18 Excel VBA Macro Shortcuts for 2018

Bottom line: Learn 18 shortcuts to save time with writing VBA macros for Excel or other Office applications.

Skill level: Intermediate

18 VBA Macro Tips & Shortcuts

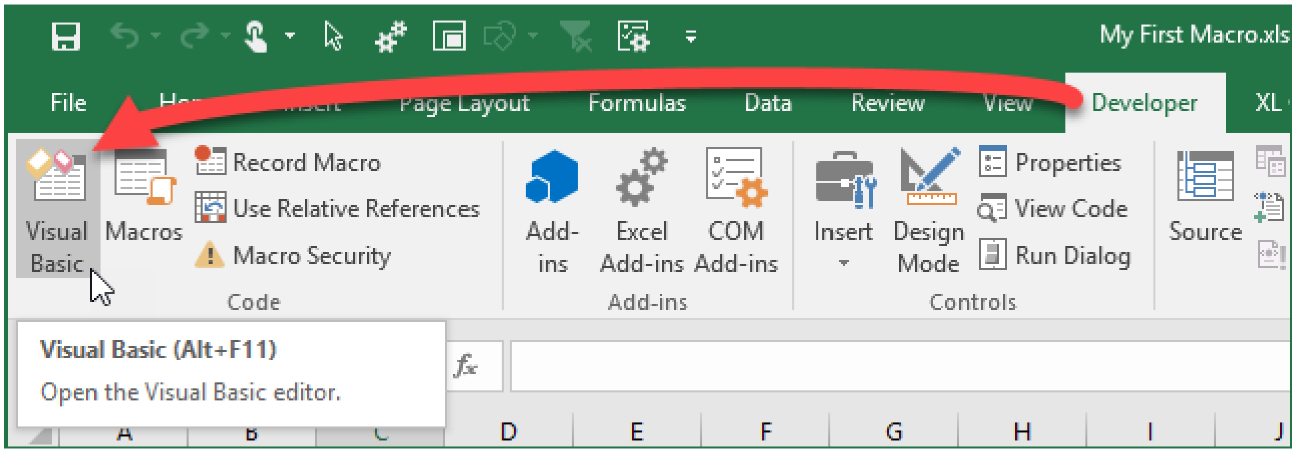
At the beginning of last year I wrote a popular post on 17 Excel Shortcuts for 2017. I decided to continue the tradition this year and write a post on tips & shortcuts that will save you time when writing VBA macros.

These are some of my favorite tips, and also some favorites from the blog, YouTube Channel, and my VBA Pro Course. If you're not familiar with VBA yet, but always wanted to learn, then checkout my upcoming webinar on "The 7 Steps to Getting Started with Macros & VBA".

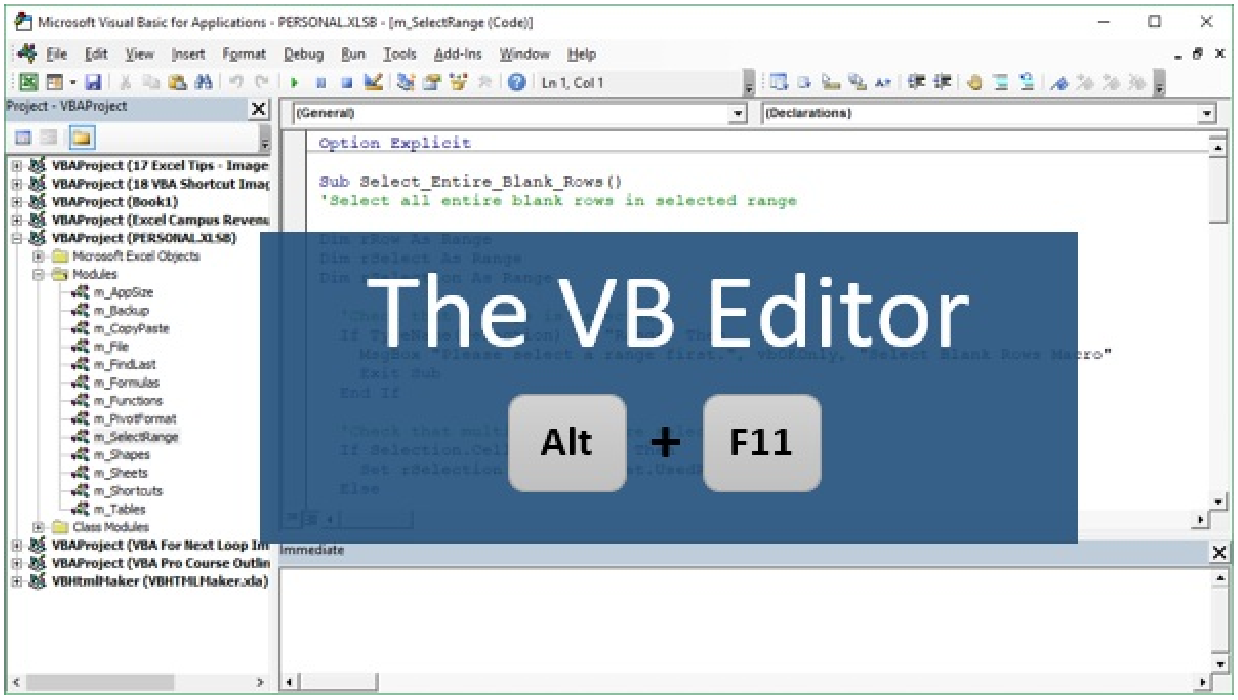
Obviously there are a TON of shortcuts & tips for VBA. So please leave a comment at the bottom of the post with your favorite tips. This way we can all learn from each other! ☺

#1 – Alt+F11 to Open the VB Editor

The VB Editor is the application we use to write macros and create userforms. It can be opened by clicking the Visual Basic button on the Developer tab in Excel.



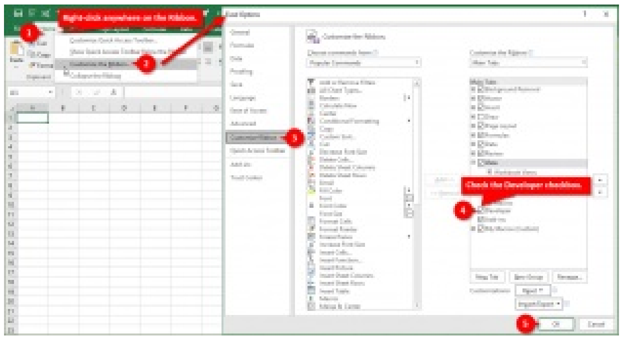
The keyboard shortcut to open the VB Editor in any Windows version of Excel is `Alt`+`F11`.


Microsoft Visual Basic for Applications - PERSONAL.XLSB - [m_SelectRange (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 1, Col 1
Project - VBAProject
VBAProject (17 Excel Tips - Image
VBAProject (18 VBA Shortcut Imag
VBAProject (Book1)
VBAProject (Excel Campus Revenu
VBAProject (PERSONAL.XLSB)
Microsoft Excel Objects
Modules
m_AppSize
m_Backup
m_CopyPaste
m_File
m_FindLast
m_Formulas
m_Functions
m_PivotFormat
m_SelectRange
m_Shapes
m_Sheets
m_Shortcuts
m_Tables
Class Modules
VBAProject (VBA For Next Loop Im
VBAProject (VBA Pro Course Outlin
VBHtmlMaker (VBHTMLMaker.xla)
(General)
(Declarations)
Immediate
The VB Editor
Alt
+
F11

```
Option Explicit

Sub Select_Entire_Blank_Rows()
'Select all entire blank rows in selected range
```

The shortcut in the Mac version is `Opt`+`F11` or `Fn`+`Opt`+`F11`.

If you don't see the Developer tab in the Ribbon, click the image to learn how to enable it:



The Fn (Function) Keys on Laptops

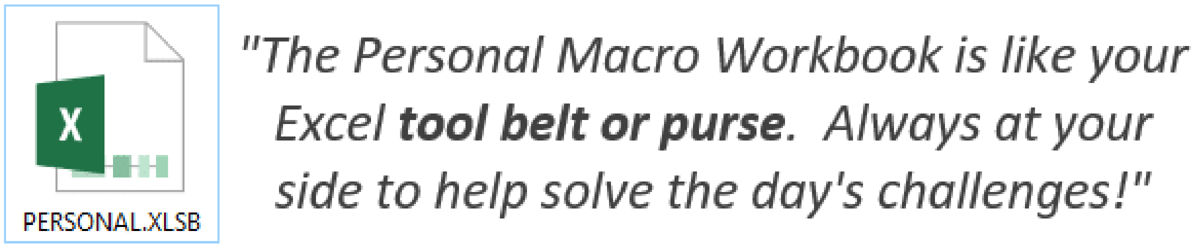
If you are using a laptop keyboard then you might also need to press & hold the `Fn` key before pressing `F11`. The function keys on laptops are typically multi-use keys, and require the `Fn` key to be pressed to activate the function keys (`F1` – `F12`).

