# VBA in Excel

## VBA for Excel Lesson 1: The Visual Basic Editor in Excel (VBE)

When you want somebody to do some work for you, you open your Email program and you send him a message in a language that he understands (English, Spanish, French...). When you want Excel to do some work for you, you open the Visual Basic Editor and you write the instructions in a language that Excel understands VBA (**V**isual **B**asic for **A**pplication).

You will develop, test and modify VBA procedures (macros) in the Excel Visual Basic Editor (VBE). It is a very user-friendly development environment. VBA procedures developed in the VBE become part of the workbook in which they are developed and when the workbook is saved the VBA components (including macros, modules, userforms and other components that you will discover in the next 32 lessons) are saved at the same time. So, when you send the workbook to the "Recycling Bin" the VBA procedures (macros) are destroyed.

### Notes

**Special note for users of Excel 2007:** Until the 2007 versions of Excel the user did not need to install anything to work with macros in Excel. If you are using Excel 2007 see how to install the **Visual Basic Editor for Excel from your Office CD**.

**IMPORTANT NOTE 1:** There are **no risks** to your computer or to Excel in completing the task below. At any time if you feel uncomfortable, just close Excel without saving the workbook and try again later.

**For users of Excel 1997 to 2006:** The first thing that you need to do is to make sure that the security level of Excel is set at either "Low" or "Medium" so that you can use the macros (VBA procedures) that you develop. From the menu bar of Excel select "Tools" then "Macro" then "Security" and select "Medium".

**For users of Excel 2007 to 2010:** From the "Developer" ribbon click on the "Macro Security" button. Check the second level "Disable all Macros with Notification" and you are set.

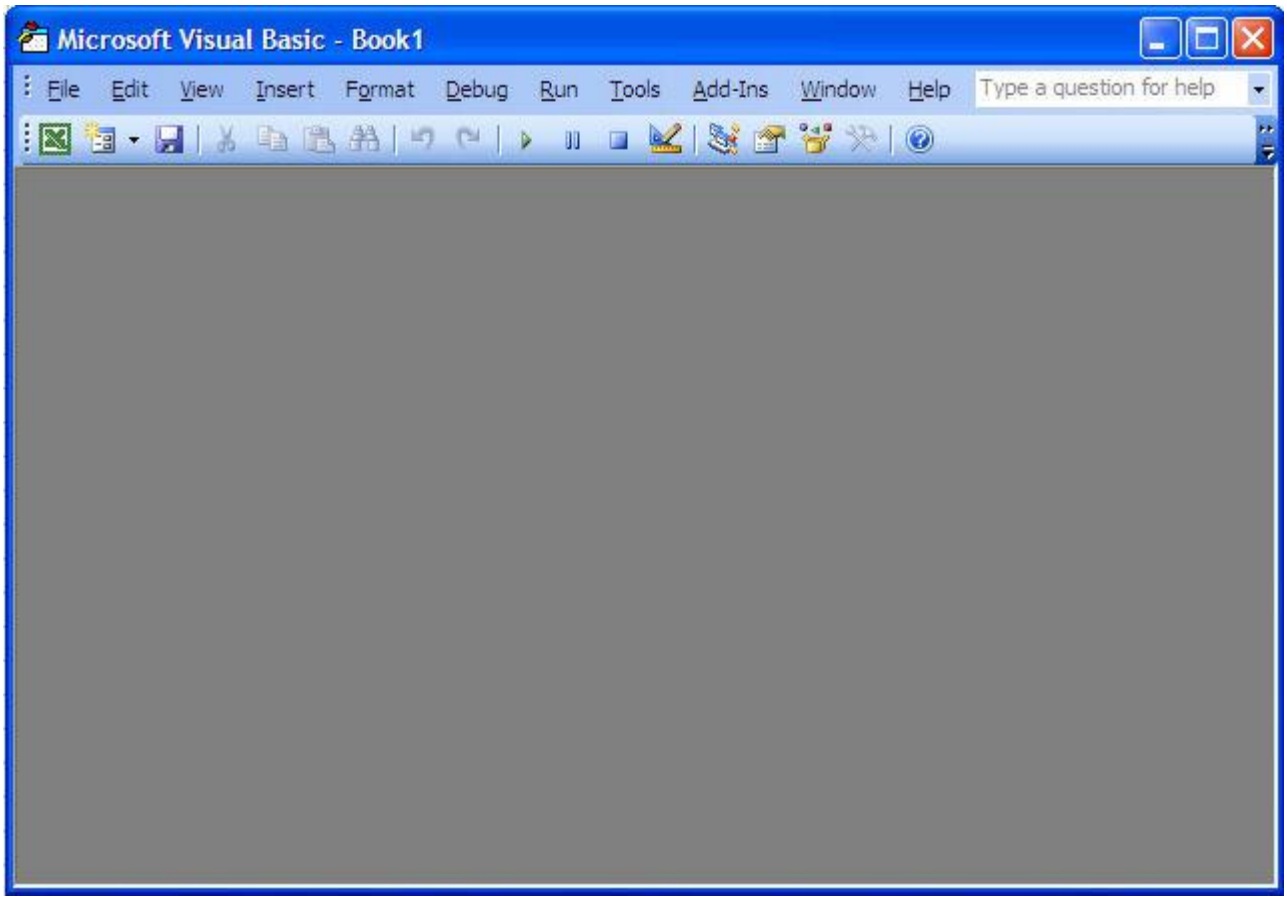
## Setting up the Visual Basic Editor in Excel (VBE)

The Visual Basic Editor is a program within Excel that allows you to communicate with Excel. We will open it and start by setting it up so that working within it becomes easy and efficient.

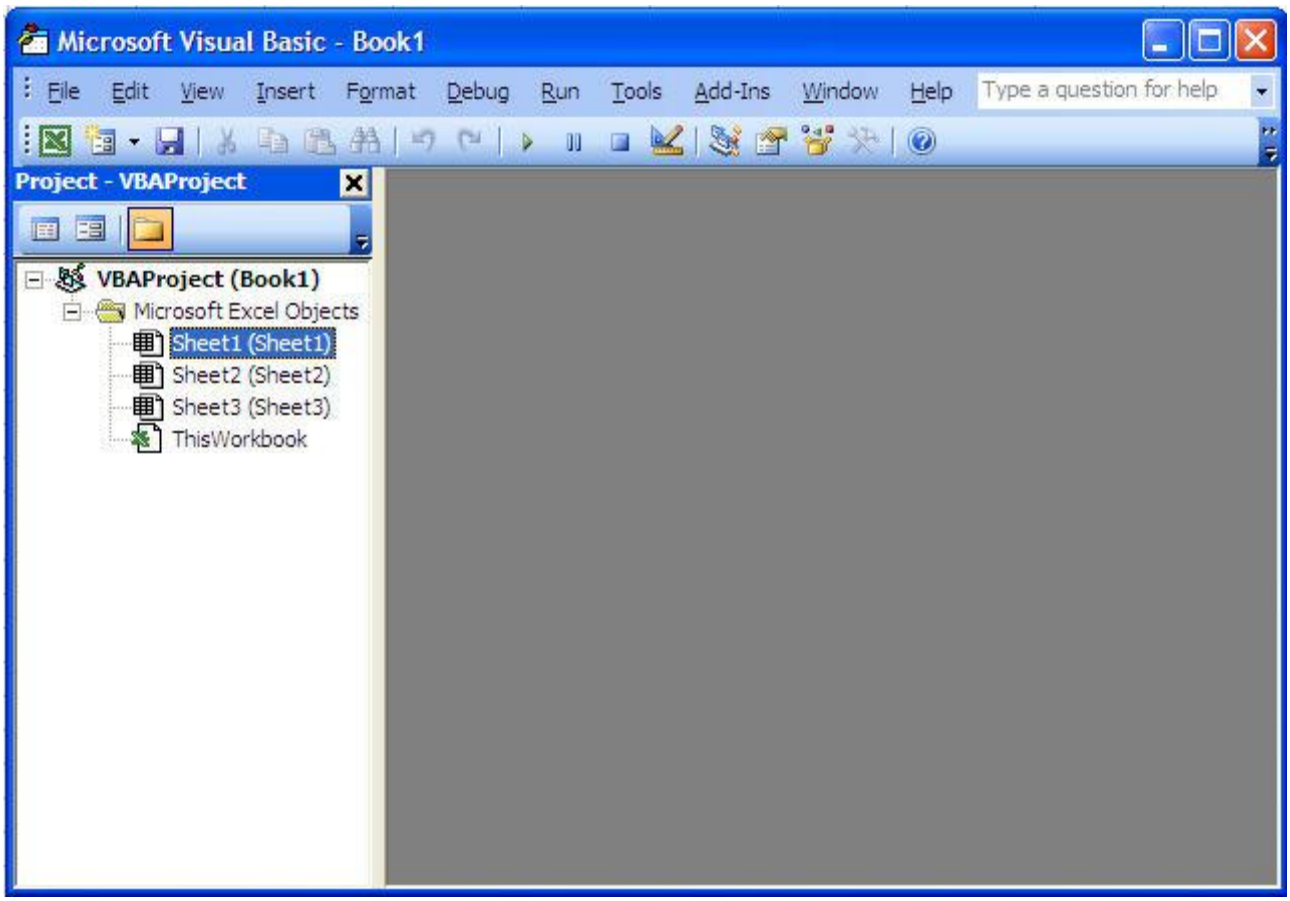
Print this page, open Excel and open a new workbook (Book1).

On your keyboard press the "ALT" key (left of the space bar) and hold, strike the "F11"key (the F11 key is at the top of your keyboard).You now see the Visual Basic Editor. Again press "ALT/F11" and you are back into Excel. Use the "ALT/F11" key to go from Excel to the VBA and back.

When you first open the VBE you will see is a window somewhat like the image below.

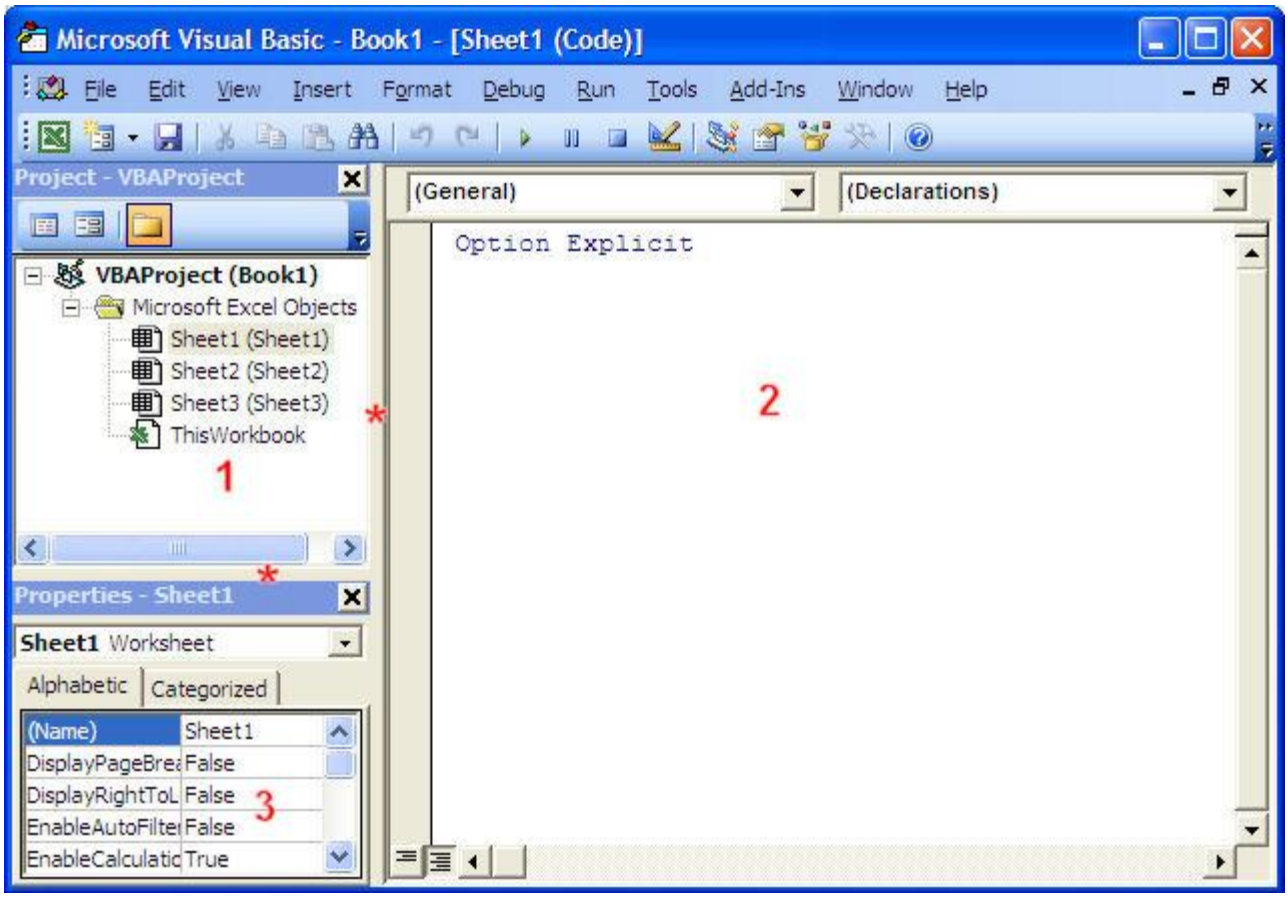



If there are any open windows within the VBE like in the image below click on the Xs to close them and see a gray rectangle filling up the bottom part of the screen like in the image above.




## The Three Windows in the Visual Basic Editor

To be efficient when working with the VBE there should always be 3 windows showing like in the image below; the Project Window (1), the Code Window ( 2), and the Properties Window (3), arranged as in the image below. You can resize the windows by left-clicking where the red stars are, holding and moving sideways or up and down. We will study each of the three windows in lessons 2, 3 and 4 but first we will set them up in the VBE.

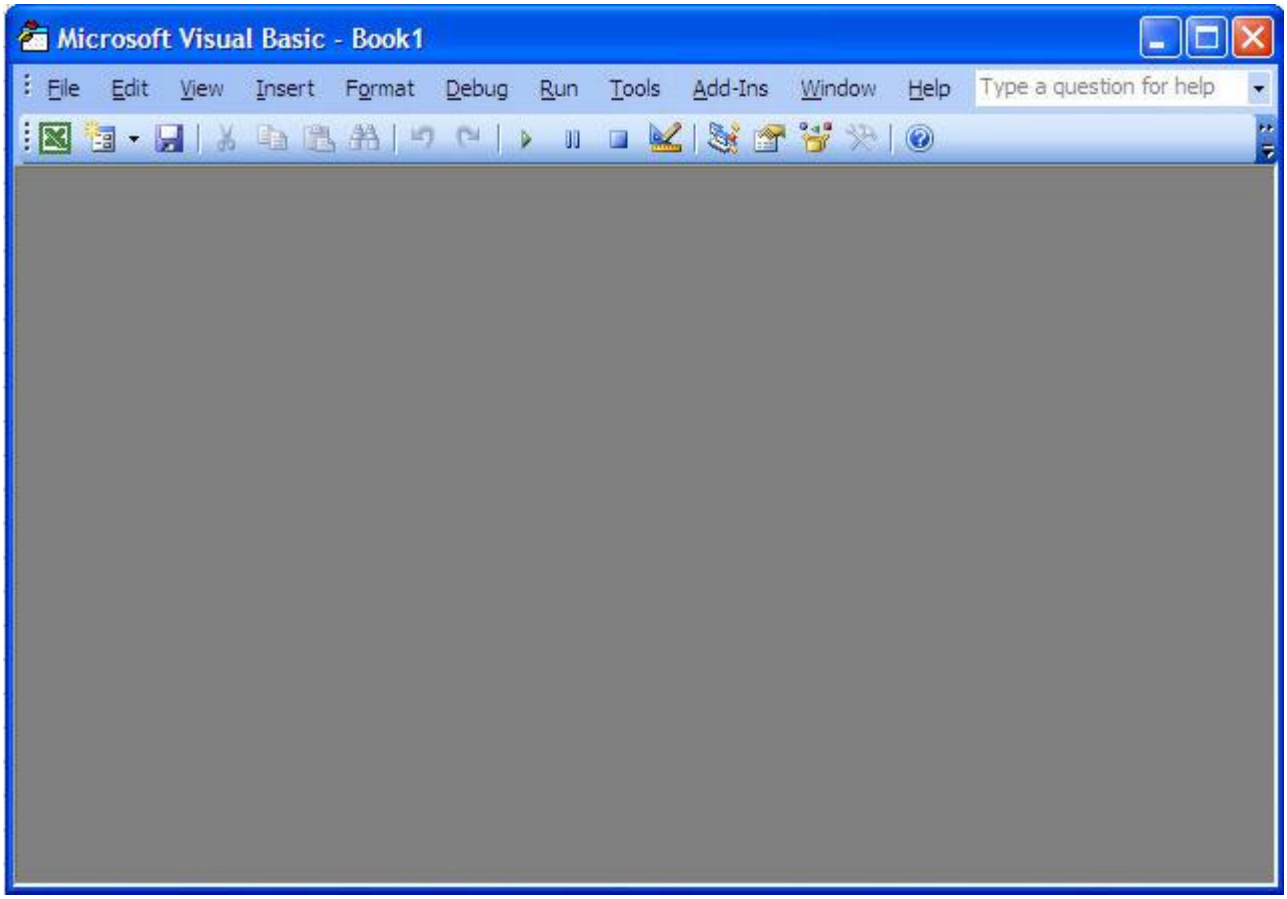



In the exercise below we will setup the 3 windows of the VBE.

