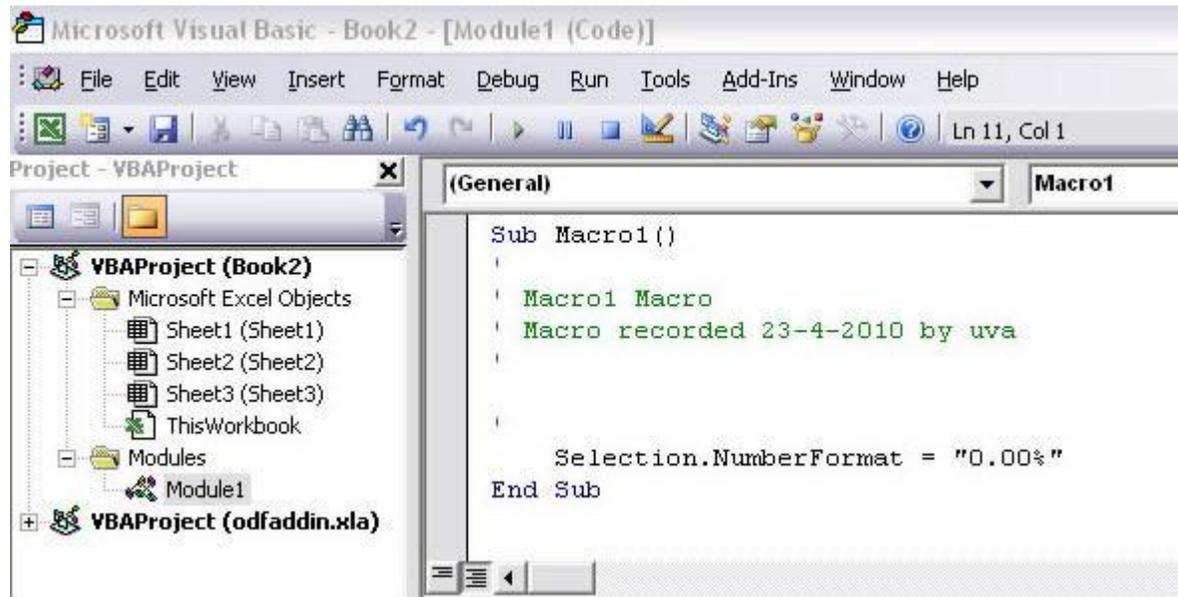
The Visual Basic Editor appears:



The window on the left with the names Sheet1, Sheet2 and Sheet3 in it is called the Project Explorer. If you can't see the Project Explorer, you can click on View and then Project Explorer. Most probably the Project Explorer will already appear as a column on the left side of the screen. If not, execute the following three steps to achieve this.

1. Right click on the white part of the Project Explorer.
2. Check Dockable (if not already done so).
3. Click on Project - VBAProject and drag it to the left until your cursor (white arrow) touches the middle of the left side of the screen.

To add the Code window, simply click on one of the sheets. Maximize the Code window so it will cover the whole screen. Once this has been completed, the configuration should be consistent with the picture above. We think this is the best configuration of the Visual Basic Editor.

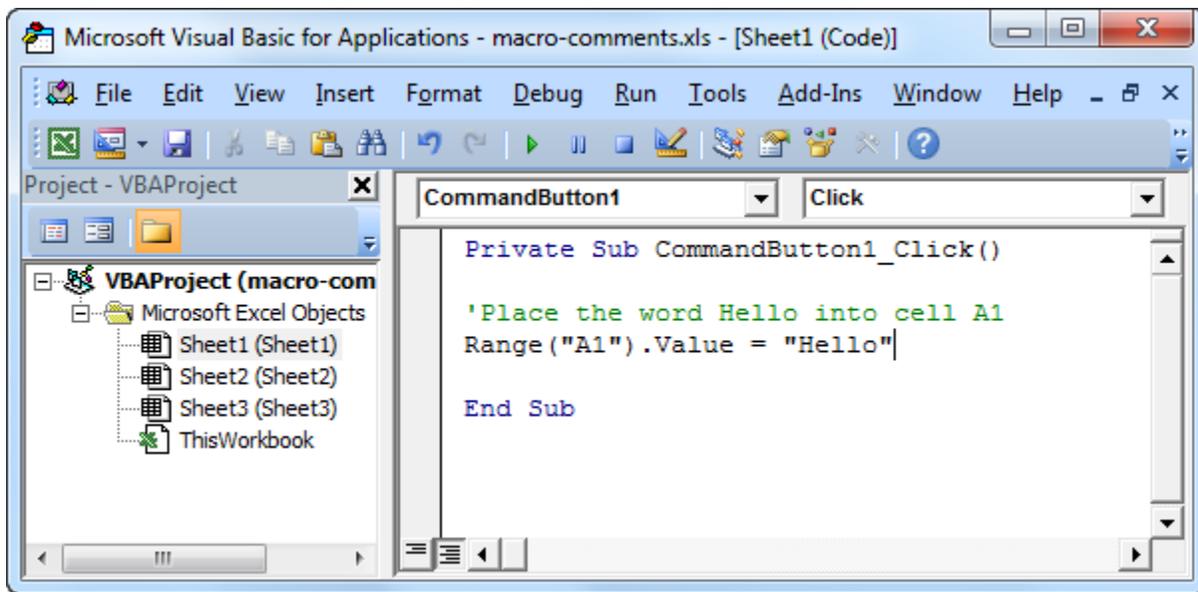


In the Project Explorer we see that a new map with the name Modules has been created. Click on Module1 to see the created macro. Code recorded with the Macro Recorder will always be placed into a module (here Module1). Earlier we placed our code created without the Excel Macro Recorder directly on Sheet1. Code placed into a module is available to the whole workbook, while code placed on a sheet is only available for that particular sheet.

Macro Comments

A macro comment is a piece of text in a macro which will not be executed by Excel VBA. It is only there to provide you information about the macro. To let Excel VBA know that it is a comment, place an apostrophe at the start of the text. Execute the following steps to place a comment.

1. Launch the Visual Basic Editor.
2. Insert the line: 'Place the word Hello into cell A1' before the code line. After the line is inserted, Excel VBA colors the line green to indicate that it is a comment.



It is good practice to use comments. Macro comments become more useful as as program size increases.

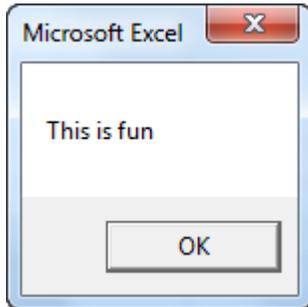
MsgBox

The Excel VBA MsgBox (message box) is a dialog box you can use in Excel VBA to display information to the users of your program. Below you can find three easy examples on how to create a MsgBox in Excel VBA.

1. To show a MsgBox with the message "This is fun", place a [command button](#) on your worksheet and simply add the following code line:

```
MsgBox "This is fun"
```

Result when you click the command button on the sheet:



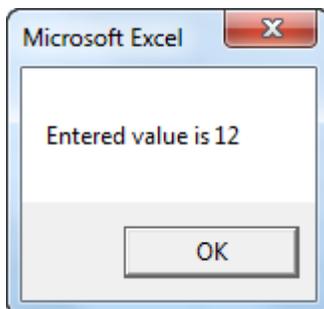
2. Now we try to create a little more advanced MsgBox.

Enter a random number into cell A1.

Instead of "This is Fun", add the following code line:

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

Result when you click the command button on the sheet:

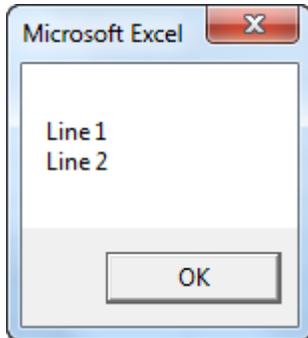


We used the & operator to concatenate (join) two strings. Although Range("A1").value is not a string, it works here.

3. To start a new line in a message box, you can use vbNewLine. Add the following code line:

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

Result when you click the command button on the sheet:



Macro Errors

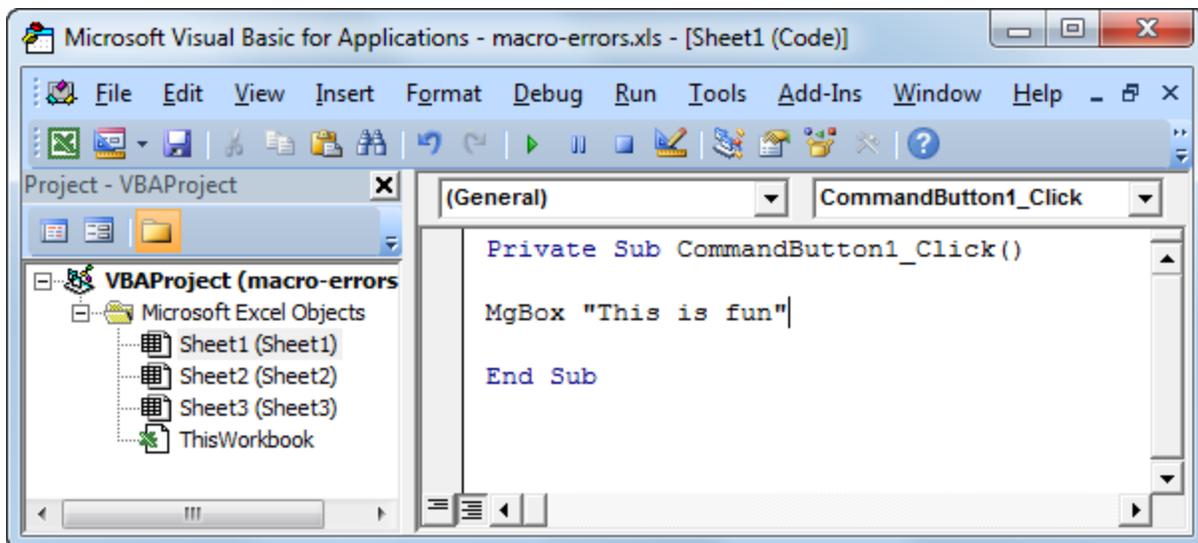
Unfortunately, not everything goes right the first time. Excel VBA will sometimes give you a macro error saying that something isn't programmed correctly. There are too many Excel Visual Basic errors to explain them all at this stage. However, here is a good tip to deal with errors.

First, we want to create a macro error.

1. Place a [command button](#) on your worksheet and add the following incorrect code line:

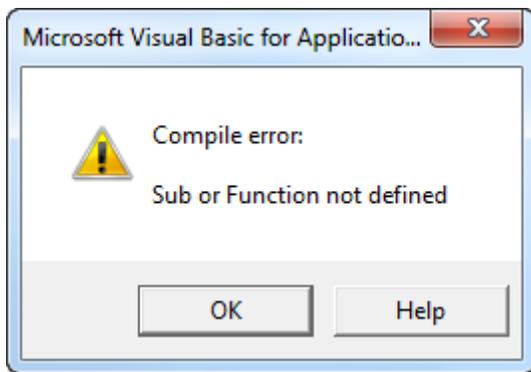
```
MgBox "This is fun"
```

(instead of the correct line `MsgBox "This is fun"`)



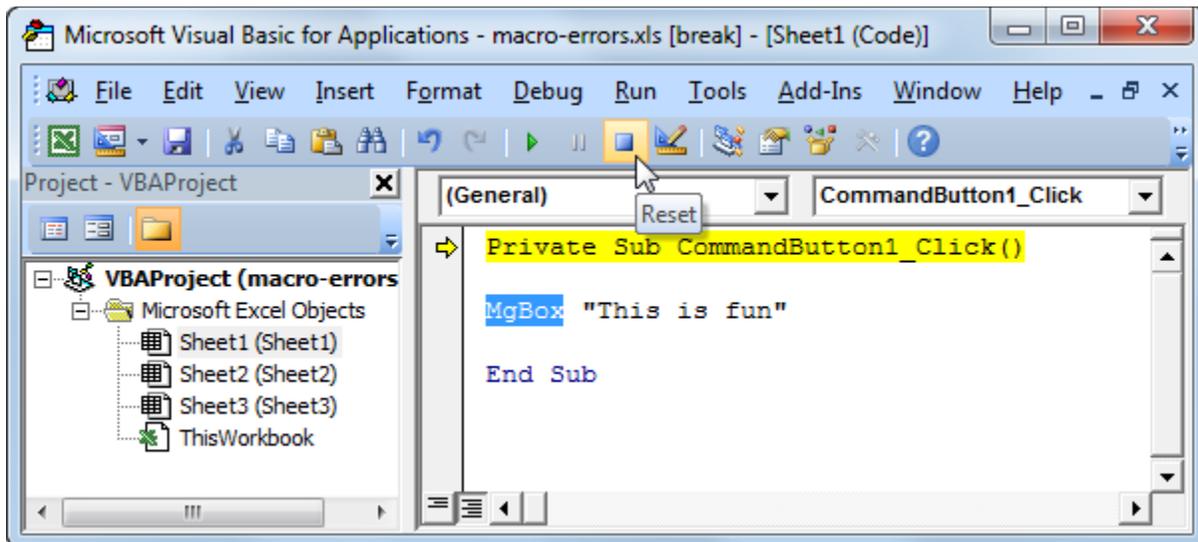
2. Click the command button on the sheet.