Some laptops have a Fn Lock feature that makes the function keys primary, which means you won't have to press the Fn key when pressing F1-F12.

Checkout my article on the Best Keyboards for Excel Keyboard Shortcuts to learn more.

#2 – Store Your Macros in The Personal Macro Workbook

The Personal Macro Workbook (PMW) is a great place to store the macros you use often. This workbook opens in the background every time we open Excel, making our macros easily accessible.

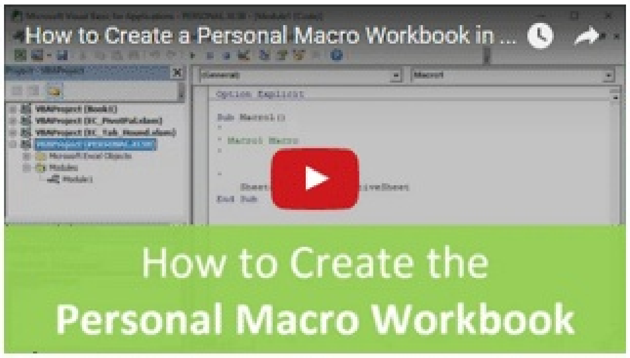

PERSONAL.XLSB
"The Personal Macro Workbook is like your Excel **tool belt or purse**. Always at your side to help solve the day's challenges!"

We can also create custom toolbars with buttons to run our macros. I like to think of the PMW as our Excel tool belt that can save us time with task we do every day.

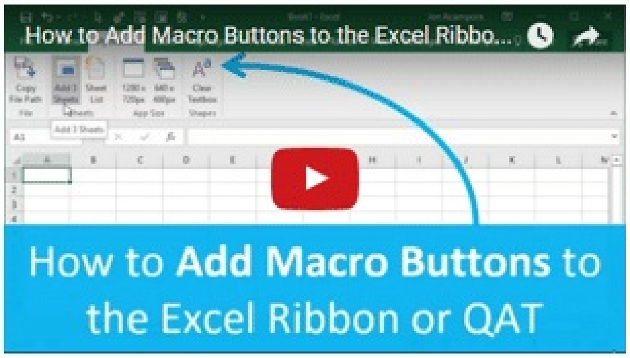
For example, I have an article with a macro that creates a list of unique values on a new blank sheet. We can store this macro in our PMW and assign it to a button in the ribbon or keyboard shortcut, and then run it any time on any open workbook.

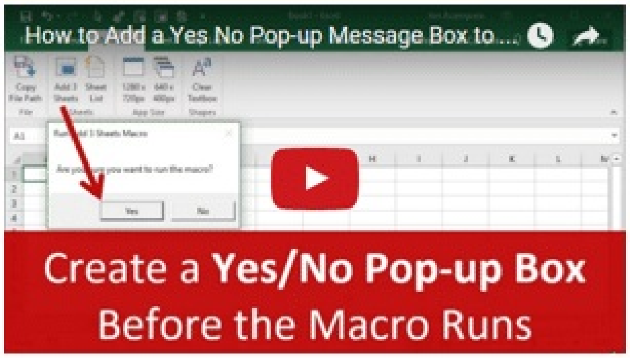
Checkout my free video series on how to create your Personal Macro Workbook to learn more. There is also a video in that series on how to add macro buttons to a custom ribbon.

Watch the Personal Macro Workbook Video Series


How to Create a Personal Macro Workbook in ...
How to Create the Personal Macro Workbook

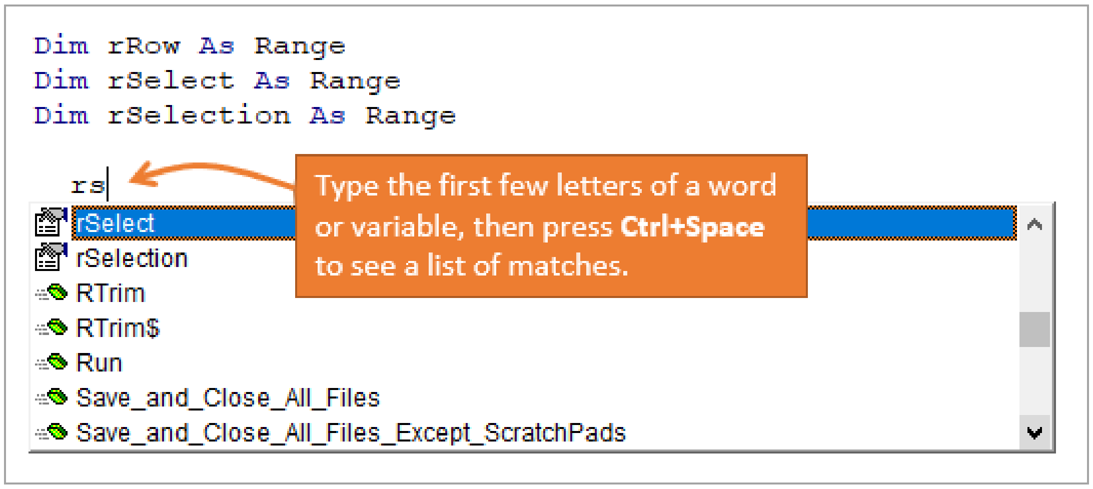
Where Is The Personal Macro Workbook Store...
Where is the Personal Macro Workbook file stored?


How to Add Macro Buttons to the Excel Ribbo...
How to Add Macro Buttons to the Excel Ribbon or QAT


How to Add a Yes No Pop-up Message Box to ...
Create a Yes/No Pop-up Box Before the Macro Runs

#3 – Ctrl+Space to Auto Complete

This is definitely one of the keyboard shortcuts I use the most in VBA. When we are typing code, `Ctrl`+`Space` opens the Intellisense drop-down menu that contains a list of matching objects, properties, methods, constants, and variables.



```
Dim rRow As Range
Dim rSelect As Range
Dim rSelection As Range

  rs
```

rSelect
rSelection
RTrim
RTrim$
Run
Save_and_Close_All_Files
Save_and_Close_All_Files_Except_ScratchPads
Type the first few letters of a word or variable, then press Ctrl+Space to see a list of matches.

To use the `Ctrl`+`Space` shortcut in the VB Editor:

1. Start typing a line of code like ActiveCell.
2. After typing the first few letters, press Ctrl+Space
3. You will see a list of all VBA words that start with Act.
4. Press the Up/Down arrows to select the word
5. Then press Tab or Enter to complete the word.

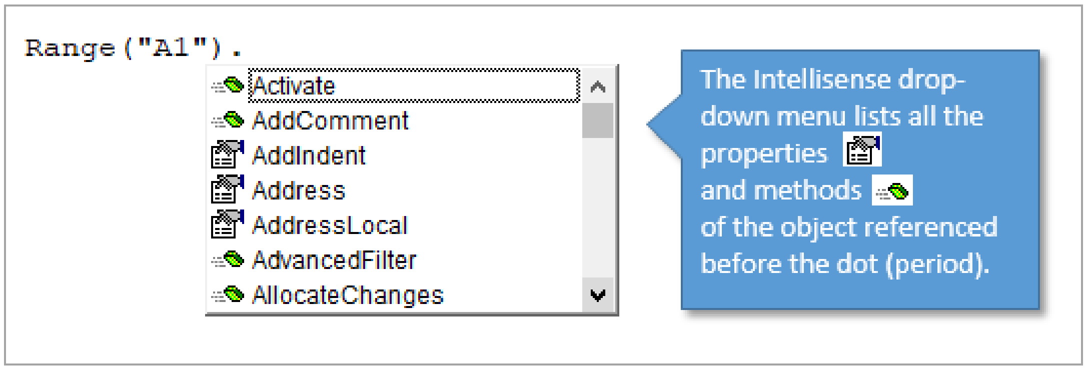
There are two major benefits to this shortcut:

1. It saves time with having to type long words and variable names.
2. It prevents typos because VBA is completing the words for you.

These benefits can be a **HUGE time saver** when debugging your code.

#4 – Intellisense for Worksheets

We typically also see the Intellisense drop-down menu after typing a period (.) in the VB Editor.

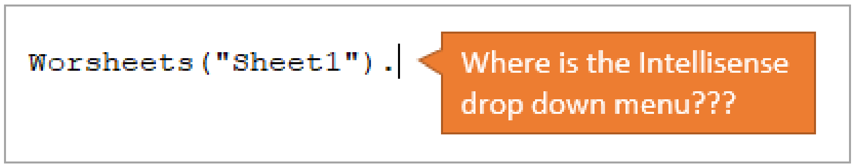


```
Range("A1").
```

Activate
AddComment
AddIndent
Address
AddressLocal
AdvancedFilter
AllocateChanges
The Intellisense drop-down menu lists all the properties and methods of the object referenced before the dot (period).

