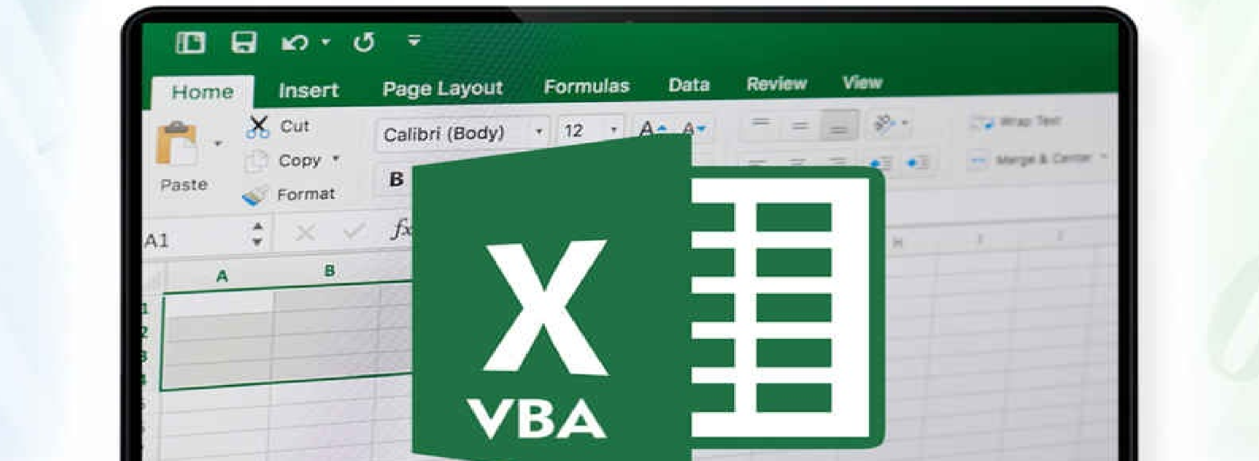
EXCEL VBA

A STEP-BY-STEP SIMPLIFIED GUIDE TO EXCEL VBA PROGRAMMING TECHNIQUES, DATA REPORTING, BUSINESS ANALYSIS AND TIPS AND TRICKS FOR EFFECTIVE STRATEGIES



PETER BRADLEY

EXCEL VBA

A Step-by-Step Simplified Guide to Excel VBA Programming Techniques, Data Reporting, Business Analysis and Tips and Tricks for Effective Strategies

EXCEL VBA

A Step-by-Step Simplified Guide to Excel VBA Programming Techniques, Data Reporting, Business Analysis and Tips and Tricks for Effective Strategies

TABLE OF CONTENTS

Excel VBA
A Step-by-Step Comprehensive Guide on
Excel VBA Programming Tips and Tricks for Effective Strategies

Excel VBA

A Step-By-Step Comprehensive Guide on Advanced Excel VBA Programming Techniques and Strategies

Introduction

VBA is a tool that helps you perform tasks in the easiest way possible. You can perform these tasks in less than a minute when you automate them using VBA. For instance, you can create custom reports, add new toolbars or perform different types of data analysis using VBA. When you learn to write VBA codes, you will become an expert at all the tasks you perform, and you can absorb more work since you can finish a job quickly.

If you want to gather more information on VBA programming, you have come to the right place. This book provides more information about VBA and also talks about the different ways you can use VBA to automate processes. If you write code, you should also know what data types you should use and how you can use them in functions and modules. This book will provide all the information you need to know about VBA.

Over the course of the book, you will gather information on the different data types used in VBA, different types of collections in the VBA, some exercises on conditional and looping statements, arrays and other necessary information. You will also learn how to redirect the flow of programs and also how you should handle the errors.

To make the learning interesting, there are some exercises provided at the end of some chapters. You should try to write the code yourself before you look at the solutions provided to you at the end of the book. Remember that practice will make you better at coding. You will make errors, and these errors will help you become a better programmer. This book is a continuation of the beginner's book in the series and provides a little more information about VBA.

Thank you for purchasing the book. I hope you gather all the information you are looking for.

Chapter One

What Can You Do With VBA?