Result:



3. Click on OK.

4. Next, in the Visual Basic Editor, click on Reset to stop the debugger! (More about the debugger in the next chapter) Now change the code. Excel VBA has colored the word MsgBox blue to indicate the macro error.



Macro Errors

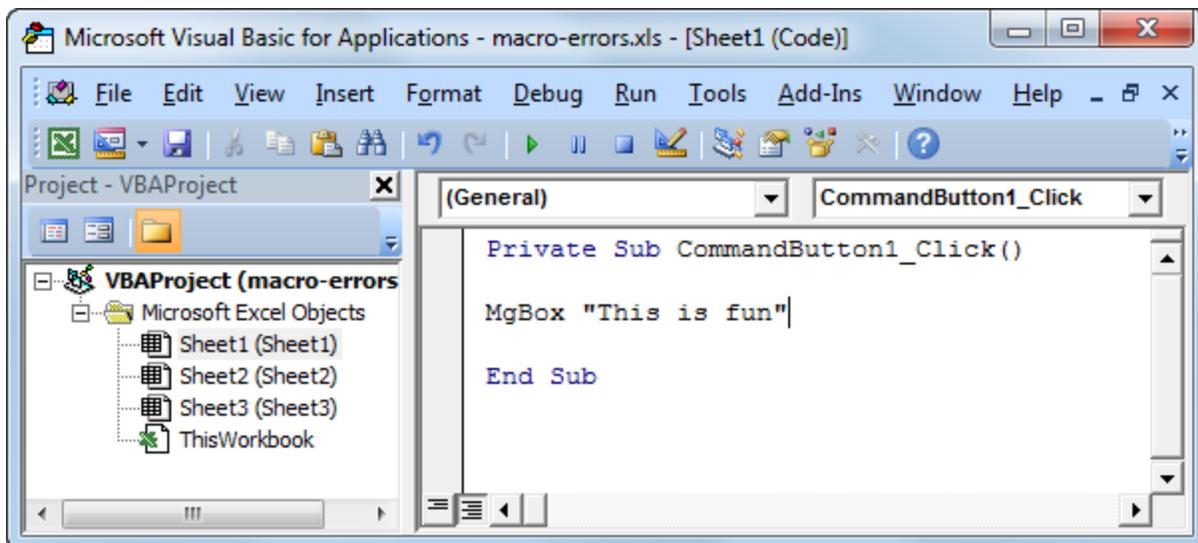
Unfortunately, not everything goes right the first time. Excel VBA will sometimes give you a macro error saying that something isn't programmed correctly. There are too many Excel Visual Basic errors to explain them all at this stage. However, here is a good tip to deal with errors.

First, we want to create a macro error.

1. Place a [command button](#) on your worksheet and add the following incorrect code line:

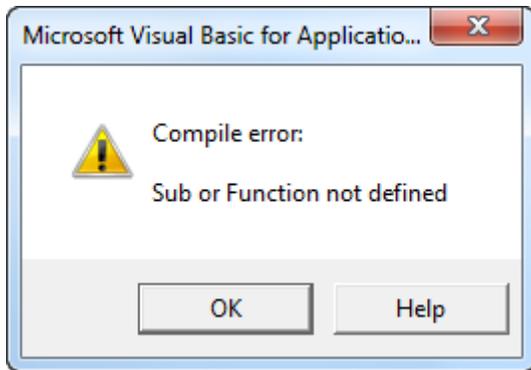
```
MgBox "This is fun"
```

(instead of the correct line MsgBox "This is fun")



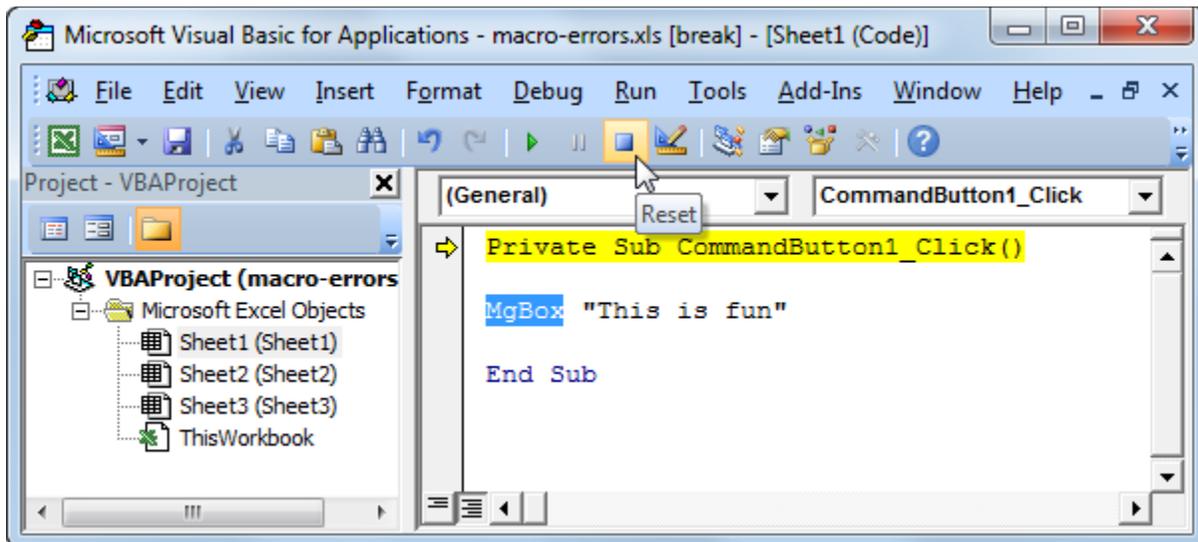
2. Click the command button on the sheet.

Result:



3. Click on OK.

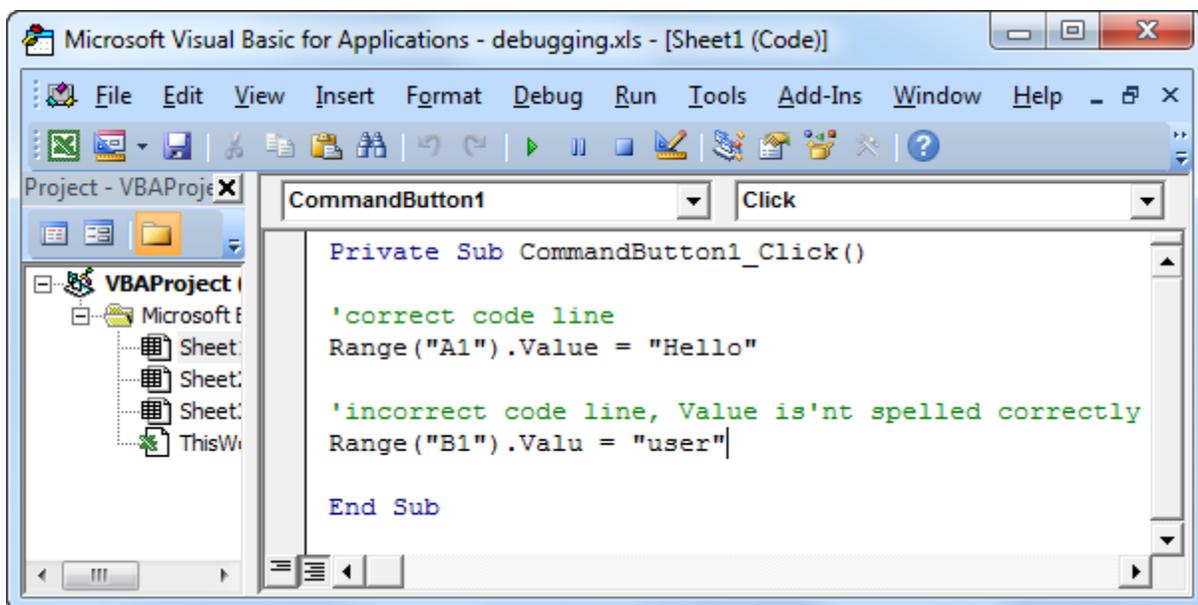
4. Next, in the Visual Basic Editor, click on Reset to stop the debugger! (More about the debugger in the next chapter) Now change the code. Excel VBA has colored the word MsgBox blue to indicate the macro error.



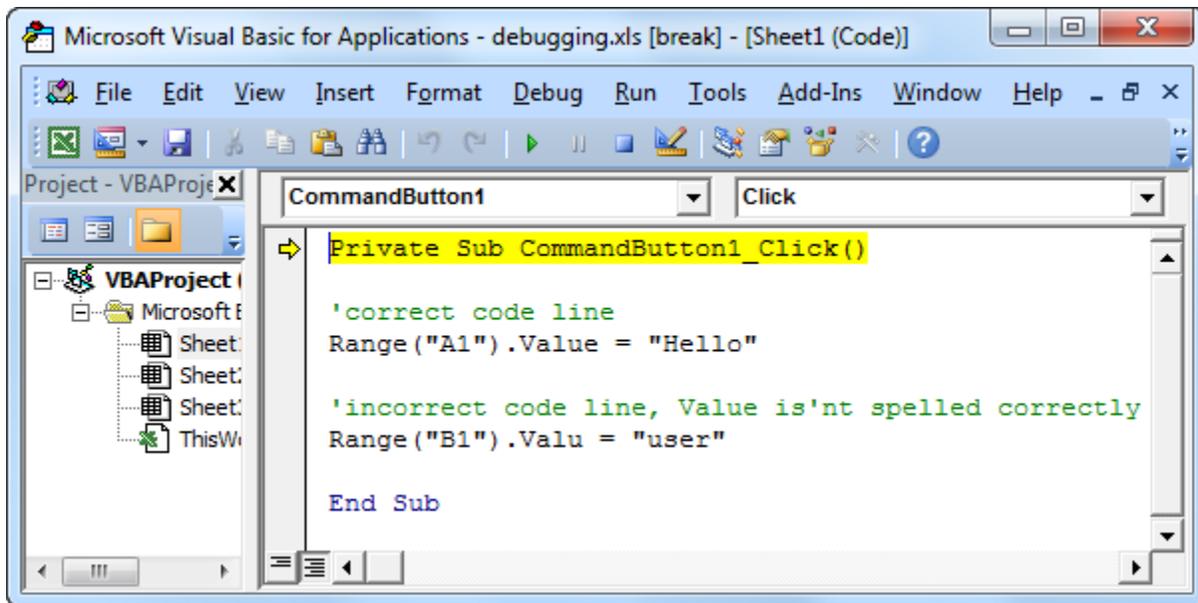
Debug Macros

You may have heard of the technique called debugging before. With this technique you can find errors in your Excel VBA code before you execute the code. To show you how to debug a macro, execute the following steps.

1. Launch the Visual Basic Editor.
2. Create the following incorrect macro:

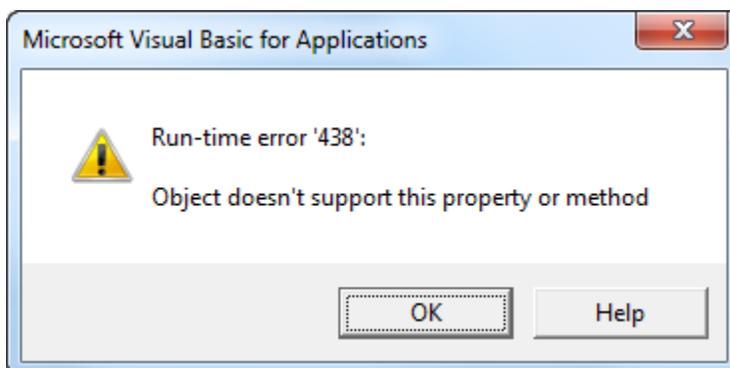


2. Place your cursor before Private.
3. There are two ways to debug a macro. Either press F8 or choose Debug from the Menu and then Step Into. Debug your macro. The first line will turn yellow.



4. Press F8 again. The second code line will turn yellow. As long as the debugger thinks everything is okay, nothing will happen.

5. Press F8 twice. The following error will show up.



6. It says that the object doesn't support the property or method. The Range object has a property called Value. Value isn't spelled correctly here, so the debugger doesn't recognize the property. More about objects, properties and methods in the [Object, Properties and Methods](#) chapter. You may find debugging a lot of work at this stage, but it will definitely pay off once your programs become more complicated.

Objects, Properties & Methods

[Excel VBA Property](#) | [Excel VBA Method](#) | [See all the properties and methods of an Object](#)

In Excel VBA, objects, properties and methods are connected with a dot. Properties are something which an object has (they describe the object), while methods do something (they perform an action with an object).

Excel VBA Property

Let's take a look at an Excel VBA object and an Excel VBA property. We will use the Range object and the Formula property. The Range object is nothing more than a cell (or cells) on your worksheet. We already know from Excel that a cell can contain a formula.

1. Place the value 10 into cell A1.
2. Create a command button.
3. Add the line:

```
Range("B1").Formula = Range("A1") * 2
```

4. Execute the macro. This macro places the formula into cell B1 and the result will be calculated (20).

Excel VBA Method

Now let's take a look at an Excel VBA object and an Excel VBA method. We will use the Range object again and the ClearContents method.