However, sometimes it doesn't work. One common case is with the Worksheets property.

If we type `Worksheets("Sheet1").` , we do NOT see the Intellisense menu. This can be frustrating and cause you to think that Intellisense is not working.

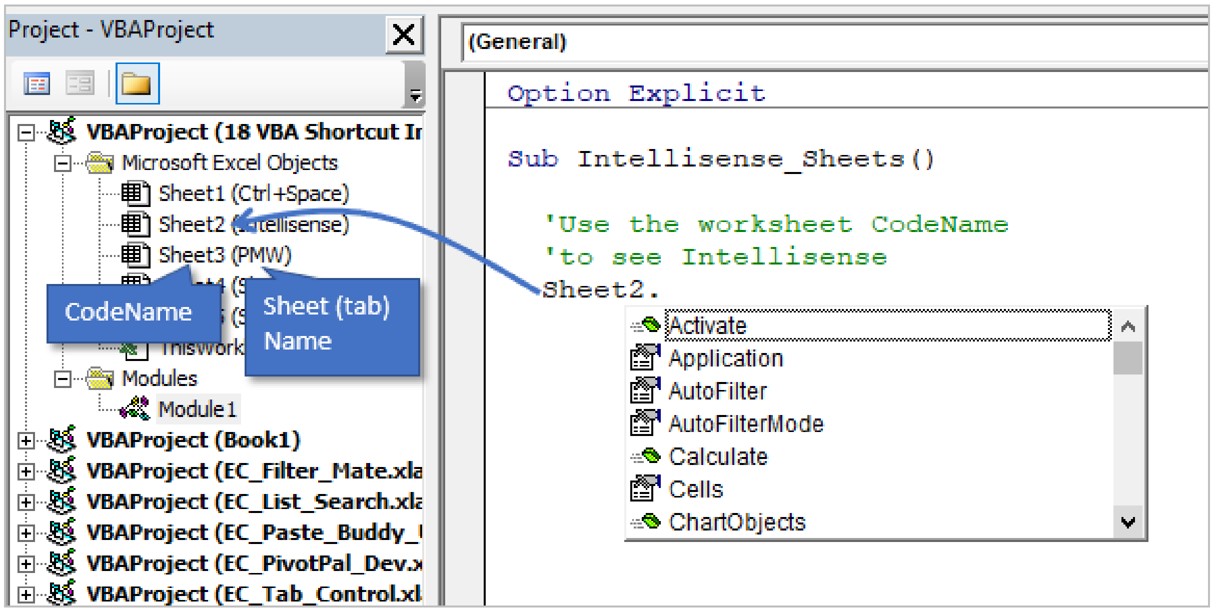


```
Worsheets("Sheet1").
```

The reason it doesn't work is because the Worksheets property can contain reference to one or multiple sheets. Depending on the reference, the properties and methods will be different for each case. It would be great if Intellisense was still intelligent enough to recognize this, but it's just one of those things we have to live with…

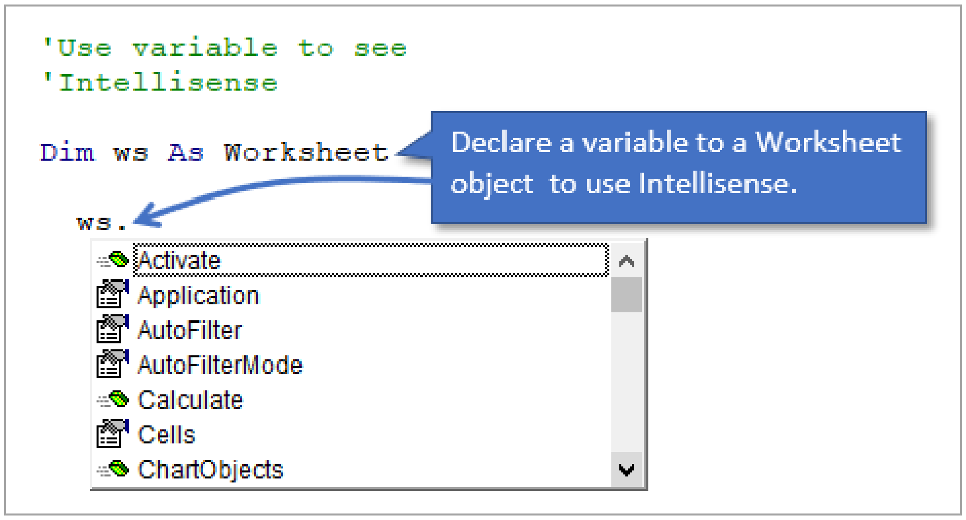
There are two ways to get around it and see Intellisense for worksheets:

1. Use the CodeName of the worksheet we want to reference. This is my preferred method for referencing worksheets because the code will not break if a user changes a sheet name.



```
Option Explicit

Sub Intellisense_Sheets()

  'Use the worksheet CodeName
  'to see Intellisense
  Sheet2.
```

2. Set the worksheet to a Worksheets object variable first. Then when we type the variable name followed by a dot (`ws.`), the Intellisense menu will appear.



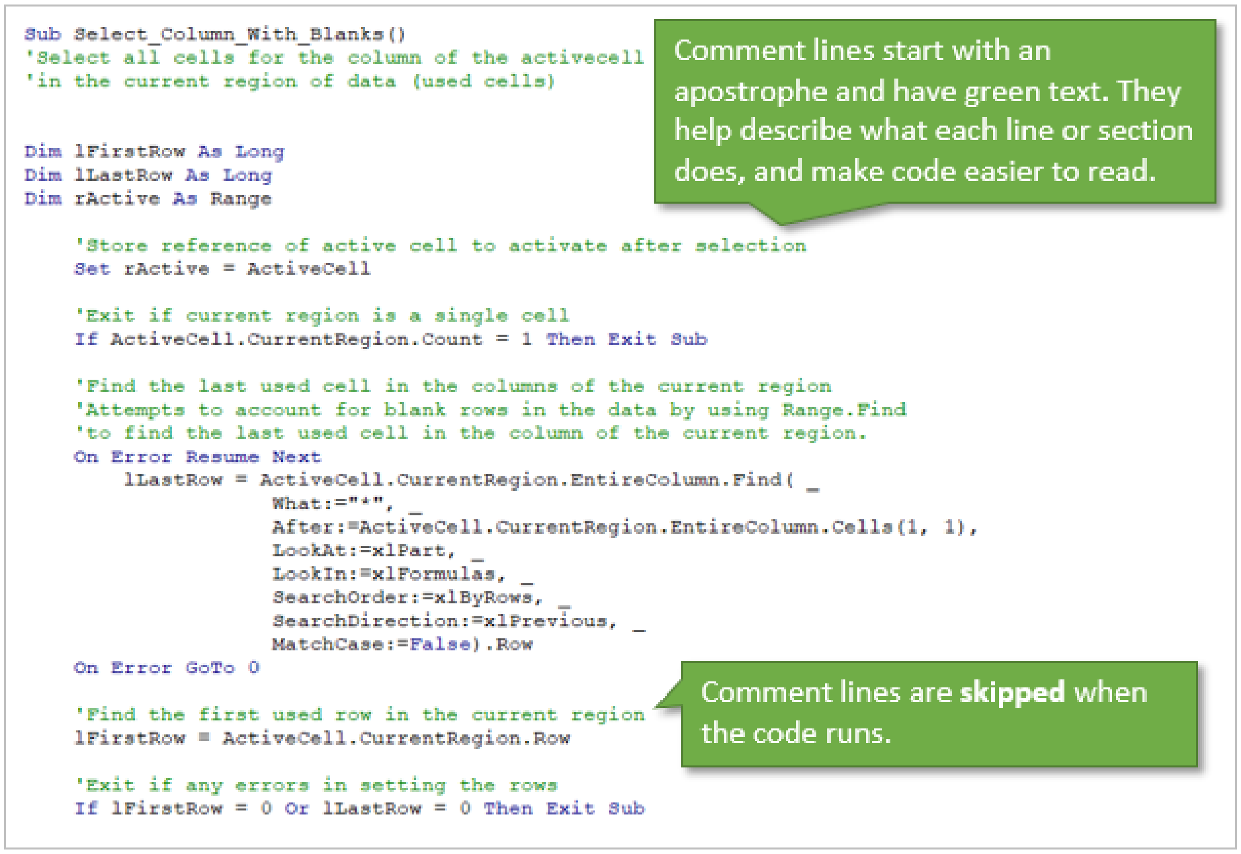
```
'Use variable to see
'Intellisense

Dim ws As Worksheet

   ws.
```

#5 – Use Comments Liberally

We can add comments to our code that help explain what each section of code does.