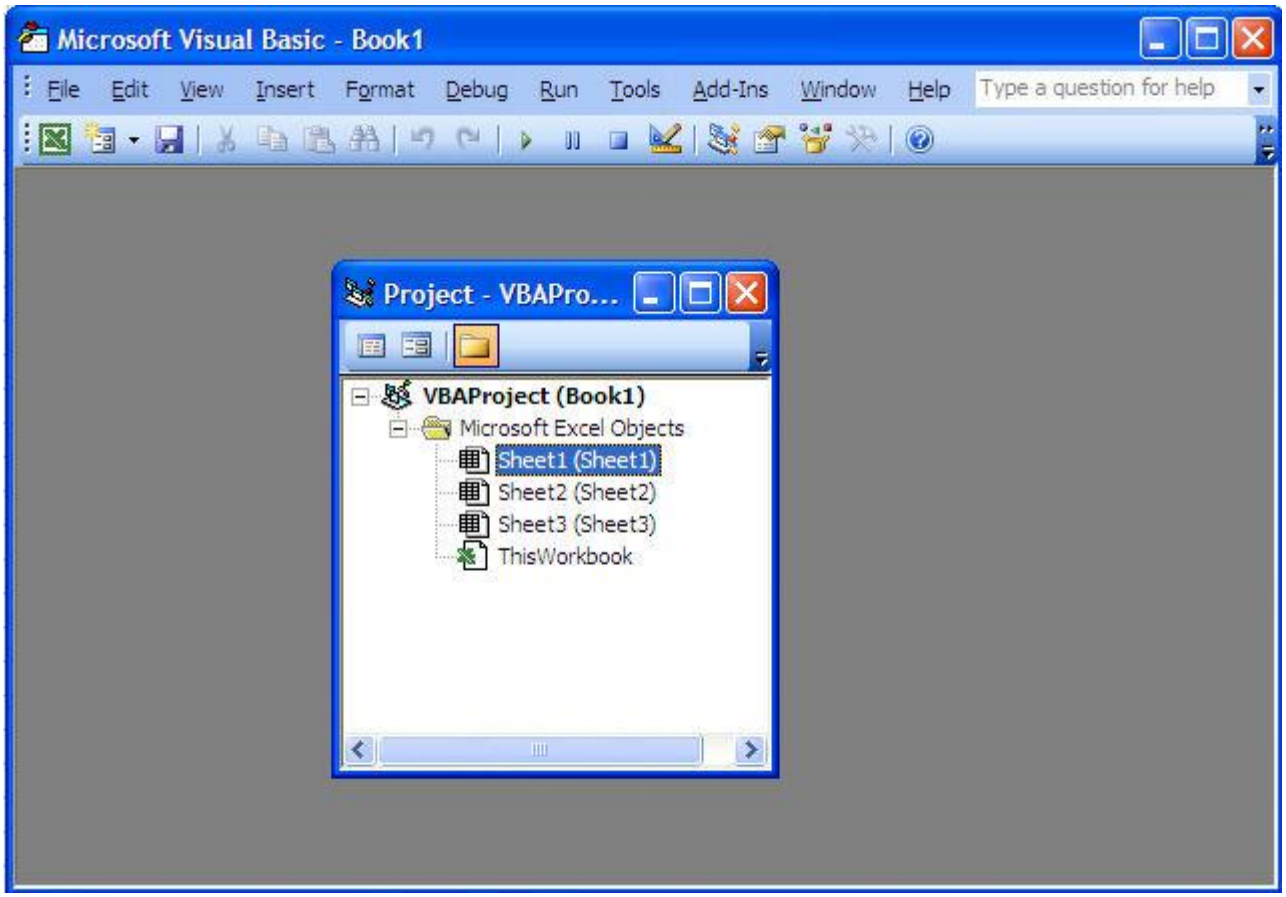
**Exercise 1** (Create your first macro and use it)

Remember that you will perform this task only once as each time you will open the VBE it will remain setup.

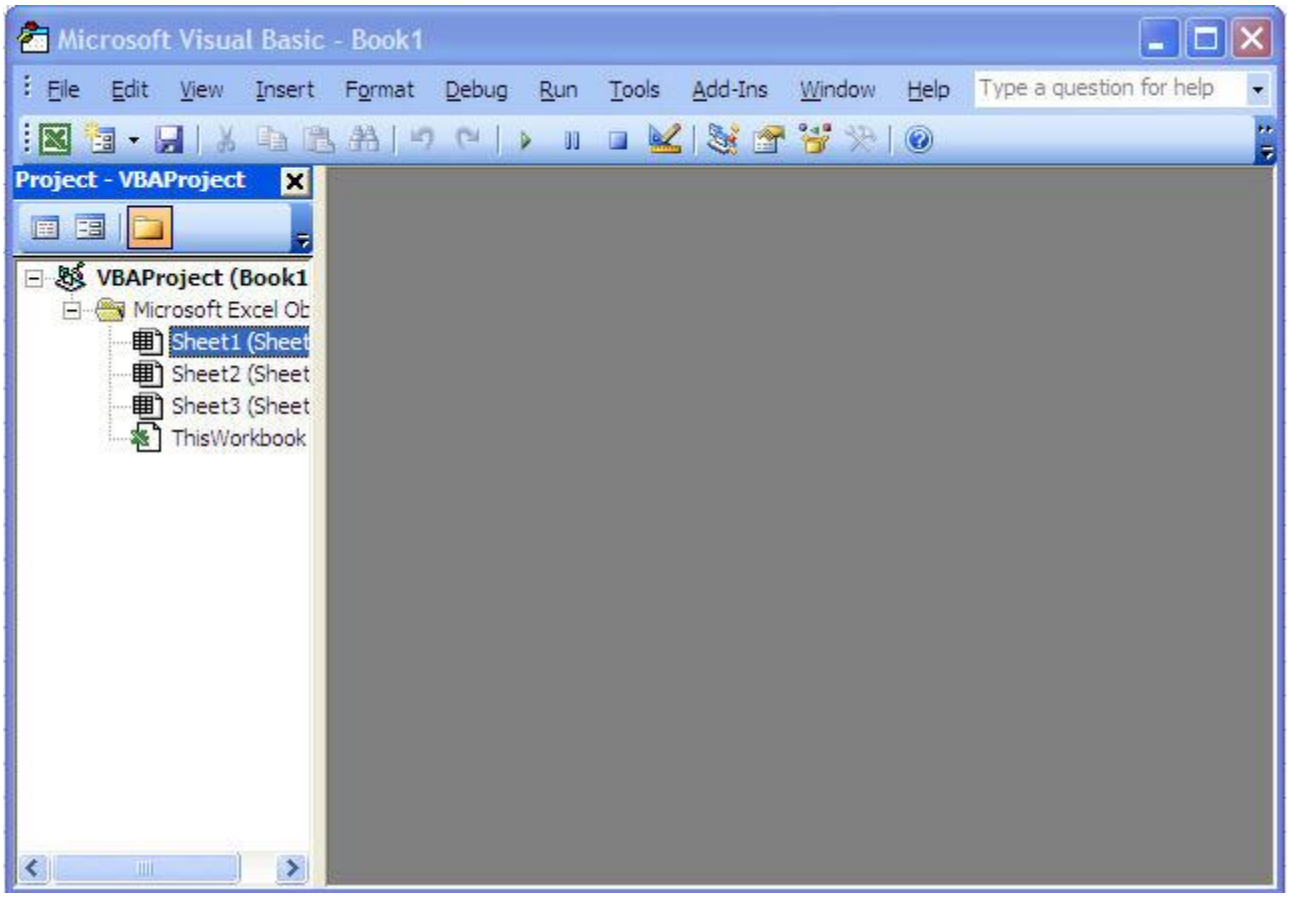
**Step 1:** Close all the windows that are open in the VBE to end up with this:




**Step 2:** Go to the menu bar "View" and click "Project Explorer". The result will be somewhat like the image below:

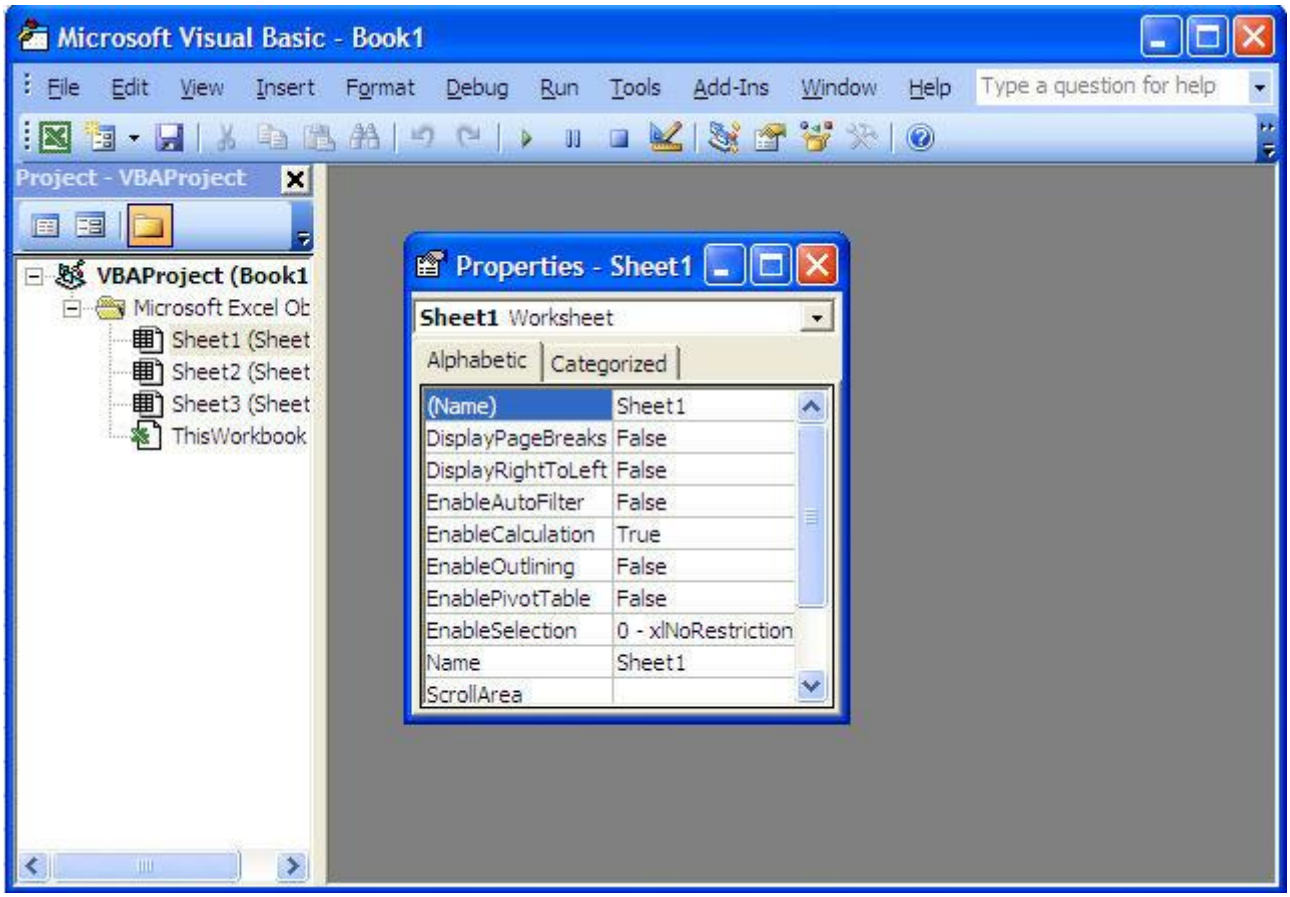



If the project window already appears as a column on the left side of the screen there is nothing else that you have to do for now. If the project window appears in the middle of the gray area like above, right-click in the white space in the middle of the project window and check "Dockable". Then click on the top blue bar of the Project window, hold and drag it left until the cursor (white arrow) touches the middle of the left side of the screen. When you let go of the mouse button the end result should be like shown in the image below. Congratulations you have setup the first major window of the VBE.

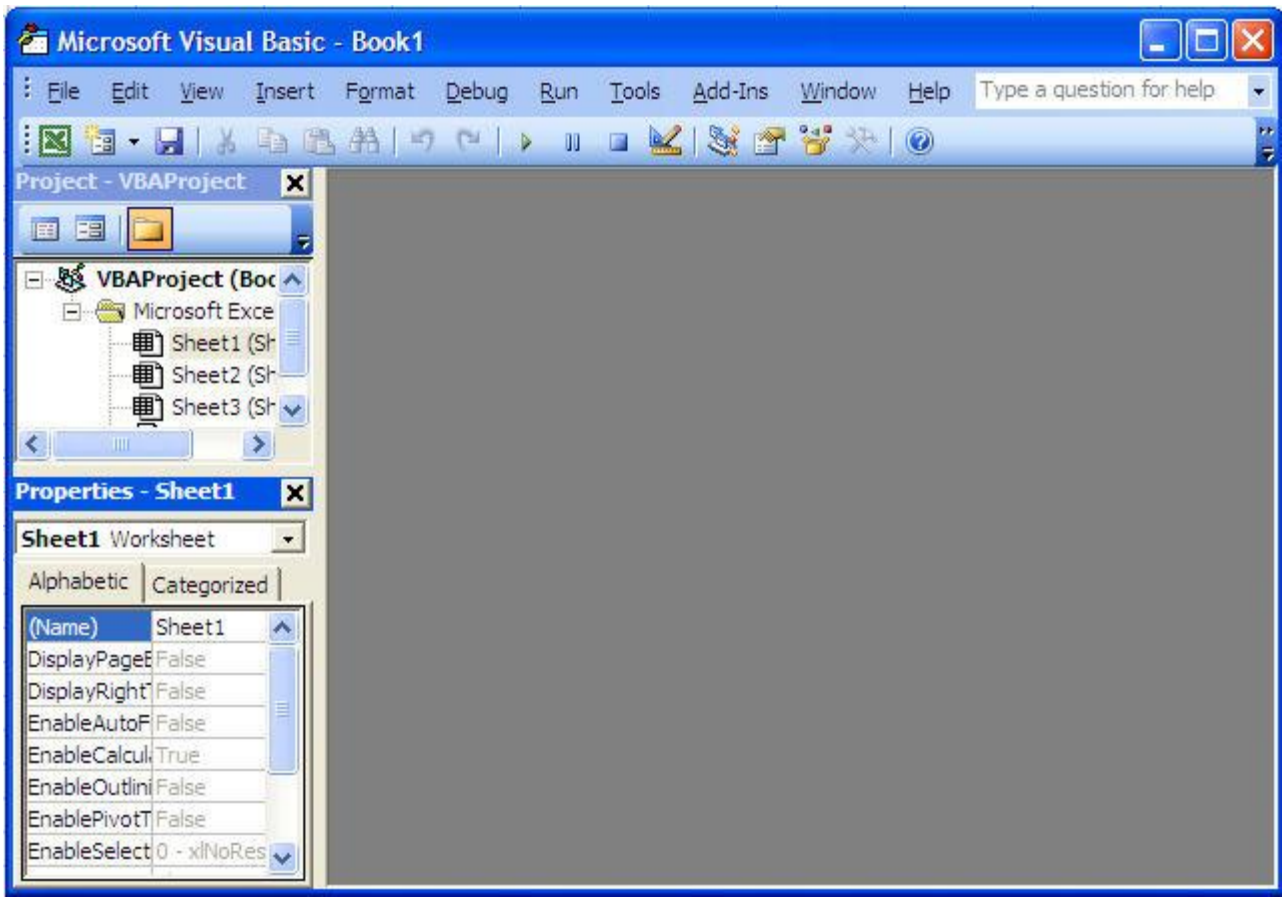



**Step 3:** Move your cursor on the line separating the project window and the gray rectangle. When it turns to two small parallel lines and arrows click, hold and move the lines sideways. Resize the two windows as you want them.