1. Place the value 10 into cell A1.
2. Create a command button.
3. Add the line:

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

4. Execute the macro. Cell A1 is now empty!

That's how easy object-oriented programming in Excel Visual Basic is!

See all the properties and methods of an object.

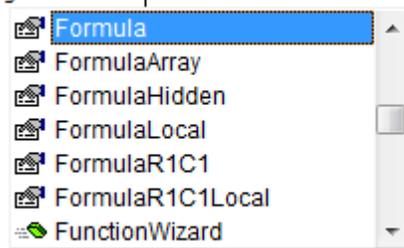
The Range object has many more properties and methods. Want to see more?

1. Launch the Visual Basic Editor.

2. Type in: Range.

3. A list will appear showing you all the Excel VBA methods and properties of the Range object. The fingers are the properties and the green boxes are the methods of the Range object.

`range.formul`



Workbook and Worksheet

[Object Hierarchy](#) | [Properties and Methods of the Workbook and Worksheet Object](#)

If you are not familiar with [Objects, Properties and Methods](#) yet, we highly recommend you to read this chapter first.

Object Hierarchy

Now that you have seen the Range object, you can understand the Workbook and Worksheet object better. In Excel Visual Basic each 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. An example of an object of the Application object is 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.

We have used the following code line a lot:

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

but what we really meant was cell A1 on the first worksheet of Book1. Thus we should actually add the following line in Excel VBA:

```
Application.Workbooks("Book1").Worksheets(1).Range("A1").Value
```

Fortunately we do not have to add a code line this way. This is because Excel Visual Basic knew we meant Book1 and the first worksheet because we placed our command button there (remember?). Now also remember the automatically created module when we recorded a macro with the [Excel Macro Recorder](#). Code placed into a module is available to all workbooks and worksheets.

Place the Sub test into a module (In the Visual Basic Editor, click on Insert and then Module).

```
Sub test()  
  
Range("A1").Value = "code placed here"  
  
End Sub
```

1. Execute the code (Click on Macros and then Run, or click on Run from the Visual Basic Editor). The words "code placed here" will be placed into cell A1. No surprise.

2. Now go to the second worksheet. Execute the code again. You will see that the words will be placed on the second worksheet as well!

3. Now even open a new workbook and execute the macro again. You will see that the words will be placed there as well! That is because we didn't specify a workbook or worksheet name and Excel VBA automatically takes the active workbook and active worksheet. Be aware that if you want to change different things on different sheets to include the Worksheet object.

Properties and methods of the Workbook and Worksheet object

You may have noticed that worksheets and workbooks are both plural (see the complete code line mentioned earlier). That's because they are actually 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, that is: a single workbook or single worksheet, in two ways. Using the index number, Worksheets(1) is the first worksheet starting from the left. Using the member's name: Worksheets("sheet1").

It probably doesn't surprise you that the collections and the members of the collections have properties and methods as well. Here are some examples.

1. The Count property of the Worksheets collection and Workbooks collection. The following code line counts the number of worksheets of a workbook. Place a [command button](#) on your worksheet and add the code line:

```
MsgBox Worksheets.Count
```

Result when you click the command button on the sheet:



You can also use the Count property to count the number of active workbooks.

2. The Add method of the Workbooks collection and Worksheets collection. The following code line creates a new worksheet.

```
Worksheets.Add
```

You can also use the Add method to add a new workbook.

3. The Worksheet object contains more interesting collections, such as the Rows collection. In Excel VBA you can use the Select method to select a row. The code line below selects row 2.

```
Worksheets(1).Rows(2).Select
```

In a similar way, you can select a column. The code line below selects column 7.

```
Worksheets(1).Columns(7).Select
```

Workbook and Worksheet

[Object Hierarchy](#) | [Properties and Methods of the Workbook and Worksheet Object](#)

If you are not familiar with [Objects, Properties and Methods](#) yet, we highly recommend you to read this chapter first.

Object Hierarchy

Now that you have seen the Range object, you can understand the Workbook and Worksheet object better. In Excel Visual Basic each 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. An example of an object of the Application object is 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.

We have used the following code line a lot:

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

but what we really meant was cell A1 on the first worksheet of Book1. Thus we should actually add the following line in Excel VBA:

```
Application.Workbooks("Book1").Worksheets(1).Range("A1").Value
```

Fortunately we do not have to add a code line this way. This is because Excel Visual Basic knew we meant Book1 and the first worksheet because we placed our command button there (remember?). Now also remember the automatically created module when we recorded a macro with the [Excel Macro Recorder](#). Code placed into a module is available to all workbooks and worksheets.

Place the Sub test into a module (In the Visual Basic Editor, click on Insert and then Module).

```
Sub test()  
  
Range("A1").Value = "code placed here"  
  
End Sub
```

1. Execute the code (Click on Macros and then Run, or click on Run from the Visual Basic Editor). The words "code placed here" will be placed into cell A1. No surprise.

2. Now go to the second worksheet. Execute the code again. You will see that the words will be placed on the second worksheet as well!

3. Now even open a new workbook and execute the macro again. You will see that the words will be placed there as well! That is because we didn't specify a workbook or worksheet name and Excel VBA automatically takes the active workbook and active worksheet. Be aware that if you want to change different things on different sheets to include the Worksheet object.

Properties and methods of the Workbook and Worksheet object

You may have noticed that worksheets and workbooks are both plural (see the complete code line mentioned earlier). That's because they are actually collections. The Workbooks collection contains all the Workbook objects that are currently open. The Worksheets collection contains all the Worksheet objects in a workbook.

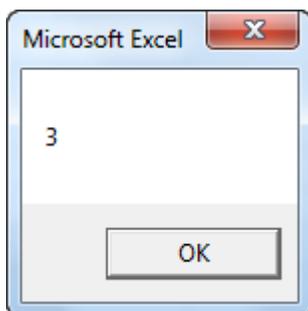
You can refer to a member of the collection, that is: a single workbook or single worksheet, in two ways. Using the index number, Worksheets(1) is the first worksheet starting from the left. Using the member's name: Worksheets("sheet1").

It probably doesn't surprise you that the collections and the members of the collections have properties and methods as well. Here are some examples.

1. The Count property of the Worksheets collection and Workbooks collection. The following code line counts the number of worksheets of a workbook. Place a [command button](#) on your worksheet and add the code line:

```
MsgBox Worksheets.Count
```

Result when you click the command button on the sheet:



You can also use the Count property to count the number of active workbooks.

2. The Add method of the Workbooks collection and Worksheets collection. The following code line creates a new worksheet.

```
Worksheets.Add
```

You can also use the Add method to add a new workbook.

3. The Worksheet object contains more interesting collections, such as the Rows collection. In Excel VBA you can use the Select method to select a row. The code line below selects row 2.

```
Worksheets(1).Rows(2).Select
```

In a similar way, you can select a column. The code line below selects column 7.

```
Worksheets(1).Columns(7).Select
```

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.

WorksheetFunction

You can access almost any Excel function through the Application object.

1. For example, place a [command button](#) on your worksheet and add the following code line:

```
Range("A3").Value = Application.WorksheetFunction.Average(Range("A1:A2"))
```

When you click the command button on the worksheet, Excel VBA calculates the average of the values in Cell A1 and Cell A2 and places the result into cell A3.

	A	B	C	D	E
1	5				
2	10				
3	7.5				
4					



Note: instead of `Application.WorksheetFunction.Average`, you can also just use `WorksheetFunction.Average`.

There are many more worksheet functions you can use in Excel VBA. Want to see more?

1. Launch the Visual Basic Editor.
2. Type in: `worksheetfunction`.
3. A list will appear showing you all the worksheet functions available.

worksheetfunction.m

- Match
- Max
- MDeterm
- MDuration
- Median
- Min
- MInverse

Programming

This section is for users who want to get the most out of Excel VBA. Excel VBA Programming is not difficult, but you do need to know the keywords used in Excel VBA.

1 [Variables](#): Excel VBA uses variables just like any other programming language. Learn how to declare and initialize an excel vba variable of type Integer, String, Double, Boolean and Date.

2 [String Manipulation](#): There are many functions in Excel VBA we can use to manipulate strings. In this chapter you can find a review of the most important functions.

3 [Calculate](#): Calculate with Excel VBA and add, subtract, multiply and divide values just like you are used to doing in Excel.

4 [If Then Statement](#): In many situations you only want Excel VBA to execute certain code lines when a specific condition is met. The If Then statement allows you to do this. Instead of multiple If Then statements, you can use Select Case.

5 [Cells](#): Instead of the more common Range object we could also use Cells. Using Cells is particularly useful when we want to loop through ranges.

6 [Loop](#): Looping is one of the most powerful programming techniques. A loop (or For Next loop) in Excel VBA enables you to loop through a range of data with just a few lines of code.

7 [Logical Operators](#): Do you want to execute code in Excel Visual Basic when more conditions are met? Or just one? Or none? Logical operators are what you need! Logical operators such as And, Or and Not are often used in Excel VBA.

8 [Range](#): The Range object which is the representation of a cell (or cells) on your worksheet is the most important object of Excel VBA. It has many properties and methods and they are essential to manipulate the content of your Excel worksheet. In this chapter you will discover the most useful properties and methods of the Excel VBA Range object. They enable you to obtain control over your Excel worksheet.

9 [Events](#): This chapter teaches you how to program workbook and worksheet events. Events are actions performed by users which trigger Excel VBA to execute a macro. For example, when you open a workbook or when you change something on an Excel worksheet, Excel VBA can automatically execute a macro.

10 [Array](#): An Excel VBA array is a group of variables. You can refer to a specific variable (element) of an array by using the array name and the index number.

11 [Date and Time](#): Dates and Times in Excel VBA can be manipulated in many ways. Easy examples are given in this chapter.

12 [Function and Sub](#): The difference between a function and a sub in Excel VBA is that a function can return a value and a sub cannot. In this chapter we will look at an easy example of a function and a sub. Functions and subs become very useful as program size increases.

Variables

[Integer](#) | [String](#) | [Double](#) | [Boolean](#) | [Date](#)

This chapter teaches you how to declare, initialize and display an Excel VBA variable. A variable is used to store a value. A variable can be of any type. In Excel VBA, we have variables of type Integer to store whole numbers, variables of type Double which can also store numbers after the comma, variables of type String to store text, variables of type Boolean to hold the value True or False and variables of type Date to store dates.

Place a [command button](#) on your worksheet and add the code lines described in this chapter. To execute the code lines, click the command button on the sheet.

Variable of type Integer

Integer variables are used to store whole numbers.

Code:

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

Result:

	A	B	C	D	E
1	6				
2					
3					
4					



1. The first code line 'Dim x As Integer' declares an Excel VBA variable with name x of type Integer. Letting Excel VBA know we are using a variable is called declaring a variable.

2. Next, we initialize the variable. In Excel VBA (and in many other programming languages), initializing simply means assigning a beginning (initial) value to a variable. This is done by adding the line 'x =6'.

3. Finally, we place the value assigned to the variable x into cell A1. This can be done by adding the line: 'Range("A1").Value = x'.

Variable of type String

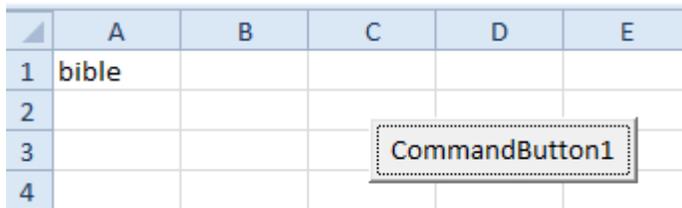
String variables are used to store text.

Code:

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

Result:

	A	B	C	D	E
1	bible				
2					
3					
4					



1. The first code line 'Dim book As String' declares an Excel VBA variable with name book of type String.
2. Next, we initialize the variable. For example, add the line: book = "bible". Always use apostrophes to initialize String variables.
3. Finally, we place the text assigned to the variable book into cell A1. You can achieve this by adding the line 'Range("A1").Value = book'

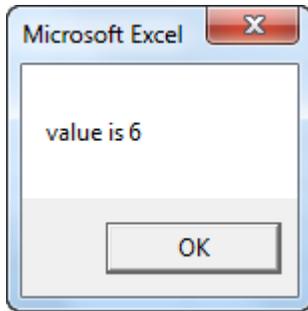
Variable of type Double

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: You might wonder why you would use Integer variables, if you could use the more accurate Double variables. That is because Double variables, and even more accurate variables, need more space and as a result your code will run slower (as program size increases). Apart from this, you will see that errors are easier to find when you use variables of the right type.

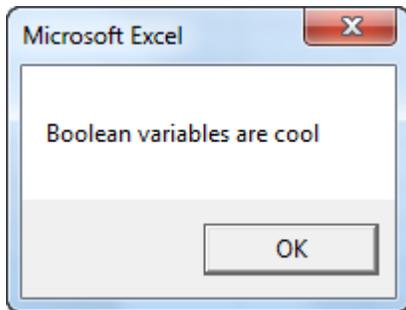
Variable of type Boolean

In Excel VBA, you can use Boolean variables to hold the value True or False.

Code:

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

Result:



1. The first code line declares a variable of type Boolean.
2. Next, we initialize the Boolean variable with value True.
3. Finally, we can use the Boolean variable to only display a MsgBox if the variable holds the value True.

