Advance Excel Assignment 16

1. What is a Macro? How is it useful in excel or in your daily work?

A macro is shorthand for a programming term: macroinstruction. In the most basic terms, a macro is a way to automate simple tasks. They can be used in a variety of ways, but most of us will learn to love them as helpful shortcuts in Microsoft Excel.

Why do I need them?

A macro is a set of instructions used to execute repetitive tasks. You can record a set of commands and then play them back with one or two keystrokes.

That means that you can save A LOT of time when doing routine and repetitive tasks. Before learning about macros, most of us just resign ourselves to the fact that there will be certain mundane tasks in Excel that we can't escape. But once you discover the power of macros, this can all change. If you've not used macros before you might be completely confused about how they can help you, so let's look at a simple example:

Every week you must update a central budget spreadsheet, pulling in numbers from three different spreadsheets, all with different formatting that you then have to fix.

Every time you do this, you start by opening the main spreadsheet. Then you open spreadsheet one, copy over the data onto the central spreadsheet, and format it so it's in the same style as the rest of the data. And then you do the same with spreadsheets two and three.

It's a boring, monotonous task that could take 15-30 minutes each time.

But if you were to use a macro, you could do all of this automatically with a few clicks.

2. What is VBA? Write its full form and briefly explain why VBA is used in excel?

Visual Basic for Applications (VBA) is part of Microsoft Corporation's (NASDAQ: MSFT) legacy software Visual Basic. VBA is used to write programs for the Windows operating system and runs as an internal programming language in Microsoft Office (MS Office, Office) applications such as Access, Excel, PowerPoint, Publisher, Word, and Visio. VBA allows users to customize beyond what is normally available with MS Office host applications.

Uses of VBA:

At its core, finance is about manipulating huge amounts of data; hence, VBA is endemic to the financial services sector. If you work in finance, VBA is likely running within applications that you use each day, whether you're aware of it or not. Some jobs in the sector require prior knowledge of VBA, and some do not. With VBA, you can:

Write macros. Macros allow financial professionals—whether accountants, commercial bankers, investment bankers, research analysts, salesmen, traders, portfolio managers, clerks, or administrators—to analyze and adjust huge amounts of data quickly.

Update data. You can use VBA in Excel to create and maintain complex trading, pricing, and risk-management models, forecast sales and earnings, and to generate financial ratios.

Perform scenario-analysis. With Visual Basic for Applications, you can create various portfolio-management and investment scenarios. This includes filtering through different situations that may impact outcomes differently.

Organize information. You also may use VBA to produce lists of customers' names or any other content; create invoices, forms, and charts; analyze scientific data, and manage data display for budgets and forecasting.

Be unconventional. VBA can be used to copy and paste values, adjust cell styles for an entire workbook, and strike accelerator keys. You can perform very normal tasks but perform them in an easier, automated manner.

Prompt action. VBA can be used to interact with users. For example, you may need a user's input for their first and last name to put on a form. VBA can be used to prompt a user in a way that makes this input unavoidably mandatory.

3. How do you record a macro? Write detailed steps to create a macro to automatically make the following table in bold and to create borders for it in excel.

hi	78
hello	69
ineuron	45

Recording a Macro in Excel

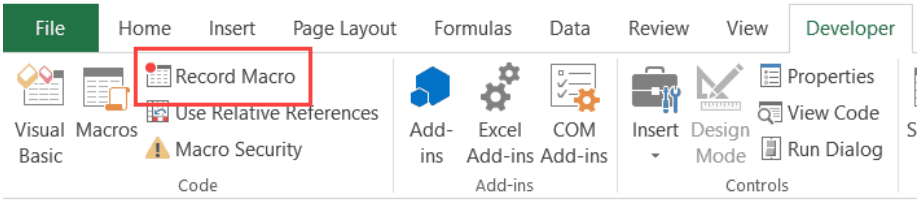
Now that we have everything in place, let's learn how to record a macro in Excel.

Let's record a very simple macro – one that selects a cell and enters the text 'Excel' in it. I am using the text 'Excel' while recording this macro, but feel free to enter your name or any other text that you like.

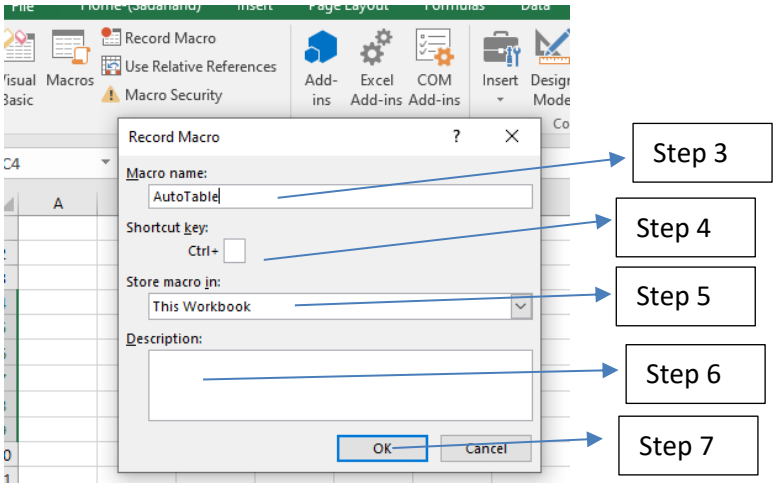
Here are the steps to record this macro:

1. Click the Developer tab.

2. In the Code group, click on the Macro button. This will open the 'Record Macro' dialog box.



3. In the Record Macro dialog box, enter a name for your macro. I am using the name **AutoTable**. There are some naming conditions that you need to follow when naming a macro. For example, you can not use spaces in between. I usually prefer to keep my macro names as a single word, with different parts with a capitalized first alphabet. You can also use underscore to separate two words – .Auto_Table.



4. (Optional Step) You can assign a keyboard shortcut if you want. In this case, we will use the shortcut Control + Shift + N. Remember that the shortcut you assign here would override any existing shortcuts in your workbook. For example, if you assign the shortcut Control + S, you will not be able to use this for saving the workbook (instead, everytime you use it, it will execute the macro).

5. In the 'Store macro in' option, make sure 'This Workbook' is selected. This step ensures that the macro is a part of the workbook. It will be there when you save it and reopen again, or even if you share it with someone.

6. (Optional Step) Enter a description. I usually don't do this, but if you're extremely organized, you may want to add what the macro is about.