**Step 4:** Go back to the menu bar "View" and click "Properties Window". The Properties window will appear somewhat like in the image below.



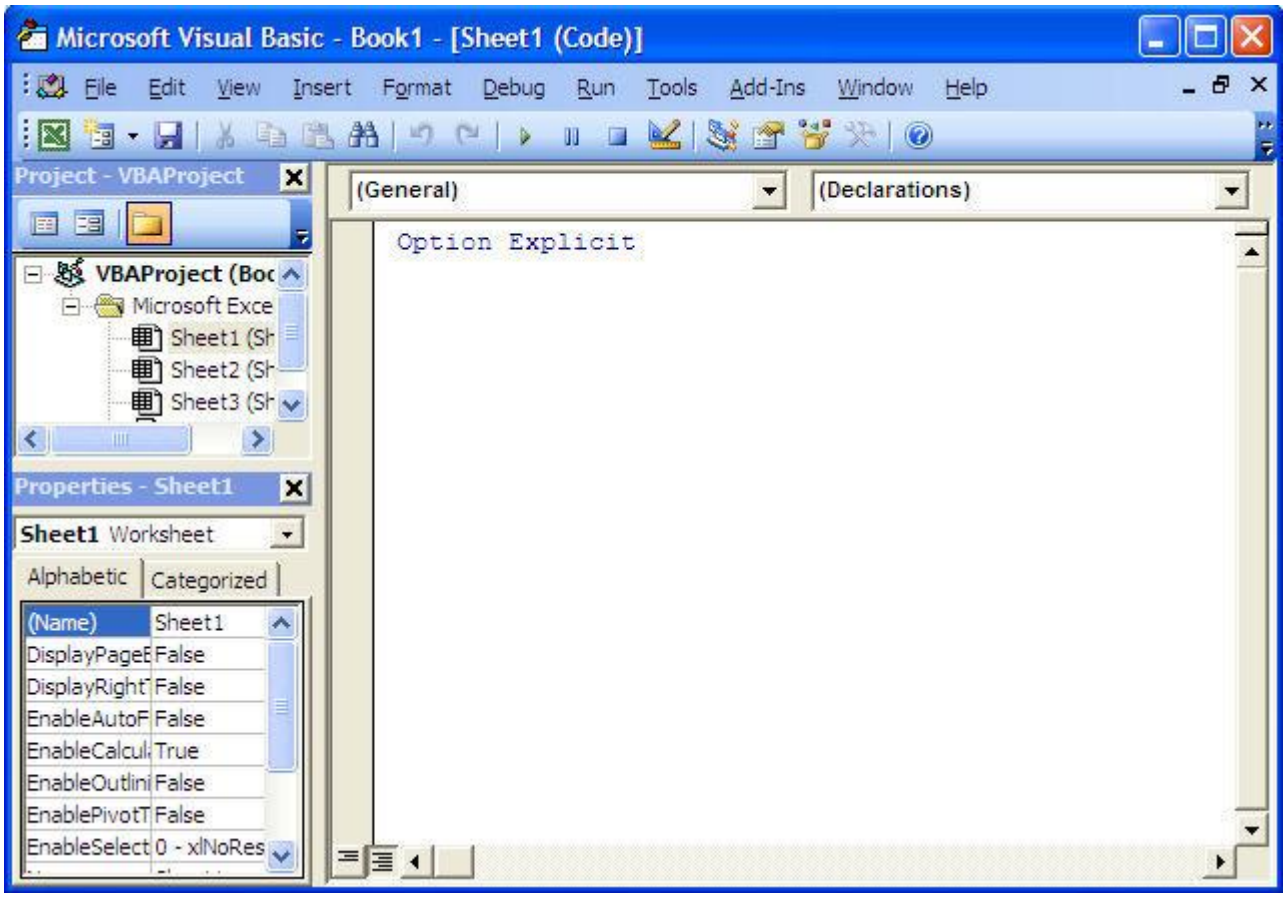

If the Properties window is already located below the Project window there is nothing left to do. If it shows like in the image above, right-click in the white space in the middle of the Properties window and check "Dockable". Then click on the top blue bar of the Properties window and drag it left and down until the cursor (white arrow) touches the center of the bottom of the Project window. When you let go of the mouse button the end result should be as the image below. Congratulations you have setup the second major window of the VBE.




**Step 5:** Move your cursor on the line separating the project window and the properties window. When it turns to two small parallel lines and arrows click, hold and move the lines vertically. Resize the two windows as you want them.

**Step 6:** To add the code window to the setup, you just have to double click on the name of a component in the Project window (Sheet1, Sheet2, Sheet3 or ThisWorkbook) and its code window appears within the gray rectangle. You can maximize any Code window by clicking on its "Maximize" button  .

The final result looks like the image below. The words "Option Explicit" might not be present in your Code window. We will address this issue later in the lesson on variables (Lesson 19). You might also have a VBAProject named FUNCRES.XLA or FUNCRES.XLAM in the project window. Forget about this project for now.




**Step 6:** Now go to Excel and close it. Re-open Excel, go to the VBE (ALT/F11) and you will see that the VBE setup persists. Congratulations, you are now ready to work in the Visual Basic Editor.

We will discover more about each of these three windows in lessons 2 (**Project Window**), 3 (**Properties Window**) and 4 (**Code Windows**).

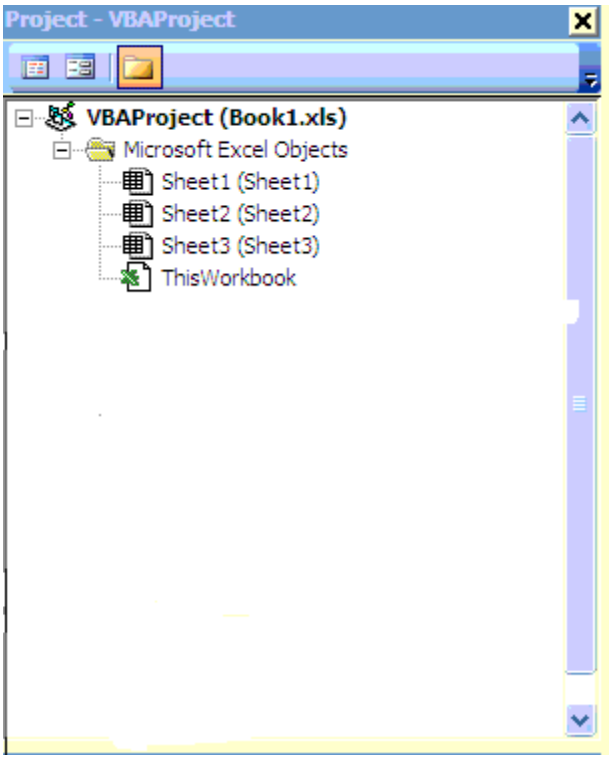
**- Lesson 2: Project  Window in the Visual Basic Editor**

## VBA for Excel Lesson 2: The Project Window in the Visual Basic Editor of Excel

**Note:** Print this page, open Excel and a open a new workbook. Use ALT/F11 to open the Visual Basic Editor as you learned in lesson 1.

As you can see, the Project window shows you all the workbooks that are open ("Book1") in the example below) and their components. You can use the + and - signs to show the details.

A new Excel workbook includes three sheets and another component named "ThisWorkbook". As we will see later in lesson 9 on events "ThisWorkbook" is a component in which you will store the macros ( also called VBA procedures) that should start automatically when the workbook is opened.



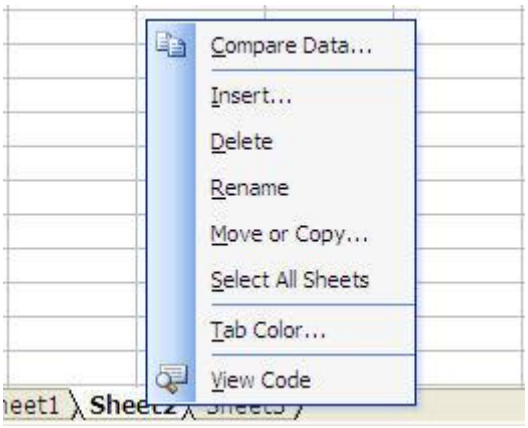

**Working within the Project Window**

We will now complete a brief exercise to learn how easy it is to work within the Project Window.

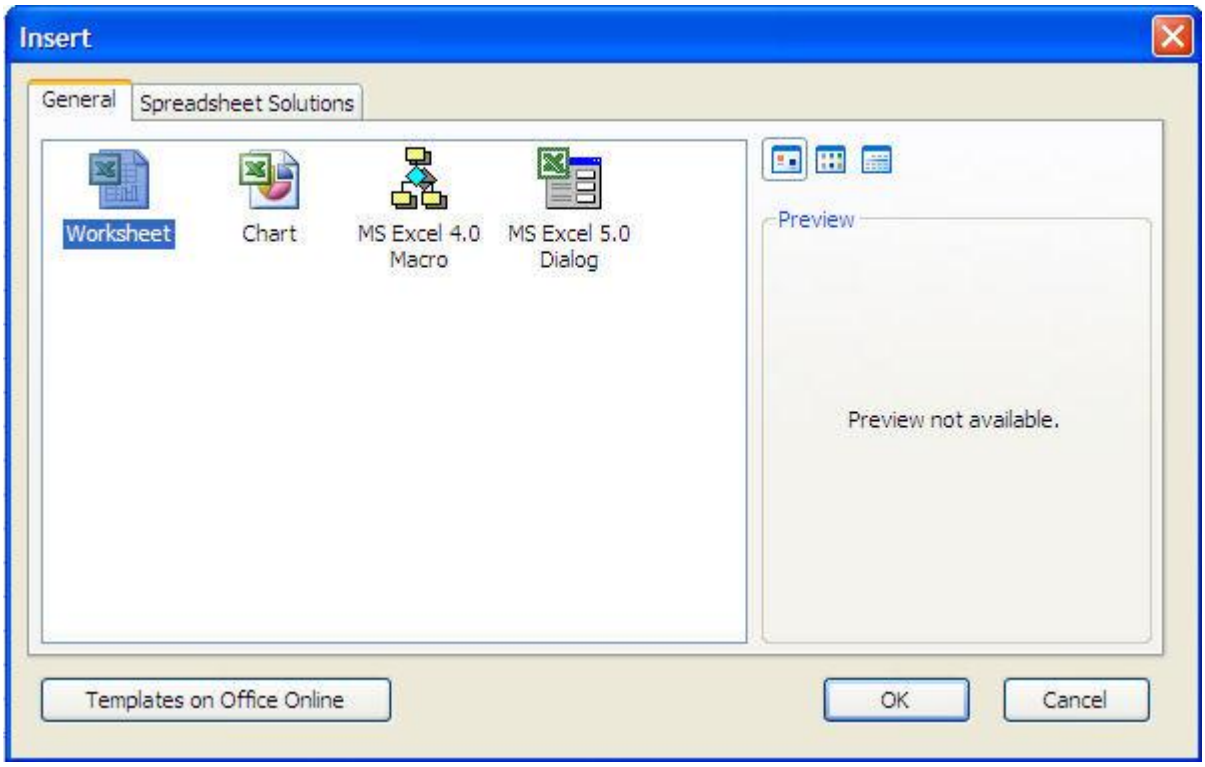
**Exercise 2** (Create your first macro and use it)

Step 1: Using the ALT/F11 key go back to Excel.

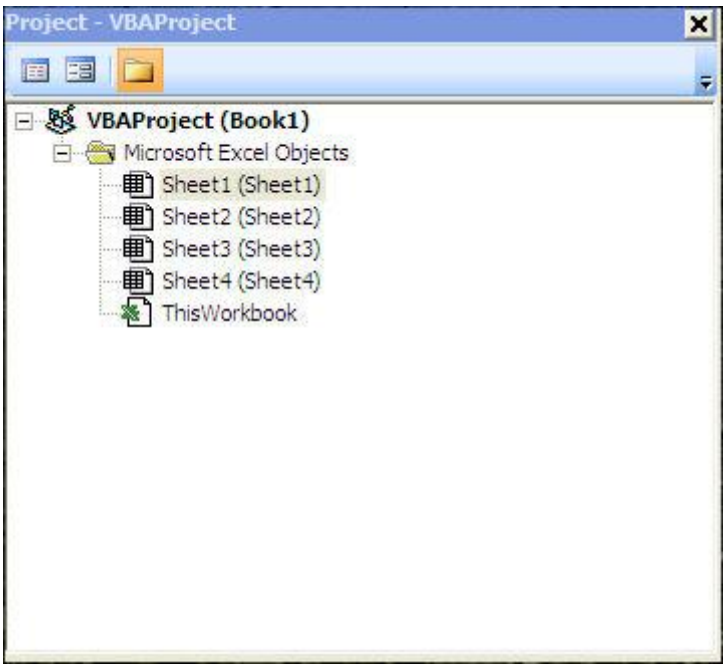
Step 2: Add a sheet. Right-click on the tab of Sheet2 and select "Insert".



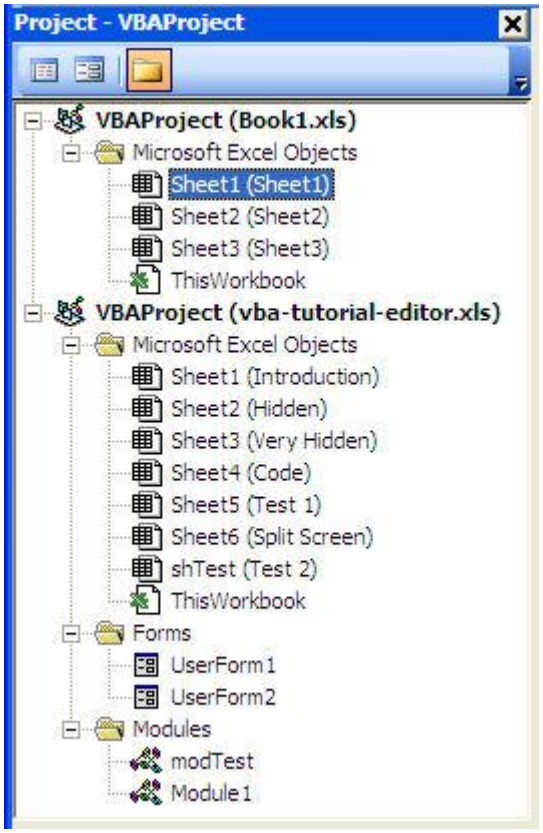

Step 3: In the dialog window that appears, click on "OK".




Step 4: Using the "ALT/F11" key, go back to the Visual Basic Editor and see that a sheet has been added to the workbook. Notice that the worksheets are sorted alphabetically in the Project window even if they are not in the workbook.

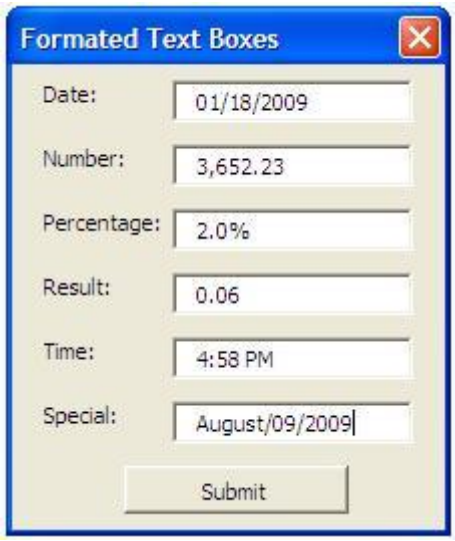



If you have purchased and **downloaded the course on Excel Macros** and opened the Excel file "vba-tutorial-editor.xls" plus a new workbook you will see this:



In the picture above you can see that the VBAProject named "Book1.xls" has 3 sheets and ThisWorkbook. The workbook "vba-tutorial-editor.xls" has 7 sheets, two userforms, two modules plus the "ThisWorkbook" object.

- Userforms are dialog windows (see example image below) that you develop to communicate with the users of your Excel programs and ask them to supply information or make choices.




- Modules are folders in which you save one or many of your macros. You can export and save these modules to be used later in other workbook.