To create a comment in VBA you type an apostrophe at the beginning of the line. Once you move the text cursor off the line, the text will turn green.

The green text makes it easy to distinguish comments when reading code. VBA completely ignores comment lines, and you can add as many as you'd like.



```
Sub Select_Column_With_Blanks()
'Select all cells for the column of the activecell
'in the current region of data (used cells)


Dim lFirstRow As Long
Dim lLastRow As Long
Dim rActive As Range

    'Store reference of active cell to activate after selection
    Set rActive = ActiveCell

    'Exit if current region is a single cell
    If ActiveCell.CurrentRegion.Count = 1 Then Exit Sub

    'Find the last used cell in the columns of the current region
    'Attempts to account for blank rows in the data by using Range.Find
    'to find the last used cell in the column of the current region.
    On Error Resume Next
        lLastRow = ActiveCell.CurrentRegion.EntireColumn.Find( _
                    What:="*", _
                    After:=ActiveCell.CurrentRegion.EntireColumn.Cells(1, 1),
                    LookAt:=xlPart, _
                    LookIn:=xlFormulas, _
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlPrevious, _
                    MatchCase:=False).Row
    On Error GoTo 0

    'Find the first used row in the current region
    lFirstRow = ActiveCell.CurrentRegion.Row

    'Exit if any errors in setting the rows
    If lFirstRow = 0 Or lLastRow = 0 Then Exit Sub
```

Comment lines start with an apostrophe and have green text. They help describe what each line or section does, and make code easier to read.
Comment lines are **skipped** when the code runs.

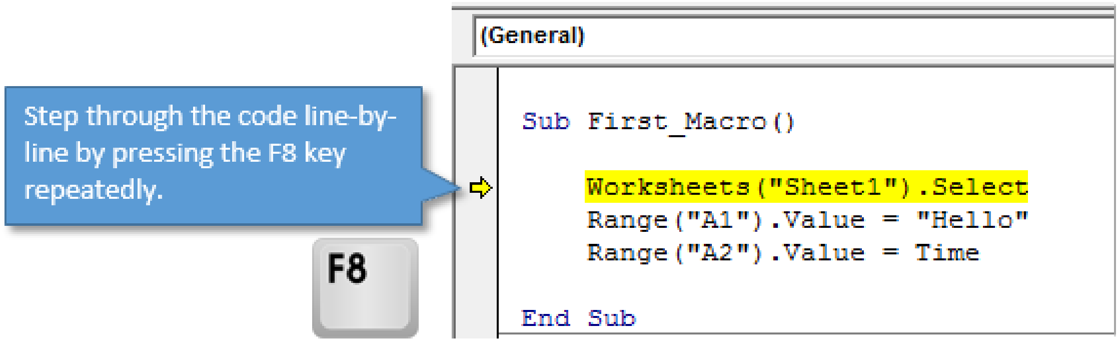
Commenting your code is somewhat of a controversial topic. Some developers believe that properly written code should speak for itself, and that you don't need to add extra comments. I see their point, but this **doesn't work for me for two reasons**.

1. When I come back to my own projects months/years later, I don't remember what the entire macro does. Comments are like headings in this blog post, and make it easy to scan through the macro to find the section you are looking for. They also quickly tell us what each section does.
2. If you are going to share your VBA project or eventually hand it off to someone else for maintenance, then it will be much easier for them to learn your code if you add a lot of comments. I call this "politely planning your legacy". 🙂

#6 – F8 to Step Through Each Line of Code

The keyboard shortcut to step through each line of code is `F8`. The Mac equivalent for Step Into/Through is `Cmd`+`Shift`+`I`.

This allows us to test and debug each line of code in our macros. We can also open Excel side-by-side with the VB Editor (or on a separate monitor) to see the actions being performed in Excel as each line runs.


Step through the code line-by-line by pressing the F8 key repeatedly.
F8
(General)

```
Sub First_Macro()

    Worksheets("Sheet1").Select
    Range("A1").Value = "Hello"
    Range("A2").Value = Time

End Sub
```

A lot times this can help you quickly find an error with a sheet or range reference.

To use the Step Into/Through shortcut:

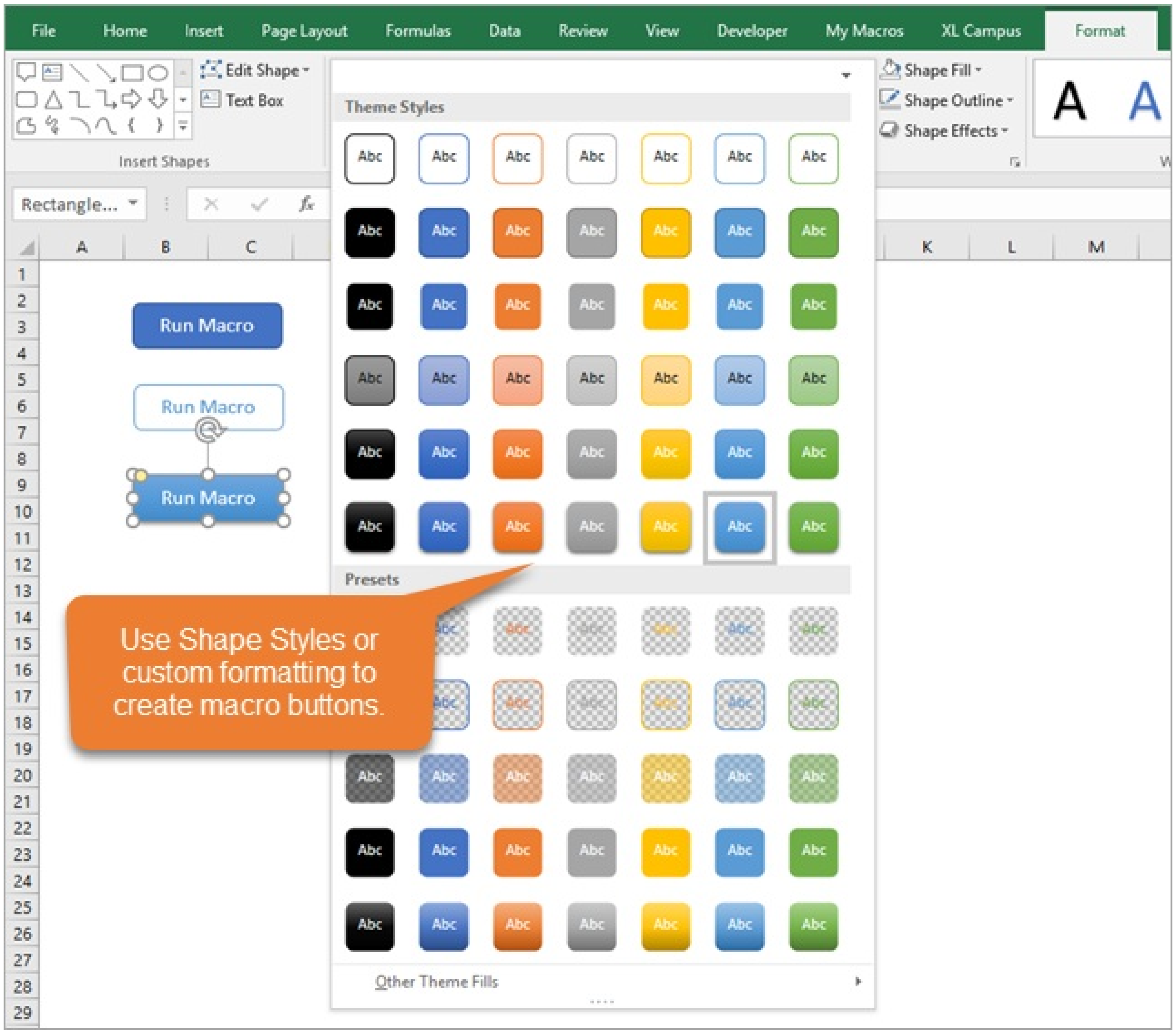
1. Click inside the macro you want to run. You can click any line of code. The macro will always start at the top.
2. Press F8.
3. The macro name will be highlighted yellow.
4. Press F8 again to run that line and highlight the next line.
5. Continue to press F8 to run each line.

It's important to note that the yellow highlighted line has NOT been run yet. It will be run when you press F8 again.

I cover this technique in more detail in my **free** upcoming webinar on "The 7 Steps to Getting Started with Macros & VBA".