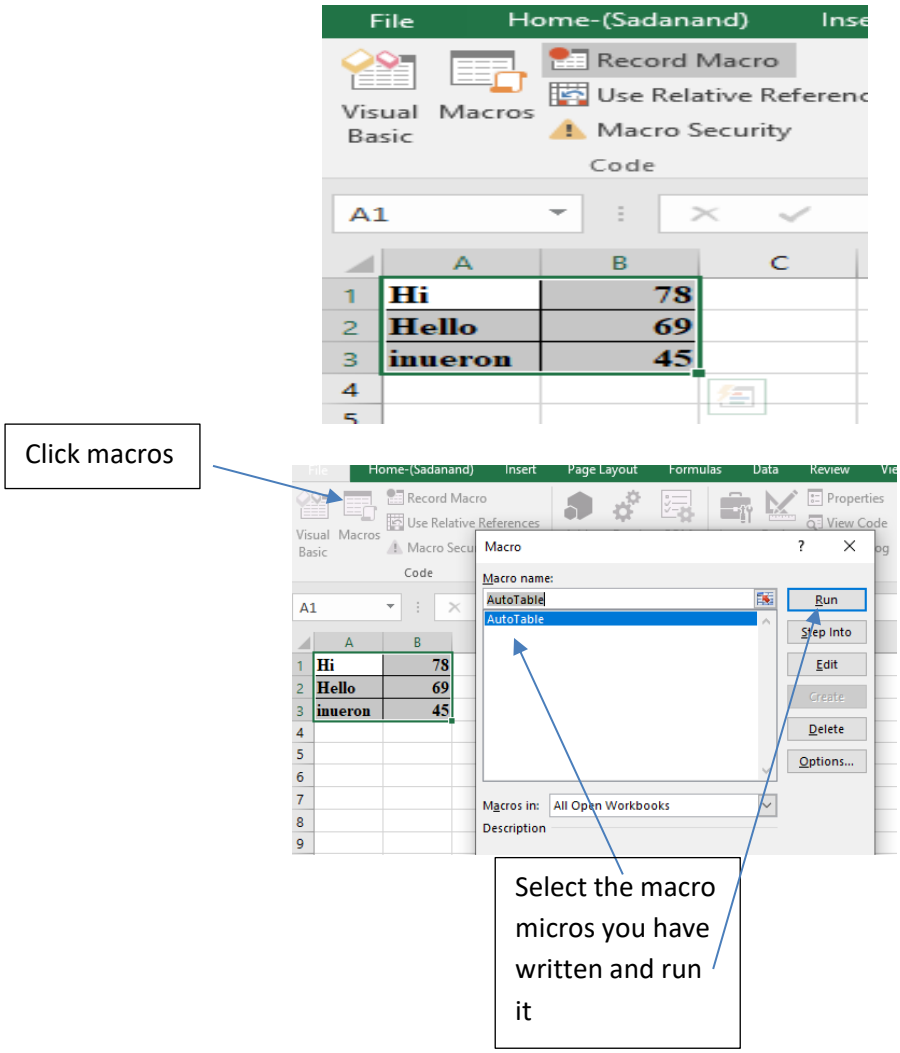
7. Click OK. As soon as you click OK, it starts to record your actions in Excel. You can see the 'Stop recording' button in the Developer tab, which indicates that the macro recording is in progress.

8. Select cell A2.

9. Enter the create table with bold letters and boarders**.**

10. Hit the Enter key. This will select cell A3.

11. Click on the Stop Recording button the Developer tab.



4. What do you mean when we say VBA Editor?

The first step to working with VBA in Excel is to get yourself familiarized with the Visual Basic Editor (also called the VBA Editor or VB Editor).

In this tutorial, I will cover all there is to know about the VBA Editor and some useful options that you should know when coding in Excel VBA.

Visual Basic Editor is a separate application that is a part of Excel and opens whenever you open an Excel workbook. By default, it's hidden and to access it, you need to activate it.

VB Editor is the place where you keep the VB code.

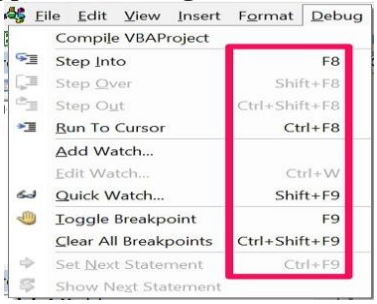
There are multiple ways you get the code in the VB Editor:

- When you record a macro, it automatically creates a new module in the VB Editor and inserts the code in that module.
- You can manually type VB code in the VB editor.
- You can copy a code from some other workbook or from the internet and paste it in the VB Editor.

5. Briefly describe the interface of a VBA editor? What is properties window? And what is watch window? How do you display these windows?

Let's dive right in and understand the 6 main components of the Visual Basic Editor.
Component #1: Menu Bar

If you've been using computers for a reasonable amount of time, you're probably quite familiar with menu bars. If that's the case, the VBE menu bar is not very different from the other menu bars you've seen before.
The menu bar, basically, **contains several drop-down menus**. Each of the drop-down menus contains commands that you can use to interact and do things with the different components of the Visual Basic Editor.
One thing you'll notice when clicking on any menu, is that several commands have a keyboard shortcut that is displayed at that point. Take a look, for example, at the Debug menu and notice all the keyboard shortcuts that appear on the right side of this image:



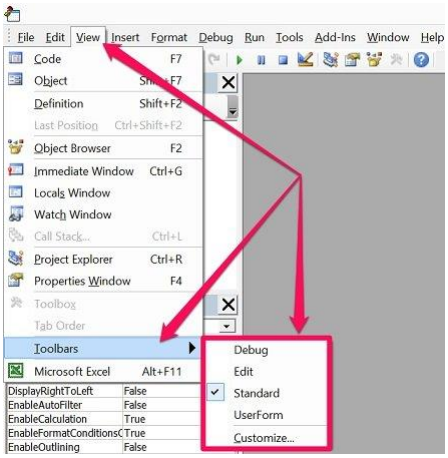
Component #2: Toolbar

Again, if you're a computer user, a toolbar is an item that you've probably seen many times before. You're probably aware that a **toolbar contains on-screen buttons, icons, menus and other similar elements that you can use while working with the VBE**.