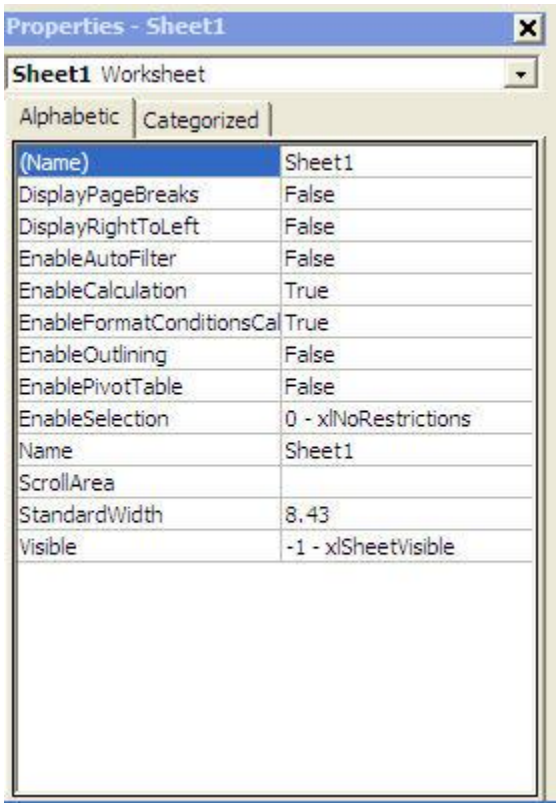
In the complete lesson 2 you will learn how to add any type of components and how to remove, import, export and manage them from the Project window.

**- Lesson 3: Properties  Window in The Visual Basic Editor**

## VBA for Excel Lesson 3:The Properties Window in the Visual Basic Editor of Excel

**Note:** Print this page, open Excel and a open a new workbook. Use ALT/F11 to open the Visual Basic Editor as you learned in lesson 1.

The Properties window shows you the properties of the component that is selected in the Project Window (single click).  For example in the new workbook if you single click on "Sheet1" in the Project Window you see the properties of sheet1 in the Properties Window like in the image below.

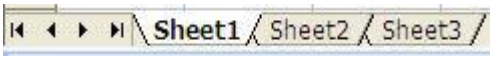



As you can see, a worksheet has 12 properties that you can change in this Properties window. Notice that there are 2 "Name" properties. On the first line there is the programmatical name of the sheet (Sheet1). You will discover later the advantages and disadvantages of changing this property. The second "Name" property (9th line) is the name (or caption) that appears on the tab of the sheet in Excel.

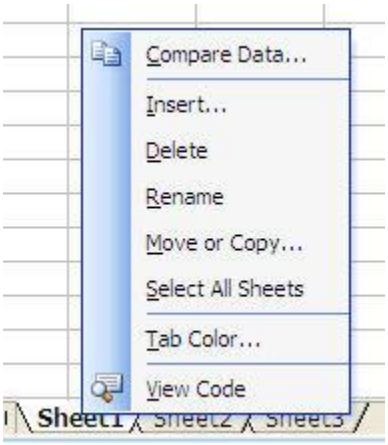
**Changing the "Name" Property**

**Exercise 3** (Create your first macro and use it)

Step 1: Go to Excel (ALT/F11) and notice the names on the three tabs of "Sheet1" as in the image below.

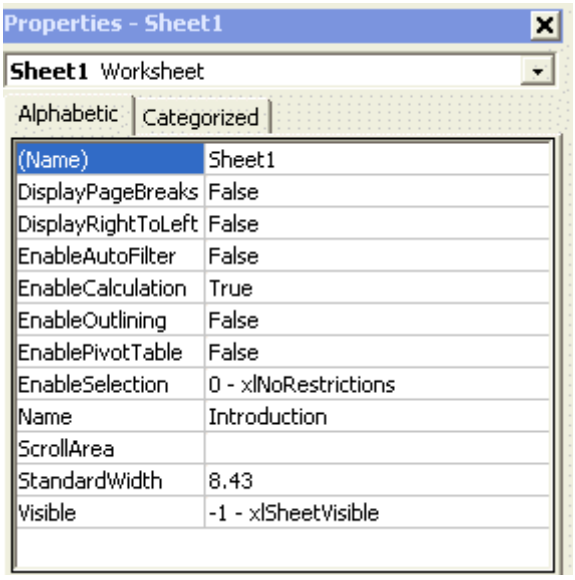



Step 2: We will change the name (Caption) on the tab of "sheet1" to "Introduction". To do so right-click on the tab of the sheet and the following dialog window appears:




Step 3: Select "Rename". The menu disappears and the name of Sheet1 is highlighted. Enter "Introduction" and this new name will replace "Sheet1" when you click "Enter". The end result is illustrated in the image below.


Step 4: Come back to the Visual Basic Editor (ALT/F11) and notice in the Properties window that the property "Name" (the ninth property, the one without the parentheses) has changed to "Introduction

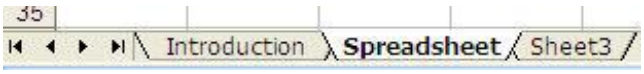



As you have now learned the name of the sheet can be changed from Excel. We will now complete another smal exercise to change the name from the VBE Properties window.

**Exercise 4** (Create your first macro and use it)

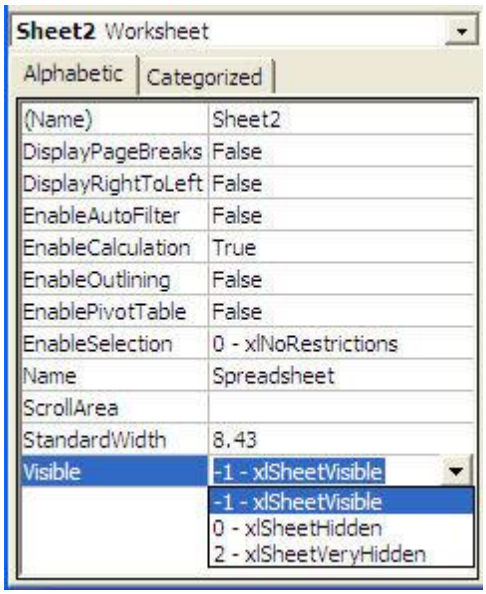
Step 1: In the VBE select "Sheet2" in the Project window. On line 9 of the Properties window double-cllck on "Sheet2" and enter the name Spreadsheet. Click "Enter"

Step 2: Go to Excel and notice that you now have a sheet named "Spreadsheet" .




Setting and modifying properties of objects in the Properties Windows is something that you will have to do a lot when you start developing userforms (see lessons 24 to 33).

Until then you will change a small number of properties including the very important "Visible" property of the sheets to one of its three values. To see the equivalent of the image below, select Sheet2(Spreadsheet) in the Project window. Click on the word "Visible" on the 12th line of the Properties window. A dropdown arrow appears in the cell to the right. Click on the arrow and you can select one of the three properties.




In lesson 3 of the **downloadable the tutorial on VBA for Excel** you will discover how useful the "xlSheetVeryHidden" property can be. This property of a sheet can be used -- for example, to hide salaries in a budgeting application or prices in an estimation application -- making sensitive data inaccessible to the unauthorized users of your workbooks.

**- Lesson 4: Code Window in the Visual Basic Editor**

## VBA for Excel Lesson 4: The Code  Window in the Visual Basic Editor of Excel

**Note:** Print this page, open Excel and a open a new workbook. Use ALT/F11 to navigate from the Visual Basic Editor to Excel as you learned in lesson 1.

The Code Window is where 90% of the VBA work is done; writing VBA sentences, testing your VBA procedures (macros) and modifying them when needed.

To illustrate everything that you can do in the Code window we will start by creating a small macro in an empty workbook.

**Exercise 6** (Create your first macro and use it)

Step 1: In Excel notice that cells A1, A2 and A3 of "Sheet1" are empty. Go to the Visual Basic Editor.
Step 2: Double click on "Sheet1" in the Project Window. On the right is the Code window of "Sheet1"

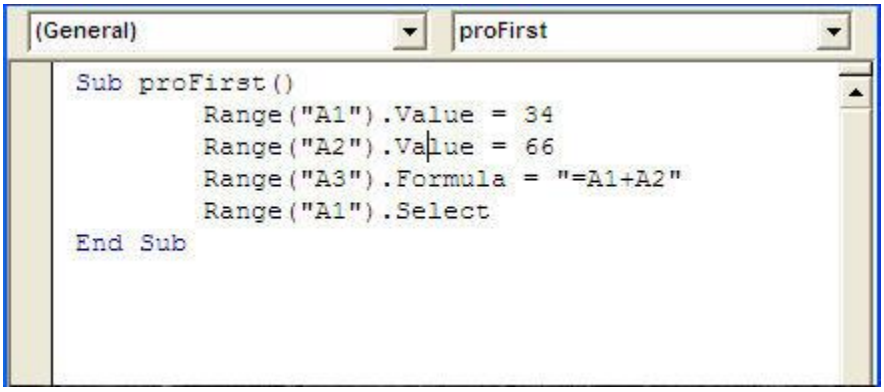
For the purpose of this exercise we will develop a small macro within the code window of a sheet. You will later develop the habit of creating modules and organizing your macros within them.

Step 3: Click anywhere in the Code window

Step 4: You can either copy/paste the following macro from your browser to the code window of "Sheet1" or key it in.

If you decide to key it in you will start by entering the first line and then when you press enter the VBE will add the final Line "End Sub". Enter the rest of the code within the two lines. Make sure that everything is there including all the quotation marks, periods, parentheses, equal signs and spaces.

```
Sub proFirst()
    Range("A1").Value = 34
    Range("A2").Value = 66
    Range("A3").Formula = "=A1+A2"
    Range("A1").Select
End Sub
```




Step 5: Click on any line of the macro, go to the menu bar at the top of the VBE screen and click "Run" then click "Run Sub/Userform".

Step 6: Go to Excel (ALT/F11) and see what has happened to cells A1, A2 and A3

Congratulations you have run and tested you first macro. Go to Excel and "Sheet1" and see that what the macro was ordering Excel to do has been done. The value of cell "A1" is 34, the value of cell "A2" is 66 and there is a formula in cell A3 that sums cells A1 and A2.

Step 7: Go to Excel and clear the cells A1, A2 and A3 of "Sheet1". On the menu bar go to "Tool" and click on "Macros". In the dialog window select "proFirst" and click on run.

You have run the macro from the menu bar of Excel. In lesson 9 on Events you will discover many other ways to start a macro.

**NOTE: You cannot change the font or its color in the code window. You input appears in black, comments appear in green, reserved words in blue and when you make a mistake the font color turns to red.**

**NOTE: For many users of an earlier versions of Excel the wheel of the mouse wheel does not work in the code window. To enable your mouse, download and install the free fix offered in the downloadable tutorial.**

There are plenty of other operations that you can execute in the code window. For example, you can test a macro line by line (step by step), go back a few lines and make corrections, use breakpoints to test only part of a macro.

In section 2 (VBA lessons 11 to 23) you will learn the **VBA vocabulary** to write macros.

**- Lesson 5: Building Macros in Excel**

## VBA for Excel Lesson 5: Developing Macros in Excel

**Note:** Print this page, open Excel and a open a new workbook. Use ALT/F11 to open the Visual Basic Editor as you learned in lesson 1.

Most macros are developed in the code window of modules. For the purpose of this exercise double click on "Sheet1" in the project window

Enter sub proTest() without using a capital "S" as the beginning of "sub". After entering the closing parenthesis click on "Enter". You get these two lines of code:

```
Sub proTest()

End Sub
```

VBE adds the line "End Sub" and capitalizes the "S" of "Sub" . The VBE capitalizes letters appropriately when the word is spelled correctly. This is one interesting feature that you should always use when writing macros. Make it tour habit never to use capital letters when writing code. In this way, whenever VBE unexpected fails to capitalize a letter, you will know that something is wrong.

Two exceptions to your otherwise consistent use of lower-case are: (1), when you declare variables (lesson 19); and (2), when you name macros (as you did above). You will see why in later lessons.

You may now write a procedure within the two lines of code above. For example your VBA procedure could look like this. You can copy/paste the macro below from your browser to the VBE Code window, or key it in. Make sure that everything is there including all the quotation marks and periods, parentheses, equal signs, and spaces.

**Note:** Make sure that you copy/paste this code in a NEW workbook not one created in a previous exercise.

```
Sub proTest()

    Sheets("Sheet1").Select
    Range("C1").Select

    Do Until Selection.Offset(0, -2).Value = ""
        Selection.Value = Selection.Offset(0, -2).Value & " " & Selection.Offset(0, -1)
```

```
        Selection.Offset(1, 0).Select
    Loop

    Range("A1").Select

End Sub
```

The procedure above will go down column "C" and assemble the first names of column "A" and the last names of column "B" with a space in between. It will perform this task all the way down until there are no more first names in column "A" . It will then place the cursor in cell "A1".

To test this macro (VBA procedure) follow the steps below:

Step 1: Go to Excel (ALT/F11) and enter first names in cell A1 to A5.

Step 2: Enter surnames in cells B1 to B5.

| | A | B | C |
|---|---|---|---|
| 1 | Peter | Smith | |
| 2 | John | Ruther | |
| 3 | Mary | Cohen | |
| 4 | Mark | Winthrop | |
| 5 | Susan | Langdon | |

Step 3: Come back to the VBE (ALT/F11) and click within the macro in the code window.

Step 4: From the menu bar select "Run/Run Sub/Userform".

Step 5: Go back to Excel and see the result.

| | A | B | C |
|---|---|---|---|
| 1 | Peter | Smith | Peter Smith |
| 2 | John | Ruther | John Ruther |
| 3 | Mary | Cohen | Mary Cohen |
| 4 | Mark | Winthrop | Mark Winthrop |
| 5 | Susan | Langdon | Susan Langdon |

You can erase everything in column C Excel and retry with more names and surnames.

Try it again removing the first name in cell A3. Notice that the macro stops on line 2.

**- Lesson 6: Testing Macros in Excel**

## VBA for Excel Lesson 6: Testing Macros in the Visual Basic Editor for Excel

### Testing the VBA procedure step by step

**NOTE: While you are running the macro step by step you can stop the execution at any time by clicking on the stop button in the toolbar.** 

Testing is the most time-consuming part of any VBA project. During the development of a project you will use 20% of your time analysing and designing, 15% programming and 65% testing.

During the testing phase, you will correct bugs, typos and the logical errors. More importantly you will improve your original project, fine tune it, discover better ways to do things and add code.

In lesson 4 you have created your first macro and tested it using the "Run" button. You can also test a macro step by step.

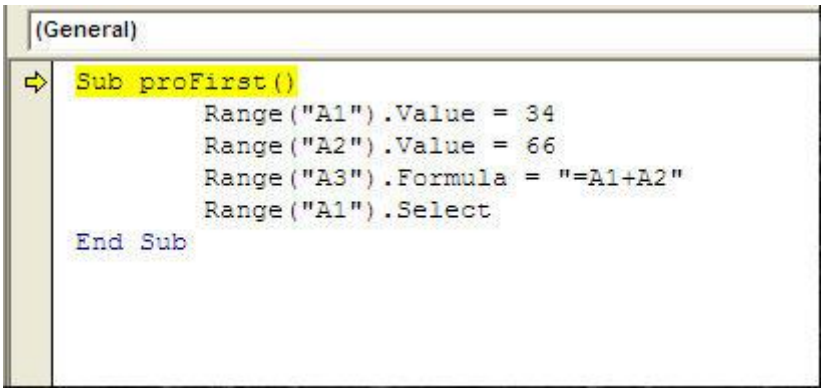
Print this page, open Excel and a open a new workbook. Use ALT/F11 to open the Visual Basic Editor as you learned in lesson 1.

Step 1: Go to Excel and make sure that cells A1, A2 and A3 of Sheet1 are empty.

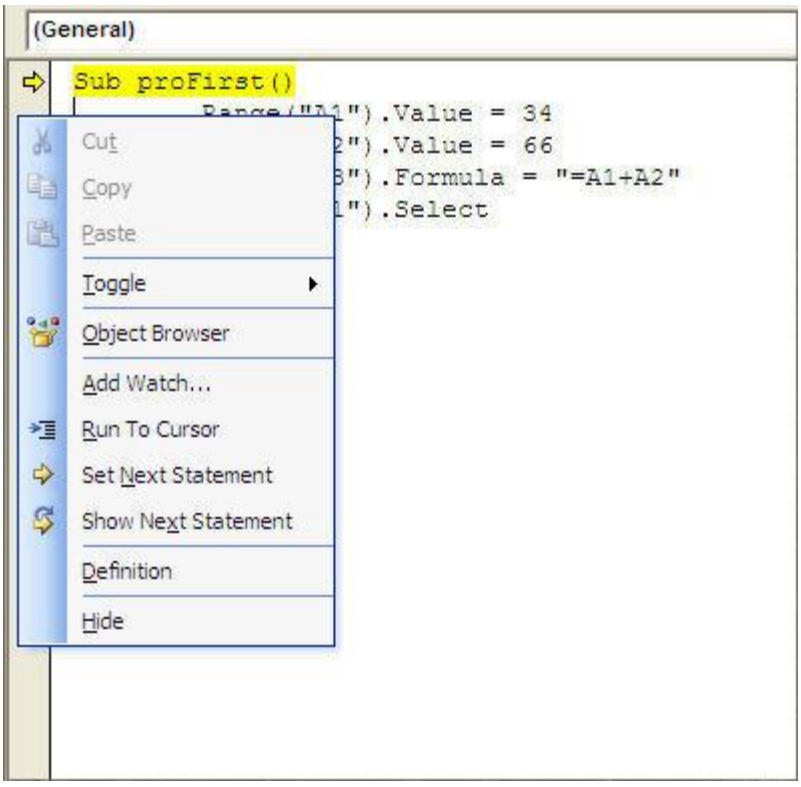
Step 2: In VBE go to the Code window of Sheet1 and copy/paste the following macro:

```
Sub proFirst()
    Range("A1").Value = 34
    Range("A2").Value = 66
    Range("A3").Formula = "=A1+A2"
    Range("A1").Select
End Sub
```

Step 3: Click anywhere within the macro and then press the F8 key at the top of your keyboard. VBE highlights the first line of code in yellow.