#7 – Assign Macros to Shapes

The sheet controls for buttons that run macros are a bit dated looking. Fortunately we can also use any shape in Excel to run a macro. The shapes can be colored and formatted to make them look more like modern buttons you find on the web and mobile apps.



To assign a macro so a shape:

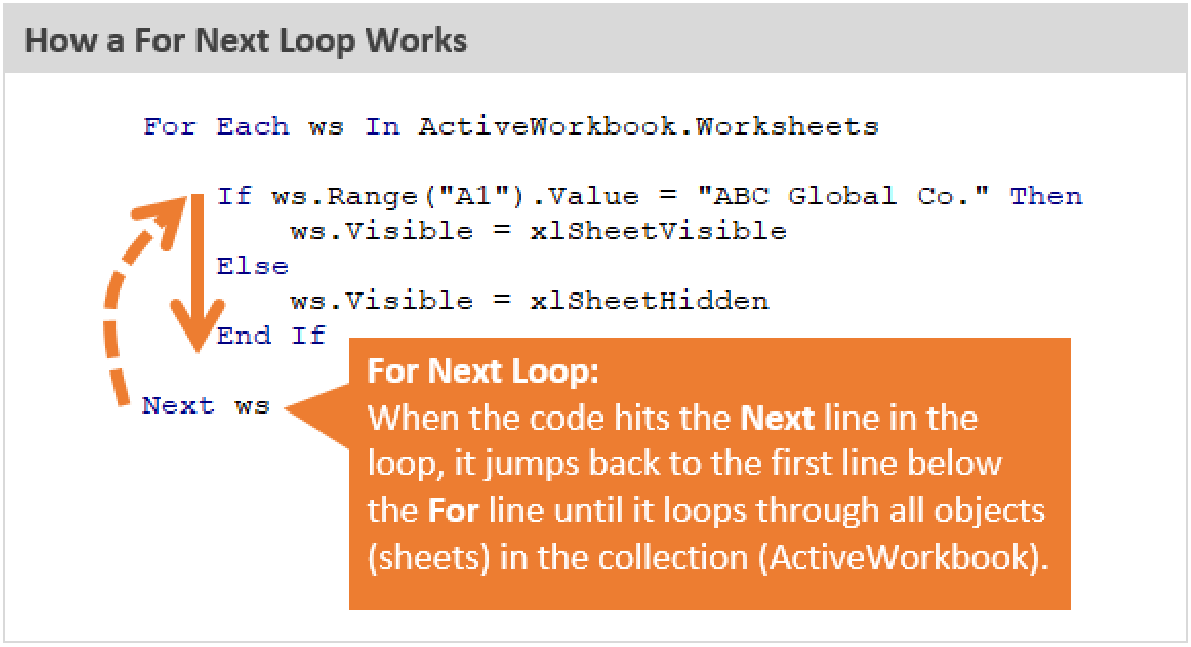
1. Insert a shape on a worksheet, and format it to your liking. This will usually be a rectangular or circle shape that contains text.
2. Right-click the shape and choose "Assign Macro…".
3. Select the macro from the list and press OK. The macro will usually be one that is stored in the same workbook as the shape.
4. Click off the shape by selecting a cell in the worksheet.
5. When you hover the shape the cursor will change to the hand pointer. Clicking the shape will run the macro.

I recommend having a Yes/No message box appear before the macro actually runs. This prevents any accidental button presses. Checkout my video on adding a Yes/No message box to your macros. It's part of my video series on the Personal Macro Workbook.

#8 – Automate Repetitive Tasks with The For Next Loop

We tend to do a lot of the same tasks over and over again in Excel. This can be tasks like: applying formatting to multiple ranges/sheets, creating a list of sheets, copying data to each workbook, setting filters on each pivot table, etc.

Loops are one of the most powerful tools in VBA that allow us to automate these tasks. The loop will loop through each item in a collection (think worksheets in a workbook or cells in a range), and perform whatever code you like on each item.



```
For Each ws In ActiveWorkbook.Worksheets

    If ws.Range("A1").Value = "ABC Global Co." Then
        ws.Visible = xlSheetVisible
    Else
        ws.Visible = xlSheetHidden
    End If

Next ws
```

For Next Loop:
When the code hits the **Next** line in the loop, it jumps back to the first line below the **For** line until it loops through all objects (sheets) in the collection (ActiveWorkbook).

There are a few different types of loops, but the For Next Loop is the most common. Checkout my in-depth article on The For Next Loop in VBA for more details on this must-know coding technique.

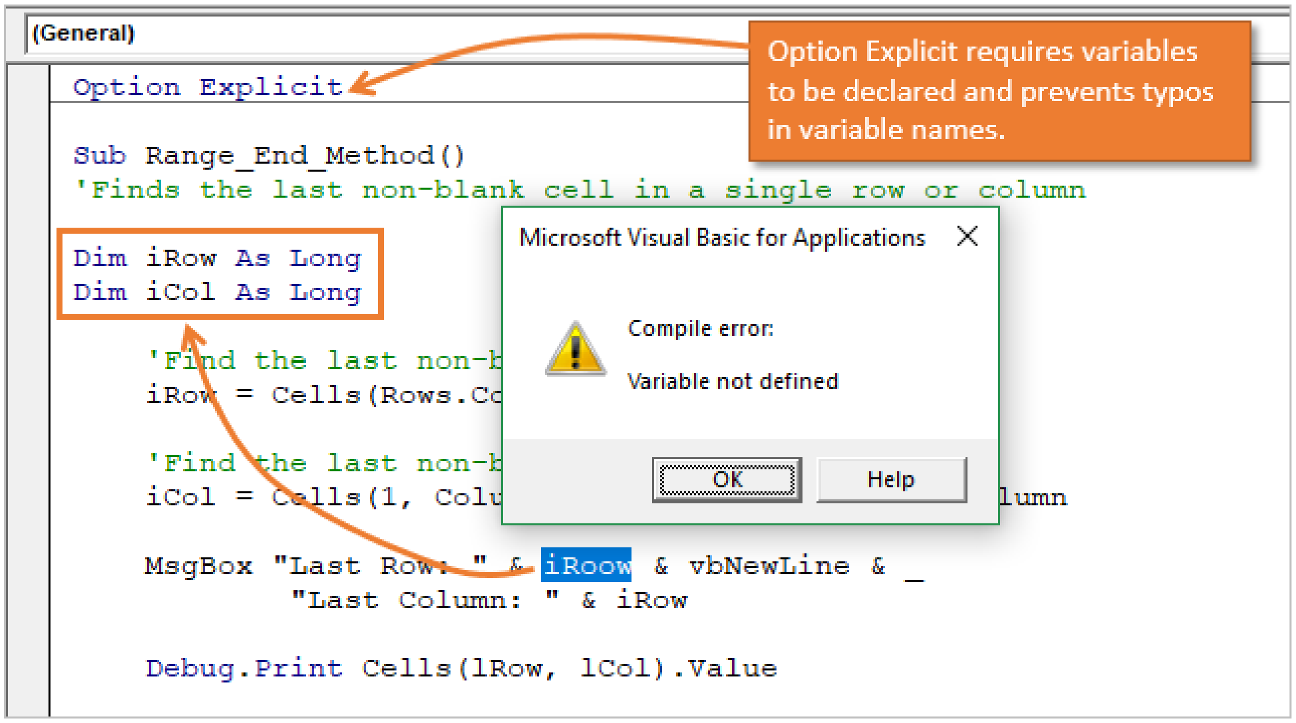
In my **free** upcoming webinar on "The 7 Steps to Getting Started with Macros & VBA", I explain how to use a For Next Loop to list all the sheets in any workbook. This creates a quick table of contents with a click of a button.

#9 – Use Option Explicit

This is another controversial topic, but I require (politely ask) that all members of my VBA Pro Course use Option Explicit.

So, what is it and why?

Option Explicit requires us to declare all variables. When we see lines of code with Dim statements at the top of a macro, this is declaring a variable.


(General)
Option Explicit requires variables to be declared and prevents typos in variable names.

```
Option Explicit

Sub Range_End_Method()
'Finds the last non-blank cell in a single row or column

Dim iRow As Long
Dim iCol As Long

    'Find the last non-b
    iRow = Cells(Rows.Co

    'Find the last non-b
    iCol = Cells(1, Colu                                  lumn

    MsgBox "Last Row: " & iRoow & vbNewLine & _
            "Last Column: " & iRow

    Debug.Print Cells(lRow, lCol).Value
```

```
Microsoft Visual Basic for Applications

Compile error:

Variable not defined
```

We are basically telling VBA to create the variable in memory to be used later while the code is running. We can then set values or references to objects to these variables in the macro below the Dim statement.

The MAJOR benefit with Option Explicit is that it **prevents typos and saves time**. The VB Editor will throw a Compile Error: Variable Not Defined when you try to run the code if a variable in the code is not declared. It will also highlight the variable so you can declare it or fix a typo.