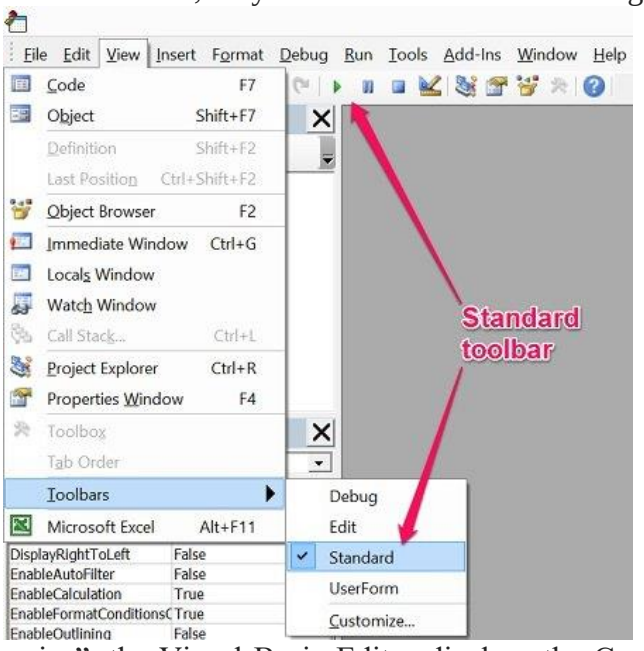
The toolbar that appears in the screenshot above is called the Standard toolbar. This is the only toolbar that the Visual Basic Editor displays by default. There are, however, 3 other basic toolbars:

- The Debug toolbar.
- The Edit toolbar.
- The UserForm toolbar.

In addition to the above, **the VBE gives you the possibility to customize the toolbars** in several ways.
You can change all of these settings by going to the View menu and selecting "Toolbars". The Visual Basic Editor displays a menu with the 4 different toolbars and the option to access the Customize dialog.



The toolbars with a checkmark to their left are those currently displayed by Excel. You can add or remove a checkmark in order to display or hide a particular toolbar by clicking on its name. For example, in the screenshot below, only the Standard toolbar is being displayed.

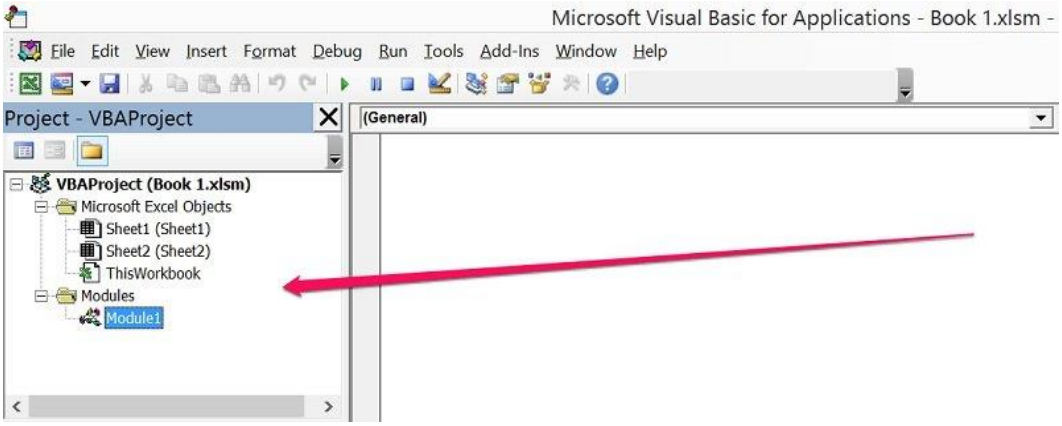


If you click on "Customize", the Visual Basic Editor displays the Customize dialog, which looks as follows:

Using this dialog box, you can control additional aspects regarding the toolbars that are displayed by the VBE. This includes, for example, the possibility of controlling the display of the Shortcut Menus toolbar or adding new toolbars.
You may be wondering **what toolbar display set up is commonly applied by VBA users**. In practice, there are different opinions.

- Some advanced VBE users use the default settings.
- However, other advanced VBA users display several toolbars.

You can also add commonly used commands that aren't by default in the Standard toolbar.
Component #3: Project Window / Project Explorer

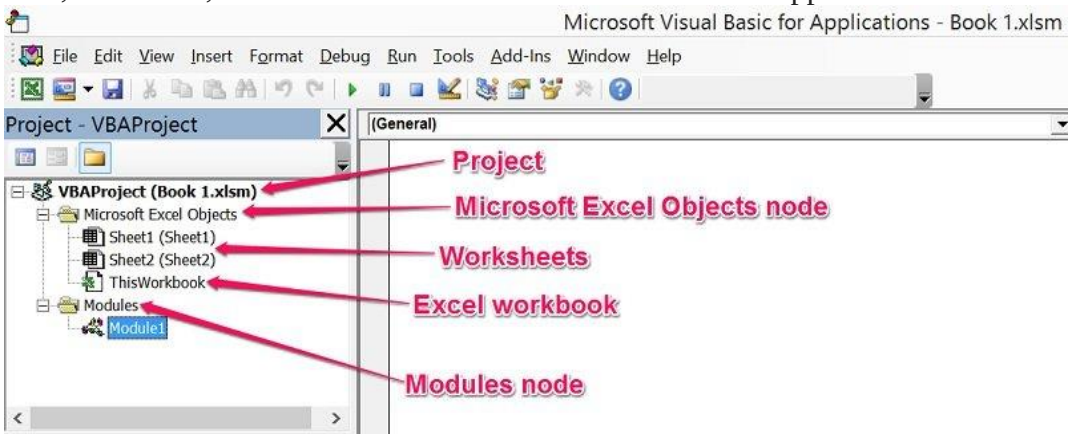


The Project Window, also known as the Project Explorer, is **useful for navigation purposes**. This is the section of the Visual Basic Editor where you'll be able to find every single Excel workbook that is currently open. This includes add-ins and hidden workbooks. More particularly, each Excel workbook or add-in that is open at the moment appears in the Project Explorer as a separate project.
A project is (basically/simply) a set of modules. If it makes it easier to understand you can take John Walkenbach's explanation in *Excel VBA Programming for Dummies*, who says that a project can be seen as "a collection of objects arranged as an outline".
As explained by Walkenbach in *Excel 2013 Power Programming with VBA*, each project may have the following nodes:

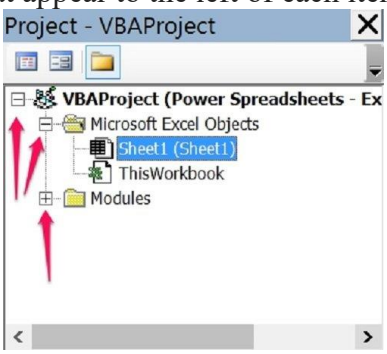
- A node called "Microsoft Excel Objects" always appears in any project. This node usually contains 2 types of objects:
  - **#1:** Each worksheet in the relevant Excel workbook. In other words, each of the worksheets is considered a separate object.