Step 4: Right-click on the small yellow arrow and see a menu appear




In lesson 4 of the **downloadable Tutorial on VBA for Excel** you will learn about these precious menu items and everything else that you can do in the Code window. For now let's finish testing this macro step by step.

Step 5: Press on "F8" a second time. No line has been executed yet and if you go to Excel you will see that cells A1 to A3 are still empty. The next time you press "F8" , VBE will execute the yellow-highlighted line.

Step 6: Press "F8" a third time. The yellow-highlighted line is now "Range("A2").Value = 66". VBE has executed the previous line "Range("A1").Value = 34" has been executed so if you go to Excel (ALT/F11) you will see 32 in cell A1.

Step 7: Come back to VBE (ALT/F11) and press "F8" again. Go to Excel and see what happened in cell A2.

Step 8: Come back to VBE (ALT/F11) and press "F8" again. Go to Excel and see that there is a formula in cell A3.

Step 9: Come back to the VBE (ALT/F11) and press "F8" again, cell A1 is now selected in Excel.

Step 10: Press "F8" again. Nothing happens in Excel but "End Sub" is highlighted in yellow

Step 11: Press "F8" again. Nothing happens in Excel no more lines in VBE are highlighted in yellow.

The macro hac been tested, the test is over.

In the code change the addresses A1, A2 and A3 respectively to B1, B2 and B3. Test the macro again. Do it as many times as you want.

**- Lesson 7: Macro Recorder in Excel**

**VBA for Excel Lesson 7: The Macro Recorder in Excel 2007 to 2010**

**Note:** If you are using Excel 1997 to 2006 see **lesson 7 here**

**IMPORTANT NOTE 1:** There are no risks to your computer or to Excel in completing the exercises below. At any time if you feel uncomfortable just close Excel without saving the workbook and retry later.

**IMPORTANT NOTE 2 (for Excel 2007 ONLY) :** You can only complete the exercises below if you have installed VBA for Excel on your computer. If you do not have, **click here**.

One of the tools that makes the programming environment in Excel unique is the **Excel Macro Recorder**. When you start the macro recorder anything you do in Excel is recorded as a new macro. That makes the macro recorder the best VBA teacher and also a great assistant who will write a lot of the words and sentences that you need without a single typo. It will also be there when you do not remember something that you do not use often. Even after many years of programming you will still use the macro recorder daily not to learn anymore but to write code (VBA words and sentences).

**With the Excel macro recorder you can not develop a macro that will damage Excel or your computer so try anything and learn.**

In this lesson on line you will record a macro and run it.

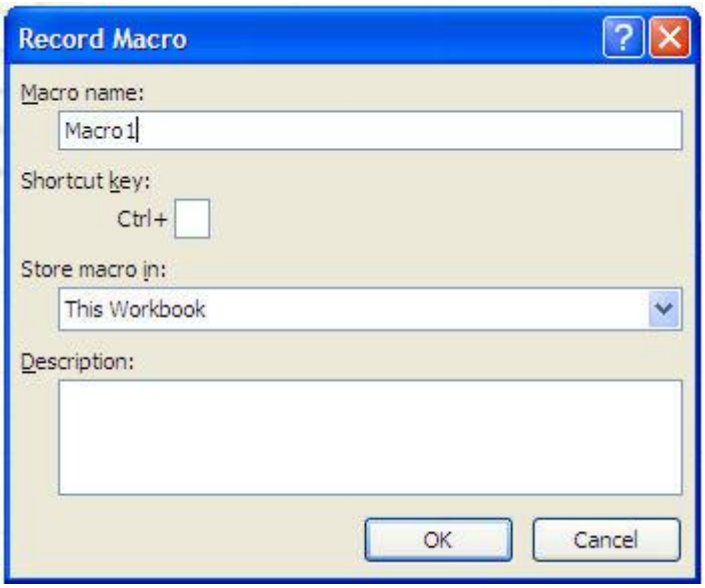
**Recording Your First New Macro:**

**Step 1:** Print this page.

**Step 2:** Open Excel and a new workbook.

**Step 3:** Go to the "Developer" ribbon to click on 


**Step 4:** A small window appears titled "Record Macro". We will review its components in the **downloadable tutorial**. For now just click on "OK".




When you do so the small window disappears and in the "Developer" ribbon 

is replaced by 

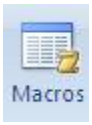
telling you that you are going in the right direction. The macro recorder is ON.

**Step 5:** In the sheet below (Sheet1) select cells B1 to B5, go to "Sheet2", select cell B6, come back to "Sheet1" and select cells D2 to D5.

**Step 6:** In the "Developer" ribbon click on 


**Running your first recorded macro**

**Step 1:** Select cell "A1" of "Sheet1".

**Step 2:** In the "Developer" ribbon click on 


**Step 3:** In the window that appears Macro1 is selected.


Again we will forget about the components of this window because we will study them in the **downloadable tutorial**. For now, just click "Run".

**Step 4:** See how fast the macro runs. You do not even see Excel go to Sheet2 (but it does). At the end of the execution cells D2 to D5 are selected.

What took you around 5 seconds to do manually (step 5 of the first exercise) took Excel a fraction of a second. Excel can work much faster than you can. Welcome to the marvelous world of VBA for Excel (Macros).

You can repeat steps 1 to 4 of this second exercise as often as you like.

**Looking at your first recorded macro**

To complete this third exercise you must have studied lessons 1 to 4.

Go to the Visual Basic editor and you will see the following macro in the code window when you double click on Module 1 in the Project Window:

```
Sub Macro1()

'

' Macro1 Macro
```

Rim

'

```
    Range("B1:B5").Select
    Sheets("Sheet2").Select
    Range("B6").Select
    Sheets("Sheet1").Select
    Range("D2:D5").Select

End Sub
```

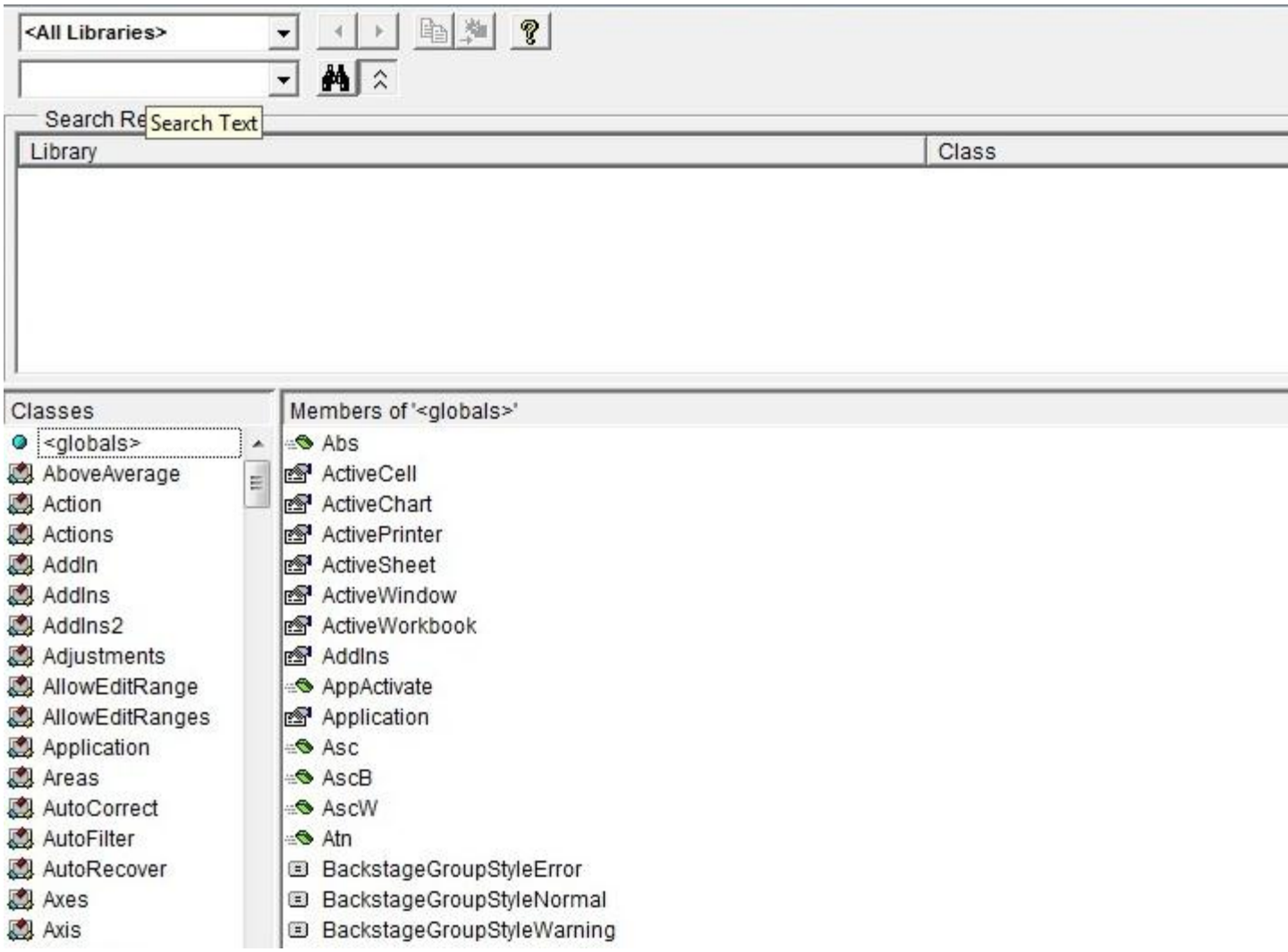
As you can see the macro recorder recorded your instructions in a language that Excel understands (VBA). You can now use VBA's written code to have Excel perform this task.

**- Lesson 8: Macros Help and Assistance**

### VBA for Excel Lesson 8: Macros Help and Assistance

There is plenty of help and assistance available within Excel when you develop macros. As you have discovered in the previous lesson the Macro Recorder is a great teacher and assistant. In this lesson we investigate the two other sources of assistance within the Visual Basic Editor of Excel: the Help Files and the Object Browser.

Here is how the Object Browser appers when you call it. ALL the VBA words are presented in this tool including useful examples. The search function is powerful.




**- Lesson 9: [Events in VBA for Excel]**

## VBA for Excel Lesson 9: Starting, Triggering a Macro in Excel 2007 to 2011 (The Events)

**Note 1:** If you are using Excel 2007 see **[lesson 9 here]**

**Note 2:** Print this page, open Excel and a open a new workbook. Use ALT/F11 to open the Visual Basic Editor as you learned in lesson 1.

When does the VBA procedure (macro) start? When an EVENT happens. The event is what triggers the VBA Excel procedure. In earlier lessons you have used an event to start your macros. In the Visual Basic Editor you have gone to the menu bar and clicked

on "Run/Run Sub/Userform" and the macro was executed. You have also clicked on the F8 key at the top of your keyboard and the macro got executed line by line.

You do not want your user to go to the Visual Basic Editor to trigger a macro. A lot of other events can happen to start a macro. The event that is mostly (85%) of macros used is clicking on a button. The button can be on the worksheet or on a userform that you would develop. The event can also be: opening the workbook, selecting a sheet, the value of a cell changing due to a manual input or due to the recalculation of a formula, clicking on a selected keystroke or going to the right menu item in Excel.

**Preparing the Exercise on Events**

To complete the following exercises, copy paste the code below from your browser to the code window of "Sheet1" of the new Excel workbook as you have learned in previous lessons.

```
Sub proFirst()

    Range("A1" ).Value = 34
    Range("A2" ).Value = 66
    Range("A3" ).Formula = "=A1+A2"

    Range("A1" ).Select

End Sub
```

**Macros Triggered from the Developer Ribbon**

Step 1: Select "Macros" from the "Developer" ribbon. You will see the "Macro" dialog window below.


Step 2: "Sheet1.proFirst" being selected in the list box and its name appearing in the text box above the list box just click "Run". The macro is automatically executed

Step 3: Erase the contents of cells A1, A2 and A3

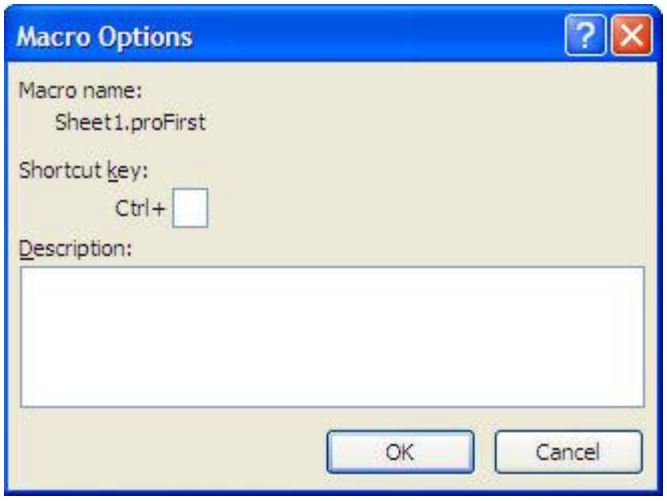
You now see that colleagues must have installed VBA on their own computer to be able to use your macros from the "Developer" ribbon.

**Macros Triggered by a Keystroke**

In this second first exercise on events we will get the macro to be keyboard activated by capital "s" (Shift/S). First you need to program a key. To do so:

Step 1: Select "Macros" from the "Developer" ribbon. You will see the "Macro" dialog window below.


Step 2: "Sheet1.proFirst" being selected in the list box and its name appearing in the text box above the list box just click on "Options". A new dialog window "Macro Options" appears:




Step 3: In the shortcut key text box enter a capital "s" "SHIFT/s" and then click "OK". Click "Cancel" in the dialog window

Step 4: If you now click "CTRL/SHIFT/S" the macro will be executed instantly.

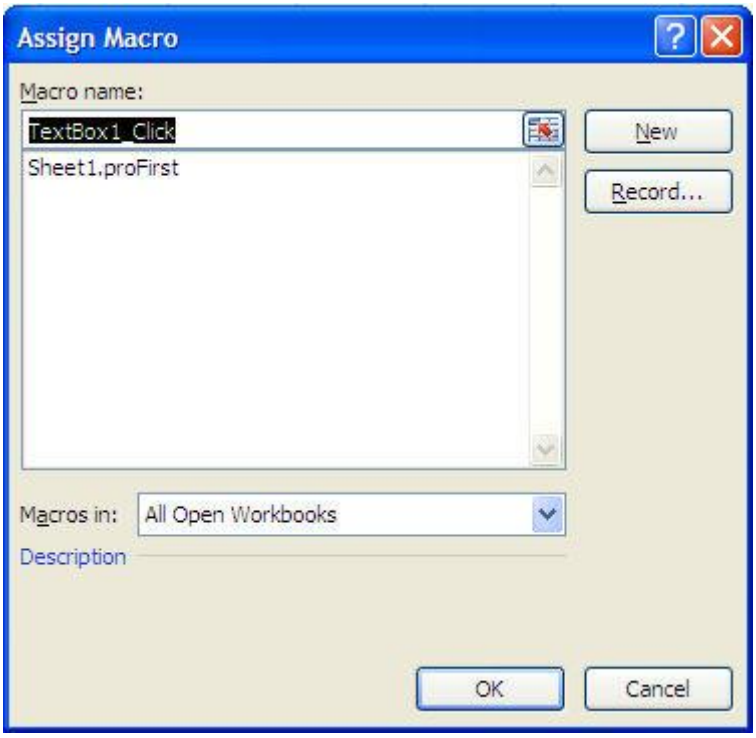
**Macros Triggered by Clicking on a Text Box on the Worksheet**

More than 90% of the macros are  triggered by a click on a button located on a worksheet.