If you don't have Option Explicit on and misspell a variable, the code will still run and can produce errors in the results. If your macro is long then then it can take A LOT of time to find these typos. Trust me. Been there done that!

Option Explicit prevents these errors and helps keep you sane. ☺

To turn Option Explicit on you simply type the words Option Explicit at the top of the code module. You can also have the VB Editor automatically add the words to new code modules by going to Tools > Options > check the "Require Variable Declaration" checkbox. Option Explicit will now appear at the top of each NEW code module you create.

#10 – Excel Tables (ListObjects)

There are a lot of benefits of using Excel Tables in our workbooks. They save time with formatting data, auto-fill formulas, and work great as the source of a pivot table.

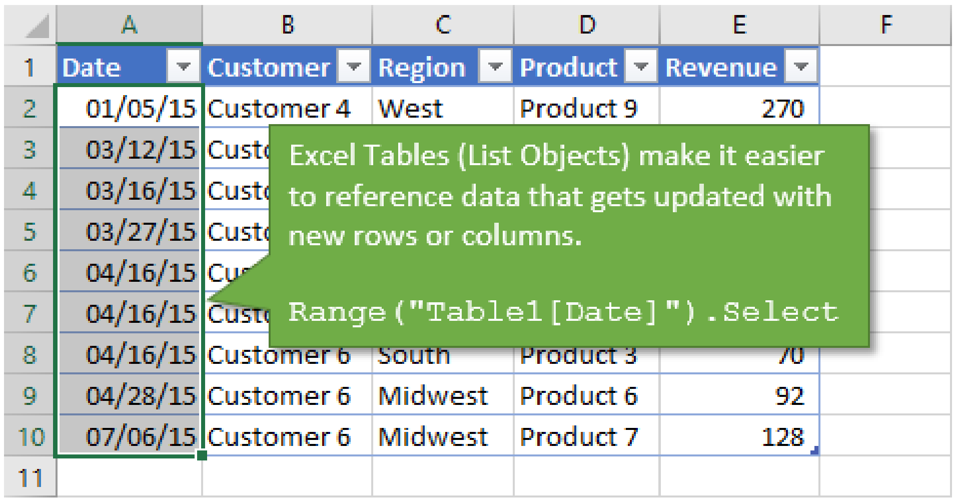
Excel Tables also make it much easier to write VBA code for dynamic ranges of data. This is a list or data set where the number of rows or columns is constantly changing as you get new/updated data.

For example, the following line of code references the cells in range A2:A15.

```
Range("A2:A10").Font.Bold = True
```

That is a hard-coded range reference. If you add new data to the bottom, you will have to manually change the code to include the new rows.

However, if we store the data in an Excel Table and reference the Table column, we don't have to worry about this.



	A	B	C	D	E	F
1	Date	Customer	Region	Product	Revenue	
2	01/05/15	Customer 4	West	Product 9	270	
3	03/12/15	Custo				
4	03/16/15	Custo				
5	03/27/15	Custo				
6	04/16/15	Cu				
7	04/16/15	Custo				
8	04/16/15	Customer 6	South	Product 3	70	
9	04/28/15	Customer 6	Midwest	Product 6	92	
10	07/06/15	Customer 6	Midwest	Product 7	128	
11						

Excel Tables (List Objects) make it easier to reference data that gets updated with new rows or columns.

```
Range("Table1[Date]").Select
```

The following line of code references the same column.

```
Range("Table1[Date]").Font.Bold = True
```

The **advantage** here is that the code will **automatically include new rows added to the Table**. No manual updating or maintenance of the code is needed.

We can also reference Excel Tables with the **ListObjects** object, properties, and methods in VBA.

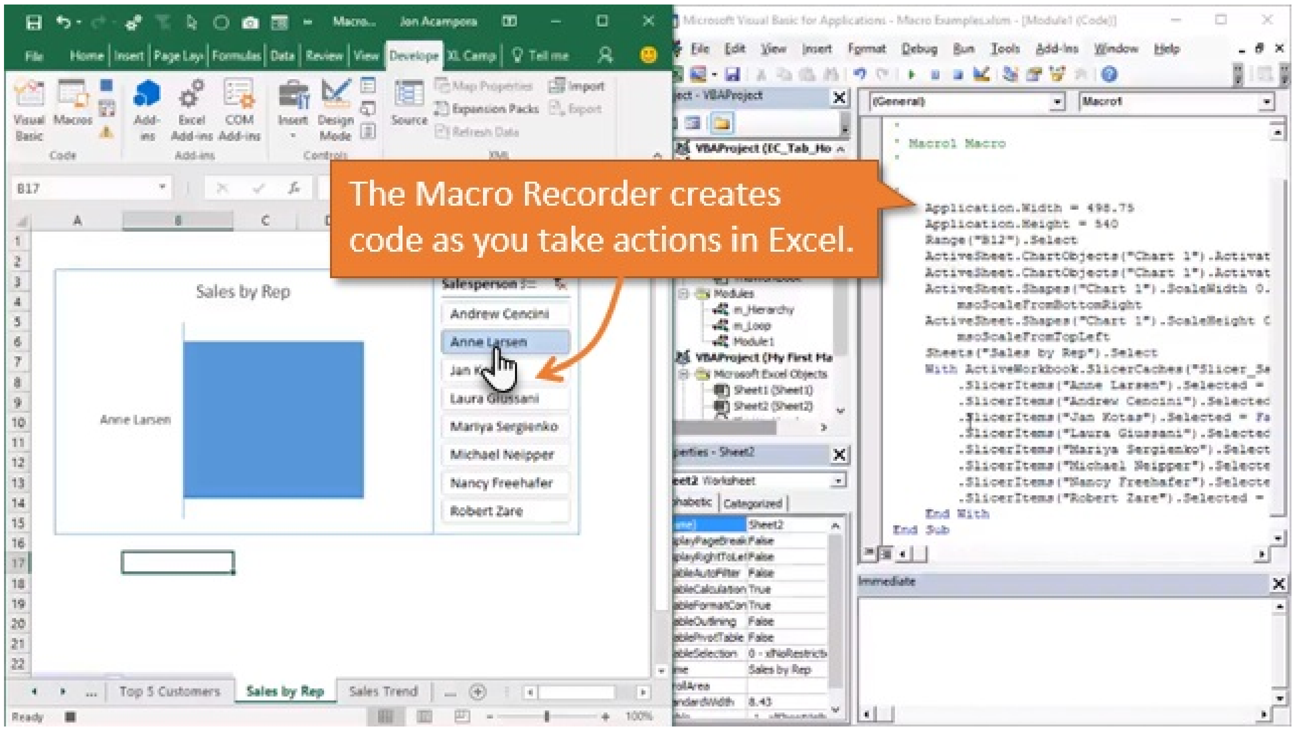
```
ActiveSheet.ListObjects("Table57").ListColumns("Date").DataBodyRange.Font.Bold = True
```

There are definitely some advantages to using ListObjects when it comes to modifying the Table structure (adding/deleting rows/columns) and properties, and looping through the Table. Checkout my good friend Chris Newman's article on ListObjects in VBA for more examples.

#11 – Get Code with the Macro Recorder

The **Macro Recorder** is an amazing feature of Excel and VBA. It **creates VBA code as we take actions in Excel**.

For example, after we turn the macro recorder on, we can go do our normal work in Excel like writing formulas or copying and pasting data. The macro recorder will create all the VBA code for these actions, and store it in a code module.



```
'
' Macro1 Macro
'

    Application.Width = 498.75
    Application.Height = 540
    Range("B12").Select
    ActiveSheet.ChartObjects("Chart 1").Activat
    ActiveSheet.ChartObjects("Chart 1").Activat
    ActiveSheet.Shapes("Chart 1").ScaleWidth 0.
        msoScaleFromBottomRight
    ActiveSheet.Shapes("Chart 1").ScaleHeight C
        msoScaleFromTopLeft
    Sheets("Sales by Rep").Select
    With ActiveWorkbook.SlicerCaches("Slicer_Sa
        .SlicerItems("Anne Larsen").Selected =
        .SlicerItems("Andrew Cencini").Selectec
        .SlicerItems("Jan Kotas").Selected = Fa
        .SlicerItems("Laura Giussani").Selectec
        .SlicerItems("Mariya Sergienko").Select
        .SlicerItems("Michael Neipper").Selecte
        .SlicerItems("Nancy Freehafer").Selecte
        .SlicerItems("Robert Zare").Selected =
    End With
End Sub
```