- **#2:** The Excel workbook itself, called “ThisWorkbook”.
- The Modules node appears when the project contains VBA modules.
- If the project contains UserForm objects, which are used to create custom dialog boxes, the Project Explorer displays a node called “Forms”.
- A project can also contain class modules (modules that define a class) and, in that case, the Project Window displays a node called “Class Modules”.
- Finally, if a project has references, there is a node called “References”.

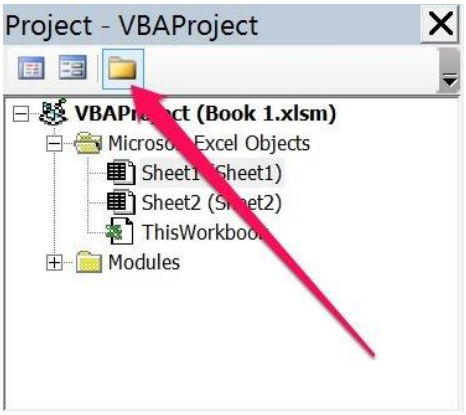
Let's take a look at how all of this looks in the VBE interface:
In the screenshot below, the only project that appears is the Excel workbook “Book 1. xlsm”. Within the Microsoft Excel Objects node, you can see that the Excel workbook has 2 worksheets. Finally, this particular project contains 1 VBA module and, therefore, the Modules node is visible. There are, however, no UserForm objects, class modules or references. Therefore, the Forms, Class Modules and References nodes don't appear.



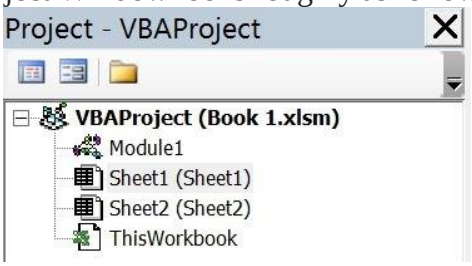
You can expand or contract the items that appear in the outline by double-clicking on them or by clicking on the “+” or “-” that appear to the left of each item, depending on the case.



You can also control whether the items that are displayed in the Project Window appear in a hierarchical or a non-hierarchical list. You change this setting by clicking on the Toggle Folders button of the Project Window.



The screenshot above shows items being displayed in a hierarchical list. When displayed in a non-hierarchical list, the Project Window looks roughly as follows:

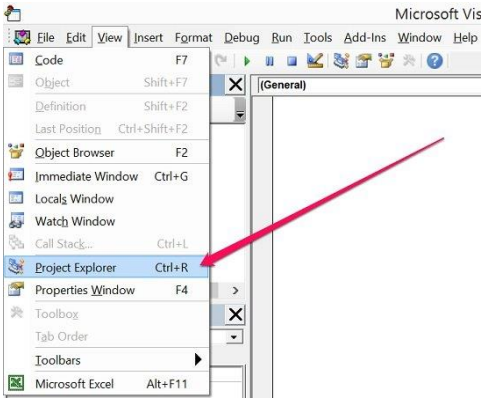


You can also hide or unhide the Project Explorer itself. I explain how to do this below.

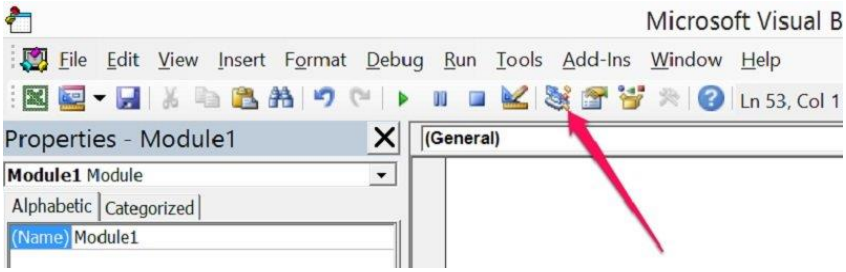
How To Display The Project Window

If you can't see the Project Explorer, you can make the Visual Basic Editor display it by using any of the following methods:

- Clicking on “Project Explorer” in the View menu.



- Clicking on the Project Explorer icon in the toolbar.

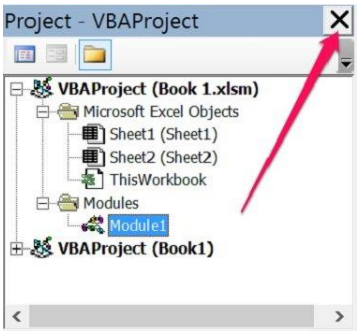


- Using the keyword shortcut “Ctrl + R”.

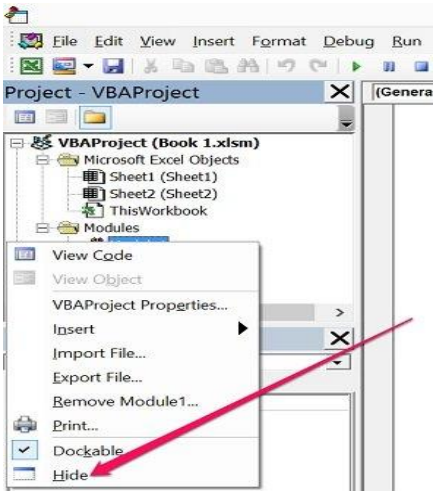
How To Hide The Project Window

You can hide the Project Explorer by using either of the following methods:

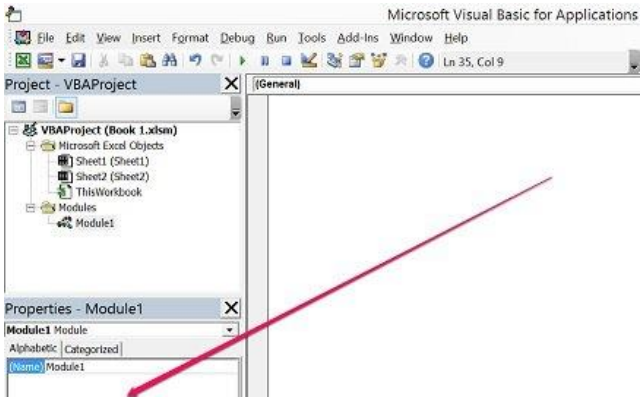
- Clicking on the close button of the Project Window.