We prefer using text boxes rather than VBA command buttons because they are much easier to maintain and allow much more creativity in the design. You can use the font that you like and the background color that fits your needs. If you are a little creative you can add 3D effects, special borders and others.

Step 1: From the "Insert ribbon" click on the "Text Box" icon once. Lower the curser toward the sheet, click and hold the left button of the mouse and stretch the text box to the desired dimension.

Step 2: Right click on the text box, select "Assign Macro" from the menu and the "Assign Macro" dialog window appears:




Step 3: Select "Sheet1.proFirst" from the list box and its name appears in the text box above the list box just click on "OK".

Step 3: Click away from the text box on the Excel sheet.

Step 4: Left click on the text box and the macro is executed.

You can assign macros to text boxes, images or WordArt using the same approach.

**- Lesson 10: Security and Protection In VBA for Excel**

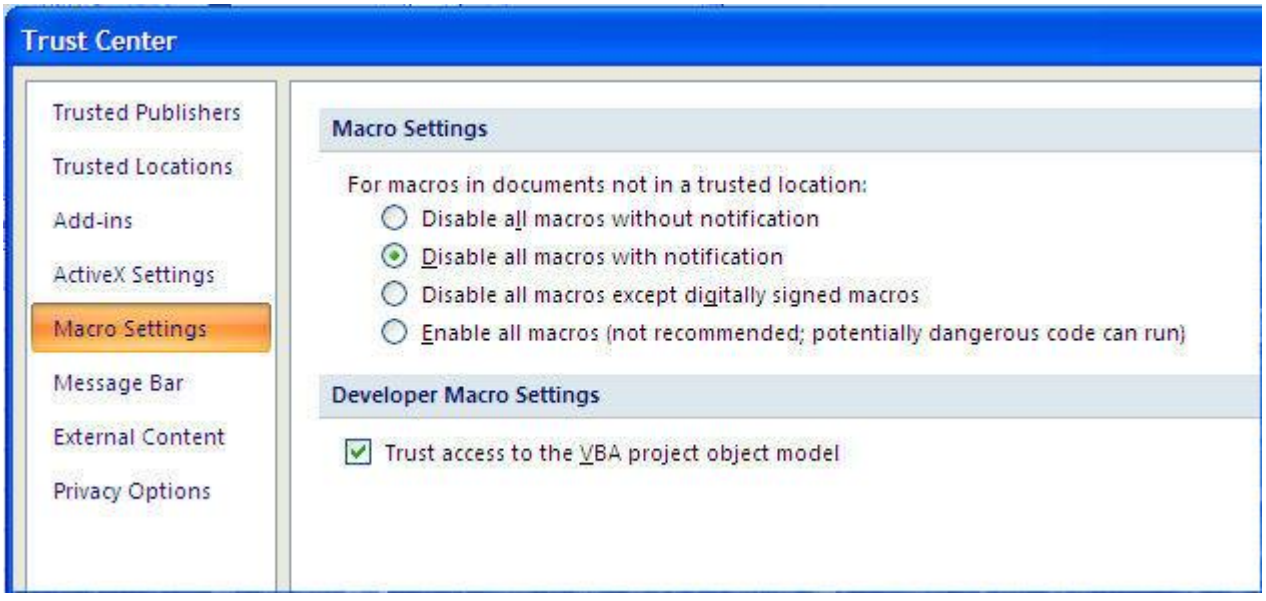
## VBA for Excel Lesson 10: VBA Macros Security and Protection in Excel (Excel 2007 and 2010)

**Note 1:** You will change the security setting one single time. You will not have to do it again. Tell your colleagues about it specially if you want to send them Excel workbooks with macros. The setting suggested here is totally safe and you will not make your computer vulnerable to any virus.

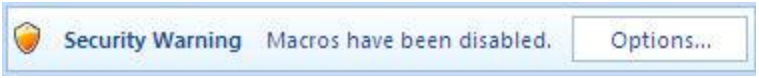
**Note 2:** If you are using Excel 1997 to 2006 see **lesson 10 here**

**Special note for users of Excel 2007:** See how to install the **Visual Basic Editor from your Office CD** and set the security level of your Excel.

If you send a workbook with macros to a colleague and he can not get them to work it is probably because his security setting is at "High" . Tell him how to change his level of security by going to the "Developer" ribbon, clicking on "Macro Security", selecting "Macro Settings and checking the second level "Disable all Macros with Notification" and you are set.




From then on each time you open a workbook that contains macros a temporary status bar appears above the grid in Excel:



Click on "Options" and the following dialog window will appear.


Adopt the same attitude as you have with documents attached to Emails. If you know the origin of the file you may enable the macros if not click on "Disable Macros" and you are fully protected. You can look at the workbook but the VBA procedures (macros) are not operational. You can go to the Visual Basic Editor to take a look at the macros. If nothing looks suspicious close the workbook and re-open it enabling the macros.

**Password Protecting the code**

As an Excel-VBA Developer you might want to protect your code so that nobody else may modify it. In the VBE editor go to "Tools/VBAProject Properties/Protection" . Check the box and submit a password. Make sure that you save the password somewhere that you will remember. If ever you loose the password for an important workbook you can always buy a program on the Internet that will allow you to view the code even if it is password protected.

- Lesson 11: VBA Coding Tips

## VBA Lesson 11: VBA Coding Tips

When you start assembling VBA words into sentences and paragraphs, it is said that you are coding or developing VBA code. In this lesson you will learn important coding tips and many special VBA words. Here is a  tip and an exercise that will give you an idea of what you will find in the complete lesson 11 of the **Downloadable Tutorial on Excel Macros.**

---

### Coding Tip 1

Always key in your code in lower case letters. If the spelling is right, the necessary letters will be capitalized.  If no letter gets capitalized .... check your spelling.

### Exercise 1-1

**Step 1:** Open a new workbook in Excel and use the ALT/F11 keys to go to the visual basic editor.

**Step 2:** In the **code window** of any of the sheets copy/paste the following macro:

```
Sub proTest()

  activecel.cop

End Sub
```

Notice that there are no capital letters in **activecel.cop** because both words are misspelled.

**Step 3:** Add a second "l" to "activecell" and an "y" to "copy" and then click "Enter". The sentence now reads: **Activecell.Copy** with a capital "A" and a capital "C" because both words are spelled correctly.

You now understand that significant letters are capitalised in each correctly spelled VBA word when you move away from the line.

**Step 5:** Close Excel without saving anything

## - Lesson 12: Working with Errors

### VBA Lesson 12: VBA for Excel to Manage Errors

The Visual Basic Editor will help you avoid errors in coding in many different ways. You will not have to wait at the end to be told that there is something wrong with your macro.

---

**Spelling Errors**

You have seen in lesson 11 the VBE capitalise letters to let you know that there are no spelling errors.

---

**Syntax Errors**

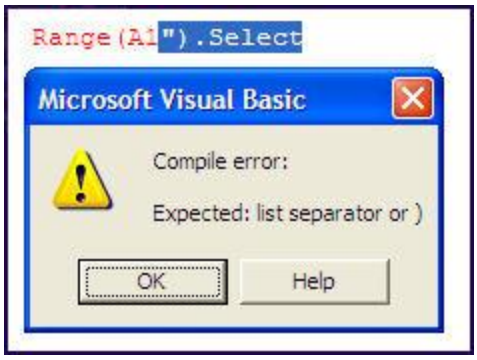
The VBE will also tell you that there is a syntax error in what you have just written by making the font red and showing you a message box.

**Exercise 1**

**Step 1:** Open a new workbook in Excel and use the ALT/F11 keys to go to the visual basic editor (VBE).

**Step 2:** In the **code window** of any of the sheet copy/paste the following line of code: **Range(A1").Select** and click "Enter".

You get the following message box telling you that you are missing a "list separator". Look for the error before the segment highlighted in blue. We can deduce that VBA is talking about the missing quotation mark.



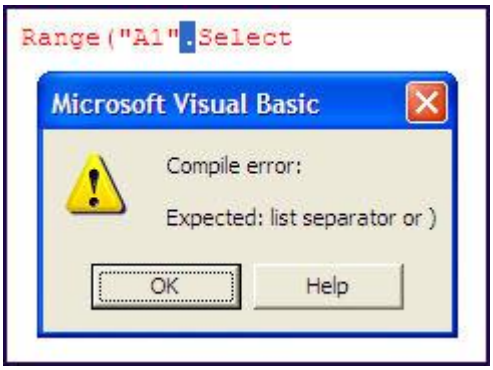

**Step 3:** Click on the "OK" button.

**Step 4:** Add the missing quotation mark, use the mouse to move the cursor to the end of the sentence and click "Enter". The font is black meaning that everything is correct.

**Exercise 2**

**Step 1:** In the **code window** that you have used for exercise 1 copy/paste the following line of code:
**Range("A1".Select** and click "Enter".

You get the following message box telling you that you are missing a "list separator". Look for the error before the segment highlighted in blue. We can deduce that VBE is talking about the missing parenthesis. Both the quotation marks in the exercise above and the parenthesis in this exercise are considered as "list separator" by the VBE.




**Step 2:** Click on the "OK" button.

**Step 3:** Add the missing parenthesis, use the mouse to move the cursor to the end of the sentence and click "Enter". The font is black meaning that everything is correct.

**Step 4:** Close Excel without saving anything

There are many other ways that the VBE uses to alert you to coding errors. You will learn about them all in the **downloadable course on Excel macros**. You will also learn how to use "If" statement to catch errors during the execution and how to use the **OnError** statement to generate user friendly error messages like the following:


## - Lesson 13: Working with the Application

### VBA Lesson 13: VBA for Excel for the Application

Application is a VBA object, IT IS EXCEL. For example: **Application.Quit** will close Excel all together.

**Exercise 1a**

**Step 1:** Open a new workbook in Excel and use the ALT/F11 keys to go to the visual basic editor (VBE).

**Step 2:** Copy the following macro in the **code window** of any sheet. As you can read, you are asking Excel to close itself.

```
Sub testLesson13a1()

    Application.Quit

End Sub
```

**Step 3:** As you have learned in lesson 7, go to Excel and run the macro from the menu bar (**Excel before 2007**) or the ribbon (**Excel since 2007**).

**Step 4:** You will be asked if you want to save the workbook. Answer "No" and Excel will close itself.

**Exercise 1b**

If you do not want to be bothered by the alert to save your workbook you will add a line of code to the small macro: **ActiveWorkbook.Saved = True**

**Step 1:** Open a new workbook in Excel and use the ALT/F11 keys to go to the visual basic editor (VBE).

**Step 2:** Copy the following macro in the **code window** of any sheet. As you can read, you are asking Excel to close itself but saying first that the workbook has already been saved.

```
Sub testLesson13a1()

   ActiveWorkbook.Saved = True
    Application.Quit

End Sub
```

**Step 3:** Run the macro from Excel as you did with the previous one.

Excel will just close itself without asking you anything.

There is a word that you can use with Application that will neutralise all the alerts that Excel can send your way. Discover this word and many others that you can use in combination with Application in the **downloadable course on Excel macros**.

---

There are many other words that you can use in combination with Application. Among them, two important words are:

**ScreenUpdating (Application.ScreenUpdating)**
When you do not want to see your screen follow the actions of your VBA procedure (macro), you start and end your code with the following sentences:
**Application.ScreenUpdating = False**
Then at the end:
**Application.ScreenUpdating = True**

**Exercise**

**Step 1:** Open a new workbook in Excel and use the ALT/F11 keys to go to the visual basic editor (VBE).

**Step 2:** Copy the following macro in the **code window** of any sheet. As you can read: starting in cell A1 a value of "99" will be entered in the selected cell then the cursor will move one cell down to enter "99", repeat the process until the row number of the selected cell is 3000 and come back to cell A1.

```
Sub testLesson13b1()
```

```
    Range("A1").Select

    Do Until Selection.Row = 3000
        Selection.Value = 99
        Selection.Offset(1, 0).Select
    Loop

    Range("A1").Select

End Sub
```

**Step 3:** Run the macro from Excel as you did with the previous one.

**Step 4:** Remove all the "99" from the cells

**Step 5:** Copy the following macro in the **code window** of a new workbook and run it. Two lines of code have been added to the previous macro to prevent all the steps of the action to be seen on the screen.

```
Sub testLesson13b2()

    Application.ScreenUpdating = False

    Range("A1").Select

    Do Until Selection.Row = 3000
        Selection.Value = 99
        Selection.Offset(1, 0).Select
    Loop

    Range("A1").Select

    Application.ScreenUpdating = True

End Sub
```

**Step 6:** Run the macro from Excel as you did with the previous one. You will see a blank sheet, no movement whatsoever and then a sheet where cells A1 to A3000 are equal to "99".

Sometimes you or the users might want to see the action. Some other times you or the user do not want to see the action. It is up to you to use the sentence or not.

You can even use the pair of sentences (as below) anywhere within a long macro to refresh the screen at significant points in the process. With the pair of sentences you

call for a refreshment with **Application.ScreenUpdating = True** and then interrupt the refreshment process until the next refreshment with **Application.ScreenUpdating = False.** Before the end of the macro you will use a final **Application.ScreenUpdating = True**.

The pair of refreshing sentences:
**Application.ScreenUpdating = True**
**Application.ScreenUpdating = False**

**Step 7:** Close the workbook without saving anything

## - Lesson 14: Working with Workbooks

### VBA Lesson 14: VBA for Excel for Workbooks

To develop a VBA procedure that is triggered by an event relating to the workbook (when you open it, when you save it, when you close it) see the **VBA lesson on events**.

**ThisWorkbook**

**ThisWorkbook** is the workbook within which your VBA procedure runs. So if you write:
**ThisWorkbook.Save**
The workbook within which your VBA procedure (macro) runs will be saved.

If you want to close the workbook within which your VBA procedure (macro) runs without saving it you will write these two lines of code:
**ThisWorkbook.Saved=True**
**ThisWorkbook.Close**

**Verifying the existence of a file**

When you want to verify if a certain file exists on your disk you will use the following code that means "If the file "C:\Stuff\toto.xls" does not exist then":
**If Dir("C:\Stuff\toto.xls") = "" Then**

You could also use a sentence that means "If the file "C:\Stuff\toto.xls" does exist then":
**If Dir("C:\Stuff\toto.xls") <> "" Then**

If you are looking in the same folder as the file in which the macro runs you can simplify the VBA code:
**If Dir("toto.xls") <> "" Then**

In the downloadable tutorial on Excel macros you will find many other uses for **Dir** including opening all the files of a folder to generate a consolidated database (whatever the number of files in the folder). You will also learn about **Path**, **ActiveWorkbook**, **Windows**, **Kill**, and many other VBA words to work with one or many workbooks.

## - Lesson 15: Working with Worksheets

### VBA Lesson 15: VBA for Excel for Worksheets

To develop a VBA procedure that is triggered by an event relating to the worksheet (when you select it, when you leave it...) see the **VBA lesson on events**.

**Sheets**

You access a worksheet named " Balance" with:
**Sheets("Balance").Select**
Note that the word "Sheets" is plural and always use the quotes within the parenthesis

You cannot select a sheet that is hidden so you will need to write:
**Sheets("Balance").Visible= True**
**Sheets("Balance").Select**
and then if you want to hide the sheet again:
**Sheets("Balance").Visible= False**

The name of a sheet must not have more than 31 characters and should not include certain special characters like " ? : \ / [ ]" . If you do not respect these rules your procedure will crash.

The following lines of code will generate an error message:
**Sheets("Sheet1").Name= "Balance and Introduction to Numbers"** because there are more than 31 characters including the spaces
**Sheets("Sheet1").Name= " Balance: Introduction"** because of the special character :
**Sheets("Sheet1" ).Name= " "** because the name cannot be blank

You can not go directly from a sheet to a cell on another sheet. For example if the active sheet is "Balance" and you want tot go to cell A1 of a sheet named " Results" you cannot write:
**Sheets("Results").Range("A1").Select**
You must take two steps:
**Sheets("Results").Select**
**Range("A1").Select**

## - Lesson 16: Range and Cells

### VBA Lesson 16: Cells, Ranges, Columns and Rows in VBA for Excel

Many beginners start their career using **Cells**. For example:
**Cells(1,1).Select** means (row 1, column 1) and is the same thing as
**Range("A1").Select** and
**Cells(14,31).Select** means (row 14, column 31) and is the same as
**Range("AE14").Select**.

We strongly recommend that you use **Range** instead of **Cells** to work with cells and groups of cells. It makes your sentences much clearer and you are not forced to remember that column AE is column 31.