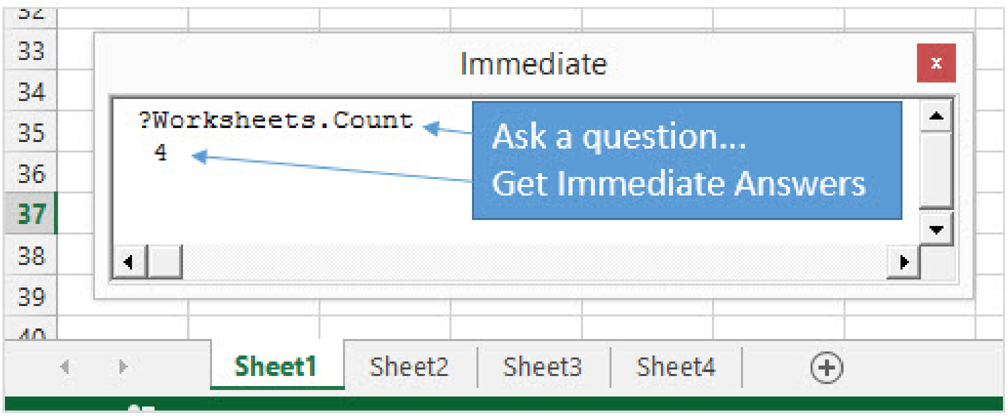
This is a great tool to use when we are first start with macros. And it's also a great tool for learning and getting snippets of code. The Excel object model is **massive**, and it's impossible (for me) to memorize all the property, method, and object references. So the macro recorder is a great way to get some code for a pivot table, list object, slicer, shape, or any other object you are not familiar with.

The macro recorder also has its **limitations**. It's NOT going to create code for loops, if statements, error handling, message boxes, etc. We need to learn how to write code to implement these more advanced techniques that allow us to fully automate processes and create applications in Excel.

Checkout my **free** upcoming webinar on "The 7 Steps to Getting Started with Macros & VBA".

#12 – The Immediate Window

The Immediate Window in the VB Editor allows us to run a single line of code. We can run an action (method) on an object, or return the result of the code back to the Immediate Window. Maybe we want to determine how many worksheets are in a workbook.



```
?Worksheets.Count
 4
```

Type ?Worksheets.Count and hit Enter. The result will be displayed on the line below.

The Immediate Window is also the place that the Debug.Print method outputs to.

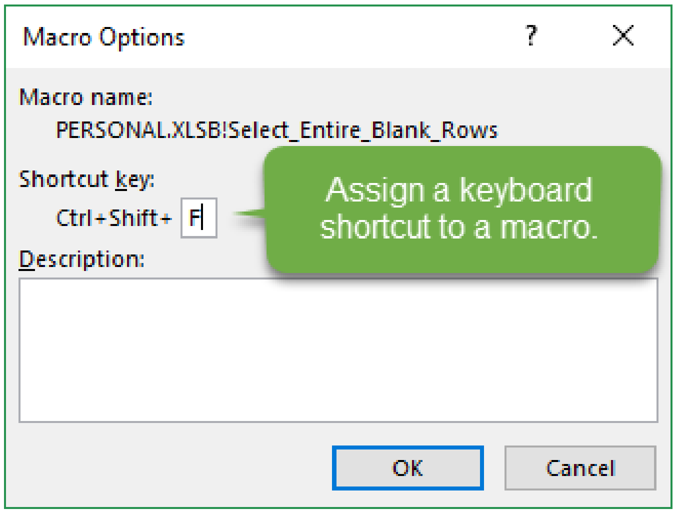
The keyboard shortcut to open the Immediate Window in the VB Editor is `Ctrl`+`G`

Checkout my article on 5 Uses for the Immediate Window in VBA for more details.

#13 – Assign a Keyboard Shortcut to a Macro

We can also assign keyboard shortcuts to run macros that we use frequently. I recommend storing these macros in your Personal Macro Workbook.

To assign the keyboard shortcut:



Macro name: PERSONAL.XLSB!Select_Entire_Blank_Rows

Shortcut key: Ctrl+Shift+ F

1. Press the Macros button on the Developer or View Tab in the ribbon.
2. Select the file that contains the macro from the Macros In drop down.
3. Select the macro from the list box.
4. Press the “Options…” button.
5. Type the letter in the Shortcut key box that you want to assign the macro to. All shortcuts will start with Ctrl. You can hold the Shift key while typing the letter to create a Ctrl+Shift shortcut. This is usually recommended because most of the Ctrl+key combinations already have dedicated keyboard shortcuts in Excel.
6. Press OK and close the macros window.
7. You can now press the shortcut key combination to run the assigned macro.

#14 – Check That a Range is Selected

Sometimes you will want to make sure the user has a range (cells) selected before your macro runs. If they have a shape (chart, slicer, etc.) selected then this might cause errors in your code.

For example, I have a macro that deletes the blank rows in the selected range. For this macro to run properly the user needs to have a range of cells selected first.

Here is code that will check if a range is selected.

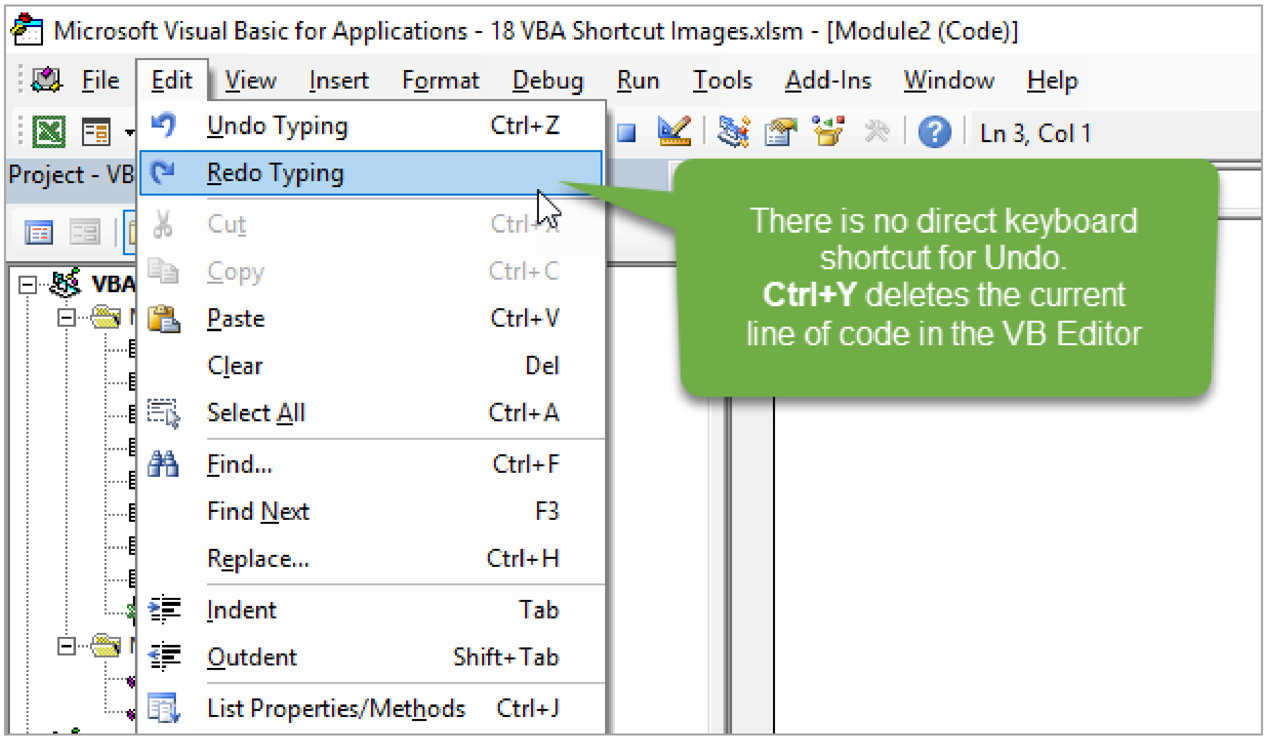
```
'Check that a range is selected
If TypeName(Selection) <> "Range" Then
  MsgBox "Please select a range first.", vbOKOnly, "Select Range"
  Exit Sub
End If
```

The TypeName function returns the data type or name of the object for a given variable or object. In this case it evaluates the Selection and returns the type of object that is selected. If it's not (<>) a range, then the If statement is true.

You typically want to put this at the top of the macro. If a range is NOT selected then a Message Box (pop-up window) will appear that instructs the user to select a range. The Exit Sub line will end the macro.

#15 – Ctrl + Y to Delete a Line of Code

In the VB Editor, `Ctrl`+`Y` **deletes the line of code** that the text cursor is in.