- Right-clicking anywhere on the Project Explorer and selecting “Hide”.



Component #4: Properties Window



The Properties Window **displays the properties of the object that is currently selected** in the Project Explorer and allows you to edit those properties.

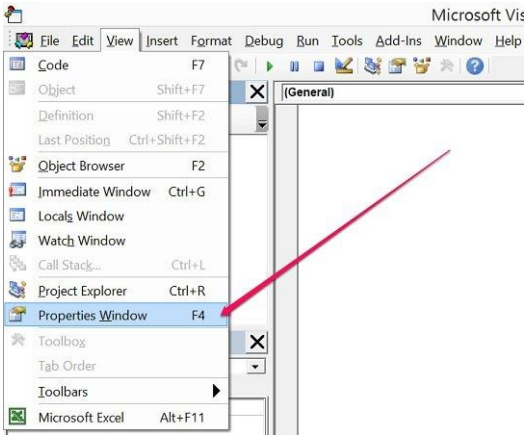
Just as with the Project Window, you can hide or unhide the Properties Window. You're likely to (eventually) work with the Properties Window, particularly in the context of creating UserForms. If you're just beginning to use the VBE, you probably won't need this window too much.

In any case, let's take a look at how you can hide or unhide the Properties Window.

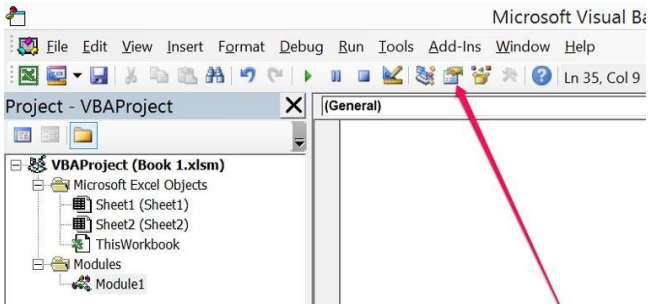
How To Unhide The Properties Window

You can get the Visual Basic Editor to show the Properties Window by using any of the following methods.

- Clicking on “Properties Window” within the View menu.



- Clicking on the Properties Window icon.

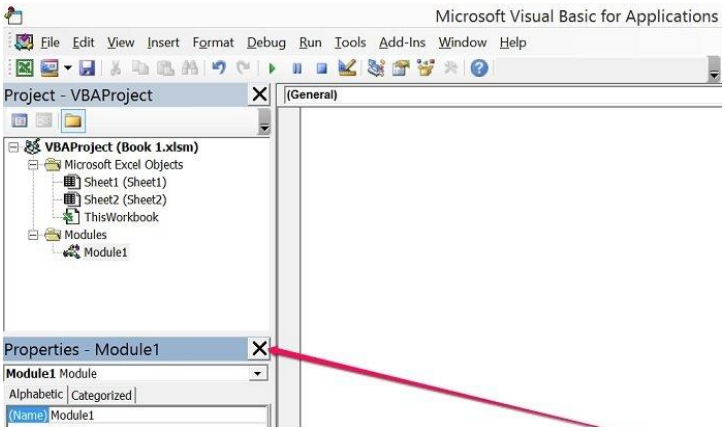


- Using the “F4” keyboard shortcut.

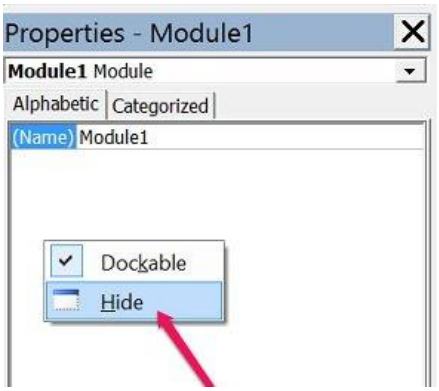
How To Hide The Properties Window

You can get the Visual Basic Editor to hide the Properties Window by doing either of the following:

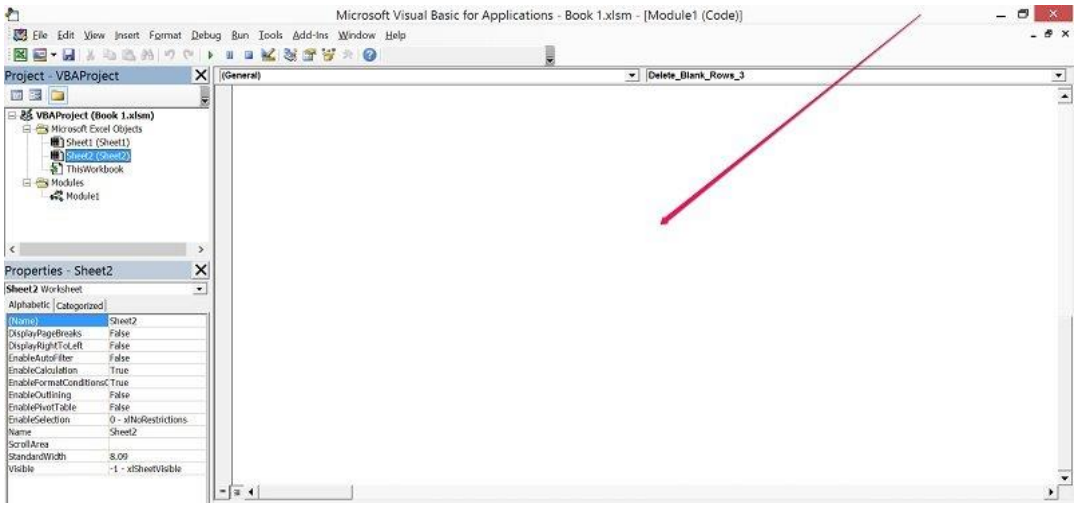
- Click on the Close button of the Properties Window.



- Right-click on the Properties Window and select “Hide”.