The only time that you will use **Cells** is when you want to select all the cells of a worksheet. For example:
**Cells.Select**
To select all cells and then empty all cells of values or formulas you will use:
**Cells.ClearContents**

## Range

To select a single cell you will write:
**Range("A1").Select**

To select a set of contiguous cells you will use the colon and write:
**Range("A1:G5").Select**

To select a set of non contiguous cells you will use the comma and write:
**Range("A1,A5,B4").Select**

To select a set of non contiguous cells and a range you will use both the colon and the comma:
**Range("A1,A5,B4:B8").Select**

## Offset

The **Offset** property is the one that you will use the most with **Range** to move around the sheet.

To move one cell down (from B2 to B3): **Range("B2").Offset(1,0).Select**
To move one cell to the right (from B2 to C2): **Range("B2").Offset(0,1).Select**
To move one cell up (from B2 to B1): **Range("B2").Offset(-1,0).Select**
To move one cell to the left (from B2 to A2): **Range("B2").Offset(0,-1).Select**

To move one cell down from the selected cell:
**ActiveCell.Offset(1,0).Select**

As you notice the first argument between the parentheses for **Offset** is the number of rows and the second one is the number of columns. So to move from A1 to G6 you will need:
**Range("A1").Offset(5,6).Select**

You will use very often the following piece of code . It selects a cell PLUS 4 more to the right to be copied/pasted somewhere else:
**Range(ActiveCell,ActiveCell.Offset(0,4)).Copy**
Notice the comma after the first **ActiveCell** and the double closing parentheses before the **Copy**.

---

There are many important VBA words to discover in the downloadable **Course on Excel Macros**. You have already read something about **Range**, **Cells**, **Offset**, **ActiveCell**, read some more about them and about many other powerful words like **CurrentRegion**, **UsedRange**, **End(xlDown)**, **Formula**, **Value, FormulaR1C1, ClearContents**, **Delete**, and many more.

## - Lesson 17: Message Boxes in VBA for Excel

### VBA Lesson 17: Message and Input Boxes (MsgBox, InputBox) in Excel

In VBA for Excel the message box (MsgBox) is the primary tool to interact with the user. For example you might want to tell the user that a long macro has finished running.

**Exercise 1**

**Step 1:** Open a new workbook and use the ALT/F11 keys to move to the Visual Basic Editor.

**Step 2:** Copy/Paste the following macro from here into the code window of any sheet.

```
Sub proLessson17a()
    Sheets("Sheet1").Select
    Range("A1").Value = 695
    MsgBox "The macro has finished running"
End Sub
```

Notice the space following **MsgBox** and the use of quotation marks surrounding the text

**Step 3:** Use the ALT/F11 keys to go back to Excel and run the macro **proLessson17a**.

The value 695 is entered in cell A1 and the following message box appears.


**Step 4:** Delete the macro in the Visual Basic Editor and the value 695 from cell A1

**Exercise 2**

You might want to tell the user where he will find the result.

**Step 1:** Use the ALT/F11 keys to move to the Visual Basic Editor.

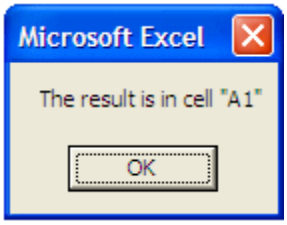
**Step 2:** Copy/Paste the following macro from here into the code window of any sheet.

```
Sub proLessson17b()
    Sheets("Sheet1").Select
    Range("A1").Value = 695
    MsgBox "The result is in cell ""A1"""
End Sub
```

Notice the space following **MsgBox,** the use of quotation marks surrounding the text and the double quotation mars around A1 because we want the address to show on the message box between quotation marks.

**Step 3:** Use the ALT/F11 keys to go back to Excel and run the macro **proLessson17b**.

The value 695 is entered in cell A1 and the following message box appears




**Step 4:** Delete the macro in the Visual Basic Editor and the value 695 from cell A1

**Exercise 3**

Instead of telling the user that the value is in cell A1, you might want to tell him what the result is in the message box itself.

**Step 1:** Use the ALT/F11 keys to move to the Visual Basic Editor.

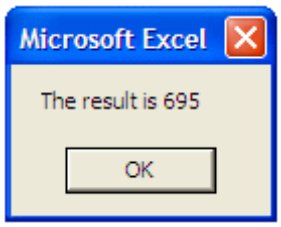
**Step 2:** Copy/Paste the following macro from here into the code window of any sheet.

```
Sub proLessson17c()
    Sheets("Sheet1").Select
    Range("A1").Value = 695
    MsgBox "The result is " & Range("A1").Value
End Sub
```

Notice the space following **MsgBox,** the use of quotation marks surrounding the text, the space at the end of the text and the spaces surrounding the ampersand.

**Step 3:** Use the ALT/F11 keys to go back to Excel and run the macro **proLessson17c**.

The value 695 is entered in cell A1 and the following message box appears




**Step 4:** Close Excel without saving anything.

You can use the message box to inform the user. You might also ask the user (with a Yes/No message box) if he is sure that he wants a certain critical procedure to run (deleting things).

There are many types of message boxes (information, alert, exclamation or questions. Then if you need an input from the user you will start using the input box.

For more elaborate message boxes and input boxes see the **downloadable course on Excel macros**.

## - Lesson 18: Excel VBA Vocabulary to Filter and Sort Data

### VBA Lesson 18: Excel VBA Vocabulary to Filter and Sort Data

When Excel recognises you **set of data as a database** it offers you very powerful database functionalities like sorting and filtering.

**Deactivating filters**

When you work in an Excel database you might want to make sure that all data filters are off. To this end you will start your procedure with two "**If**"statements. For example with a database starting in cell A1 here are the two sentences:

```
Range("A1" ).Select
        If ActiveSheet.AutoFilterMode = True Then Selection.AutoFilter

        If ActiveSheet.FilterMode = True Then ActiveSheet.ShowAllData
```

**Sorting Data**

Here is a simplified Excel macro to sort data using a criteria in one field. The following Excel macro will work with any size database starting in cell A1 and it will work in any version of Excel (1997 to 2010).

```
Sub proFilter()

Range("A1").Sort Key1:=Range("A2"), Order1:=xlAscending, Header:=xlYes

End Sub
```

Try the Excel macro above with a small table like the following (as you have leand how in the basic exercises for beginners):

| Name | Number |
|---|---|
| Jones | 1 |
| Tom | 2 |
| Barry | 3 |
| Peter | 4 |

Here is another simplified Excel macro sorting data using criteria in three different fields.

```
Sub proFilter()

Range("A1").Sort Key1:=Range("A2"), Order1:=xlAscending, Key2:=Range( _
      "B2"), Order2:=xlAscending, Key3:=Range("C2"), Order3:=xlAscending, _
      Header:=xlYes

End Sub
```

The code in the two procedures above is much simpler than the following recorded macro in Excel 2007 and 2010. This recorded macro will not work in earlier versions of Excel (1997 to 2006).

```
   ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Clear
    ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Add
Key:=Range("A2:A7"), _
      SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
```

```
    ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Add
Key:=Range("B2:B7"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Add
Key:=Range("C2:C7"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    With ActiveWorkbook.Worksheets("Sheet1").Sort
        .SetRange Range("A1:E7")
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
```

In the **downloadable course on Excel macros** we offer you much more vocabulary to work with Excel databases and also many more simplified macros that can be used in all versions of Excel. You can you can copy/paste any of them into your own workbooks.

## - Lesson 19: Working with Variables

### VBA Lesson 19: VBA for Excel Variables

You will start developing complex and sophisticated programs in Excel and you will start working with very large sets of data when you discover the variables.

A variable is an object that you create and in which you can store text, dates, numbers or almost anything else. Why should you use variable? The first good reason is to make your code dynamic, to avoid hard coding some values.

**Hard Coding vs Dynamic Coding**

You are hard coding when you write:

```
Workbooks.Open "MyFile.xls"
```

You are dynamically coding when you enter the name of the file in an cell (A1) of your Excel sheet and you write.

```
varWorkbook=Range("A1").Value
Workbooks.Open varWorkbook
```

At this point you or the user can change the name of the workbook to open in cell A1 instead of going to the VBA code in the Visual Basic Editor.

You will also create variables to count the number of rows, store the result in a variable and then do something as many time as there are rows.

```
For varCounter = 1 to varNbRows
      Selection.Value=Selection.Value*2
      Selection.Offset(1,0).select
Next
```

In the VBA procedure above the value in each cell is multiplied by 2 then the cell below is selected. This action is repeated as many times as there are rows in the set of data.

## - Lesson 20: Working with Statements

### VBA Lesson 20: VBA for Excel Statements

Among the VBA statements that you will discover in the downloadable tutorial on Excel macros, there are the "**If**" statement including **Then**, **ElseIf** and **End If**, there is the "**Do**" statement including **Loop**, **Until**, **While** and **Exit**, there is the "**For**" statement including **To**, **Step**, **Next** and **Exit**, there is the powerful "**Select Case**" statement including **Case**, **End Select** and **Exit** and other statements.

A lot of visitors ask us how they can delete the entire lines when a certain cell is empty. For example, in the table below rows 2 and 5 should be deleted:

| | A | B |
|---|---|---|
| 1 | John | 33 |
| 2 | Mary | |
| 3 | Louis | 99 |
| 4 | Mark | 75 |
| 5 | Liz | |
| 6 | Peter | 44 |

First enter xxx where you want the loop to stop (below the last value: B7). Select the cell at the top of the column containing the values to be considered (B1)and run the macro.

```
Sub proDelete()

    Range("B1").Select
    Do Until Selection.Value = "xxx"
        If Selection.Value = "" Then
            Selection.EntireRow.Delete
        Else
            Selection.Offset(1, 0).Select
```

```
        End If
    Loop

    Range("A1").Select

End Sub
```

If you have completed the free exercises "**Free Basics**", just copy/paste the macro above in the Visual Basic editor and run it.

**Exiting a Loop**

In the loop above if you want the loop to stop when it finds the value 99 you can add this line of code within the loop:

```
If Selection.Value = 99  Then Exit Do
```

Exit allows you to get out of almost anything like:

```
Exit Sub
Exit For
Exit Do
```

## VBA Lesson 21: Functions in VBA for Excel

There are three topics in this lesson:
- using Excel functions within macros,
- using VBA functions within macros,
- creating new Excel functions with VBA.

---

**Excel Functions**

Some of the functions that you find in Excel are available through macros in this form:

```
Range ("C1").Value= Application.WorksheetFunction.Sum(Range("A1:A32"))
```

this sentence sums the values of cell A1 to A32 and stores the total in cell C1.

---

**VBA Functions**

Here are two VBA functions that you will use within your Excel macros:

**LCase, UCase**
The " If" statements are case sensitive. When you test a string of characters and you do not know if the user will enter upper case or lower case letters, use the LCase or UCase functions within your " If" statement so that however the user enters his answer the statement will work.

**If LCase(Selection.Value)= "yes" then...**
or
**If UCase(Selection.Value)= "YES" then...**

## VBA Lesson 22: External Data and SQL in VBA for Excel

SQL stands for Structured Query Language and is the language used to extract data from almost all databases like Access and SQL Server from Microsoft or, Oracle, Sybase, SAP and also most accounting applications. You can also extract data from the Internet, from text files and from other Excel or CSV files.

Basically you need a connection (varConn in the macro below) and an SQL sentence (varSQL in the macro below) to automate the extraction of data for reporting purposes. In the example below an SQL query extracts all the data from a small Acces database.

Click here to **download the small Access database** and test the following code from a workbook sitting in the same folder.

```
Sub proSQLQueryBasic()
Dim varConn As String
Dim varSQL As String

   Range("A1").CurrentRegion.ClearContents

   varConn = "ODBC;DBQ=test.mdb;Driver={Driver do Microsoft Access (*.mdb)}"

   varSQL = "SELECT tbDataSumproduct.Month, tbDataSumproduct.Product,
tbDataSumproduct.City FROM     tbDataSumproduct"

       With ActiveSheet.QueryTables.Add(Connection:=varConn,
Destination:=Range("A1"))
          .CommandText = varSQL
          .Name = "Query-39008"
          .Refresh BackgroundQuery:=False
       End With

End Sub
```

Open the Excel files **vba-sql1** and **vba-sql2** for a complete explanation of the code and much more on queries. These two Excel workbooks are part of the **Tutorial on Excel Macros**.

## VBA Lesson 23: Working with Other Microsoft Programs in VBA for Excel

### Working with other Microsoft programs using VBA within Excel

Within Excel you can open another program and even develop a program within it using VBA. For example here is a short macro that opens Word, then a new document to

copy/paste the content of 2 cells from Excel to Word and save the Word document in the same directory as the workbook in which the macro runs:

**Exercise**

**Step 1:** As you have learned how to in the "**Free Basics**", copy/paste the following macro in a new workbook that you will save as word.xlsm.

```
Sub proWord()
Dim varDoc As Object

    Set varDoc = CreateObject("Word.Application")

        varDoc.Visible = True
        Sheets("Sheet1").Range("A1:B1").Copy
        varDoc.documents.Add
        varDoc.Selection.Paste
        varDoc.activedocument.SaveAs ThisWorkbook.Path & "/" &
"testWord.doc"
        varDoc.documents.Close

    varDoc.Quit
     Application.CutCopyMode = False

End Sub
```

**Step 2:** Enter values in cells A1 and B1 (your first and lat name for example).

**Step 3:** Run the macro

You end up with a Word document named testWord .Doc in the same directory as the Excel workbook in which the macro runs. The Word document consists of a single sheet with a two cells table with the values of cell A1 and B1 of the workbook.

Notice that you use VBA for Word within the object **varDoc** that you have created. If you do not know VBA for Word remember that there is also a Macro Recorder in Word. The object **varDoc** can be visible or you can work within it without bringing it on screen with: **varDoc.Visible = False**

**API Working with Windows**

API stands for Application Programming Interface and consists of a collection of functions that provide programmatic access to the features of the operating system (Windows). When you use API's within VBA for Excel not only do you control Excel but

also most parts of Windows.

---

## VBA Lesson 24: Forms (Userforms) in VBA for Excel

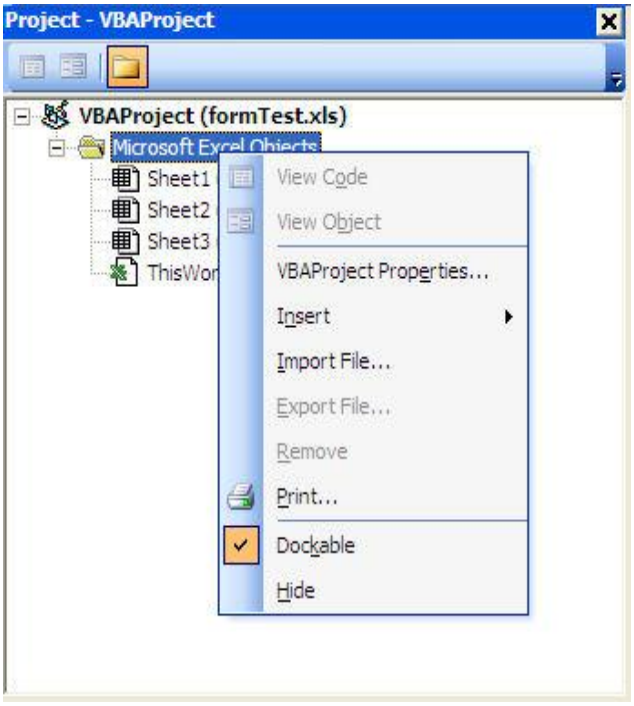
When the **message box** or the **input box** are not sufficient any more to communicate with the user you need to start developing userforms.

The form is used to require information from the user to feed the VBA procedure. Different basic controls can be added to the userform they are called: **labels**, **text boxes**, **combo boxes**, **list boxes**, **check boxes**, **option buttons**, **frames**, **command buttons**, **spin buttons** and **images** . To learn more about all the controls see lessons 26 to 33.