Variable of type Date

To declare a date, we use the Dim statement. To initialize a date, we use the DateValue function. Want to learn more about dates? Go to the [Date and Time](#) chapter.

String Manipulation

[Join Strings](#) | [Left](#) | [Right](#) | [Len](#) | [Instr](#) | [Mid](#)

There are many functions in Excel VBA we can use to manipulate strings. Below you can find a review of the most important functions.

Place a [command button](#) on your worksheet and add the code lines described in this chapter. To execute the code lines, click the command button on the sheet.

Join Strings

We use the & operator to concatenate (join) strings.

Code:

```
Dim text1 As String, text2 As String
text1 = "Hi "
text2 = "Tim"

MsgBox text1 & text2
```

Result:



Left

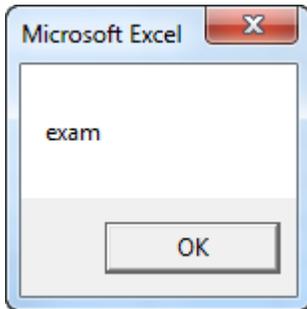
To extract the leftmost characters from a string, use Left.

Code:

```
Dim text As String
text = "example text"
```

```
MsgBox Left(text, 4)
```

Result:



Right

To extract the rightmost characters from a string, use Right. We can directly insert text in a function as well.

Code:

```
MsgBox Right("example text", 2)
```

Result:



Len

To get the length of a string, use Len.

Code:

```
MsgBox Len("example text")
```

Result:



Note: space (position 8) included!

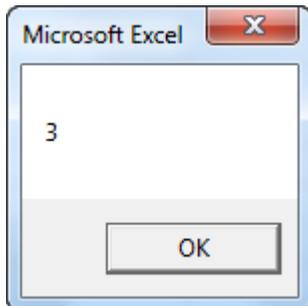
Instr

To find the position of a substring in a string, use Instr.

Code:

```
MsgBox Instr("example text", "am")
```

Result:



Note: string "am" found at position 3.

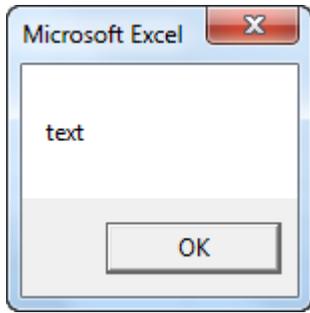
Mid

To extract a substring, starting in the middle of a string, use Mid.

Code:

```
MsgBox Mid("example text", 9, 4)
```

Result:



Note: started at position 9 (t) with length 4.

Calculate

Let's develop a small macro which involves a simple calculation (adding a value to a variable) and a very important programming technique.

Place a [command button](#) on your worksheet and add the following code lines:

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

x = x + 1
Range("A1").Value = x
```

1. The first code line declares a variable with name x of type Integer.
2. Next, we initialize this variable with the value of cell A1.
3. We want to add 1 to the variable x. You can do this by adding the line 'x = x + 1'. In Excel Visual Basic (and in other programming languages), the symbol '=' means becomes. It does not mean equal! So x = x + 1 means x becomes x + 1. In other words: take the present value of x and add 1 to it. Example: If x = 6, x becomes 6 + 1 = 7.
4. Finally, place the variable with the new value into cell A1.

Exit the Visual Basic Editor and enter a value into cell A1. Click on CommandButton1 to see how the value of cell A1 is incremented each time you click on CommandButton1.

Result:

	A	B	C	D	E
1	3				
2					
3			CommandButton1		
4					

	A	B	C	D	E
1	4				
2					
3			CommandButton1		
4					

	A	B	C	D	E
1	5				
2					
3					
4					

CommandButton1

You've just created a counter in Excel VBA. Well done!

If Then Statement

[If Then Statement](#) | [Else Statement](#)

In many situations we only want Excel VBA to execute certain code lines when a specific condition is met. The If Then statement allows you to do this. Instead of multiple If Then statements, you can use Select Case.

If Then Statement

Place a [command button](#) on your worksheet and add the following code lines:

```
Dim score As Integer, grade As String
score = Range("A1").Value

If score >= 60 Then grade = "passed"

Range("B1").Value = grade
```

1. The first code line declares two variables. One variable of type Integer and one variable of type String.
2. Next, we initialize the variable score with the value of cell A1.
3. If score is higher or equal to 60, we assign the text 'passed' to the variable grade.
4. Finally, we place the value of the variable grade into cell B1.

Exit the Visual Basic Editor, enter a value into cell A1, and click on CommandButton1.

Result:

	A	B	C	D	E
1	65	passed			
2					
3			CommandButton1		
4					

Else Statement

If score is lower than 60, the previous code lines empty cell B1. We can use an Else statement to assign the text 'failed' to the variable grade if score is lower than 60.

Code:

```
Dim score As Integer, grade As String
score = Range("A1").Value

If score >= 60 Then
    grade = "passed"
Else
    grade = "failed"
End If

Range("B1").Value = grade
```

Result when you click the command button on the sheet:

	A	B	C	D	E
1	55	failed			
2					
3			CommandButton1		
4					

Note: It is good practice to always start a new line after the words Then and Else and to end with End If (see second example). Only if there is one code line after Then and no Else statement, it is allowed to place a code line directly after Then and to omit (leave out) End If (see first example).

Cells

So far we have used the Range object to place values into cells or read values from cells. Instead of the Range object, you can also use Cells. For example, Cells(4,2).value is the same as Range("B4").value. Cells(4,2).value corresponds to the value of the cell with rownumber 4 and columnnumber 2, which is cell B4.

Place a [command button](#) on your worksheet and add the following code line:

```
Cells(4, 2).Value = 100
```

Result when you click the command button on the sheet:

	A	B	C	D	E
1					
2					
3			CommandButton1		
4		100			
5					

You might wonder why you would use Cells, if you could use the more common Range object. That is because using Cells is particularly useful when we want to loop through ranges.

Loop

The Excel VBA loop (or For Next loop) is a very useful programming statement which is often used in Excel VBA. First, we will look at an easy example of how to loop through a one-dimensional range.

Single Loop

You can use a single loop to loop through a one dimensional range.

Place a [command button](#) on your worksheet and add the following code lines:

```
Dim total As Integer, i As Integer
total = 0

For i = 1 To 4
    'ignore the two code lines below, they are only added to illustrate the loop
    Cells(i, 1).Select
    MsgBox "i = " & i

    If Cells(i, 1).Value > 40 Then total = total + 1
Next i

MsgBox total & " values higher than 40"
```

1. The first two code lines declare two variables of type Integer. One named total and one named i.
2. Next, we initialize the variable total with value 0.
3. Add the For Next loop which runs from 1 to 4.
4. Create an If Then statement which increments total by 1 if a value is higher than 40.
5. Finally, use a message box to display the total number of values higher than 40.

Result when you click the command button on the sheet a few times:

	A	B	C	D
1	39			
2	48			
3	23			
4	42			
5				
6				
7				
8				
9				



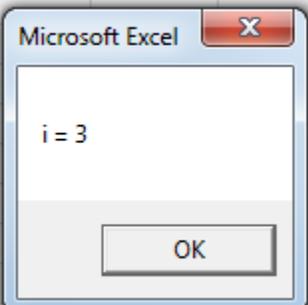
A dialog box titled "Microsoft Excel" with a close button (X) in the top right corner. The main area contains the text "i = 1". At the bottom, there is an "OK" button.

	A	B	C	D
1	39			
2	48			
3	23			
4	42			
5				
6				
7				
8				
9				



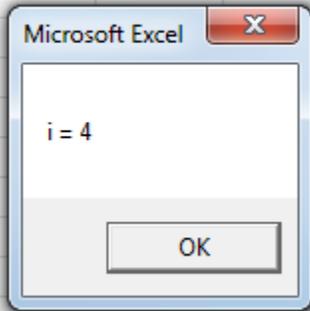
A dialog box titled "Microsoft Excel" with a close button (X) in the top right corner. The main area contains the text "i = 2". At the bottom, there is an "OK" button.

	A	B	C	D
1	39			
2	48			
3	23			
4	42			
5				
6				
7				
8				
9				



A dialog box titled "Microsoft Excel" with a close button (X) in the top right corner. The main area contains the text "i = 3". At the bottom, there is an "OK" button.

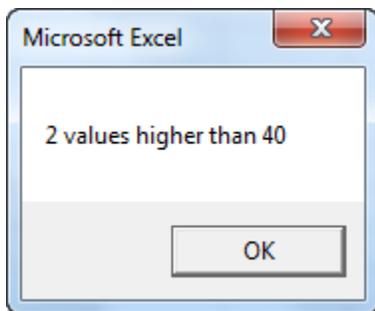
	A	B	C	D
1	39			
2	48			
3	23			
4	42			
5				
6				
7				
8				
9				



Explanation: The code lines between For and Next will be executed four times. For $i = 1, 2, 3$ and 4 . For $i = 1$, Excel VBA fills in 1 for i and gets `Cells(1,1).value`. This is the first value. When Excel VBA reaches `Next i`, it jumps back to the For statement increasing i with 1. For $i = 2$, Excel VBA fills in 2 for i and gets `Cells(1,2).value`. This is the second value. For $i = 2$, Excel VBA also increments total by 1 because the second value is higher than 40. For $i = 3$, etc.

Excel VBA loops through the code four times and after that leaves the For Next loop and executes the rest of the code.

Result:



Note: it is good practice to always indent (tab) the code between the words For and Next. This makes your code easier to read.

Logical Operators

[Logical Operator And](#) | [Logical Operator Or](#) | [Logical Operator Not](#)

In all sorts of macros we use Excel VBA logical operators. The three most used logical operators in Excel Visual Basic are: And, Or and Not. Each logical operator will now be illustrated with the help of an easy example.

Logical Operator And

For example, we hire a person for a certain job, only if his/her IQ quotient is higher or equal to 110 and the impression he or she leaves during an interview is higher or equal to 7 (on a scale of 1 to 10).

	A	B	C	D	E
1	IQ	90			
2	Impression	9			
3			CommandButton1		
4	Result				
5					

Place a [command button](#) on your worksheet and add the following code lines:

```
If Range("B1").Value >= 110 And Range("B2").Value >= 7 Then
    Range("B4").Value = "Hire"
Else
    Range("B4").Value = "Do not hire"
End If
```

Result when you click the command button on the sheet:

	A	B	C	D	E
1	IQ	90			
2	Impression	9			
3			CommandButton1		
4	Result	Do not hire			
5					

The macro uses the logical operator And to get the right result. We do not hire the person because his/her IQ quotient is lower than 110.

Logical Operator Or

For example, we hire a person for a certain job, only if his/her IQ quotient is higher or equal to 110 or the impression he or she leaves during an interview is higher or equal to 7 (on a scale of 1 to 10).

Code:

```
If Range("B1").Value >= 110 Or Range("B2").Value >= 7 Then
    Range("B4").Value = "Hire"
Else
    Range("B4").Value = "Do not hire"
End If
```

Result:

This example shows you the effect of the Or statement. Although, the IQ quotient of the person is way below 110, we hire the person because he or she leaves a very good impression during the interview.

Logical Operator Not

For example, we do not hire a person for a certain job, who is 'greedy' and 'all about the money'. The macro below uses the logical operator Not to get the right result. The Not operator designates that the condition must not be true.

Slightly adjust the previous macro. Code:

```
If (Range("B1").Value >= 110 Or Range("B2").Value >= 9) And Not Range("B3").Value =
"Yes" Then
    Range("B4").Value = "Hire"
Else
    Range("B4").Value = "Do not hire"
End If
```

Result:

	A	B	C	D	E
1	IQ	90			
2	Impression	9			
3	All about the money	Yes	CommandButton1		
4	Result	Do not hire			
5					

We want one of the first two conditions to be true (use brackets) and the third condition must not be true. We do not hire the person because the third condition is true (cell B3 contains the word Yes).

You can [debug](#) through the code using F8 (see picture below). Place the cursor on the code to see the actual values! The '_' symbol is used here to continue the statement on a new line (not necessary). Always test your macro when you use logical operators to be sure that they do what you want.