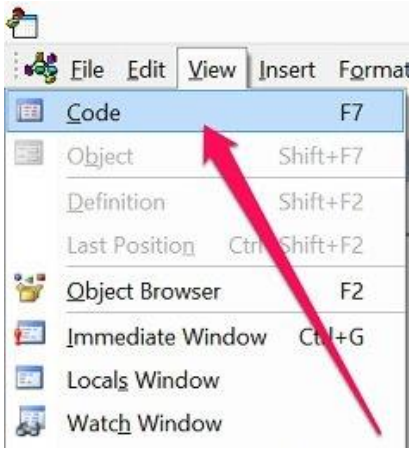
Component #5: Programming Window / Code Window / Module Window



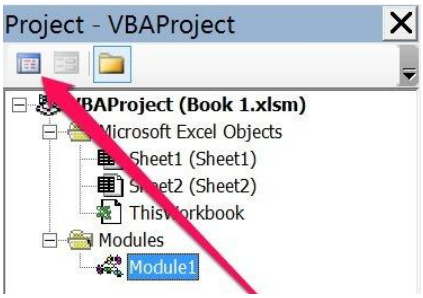
As you may expect, the Code Window of the Visual Basic Editor is **where your VBA code appears, and where you can write and edit such code**. At the beginning, though, the Programming Window is empty as in the screenshot above.

There is a Code Window for every single object in a project. You can access the window of a particular object by going to the Project Explorer and doing any of the following:

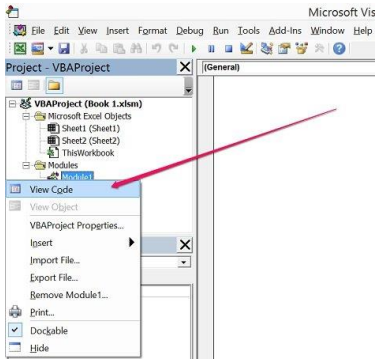
- Double clicking on the object. The main exception to this rule are UserForms. If you double-click on a UserForm, the Visual Basic Editor displays the UserForm in Design view, a topic I'll cover in future tutorials.
    - Selecting the object and, then, clicking on “Code” in the View menu.



- Selecting the object and clicking on the View Code icon that appears at the top of the Project Explorer.



- Right-clicking on the object and selecting “View Code”.



- Using the keyboard shortcut “F7”.

Component # 6: Immediate Window

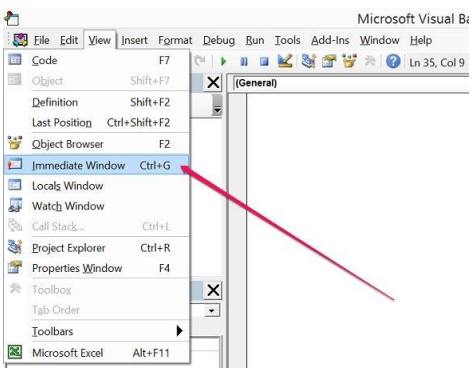
The main purpose of the Immediate Window is to **help you noticing errors, checking or debugging VBA code**.

The Immediate Window is, by default, hidden. However, as with most of the other windows, you can unhide it. Let' take a look at how you can do both the hiding and the un-hiding:

How To Unhide The Immediate Window

You can unhide the Immediate Window by doing either of the following:

- Clicking on “Immediate Window” in the View menu.



- Using the “Ctrl + G” keyboard shortcut.

However, as explained in *Excel VBA Programming for Dummies*, if you're just getting started with the VBE “this window won't be all that useful”. Therefore, if you're just beginning to work with macros and Visual Basic for Applications, you probably don't need to display the Immediate Window.

If you're a more advanced user, you'll probably want to have the Visual Basic Editor show the Immediate Window, since this can be very useful.

How To Hide The Immediate Window

You can hide the Immediate Window using either of the following methods:

- Click the Close button.

- Right-click on the Immediate Window and select “Hide”.

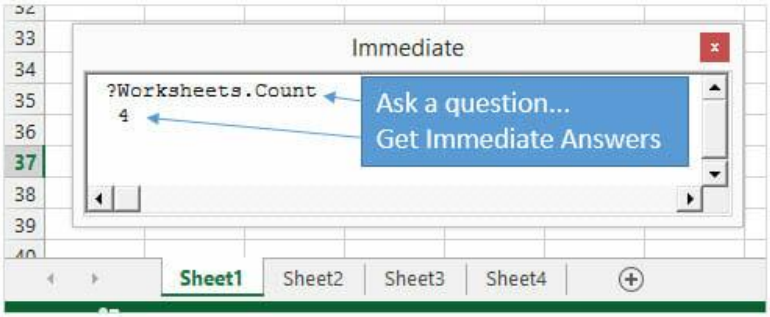
You already know that:

- The VBE allows you to customize several aspects.
- On your way to becoming a macro and VBA expert you'll probably spend a significant amount of time working with the Visual Basic Editor.

6. What is an immediate Window and what is it used for?

The VBA Immediate Window is a great tool that can help any Excel user, even if you are not writing macros. Learn how to get answers about your Excel file, quickly run macros, debug your code, and more. Free file to download contains VBA code samples.

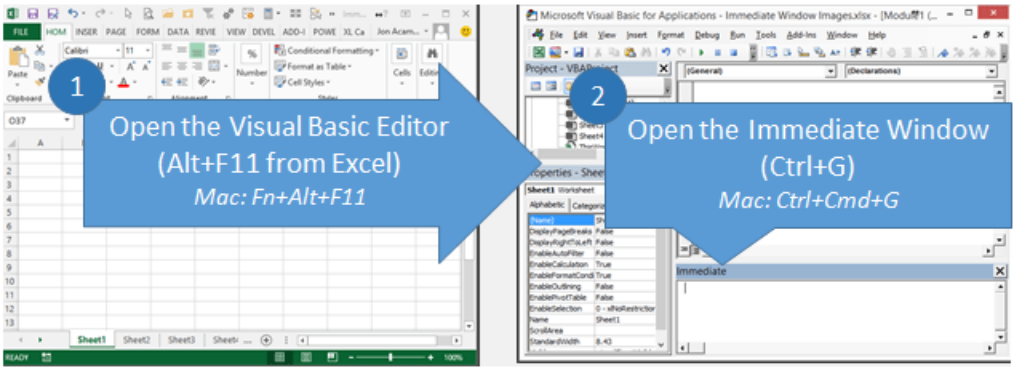
The VBA Immediate Window is an awesome tool that allows you to get immediate answers about your Excel files, and quickly execute code. It is built into the Visual Basic Editor, and has many different uses that can be very helpful when writing macros, debugging code, and displaying the results of your code.