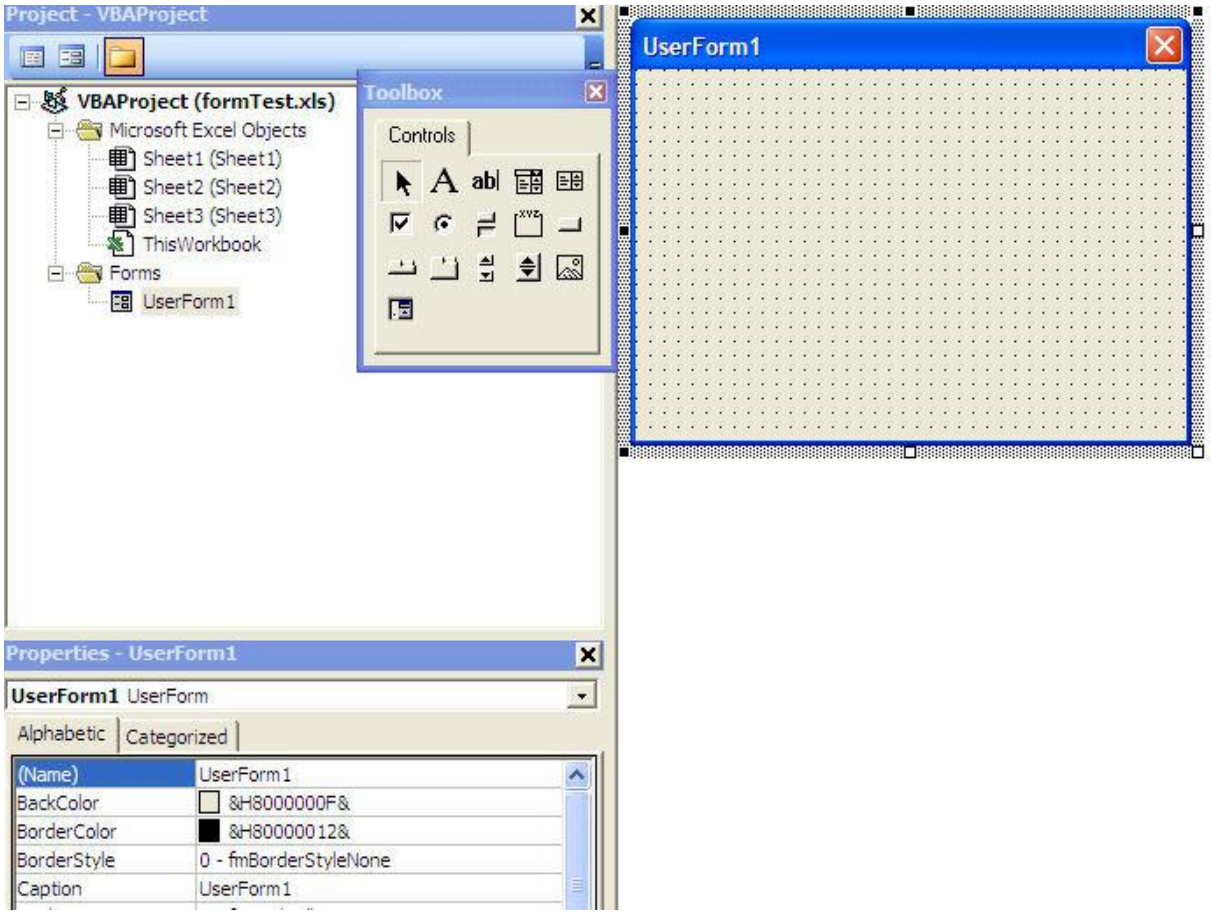
### Creating a Userform in Excel

Userforms are created in the **Project Window** of the Visual Basic Editor. You will also find the toolbox that allows you to add controls to your userforms in the Visual Basic Editor.

In the Visual Basic Editor you right click in the project window and you will see this menu appear:




Go to "Insert" and select "UserForm". You will then see the following:



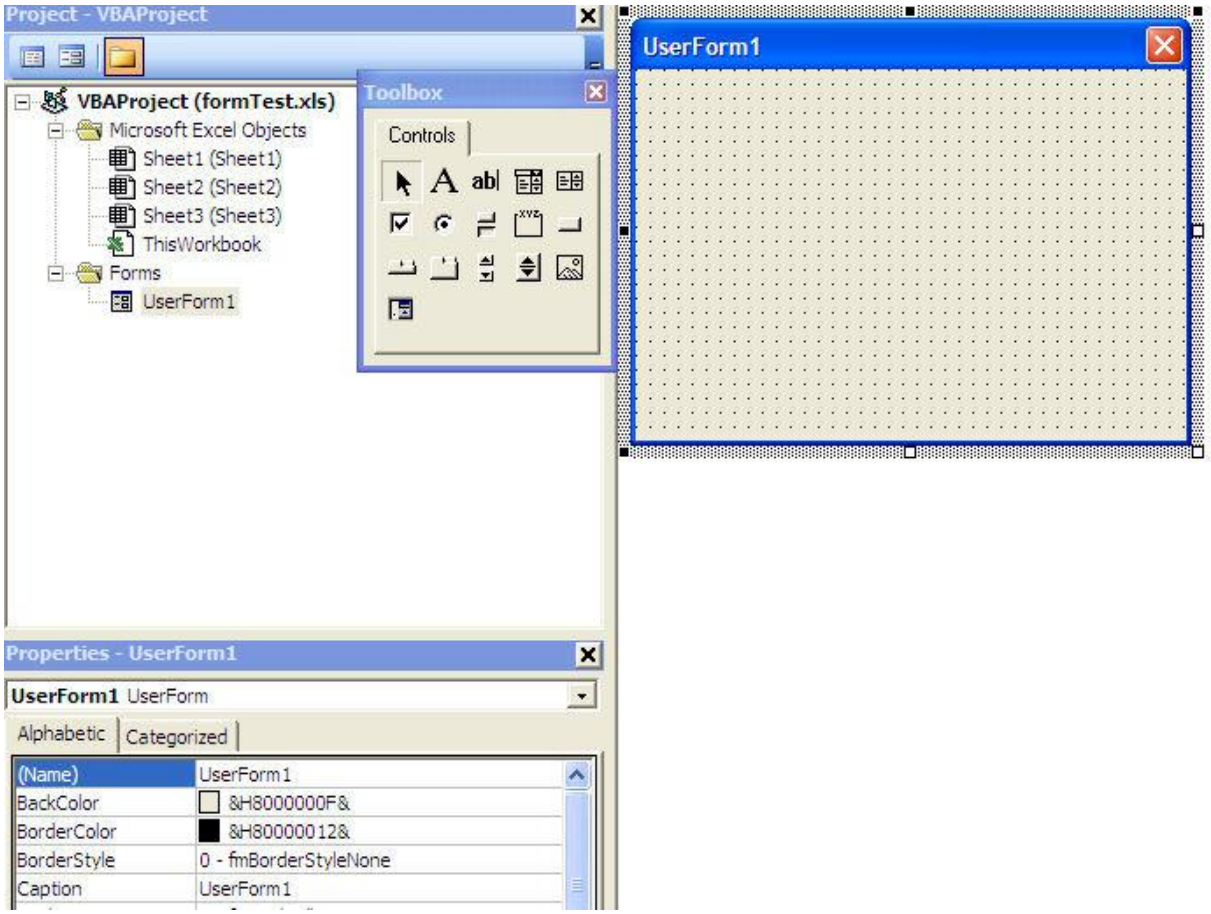

On the right you see the userform that you have just added to your workbook. On the left is the toolbox with all the controls that you can add to your userform. You can hide that toolbox by clicking on the "X" and bring it back by clicking on the toolbox icon  or by going to the menu bar "View/Toolbox". We will use the toolbox later in this section.

## VBA Lesson 25: Userforms Properties and VBA Code

In this lesson we will review some of the properties of the userform, we will develop some programming to call the userform and some other programming within the userform itself.

### Userforms Properties

When you double click on the userform name in the project window of the Visual Basic Editor the properties windows shows 35 properties of the userform. On this website we will work with two of them. For the other 33 properties see the **downloadable course on Excel macros (VBA)**




**VBA Code within the UserForm**

In lesson 9 you have learned about events. The events trigger the macros. There are many events that happen around the userform. For example, a macro can start when the userform is shown (or activated) and another macro can start when a user clicks on a command button. You will learn all these two events in the **downloadable the tutorial on Excel macros**.

## VBA Lesson 26: Labels in VBA for Excel

In the toolbox the label has this icon  . The label is a passive control meaning that the user never really acts on it. It is there to inform the user and to label other controls like text boxes, combo boxes or list boxes.

**Properties**

Among the properties of the label is:

- **WordWrap:** If you want to write more than one line of text in a label set this property to "True" .

**Adding a Label to a Userform**

To add a label to a userform you left click on its icon in the toolbox. You move the cursor to the userform, you click again and the label appears. You can then resize it to your liking. If you double click on the label icon in the toolbox you can then click on the form as many times as you need labels. When you are finished adding labels just click once on the label icon of the toolbox.

## VBA Lesson 27: Text Boxes in VBA for Excel

In the toolbox the text box icon is:  .

The text box is the simplest control that requires an entry by the user. The user types something in it and this value can then be used in your VBA procedure. You will usually add a label to accompany the text box.

For most controls including the VBA for Excel text box there are general properties that allow you to set the font, the color of the font, the color of the background, the type of background, the type of border and other design features.

As its name says it the text box carries text. To use the contents of a text box as a number, to add dollar signs, decimal and other numerical features see the **downloadable tutorial on Excel macros (VBA)**.

**Adding a Text Box to a Userform**

To add a text box to a userform you left click on its icon in the toolbox. You move the cursor to the userform, you click again and the text box appears. You can then resize it to your liking. If you double click on the text box icon in the toolbox you can then click on the form as many times as you need text boxes. When you are finished adding text boxes just click once on the text box icon of the toolbox.

## VBA Lesson 28: Command Buttons in VBA for Excel

In the toolbox the command button has this icon  . The command button is a very active control and there is always VBA code behind it.

The command buttons are usually placed at the bottom of the form and serve to complete the transaction for which the form has been created. The caption of these buttons are usually "Go" , "Run" , "Submit" , "Cancel" , etc.

### Properties

Among the other properties of the command button  are:

- **WordWrap** to be able to write more that one line on a button,
- **ControlTipText** which generates a small comment box when the user moves the mouse over the control. You can use this property to give explanations and instructions about the command button,

### Adding a Command Button to a Userform

To add a command button to a userform you left click on its icon in the toolbox. You move the cursor to the userform, you click again and the command button appears. You can then resize it to your liking.  If you double click on the command button icon in the toolbox you can then click on the form as many times as you need command buttons. When you are finished adding command buttons just click once on the command button icon of the toolbox.

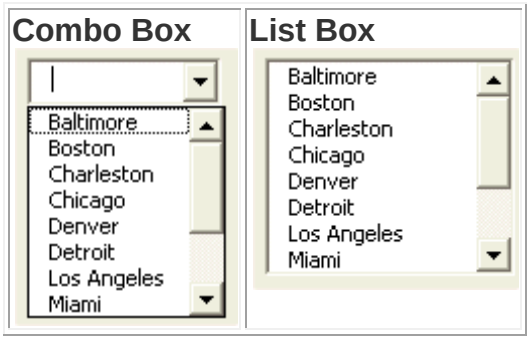
### VBA Code

Most of the VBA code (VBA sentences) is created within the command button when you develop simple userforms. Here are two exercises creating VBA code within the command button.

## VBA Lesson 29: Combo Boxes in VBA for Excel

### Before we begin on the Combo Box

The difference between a combo box and a **list box** is that the combo box is a drop-down list and the user can submit a single value from the drop-down list. The list box shows a certain number of values with or without a scroll bar and the user can select one or more values.



If you are looking for a drop-down list (also called pull-down lists) to use on a regular worksheet see the much easier and user friendly **Excel drop-down lists** in the website on Excel.

When you double click on the combo box in the **Visual Basic Editor** you will see all its properties in the **Properties window .**

No programming is needed to submit the list of values that will be offered to the user within the combo box. Look for the RowSource property.

**The RowSource Property:**

The values that should appear in the drop-down list of the combo box are submitted in the **RowSource** property. For example, if the value of the RowSource property is **Balance!A1:A12** The values residing in cell A1 to A12 of the sheet named Balance will be offered as choices to the user who clicks on the small arrow of the combo box.

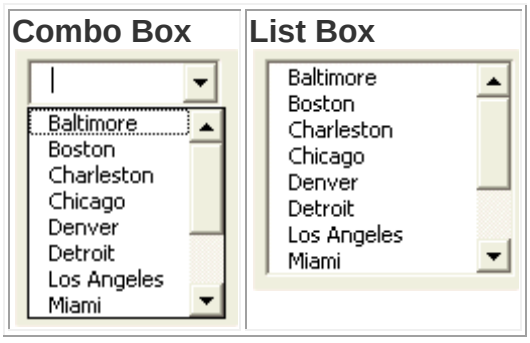
The rules to submit the **RowSource** property is the name of the sheet where the list resides **followed by an exclamation point** (!), the address of the first cell, a colon and the address of the last cell.

**IMPORTANT NOTE**: if there is a space or a special character within the name of the sheet where the list resides you must surround the name of the sheet with simple quotes. For example: **'New Balance'!A1:A12**.

## VBA Lesson 30: List Boxes in VBA for Excel

**Before we begin on the List Box**

The difference between a **combo box** and a list box is that the combo box is a drop-down list and the user can submit a single value from the drop-down list. The list box shows a certain number of values with or without a scroll bar and the user can select one or more values.



In the toolbox the list box has this icon  .

No programming is needed to submit the list of values that will be offered to the user within the combo box. Look for the RowSource property.

**The RowSource Property:**

The values that should appear in the drop-down list of the combo box are submitted in the **RowSource** property. For example, if the value of the RowSource property is **Balance!A1:A12** The values residing in cell A1 to A12 of the sheet named Balance will be offered as choices to the user who clicks on the small arrow of the combo box.

The rules to submit the **RowSource** property is the name of the sheet where the list resides **followed by an exclamation point** (!), the address of the first cell, a colon and the address of the last cell.

**IMPORTANT NOTE**: if there is a space or a special character within the name of the sheet where the list resides you must surround it with simple quotes. For example: **'New Balance'!A1:A12**.

## VBA Lesson 31: Option Buttons, Check Boxes and Frames

In the toolbox the option button has this icon  , the check box has this one  and, the frame this one  .

You do not need to add a label to accompany the check box or the option button because they come with their own.

The check boxes and the option buttons are both used to offer the user a choice. The main difference between check boxes and option buttons is that if you have 5 of each on a form a user can check all 5 check boxes but can only select one of the option buttons.

If you want to create two sets of option buttons read below on frames and option buttons. If you do not want to use frames to create groups of option buttons you will need to use the "GroupName" property of the option buttons. All option buttons with the same GroupName work together.

**Properties**

- **WordWrap** to be able to write more that one line in the caption,
- **ControlTipText** which generates a small comment box when the user moves the mouse over the control. You can use this property to give explanations and instructions

about the option button or the check box.
- **Enabled** and **Visible** are properties that you can change programmatically to disable or render invisible an option button or a check box following a previous selection in another control of the userform.

**Frames**

Frames are also a passive control. Frames are used to improve the layout of the userform. You can use them around a group of controls that have something in common.

Frames become more important to manage option buttons. If you have two sets of option buttons on a userform and you do not place them within a frame they all work together and you can choose only one. If you put each set within a frame you can choose one in each set.

## VBA Lesson 32: Excel Spin Buttons

**Spin Button**

In the toolbox the spin button has this icon  .

You can ask a user to enter a value directly in a text box but you can make things a little more attaractive by using a text box and a spin button.

The spin button is not really used by itself. Because the spin button does not show its value it is usually used with a text box. The text box shows a number and by clicking on the arrows of the spin button the value in the text box is increased (or decreased) by 1, or 5 or 10...by whatever value that is set within the properties of the spin button.

**Properties**

Among the other properties of the spin buttons are:

- **Min** is the minimum value of the spin button. It can be negative
- **Max** is the maximum value of the spin button. It can be negative
- **Small** is the value of the change when the user clicks on the arrows
- **Large** is the value of the change when the user clicks on the scroll bar of the spin button.

## VBA Lesson 33: Excel Image Controls

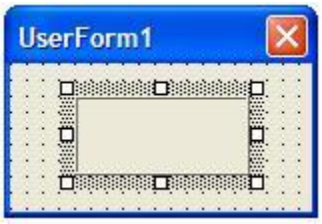
**Image Control**

There is a control in the toolbox called "Image" . Within this control you can show all types of pictures. You set an image control on  a userform and you submit a picture in the property "Picture" . The picture becomes part of the control and userform.

**Fitting the Picture**

The first thing that you want to do is to fit the picture in the image control to make the size of the control adapt to the size of the picture.

When you are in the Visual Basic Editor and you single click on an image control a frame appears around it with 8 stretchers (picture below). If you double click on the middle stretcher  (when a two tips arrow shows) of the right side or on the middle one at the  bottom or on the bottom right corner stretcher  the image control will adapt to the size of the image. Double clicking anywhere else will take you to the VBA code and will not adapt the control size to the picture size.




**PictureSizeMode Property**

Another property of the image control is the PictureSizeMode.

If the property is set to the default value 0-frmPictureSizeModeClip the control size can be changed without the picture size being modified. So you can see only part of the picture or there can be a background behind it in a clolor color you can change at will.

If the property is set to the 1-frmPictureSizeModeStretch the picture is resized as the control is. The image fills the control.

If the property is set to the 3-frmPictureSizeModeZoom the picture is resized as the control is but the picture and background are present.