```

Private Sub CommandButton1_Click()
    If (Range("B1").Value >= 110 Or Range("B2").Value >= 9)
    Range("B1").Value = 90 B3).Value = "Yes" Then
        Range("B4").Value = "Hire"
    Else
        Range("B4").Value = "Do not hire"
    End If
End Sub

```

Range

[Range Examples](#) | [Declare a Range](#) | [Select a Range](#) | [Rows](#) | [Columns](#) | [Copy/Paste a Range](#) | [Clear a Range](#) | [Count](#)

This chapter gives an overview of the properties and methods of the very important Excel VBA Range object.

Range Examples

We have already seen the Range object in the previous chapters. The Range object is the representation of a cell (or cells) on your worksheet. The code line: 'Range("A1").value = 1' places the value 1 into cell A1. You can also execute operations in Excel Visual Basic on more than one cell at the same time. See the following three macros.

Place a [command button](#) on your worksheet and add the following code line:

```
Range("A1:A4").Value = 2
```

Result when you click the command button on the sheet:

	A	B	C	D	E	F
1	2					
2	2					
3	2					
4	2					
5						



Code:

```
Range("A1:B4").Value = 5
```

Result:

	A	B	C	D	E	F
1	5	5				
2	5	5				
3	5	5				
4	5	5				
5						



Code:

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

Result:

	A	B	C	D	E	F
1	10					
2	10					
3		10	10	CommandButton1		
4		10	10			
5						

Declare a Range

In the [Variables](#) chapter, we learned how to declare a variable in Excel VBA. Besides declaring a variable, you can also declare an Excel VBA Range object. You can do this by using the keywords Dim and Set.

Place a [command button](#) on your worksheet and add the following code lines:

```
Dim example As Range
Set example = Range("A1:D1")

example.Value = 8
```

Result when you click the command button on the sheet:

	A	B	C	D	E	F
1	8	8	8	8		
2						
3				CommandButton1		
4						

Select a Range

An important method of the Excel VBA 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:

	A	B	C	D	E	F
1						
2						
3						
4						
5						

CommandButton1

Rows

The Rows property gives access to a specific row of a range. The following macro selects the third row of Range("A1:C4").

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

example.Rows(3).Select
```

Result:

	A	B	C	D	E	F
1						
2						
3						
4						
5						

CommandButton1

Note: Range("A1:C4") has been formatted for illustration.

Columns

The Columns property gives access to a specific column of a range. The following macro selects the second column of Range("A1:C4").

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

example.Columns(2).Select
```

Result:

	A	B	C	D	E	F
1						
2						
3						
4						
5						

CommandButton1

Note: Range("A1:C4") has been formatted for illustration.

Copy and Paste a Range

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

The following macro copies Range("A1:A2") and pastes it into Range("C4:C5").

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

Range("C4").Select
ActiveSheet.Paste
```

Result:

	A	B	C	D	E	F
1	1					
2	2					
3						
4			1			
5			2			
6						

CommandButton1

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

```
Range("C4:C5").Value = Range("A1:A2").Value
```

Clear a Range

To clear the content of an Excel range, you can use the Clear method. Besides emptying the range, this method also clears the format of the range. If you only want to clear the content, you can use the ClearContents method. If you only want to clear the format, you can use the ClearFormats method. The following code line clears the content of cell A1.

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

Note: Range("A1").ClearContents in Excel VBA is exactly the same as Range("A1").value = ""

Count

With the Count property, you can count the number of cells, rows and columns of an Excel range. Below are some examples.

	A	B	C	D	E	F
1						
2						
3						
4						
5						

CommandButton1

The following macro counts the number of cells of the formatted range.

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

MsgBox example.Count
```

Result:

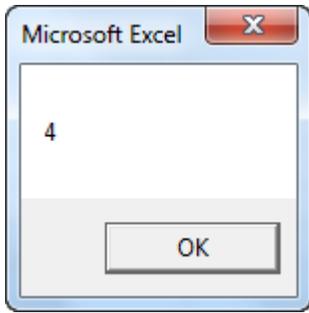


The following macro counts the number of rows of the formatted range.

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

MsgBox example.Rows.Count
```

Result:



In a similar way, you can count the number of columns of a range.

Events

[Event Examples](#) | [Workbook Event](#)

This chapter teaches you how to program a workbook event in Excel VBA. Events are actions performed by users, which trigger Excel VBA to execute a macro. First, we will look at some examples of events.

Event Examples

Below is a list of event examples.

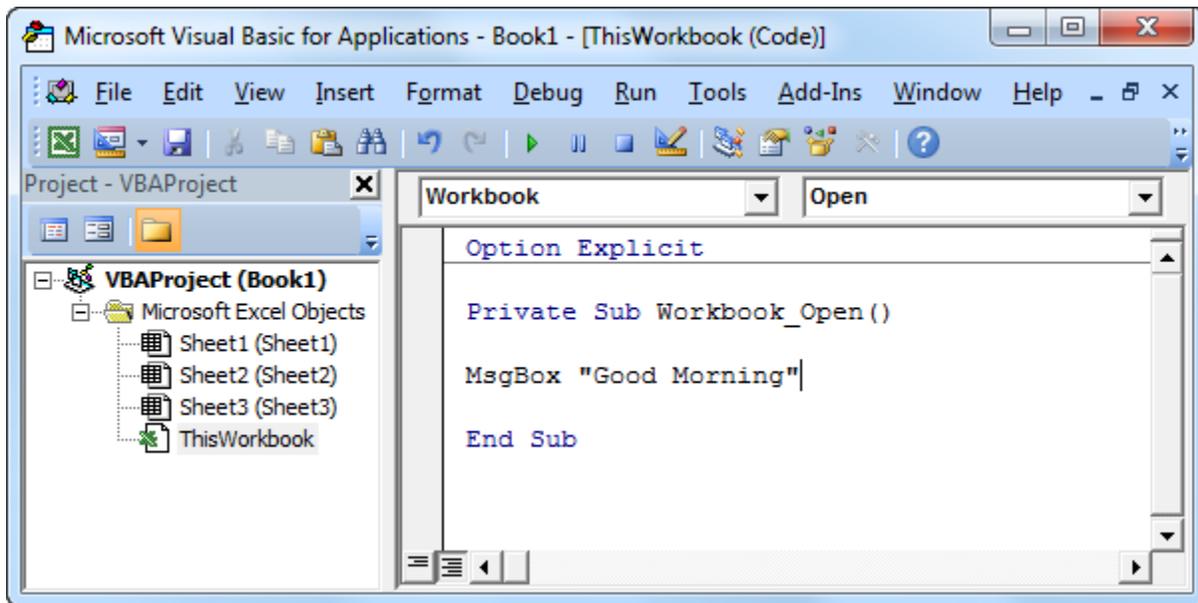
- the user opens a workbook (workbook event).
- the user activates a worksheet (worksheet event).
- the user changes the content of a cell on the sheet (worksheet event).
- the user double clicks a cell (worksheet event).

Workbook Event

Workbook events are actions performed on the workbook (= your Excel file) which trigger Excel VBA to execute a macro. To create a workbook event, execute the following steps.

1. Launch the Visual Basic Editor.
2. Double Click on This Workbook in the Project Explorer (important!). The code window will appear showing you two drop-down lists.
3. Choose Workbook from the left drop-down list.
4. The right drop-down list shows you all the workbook events (don't worry, you will never use most of these events). Choose the Open event. Excel VBA automatically places a Sub procedure for you.
5. Now you can add a line which will only be executed by Excel VBA when you open the workbook! Add the line:

```
MsgBox "Good Morning"
```



6. Close the Excel file (don't forget to save).

7. Open the Excel file and test the workbook event.

Result:



Note: There is a chance your macro has not been executed because your security settings are not set up correctly. If so, go to [Security Settings](#) and click on: enable all macros.

Array

An Excel VBA array is a group of variables. You can refer to a specific variable (element) of an array by using the array name and the index number. To create a one-dimensional array, execute the following steps.

Place a [command button](#) on your worksheet and add the following code lines:

```
Dim Films(1 To 5) As String

Films(1) = "Lord of the Rings"
Films(2) = "Speed"
Films(3) = "Star Wars"
Films(4) = "The Godfather"
Films(5) = "Pulp Fiction"

MsgBox Films(4)
```

1. The first code line declares a String array with name Films. The array consists of five elements.
2. Next, we initialize each element of the array. In other words: we assign beginning values.
3. Finally, we display the fourth film using a MsgBox.

Exit the Visual Basic Editor and test the array.

Result when you click the command button on the sheet:



Date and Time

[Year, Month, Day of a Date](#) | [DateAdd](#) | [Current Date & Time](#) | [Hour, Min, Sec](#) | [TimeValue](#)

Dates and times in Excel VBA can be manipulated in many ways. This chapter teaches you how to get the year, month and day of an Excel VBA date, how to add a number of days to a date, how to get the current date and time, how to get the hour, minute and second of the current time and how to convert a string to a time serial number.

Place a [command button](#) on your worksheet and add the code lines described in this chapter. To execute the code lines, click the command button on the sheet.

Year, Month, Day of a Date

The following macro gets the year of a date. First, we declare a date using the Dim statement. To initialize a date, we use the DateValue function.

Code:

```
Dim exampleDate As Date

exampleDate = DateValue("Jun 19, 2010")

MsgBox Year(exampleDate)
```

Result:



Note: Use Month and Day to get the month and day of a date.

DateAdd

To add a number of days to a date, use the DateAdd function. The DateAdd function has three arguments. Fill in "d" for the first argument since we want to add days. Fill in 3 for the second argument to add 3 days. The third argument represents the date, to which in this example, the number of days will be added.

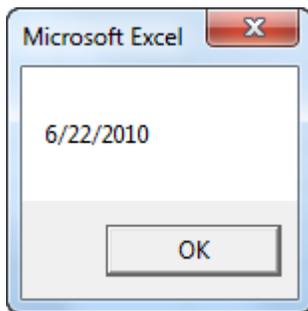
Code:

```
Dim firstDate As Date, secondDate As Date

firstDate = DateValue("Jun 19, 2010")
secondDate = DateAdd("d", 3, firstDate)

MsgBox secondDate
```

Result:



Note: Change "d" to "m" to add a number of months to a date. Place your cursor on DateAdd in Excel VBA and click on F1 for help on the other interval specifiers. The format of the date depends on your windows regional settings.

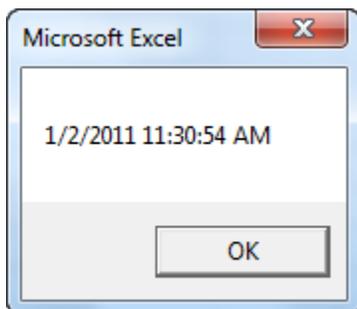
Current Date & Time

To get the current date and time, use the Now function.

Code:

```
MsgBox Now
```

Result:



Note: replace a date, such as "June 19, 2010" with Now and you can use all the functions described above on the current date!

Hour, Minute and Second

The following macro gets the hour of the current time.

Code:

```
MsgBox Hour (Now)
```

Result:



Note: Use Minute and Second to get the minute and second of the current time.

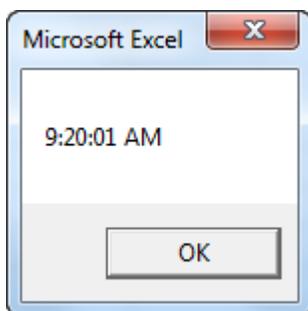
TimeValue

The TimeValue function converts a string to a time serial number. The time's serial number is a number between 0 and 1. For example, noon (halfway through the day) is represented as 0.5.

Code:

```
MsgBox TimeValue("9:20:01 am")
```

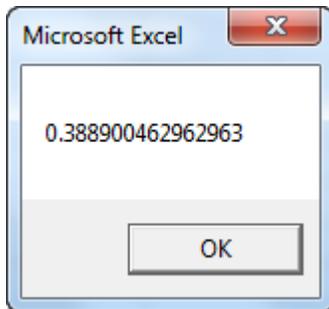
Result:



Now, to clearly see that Excel handles times internally as numbers between 0 and 1, add the following code lines:

```
Dim y As Double  
y = TimeValue("09:20:01")  
MsgBox y
```

Result:



Function and Sub

[Function](#) | [Sub](#)

The difference between a function and a sub in Excel VBA is that a function can return a value and a sub cannot. In this chapter we will look at an easy example of a function and a sub. Functions and subs become very useful as program size increases.

Function

If you want Excel VBA to perform a task that returns a result, you can use a function. Place a function into a module (In the Visual Basic Editor, click on Insert and then Module). For example, the function with name Area.

```
Function Area(x As Double, y As Double) As Double  
  
Area = x * y  
  
End Function
```

Explanation: This function has two arguments (of type Double) and a return type (the part after As also of type Double). You can use the name of the function (Area) in your code to indicate which result you want to return (here $x * y$).

You can now refer to this function (in other words call the function) from somewhere else in your code by simply using the name of the function and giving a value for each argument.

Place a [command button](#) on your worksheet and add the following code lines:

```
Dim z As Double  
  
z = Area(3, 5) + 2  
  
MsgBox z
```

Explanation: The function returns a value so you have to 'catch' this value in your code. You can use another variable (z) for this. Next, you can add another value to this variable (if you want). Finally, display the value using a MsgBox.

Result when you click the command button on the sheet:



Sub

If you want Excel VBA to perform some actions, you can use a sub. Place a sub into a module (In the Visual Basic Editor, click on Insert and then Module). For example, the sub with name Area.

```
Sub Area(x As Double, y As Double)

MsgBox x * y

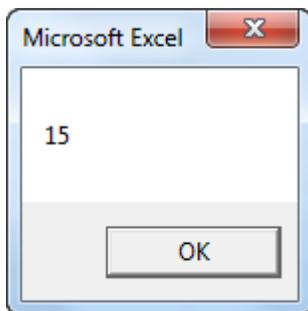
End Sub
```

Explanation: This sub has two arguments (of type Double). It does not have a return type! You can refer to this sub (call the sub) from somewhere else in your code by simply using the name of the sub and giving a value for each argument.