Every Excel user can benefit from the Immediate Window, even if you're not writing macros. This post will explain 5 different uses for the Immediate Window. Once you understand the capabilities of this tool, you will find yourself using it all the time.

Where is the Immediate Window?

The Immediate window is located in the **Visual Basic Editor window**.



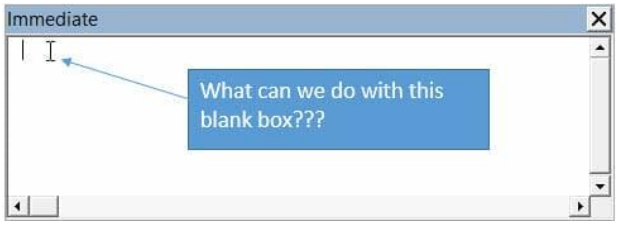
The fastest way to get to the Immediate Window is to:

1. Press **Alt+F11** (hold Alt key, then press F11 key) from anywhere in Excel. The Visual Basic Editor window will open. (Mac version is Fn+Alt+F11)
2. Pressing **Ctrl+G** opens the Immediate Window and places the text cursor in it. Begin typing your code. (Mac version is Ctrl+Cmd+G)

When you open the VB Editor (Alt+F11) you might see the Immediate Window automatically appear in the bottom right corner. This is its default location. If it's not there you can press Ctrl+G or View menu > Immediate Window.

This Blank Box is Magical!

When you click inside the Immediate Window you will just see a blank box with the text cursor flashing. At first glance this doesn't look too exciting, but the Immediate window can be a very **powerful and useful tool**.



Think of it like a blank cell in a worksheet. It's pretty boring until you add a formula to it, right? Well the Immediate Window is very similar, so let's look at **5 examples that will help you get the most out of this magical box**.

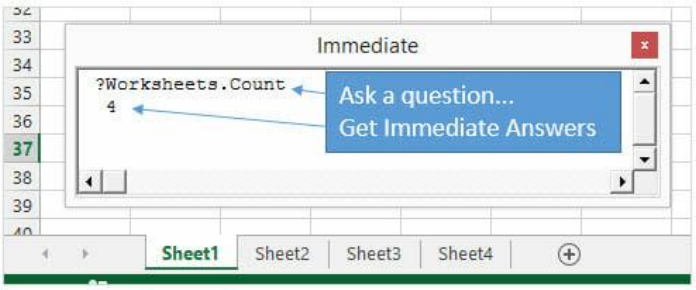
#1 – Get Info About The Active Workbook

The simplest use for the Immediate window is to quickly get information about the workbook that you currently have open and active in the background. **You can evaluate any line of VBA code in the Immediate Window**, and it will immediately give you the result.

For example, to find out how many sheets are in the active workbook, type the following line of code in the Immediate window and then press the Enter key.

Activeworkbook.Worksheets.Count

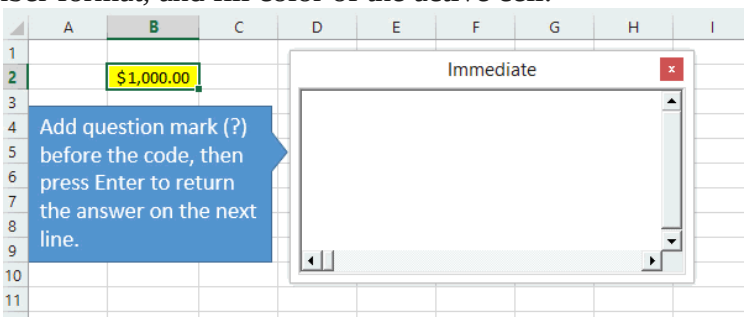
The answer will be displayed on the next line of the Immediate window, directly under the code.



Ask a question, any question…

Putting the question mark (?) at the beginning of the statement tells the Immediate window that we are asking it a question, and expecting a result.

The following screencast shows a few examples of how we can use the Immediate window to get the value, number format, and fill color of the active cell.



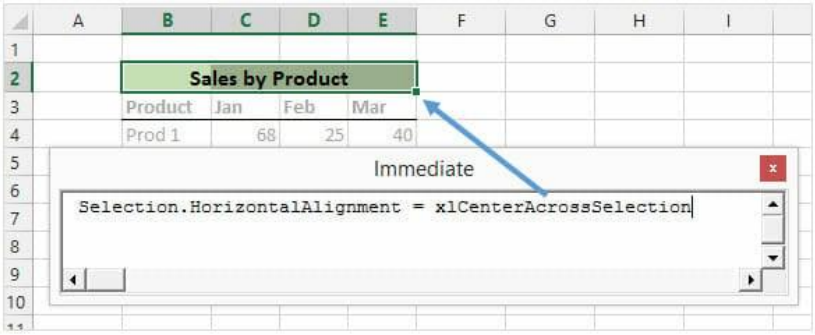
Notice that the Immediate Window **displays the intellisense** as I type. Intellisense is the drop-down menu that displays the properties, methods, and members of the object I'm referencing. This makes it very fast and easy to type code in the Immediate Window.

You can download the free sample workbook that contains a few more useful examples.

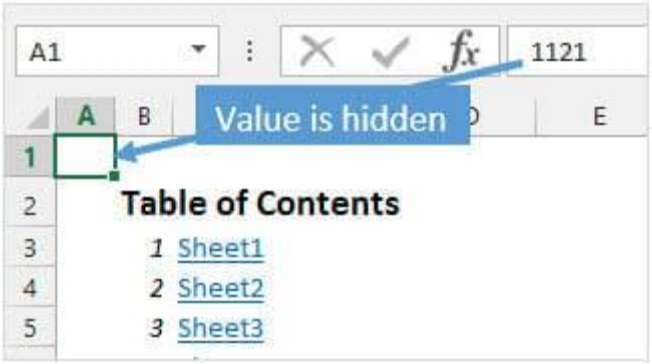
#2 – Execute a Line of VBA Code

You don't have to write a whole macro if you just need to perform one line of code to your workbook.

Remove the question mark at the front of the statement and the Immediate Window will execute or perform that line of code.