Microsoft Visual Basic for Applications - 18 VBA Shortcut Images.xlsm - [Module2 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 3, Col 1
Project - VB
Undo Typing Ctrl+Z
Redo Typing
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Clear Del
Select All Ctrl+A
Find... Ctrl+F
Find Next F3
Replace... Ctrl+H
Indent Tab
Outdent Shift+Tab
List Properties/Methods Ctrl+J
There is no direct keyboard shortcut for Undo.
Ctrl+Y deletes the current line of code in the VB Editor

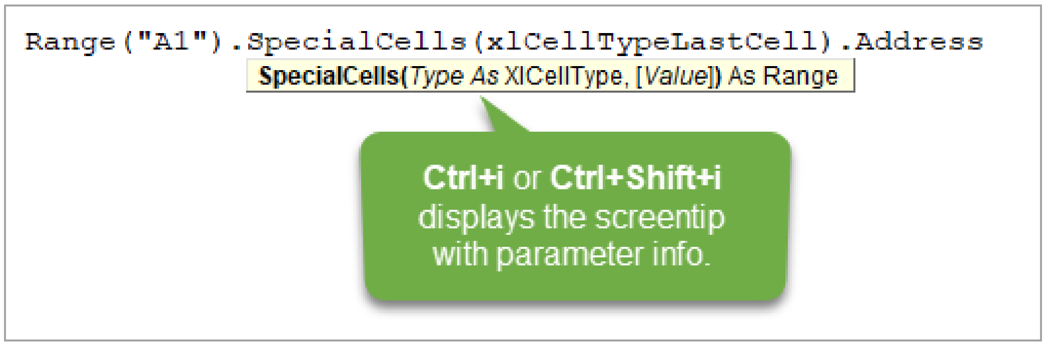
This creates a lot of **confusion** since Ctrl+Y is **typically** used for the Redo command in almost every other application, including Excel!

If you look at the Edit menu in the VB Editor you will see that there is no dedicated shortcut for Redo. We can use `Alt`, `E`, `R` as an alternative shortcut for Redo.

This is just one of those weird quirks of VBA that is really good to know. It will definitely help prevent head scratching WTFs (frustration)! ☺

#16 – Ctrl + i for Quick Info

This is another great keyboard shortcut to know. `Ctrl`+`i` displays the screentip that you see writing or running code. This allows you to see all the parameters in a property, method, or function.



```
Range("A1").SpecialCells(xlCellTypeLastCell).Address
SpecialCells(Type As XlCellType, [Value]) As Range
```

Ctrl+i or **Ctrl+Shift+i** displays the screentip with parameter info.

To use the Ctrl+i shortcut:

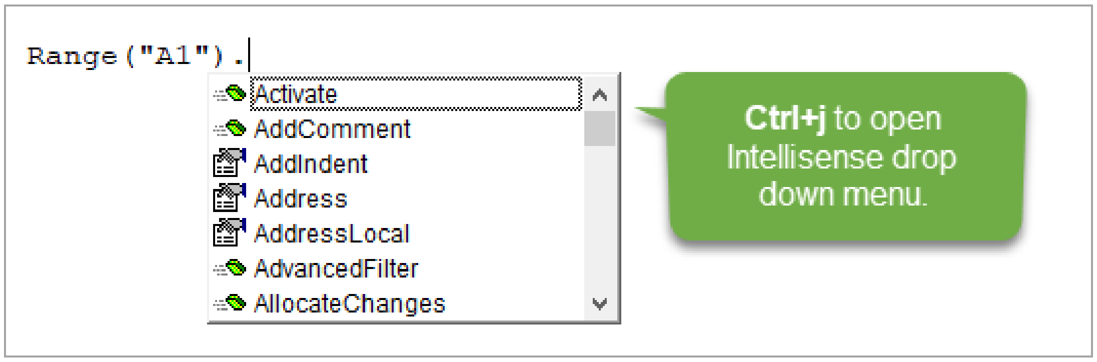
1. Place the text cursor in the word you want to display the info screentip for.
2. Press `Ctrl`+`i`.
3. The screentip will appear.
4. Press Escape to close it or move the cursor.

If you have variable selected within a line and you want to see the parameter info instead of the variable's value/data type, then press `Ctrl`+`Shift`+`i` to see parameter info.

#17 – Ctrl+J Opens the Intellisense Drop Down

The `Ctrl`+`J` shortcut will open the Intellisense drop down menu that displays a list of objects, properties, methods, variables, etc.

I typically use this when I've typed a partial line of code and ended it with a period, like Range("A1").



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

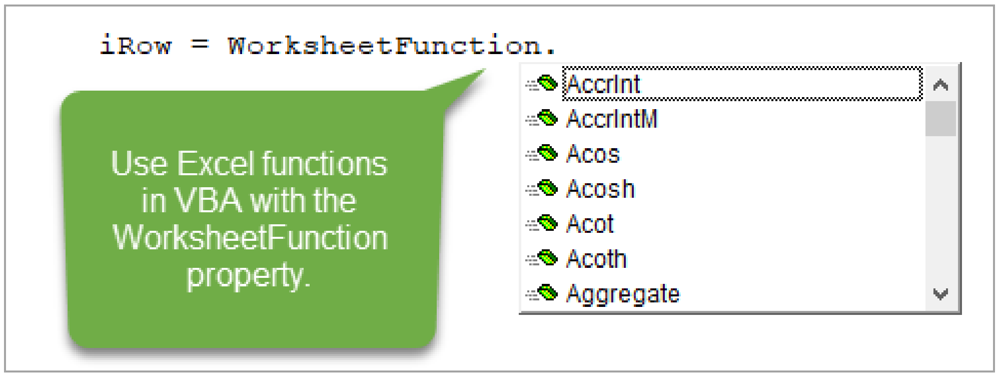
I then want to come back to that line and see the Intellisense drop down. Pressing `Ctrl`+`J` opens the menu. Otherwise, you have to delete the period and type it again.

We can also use Ctrl+J to select a different variable from the list. If you used the wrong variable name somewhere, or need to change it, hit Ctrl+J to see a list of the variable names. If you prefix your variable names (aka Hungarian notation), then the other variables should be pretty close by in that list.

#18 – Worksheet Functions

Did you know we can use worksheet functions in VBA? These are functions we use in a formula in Excel like: vlookup, match, countif, max, min, etc.

Type WorksheetFunction. in a macro to see a list of all the Excel functions that are available in VBA.

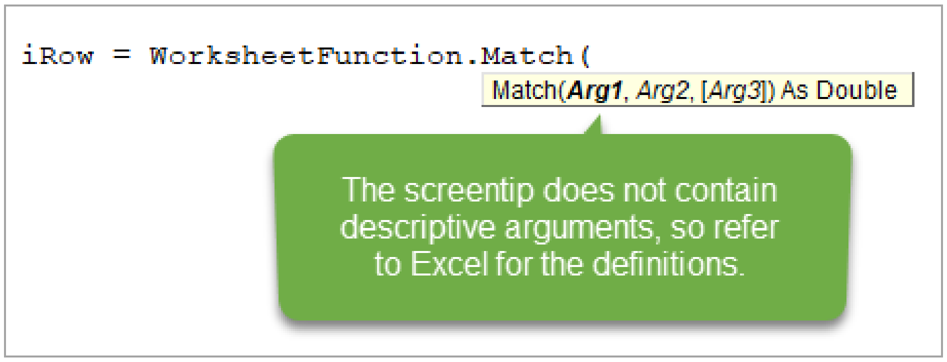


```
iRow = WorksheetFunction.
```

AccrInt
AccrIntM
Acos
Acosh
Acot
Acoth
Aggregate
Use Excel functions in VBA with the WorksheetFunction property.

This is the best of both worlds in my opinion. We can use the power of Excel functions in our VBA code.

The screentip for the function shows the number of arguments, but it does NOT show the names of the arguments. So we typically need to type a formula in Excel to determine what each argument is (unless you have them memorized).



```
iRow = WorksheetFunction.Match(
```

Match(Arg1, Arg2, [Arg3]) As Double
The screentip does not contain descriptive arguments, so refer to Excel for the definitions.

The worksheet function I use the most often in VBA is **Match**. We can use Match to lookup a value and return the row or column number of the cell that contains the matching value. This can sometimes be easier then using the Range.Find method. Checkout my article & videos on how to find the last used cell on a sheet for more on the Range.Find method.

What Is Your Favorite Shortcut or Tip for VBA?

I hope you enjoyed that list and learned something new. Maybe this has inspired you to learn more about macros & VBA. If so, please checkout my **free** upcoming webinar on “The 7 Steps to Getting Started with Macros & VBA”.

I would also like to know what your favorite tip or shortcut is that was not included in this article. Please leave a comment below and share yours. There are no right or wrong answers, and this will be a great way to learn from everyone.

Thanks again for your support and I wish you all the best in 2018! ☺

Please share