Place a [command button](#) on your worksheet and add the following code line:

```
Area 3, 5
```

Result when you click the command button on the sheet:



Can you see the difference between the function and the sub? The function returned the value 15. We added the value 2 to this result and displayed the final result. When we called the sub we had no more control over the result (15) because a sub cannot return a value!

Controls

This section is about communicating with users using controls or a Userform. Learn how to use these controls in Excel 2010, Excel 2007 or Excel 2003. You can directly place controls on a sheet or place them on a Userform.

1 [Textbox](#): A textbox is an empty field where the user can fill in a piece of text. Learn how to draw a textbox on your worksheet, how to refer to a textbox in your Excel VBA code, and how to clear a textbox.

2 [Listbox](#): A listbox, is a drop down list from where the user can make a choice. Learn how to draw a listbox on your worksheet and how to add items to a listbox.

3 [Combobox](#): A combobox is the same as a listbox but now the user can also fill in his/her own choice if it is not included in the list. Learn how to draw a combobox on your worksheet and how to add items to a combobox.

4 [Checkbox](#): A checkbox is a field which can be checked to store information. Learn how to draw a checkbox on your worksheet and how to refer to a checkbox in your Excel VBA code.

5 [Option Buttons](#): Option buttons are the same as checkboxes except that option buttons are dependent on each other while checkboxes are not. This means that when you check one option button the other option button will automatically uncheck.

6 [Userform](#): This chapter teaches you how to create an Excel VBA Userform (also known as a dialog box). You can download the Userform on this page as well.

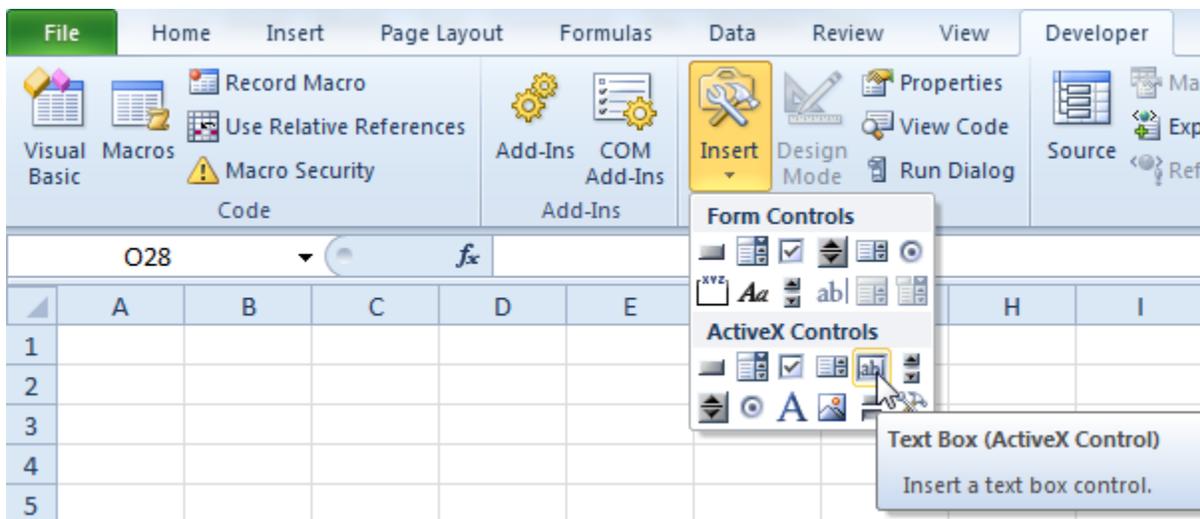
Textbox

[Draw a Textbox](#) | [Populate a Textbox](#) | [Clear a Textbox](#)

A textbox is an empty field where the user can fill in a piece of text. Learn how to draw a textbox on your worksheet, how to refer to a textbox in your Excel VBA code, and how to clear a textbox.

Draw a Textbox

Excel 2010 and Excel 2007 users. Click on Insert from the Developer tab and then click on Text Box in the ActiveX Controls section.



Excel 2003 users. Click on Text Box from the Control ToolBox.



1. Draw a textbox and a command button on your worksheet.

	A	B	C	D	E	F
1						
2		Import Data				
3						
4						
5						

2. You can change the caption of the command button by right clicking on the command button and then clicking on Properties and Caption (make sure Design Mode is selected).

3. You can change the name of the textbox by right clicking on the textbox and then clicking on Properties and Name. For now, we will leave TextBox1 as the name of the textbox.

Populate a Textbox

To populate the textbox, execute the following steps:

1. Right click on the command button. Click on View Code. Add the following code line:

```
TextBox1.Text = "Data imported successfully"
```

Result when you click the command button on the sheet:

	A	B	C	D	E	F
1						
2		Import Data	Data imported successfully			
3						
4						
5						

2. To place text from a textbox into a cell, add a code line like this:

```
Range("A1").Value = TextBox1.Text
```

Clear a Textbox

To clear a textbox, use the following code line:

```
TextBox1.Value = ""
```

Although in some situations it can be useful to directly place a textbox on your worksheet, a textbox is particularly useful when placed on a [Userform](#).

Listbox

[Draw a Listbox](#) | [Add items to Listbox](#)

An Excel VBA listbox, is a drop down list from where the user can make a choice. Learn how to draw a listbox on your worksheet and how to add items to a listbox.

Draw a Listbox

Excel 2010 and Excel 2007 users. Click on Insert from the Developer tab and then click on List Box in the ActiveX Controls section.

Excel 2003 users. Click on List Box from the Control ToolBox.



1. Draw a listbox on your worksheet.

	A	B	C	D	E
1					
2					
3					
4					
5					
6					

Add items to Listbox

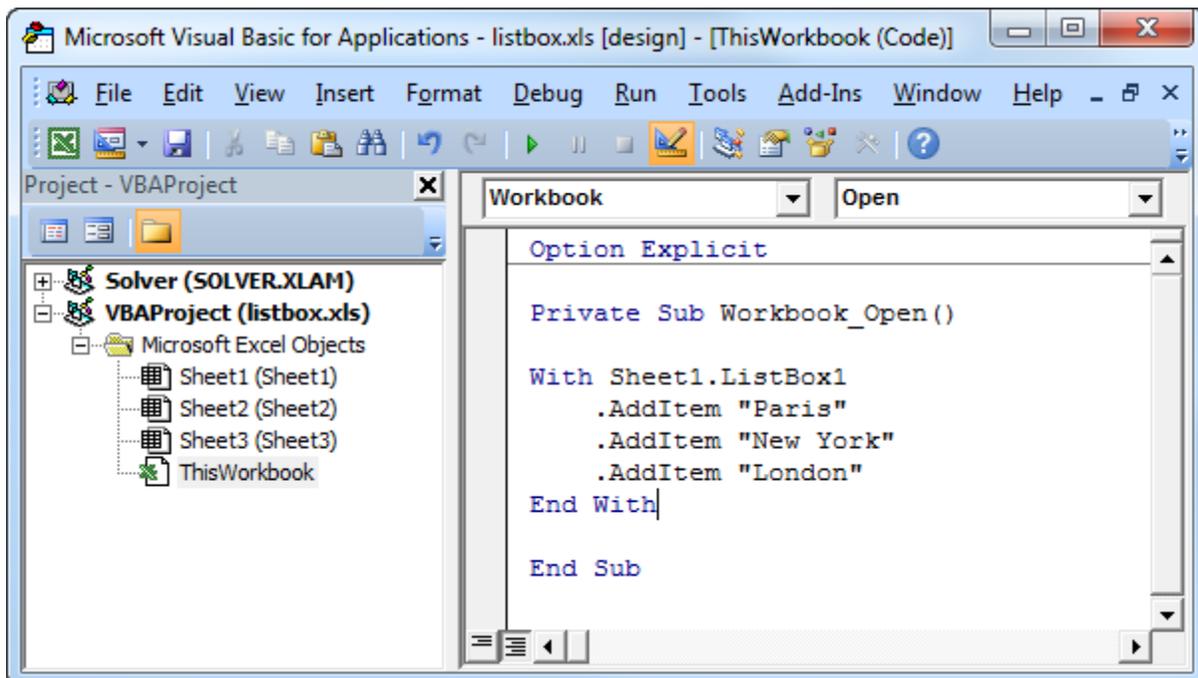
To add some items to the listbox, execute the following steps.

1. Add the following code lines to the [Workbook Open event](#) (or add them to your own code).

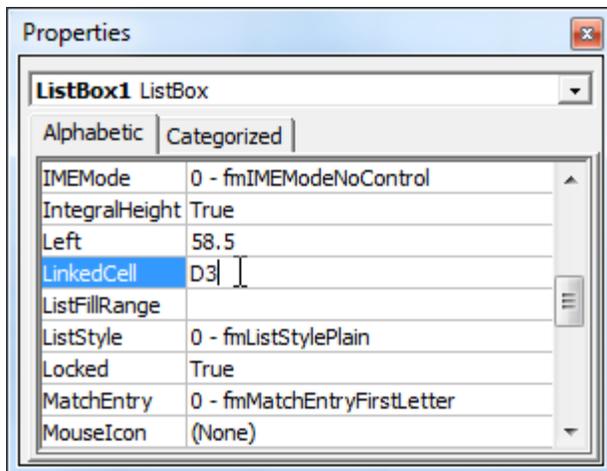
```
With Sheet1.ListBox1
    .AddItem "Paris"
    .AddItem "New York"
    .AddItem "London"
End With
```

Note: use Sheet2 if your listbox is located on the second worksheet.

Result:



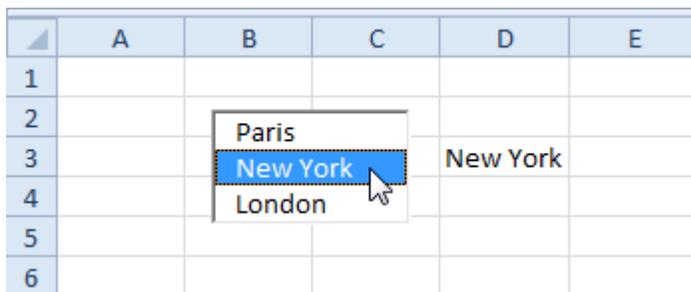
2. To link a cell to the listbox, right click on the listbox (make sure design mode is selected) and click on Properties. Fill in D3 for LinkedCell.



Note: also see the ListFillRange property, to fill a listbox with a range of cells.

3. Close Excel and reopen your Excel file.

Result:



4. Note: if you use these code lines in your own code, outside the Workbook Open event, add the following code line at the start of your code.

```
ListBox1.Clear
```

This way your items won't be added multiple times, when you execute your code more than once.

Although in some situations it can be useful to directly place a listbox on your worksheet, a listbox is particularly useful when placed on a [Userform](#).

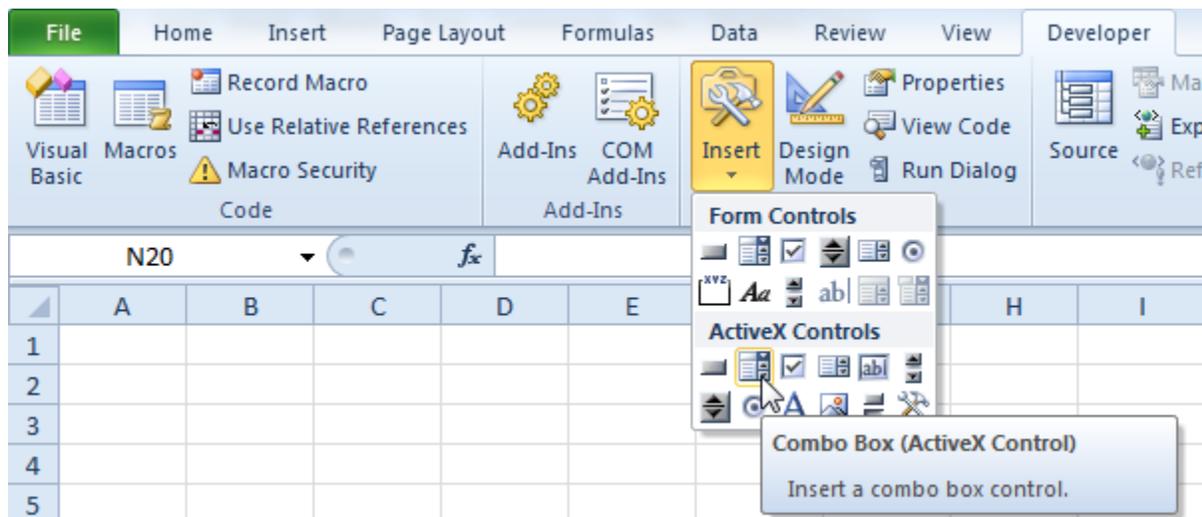
Combobox

[Draw a Combobox](#) | [Add items to Combobox](#)

An Excel VBA combobox is a drop down list from where the user can make a choice. The difference between a listbox and a combobox is that with a combobox the user can also fill in his/her own choice if it is not included in the list.

Draw a Combobox

Excel 2010 and Excel 2007 users. Click on Insert from the Developer tab and then click on Combo Box in the ActiveX Controls section.



Excel 2003 users. Click on Combo Box from the Control ToolBox.

1. Draw a combobox on your worksheet.

	A	B	C	D	E	F
1						
2						
3						
4						
5						

Add items to Combobox

To add some items to the combobox, execute the following steps.