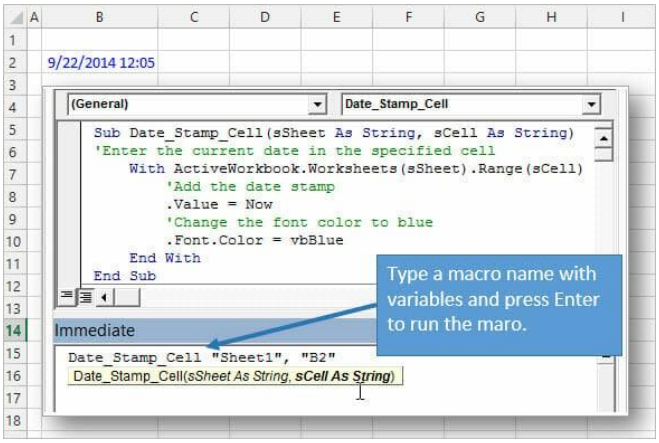


Selection.HorizontalAlignment = xlCenterAcrossSelection
The image above shows how to format the selected cells with the Center Across Selection alignment.
You can also use the following line of code to make a worksheet "very hidden".
Worksheets("Sheet1").Visible = xlVeryHidden
Another example is to hide the contents of a cell by making its font color the same as its fill (background) color.
Range("A1").Font.Color = Range("A1").Interior.Color



I use this line of code in Tab Hound's Table of Contents tool to hide some settings stored in cell A1. Even if the user changes the fill color of the sheet, the contents in cell A1 will still be hidden after the code is run.

#3 – Run a Macro
You can run a macro from the Immediate Window by typing the name of the macro (procedure), and pressing Enter.
Of course you can also do this by pressing the F5 key or the Run Macro button in the VB Editor, but what if your macro contains arguments?
A macro cannot be run from within the procedure if it contains arguments. However, you can call the macro from the Immediate Window.
The example below is a simple macro that enters the current date (Now) in the cell , and changes the font color to blue (Color = vbBlue). This macro requires two arguments to be passed to it to run, the worksheet name and cell address where the date stamp will be entered.



For a macro like this you will typically be calling it from another macro and specifying the arguments in the macro that is calling it. But if you just want to test the macro that contains arguments, you can use the Immediate Window to call it.

This is great for writing and debugging code. You might not want to run the entire stack of procedures (macros) in the code, but you can use the Immediate Window to only call that specific macro you're working on.

The example above shows how you can specify the arguments after the macro name. For arguments that are string variables (text), you will need to wrap the variable in quotation marks.

As you can see in the image, the intellisense is available in the Immediate Window, which makes it easy to specify the arguments for the macro.

The code in the image above is included in the free sample file you can download below.

VBA Immediate Window Examples.zipDownload

#4 – View Debug.Print Info

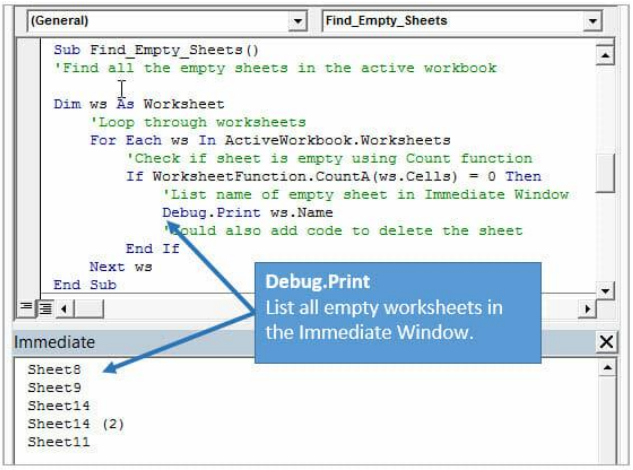
Have you ever seen VBA code on the internet that contains a line similar to the following?

Debug.Print xxxxx

With that "xxxxx" being some variable that the code is calculating or producing.

Debug.Print is telling VBA to print that information in the Immediate Window. This can be useful when you want to see the value of a variable in a certain line of your code, without having to store the variable somewhere in the workbook or show it in a message box. It is especially useful when you are writing or debugging code.

The example below is a macro that loops through all the sheets in the workbook and checks to see if each sheet is empty (not used). If the sheet is empty then it is listed in the Immediate Window using Debug.Print.

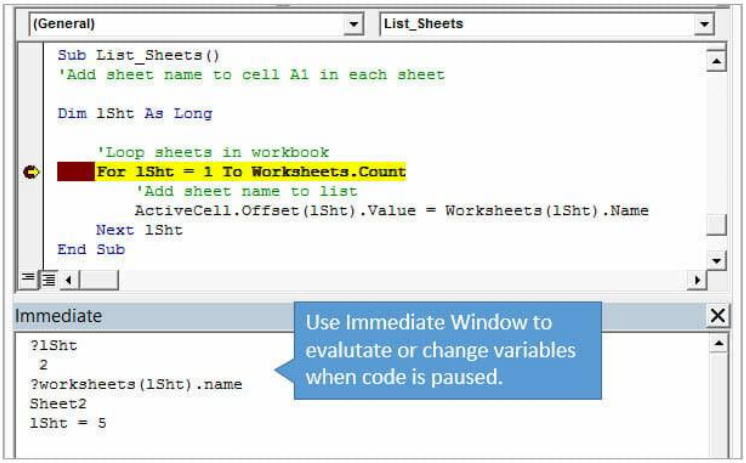


The ultimate goal of this macro may be to delete all empty (blank) sheets in the workbook, but we might want to test the code first before actually deleting any sheets. The Debug.Print line creates a list of empty sheets in the Immediate Window, and we can then manually check each of those sheets to make sure they are really blank.

#5 – Get or Set a Variable's Value

The Immediate Window can also be used to get answers about the procedure (macro) that is currently running. If you are stepping through your code (F8) or add a break point (F9) or add a STOP line in your code, then the code will pause. **When the code is paused you can use the Immediate Window to get information about any variables** or objects that are referenced in the code.

The following example is a macro that creates a list of all the sheets in the active workbook. There are plenty of different ways to write this code, but in this example I use the "lSht" variable in a For Next loop to loop through the worksheets and then add the sheet name to the active sheet.



I added a break point (F9) in the code to pause the code when that line is executed. With the code paused, the Immediate Window can be used to evaluate or change variables. In the image above I used the question mark to check the value of the lSht variable. Then I used the variable to get the sheet name of the sheet that is currently being processed in the loop.

Finally I changed the lSht variable to 5 by using the equals sign (lSht = 5). This will effectively skip some of the sheets in the loop because I changed the variable from 2 to 5.

This is a simple example, but the Immediate Window can come in handy when stepping through code.