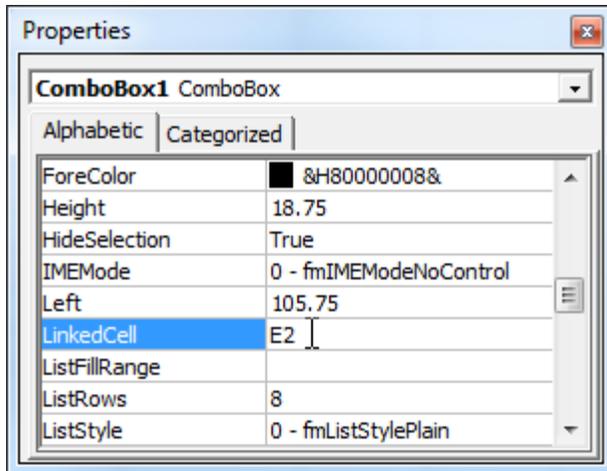
1. Add the following code lines to the [Workbook Open event](#) (or add them to your own code).

```
With Sheet1.ComboBox1
    .AddItem "Paris"
    .AddItem "New York"
    .AddItem "London"
End With
```

Note: use Sheet2 if your combobox is located on the second worksheet.

Result:

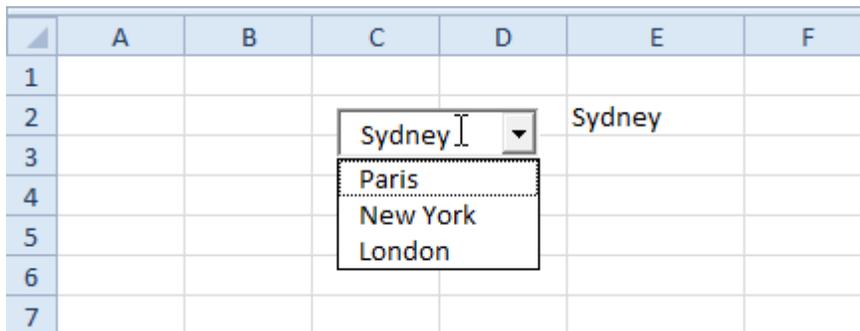
2. To link a cell to the combobox, right click on the combobox (make sure design mode is selected) and click on Properties. Fill in E2 for LinkedCell.



Note: also see the ListFillRange property, to fill a combobox with a range of cells.

3. Close Excel and reopen your Excel file.
4. Choose an item from the combobox or fill in your own choice.

Result:



5. Note: if you use these code lines in your own code, outside the Workbook Open event, add the following code line at the start of your code.

```
ComboBox1.Clear
```

This way your items won't be added multiple times, when you execute your code more than once.

6. The previous code line doesn't clear your own choice. Add the following code line to achieve this.

```
ComboBox1.Value = ""
```

Although in some situations it can be useful to directly place a combobox on your worksheet, a combobox is particularly useful when placed on a [Userform](#).

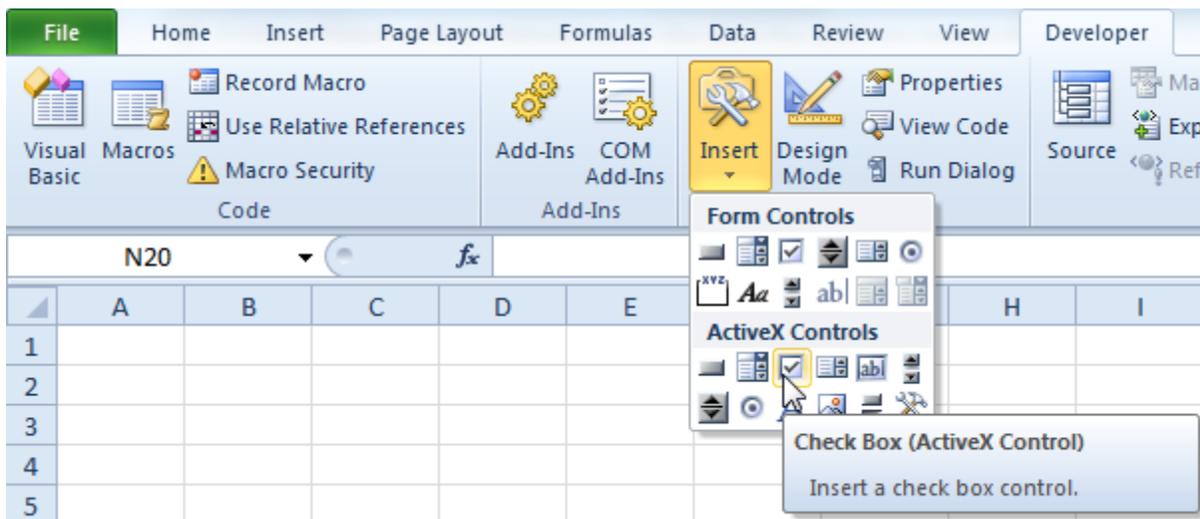
Checkbox

[Draw a Checkbox](#) | [Refer to Checkbox in your Code](#)

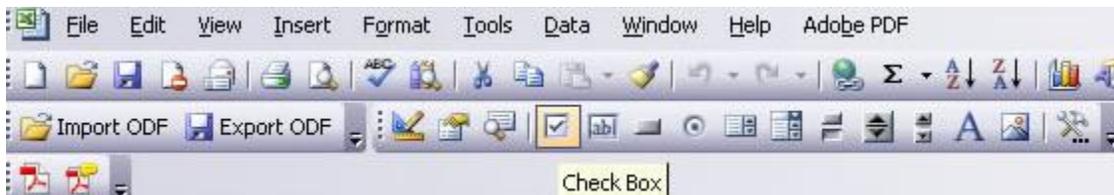
An Excel VBA checkbox is a field which can be checked to store information. Learn how to draw a checkbox on your worksheet and how to refer to a checkbox in your code.

Draw a Checkbox

Excel 2010 and Excel 2007 users. Click on Insert from the Developer tab and then click on Check Box in the ActiveX Controls section.



Excel 2003 users. Click on Check Box from the Control ToolBox.



1. Draw a checkbox on your worksheet.

	A	B	C
1			
2	<input type="checkbox"/> myCheckBox		
3			

2. You can change the caption of the checkbox by right clicking on the checkbox and then clicking on Properties and Caption (make sure Design Mode is selected).

3. You can change the name of the checkbox by right clicking on the checkbox and then clicking on Properties and Name. For now, we will leave Checkbox1 as the name of the checkbox.

Refer to Checkbox in your Code

To refer to a checkbox in your Excel VBA code, execute the following steps:

1. Right click on the checkbox. Click on View Code. Add the following code lines:

```
Private Sub CheckBox1_Click()

If CheckBox1.Value = True Then Range("C2").Value = 1
If CheckBox1.Value = False Then Range("C2").Value = 0

End Sub
```

2. Exit the Visual Basic Editor and check the checkbox.

Result:

	A	B	C
1			
2	<input checked="" type="checkbox"/> myCheckBox		1
3			

	A	B	C
1			
2	<input type="checkbox"/> myCheckBox		0
3			

Although in some situations it can be useful to directly place a checkbox on your worksheet, a checkbox is particularly useful when placed on a [Userform](#).

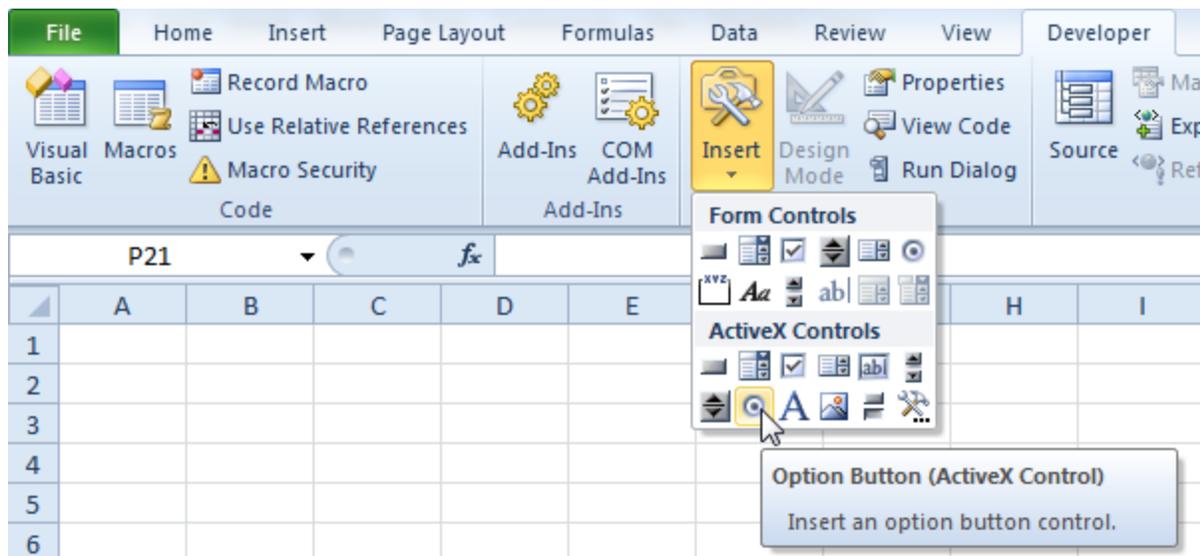
Option Buttons

[Draw an Option Button](#) | [Refer to Option Button in your Code](#)

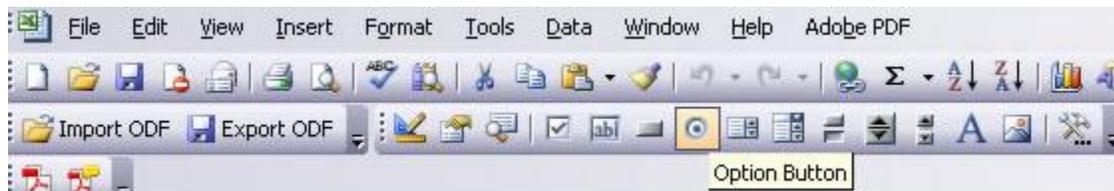
Excel VBA option buttons are the same as checkboxes except that option buttons are dependent on each other while checkboxes are not. This means that when you check one option button the other option button will automatically uncheck.

Draw an Option Button

Excel 2010 and Excel 2007 users. Click on Insert from the Developer tab and then click on Option Button in the ActiveX Controls section.



Excel 2003 users. Click on Option Button from the Control ToolBox.



1. Draw two option buttons on your worksheet.

	A	B	C	D
1				
2		<input checked="" type="radio"/> Female		
3				
4		<input type="radio"/> Male		
5				

2. You can change the captions of the option buttons by right clicking on the option button and then clicking on Properties and Caption (make sure Design Mode is selected).

3. You can change the name of the option button by right clicking on the option button and then clicking on Properties and Name. For now we will leave OptionButton1 and OptionButton2 as the names of the option buttons.

Refer to Option Button in your Code

To refer to a option button in your Excel VBA code, execute the following steps:

1. Right click on the first option button. Click on View Code. Add the following code line:

```
Private Sub OptionButton1_Click()

If OptionButton1.Value = True Then Range("D3").Value = 30

End Sub
```

2. Right click on the second option button. Click on View Code. Add the following code line:

```
Private Sub OptionButton2_Click()

If OptionButton2.Value = True Then Range("D3").Value = 50

End Sub
```

3. Exit the Visual Basic Editor and click on the option buttons.

Result:

	A	B	C	D
1				
2		<input checked="" type="radio"/> Female		
3				30
4		<input type="radio"/> Male		
5				

	A	B	C	D
1				
2		<input type="radio"/> Female		
3				50
4		<input checked="" type="radio"/> Male		
5				

You will see that when you check the 'Female' option button the 'Male' option button will automatically uncheck and vice versa.

Although in some situations it can be useful to directly place option buttons on your worksheet, option buttons are particularly useful when placed on a [Userform](#).

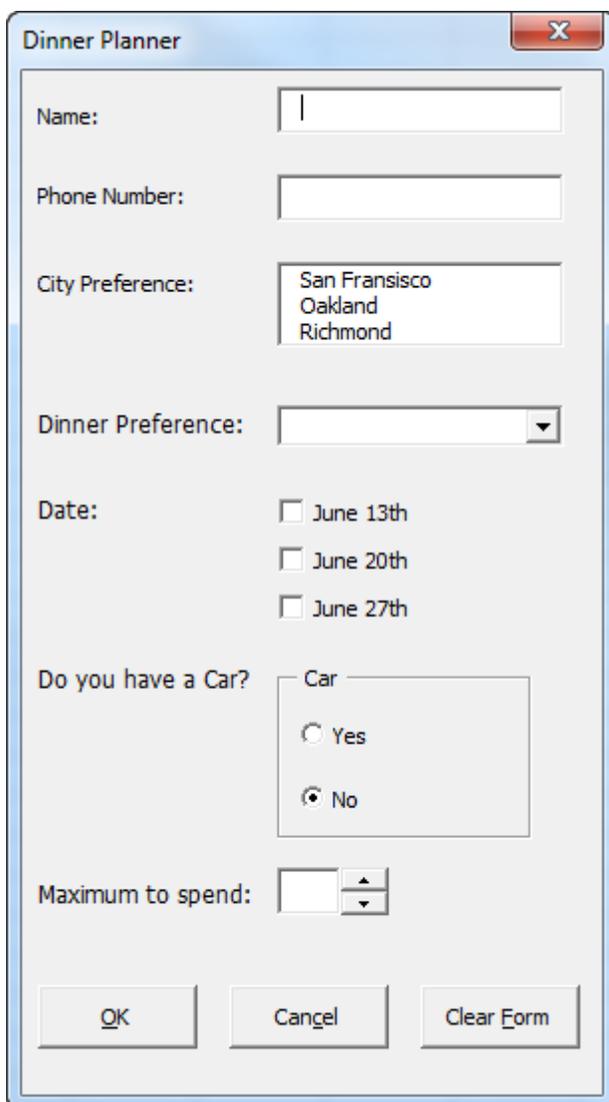
Userform

[Userform Example](#) | [Controls](#) | [Create the Userform](#) | [Show the Userform](#) | [Userform_Initialize](#) | [Assign the Macros](#) | [Test the Userform](#)

This chapter teaches you how to create an Excel VBA Userform (also known as a dialog box). You can download the Userform as well.

Userform Example

The Userform we are going to create looks as follows.



The image shows a VBA Userform titled "Dinner Planner" with a close button (X) in the top right corner. The form contains the following controls:

- Name:** A text input field.
- Phone Number:** A text input field.
- City Preference:** A list box containing "San Fransisco", "Oakland", and "Richmond".
- Dinner Preference:** A dropdown menu.
- Date:** Three radio button options: "June 13th", "June 20th", and "June 27th".
- Do you have a Car?:** A group box containing a "Car" label, a "Yes" radio button, and a "No" radio button (which is selected).
- Maximum to spend:** A spinner control.
- Buttons:** "OK", "Cancel", and "Clear Form" buttons at the bottom.

Controls

The most important Controls that can be added to an Excel VBA Userform are:

Labels

Examples of labels in our Userform are: 'Name:', 'Phone Number:', 'City Preference:', 'Dinner Preference:', etc.

Textboxes

The three boxes next to the Labels 'Name:', 'Phone Number:' and 'Maximum to spend:' are textboxes.

Listboxes

The box next to the Label 'City Preference:' is a listbox.

Comboboxes

The dropdown-list next to the Label 'Dinner Preference:' is a combobox.

Note: a combobox is the same as a listbox except that the user can now also fill in his/her own choice if his/her choice is not included in the list.

Checkboxes and Option buttons

'June 13th', 'June 20th' and 'June 27th' are examples of Checkboxes. 'Yes' and 'No' are examples of option buttons.

Note: Checkboxes and option buttons are primarily the same except that option buttons are dependent on each other while checkboxes are not. This means that when you want to check one option button the other option button will automatically uncheck.

Frames

The field with name 'Car' including the two option buttons is a Frame control.

Note: in order for option buttons to have the dependent functionality described earlier it is best to place the option buttons in a Frame control.

Command buttons

The three command buttons at the bottom of the Userform are examples of command buttons.

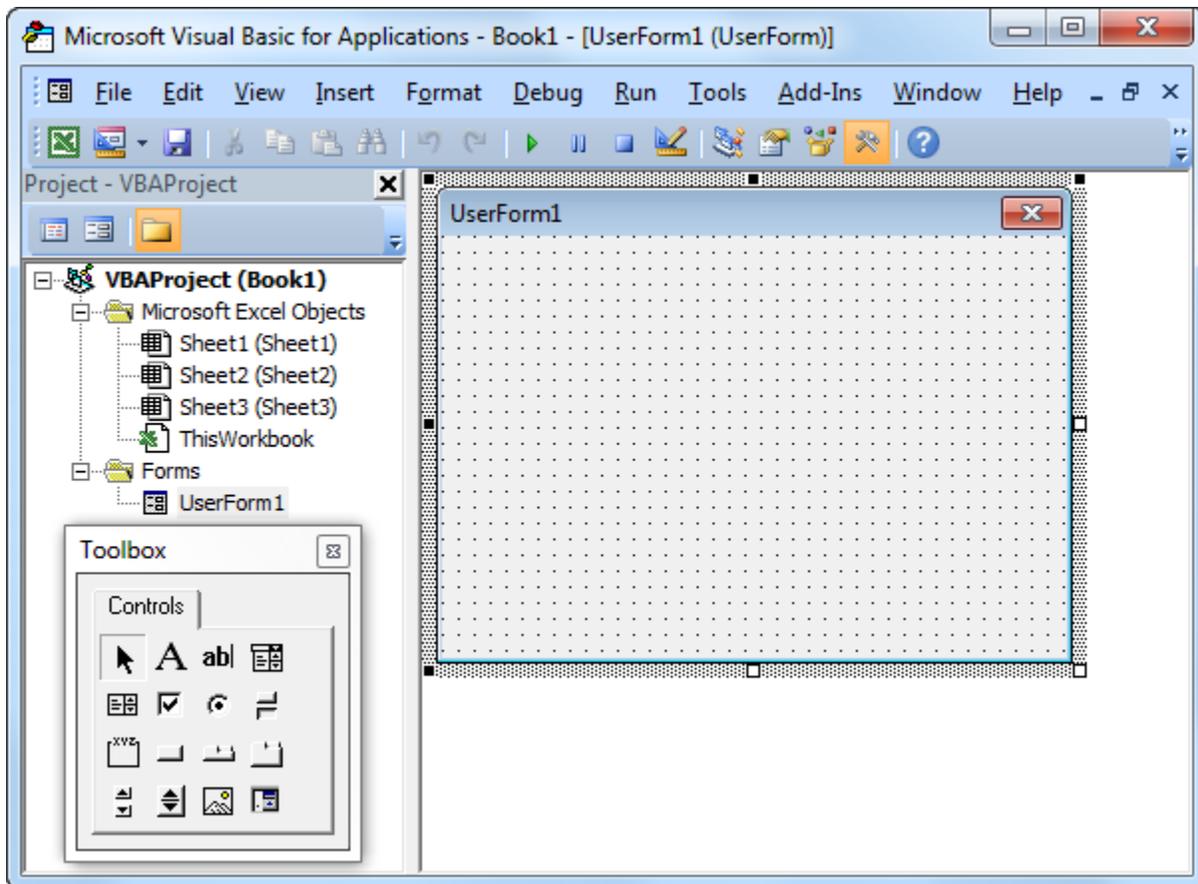
Spin buttons

The spin button is placed next to the textbox at the bottom of the Userform.

Create the Userform

Now it is time to create the Userform!

1. Launch the Visual Basic Editor.
2. Click on ThisWorkbook from the Project Explorer. If the Project Explorer is not visible, click on View and then Project Explorer.
3. From the Menu click on Insert and then Userform. Your screen should be set up as below:



4. If the Toolbox does not appear automatically, click on View and then Toolbox.
5. Add all the controls. Once this has been completed, the result should be consistent with the picture of the Userform shown earlier. For example, create a Label by clicking on Label from the Toolbox. Next, you can drag a Label on the Userform. When you arrive at the CarFrame, remember to draw this Frame first before you place the two option buttons in it.
6. Change the names and the captions of the controls. Right mouse click on each control. Then click on Properties. Names are used in the Excel VBA code. Captions are those that appear on your screen. Change the names and captions of the controls according to the table below.

Control	Name (in VBA)	Caption
Userform	DinnerPlannerUserForm	Dinner Planner
Textbox	NameTextBox	N/A
Textbox	PhoneTextBox	N/A
Listbox	CityListBox	N/A
Combobox	DinnerComboBox	N/A
Checkbox	DateCheckBox1	June 13th
	DateCheckBox2	June 20th
	DateCheckBox3	June 27th
Frame	CarFrame	Car
Option button	CarOptionButton1	Yes
	CarOptionButton2	No
Textbox	MoneyTextBox	N/A
Spin button	MoneySpinButton	N/A
Command button	OKButton	OK
Command button	CancelButton	Cancel
Command button	ClearButton	Clear Form
9 Labels	No need to change	See Picture

Note: it is good practice to change the names of controls. This will make your code easier to read.

Show the Userform

To show the Userform, place a command button on your worksheet and add the following code line:

```
Private Sub CommandButton1_Click()  
  
DinnerPlannerUserForm.Show  
  
End Sub
```

UserForm_Initialize

The Sub UserForm_Initialize runs automatically whenever the Userform is loaded. Thus, when you use the Show method for the Userform, the code will automatically be executed.

```
Private Sub UserForm_Initialize()  
  
'Empty NameTextBox  
NameTextBox.Value = ""  
  
'Empty PhoneTextBox  
PhoneTextBox.Value = ""  
  
'Empty CityListBox  
CityListBox.Clear  
  
'Fill CityListBox  
With CityListBox  
    .AddItem "San Fransisco"  
    .AddItem "Oakland"  
    .AddItem "Richmond"  
End With  
  
'Empty DinnerComboBox  
DinnerComboBox.Clear  
  
'Fill DinnerComboBox  
With DinnerComboBox  
    .AddItem "Italian"  
    .AddItem "Chinese"  
    .AddItem "Frites and Meat"  
End With  
  
'Uncheck DataCheckBoxes  
DateCheckBox1.Value = False  
DateCheckBox2.Value = False  
DateCheckBox3.Value = False  
  
'Set no car as default  
CarOptionButton2.Value = True
```

```
'Empty MoneyTextBox
MoneyTextBox.Value = ""

'Set Focus on NameTextBox
NameTextBox.SetFocus

End Sub
```

Result of the code: Fields are emptied, lists are populated and checkboxes are unchecked. CarOptionButton2 is set as default, assuming that most of the people do not have a car. Finally, the last code line sets the focus on NameTextbox as this is where we want to start when the Userform is loaded. To add this code to the Userform, execute the following steps.

1. Launch the Visual Basic Editor.
2. Click on View and then Code from the Menu or right click on the Userform and then View Code.
3. The code window will appear showing you two drop-down lists. Choose Userform from the left drop-down list. Choose Initialize from the right drop-down list.
4. Add the code.

Assign a macro to the Cancel button

To close the Excel VBA Userform when you click on the Cancel button, execute the following steps.

1. Launch the Visual Basic Editor.
2. Double click on DinnerPlannerUserForm from the Project Explorer.
3. Double click on the Cancel button.
4. Add the following code line:

```
Private Sub CancelButton_Click()

Unload Me

End Sub
```

Assign a macro to the Clear button

To call the Sub UserForm_Initialize when you click on the Clear button, execute the following steps.

1. Launch the Visual Basic Editor.
2. Double click on DinnerPlannerUserForm from the Project Explorer.
3. Double click on the Clear button.
4. Add the following code line:

```
Private Sub ClearButton_Click()  
  
Call UserForm_Initialize  
  
End Sub
```

Assign a macro to the OK button

After clicking on the OK button, the information from the Userform will be placed on your worksheet.

```
Private Sub OKButton_Click()  
  
Dim emptyRow As Long  
  
'Make Sheet1 Active  
Sheets(1).Activate  
  
'Determine emptyRow  
emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1  
  
'Export Data to worksheet  
Cells(emptyRow, 1).Value = NameTextBox.Value  
Cells(emptyRow, 2).Value = PhoneTextBox.Value  
Cells(emptyRow, 3).Value = CityListBox.Value  
Cells(emptyRow, 4).Value = DinnerComboBox.Value  
  
If DateCheckBox1.Value = True Then Cells(emptyRow, 5).Value = DateCheckBox1.Caption  
  
If DateCheckBox2.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value  
& " " & DateCheckBox2.Caption  
  
If DateCheckBox3.Value = True Then Cells(emptyRow, 5).Value = Cells(emptyRow, 5).Value  
& " " & DateCheckBox3.Caption  
  
If CarOptionButton1.Value = True Then  
    Cells(emptyRow, 6).Value = "Yes"  
Else  
    Cells(emptyRow, 6).Value = "No"
```

```

End If

Cells(emptyRow, 7).Value = MoneyTextBox.Value

End Sub

```

Explanation of the code: First, we activate Sheet1. Next, we determine emptyRow. The information from the Userform will be placed in this row. EmptyRow increases every time a record is added. Finally, we take the information from the Userform to the specific columns of emptyRow. Double click on the OK button to add the code just like we did with the Cancel and the Clear button.

Assign a macro to the Money Spin Button

By using the Money Spin Button the user can indicate how much he/she wants to spend. Execute the following steps to program this:

1. Launch the Visual Basic Editor.
2. Double click on DinnerPlannerUserForm from the Project Explorer.
3. Double click on the Money Spin Button.
4. Add the following code line:

```

Private Sub MoneySpinButton_Change ()

MoneyTextBox.Text = MoneySpinButton.Value

End Sub

```

Test the Userform

Exit the Visual Basic Editor, populate row 1, and test the Userform.

Result:

	A	B	C	D	E	F	G
1	Name	Phonenumber	City	Meal	Dates	Car	Maximum to spend
2	Niels	070 540 546	Oakland	Italian	June 13th	No	30
3	Bregje	070 748 847	San Fransisco	Chinese	June 13th June 20th	Yes	40
4							