People use Excel and VBA for a variety of reasons. Some examples are:

- Analyzing data
- Creating lists
- Developing diagrams and charts using data
- Forecasting and Budgeting
- Creating forms and invoices

The list is endless since you can use Excel for many reasons, but I am sure you get the idea. In simple words, you can use Excel to perform different tasks, and I am sure you read the first book in the series and have picked up this book with a set of expectations. If there is any function in Excel that you want to automate, you can use VBA.

For instance, you may want to create or develop a program that will help you import some data or numbers and then format that data to print a report. Once you develop the code, you can execute the macro using a command or a button. This will ensure that Excel performs the task in a few seconds or minutes.

Common Uses of VBA

In the first book, you gathered information on the different functions in VBA. Before you apply those functions, you should understand why you want to use VBA. You have to take some time out of your day and write the code to automate processes. You must also look for different ways to use VBA. This section covers some processes that you can automate using VBA.

Automating Documents

People hate having to prepare documents, and if the documents they prepare always contain the same information, they will certainly not want to put more work into that document. In this instance, you can use the Excel Ad-in called Mail Merge, which is used to automate letters and documents, but this is not an option you can use when you need to write individual documents or letters. At such times, you should write a VBA code that will help you create a form, which will include common information. You can then include check boxes in your code to help you write the documents.

Word processing is not the only task you can automate using VBA. You can also automate the spreadsheet and there are numerous programs you can create for the same. For example, you can extract information or data from the internet into a spreadsheet by clicking a button. Therefore, you can limit the time you spend on simply copying the data from the web and pasting it according to the required format in your Excel worksheet.

Customizing Application Interfaces

There are some features in an application that will not help your cause, and you can learn to turn those features off. You cannot turn off these features if you need to use them for your other you work. Instead of disabling that feature, you can use VBA to create a new feature, which has all the functions that you need. For example, you can write a VBA code to help you if you want to use conditional formatting when you make some changes in your worksheet.

It is easy to change the interface of an application, so it works better for you. You can customize toolbars or menu systems, and can also move some elements around in the interface to make it look presentable. Additionally, you can use multiple interfaces and use a VBA code to shift between those interfaces.

A common application of VBA is to perform a variety of calculations in a few seconds. You can create different graphs and equations using the data that you store in Excel. You must make some changes or modifications to the data you are using so you can perform these calculations on it. If the equation is slightly complicated, you can use VBA and develop a code that will help to simplify the process. You can also use certain iterative functions to perform different calculations.

There are times when the numbers you obtain as a result of a calculation do not mean too much. This value is a just a number until someone decides what to do with that number. Some of the decisions a user makes are repetitive, and a smart application will save time and allow you to enjoy a nice game of Solitaire.

Adding new application features

Most developers and vendors do not use the applications they develop or build. Therefore, they forget to update the code for their application. You can tweak those applications by adding new features or develop a new application using

VBA. When you develop these applications, you can complete some of your work in a few minutes or less, and you can impress your colleagues and your boss. This is an added use of VBA.

Chapter Two

VBA, A Primer

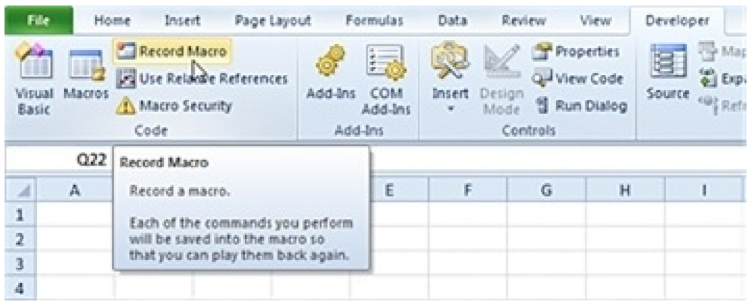
Macro Recorder

As mentioned in the previous book, the macro recorder is an important and useful tool in Excel. This tool will record every action that you perform in Excel. You only need to record a task once using the macro recorder, and you can execute that same task a million times by clicking a button. If you do not know how to program a specific task in Excel, you can use the Macro Recorder to help you understand what you need to do. You can then open the Visual Basic Editor once you have recorded the task to see how you can program it.

You cannot perform many tasks when you use the Macro Recorder. For instance, you cannot use the macro recorder to loop through data. The macro recorder also uses more code than you need, which will slow the process down.

Record a Macro

- Go to the Menu Bar and move the Developer Tab, and click the button to Record the Macro.

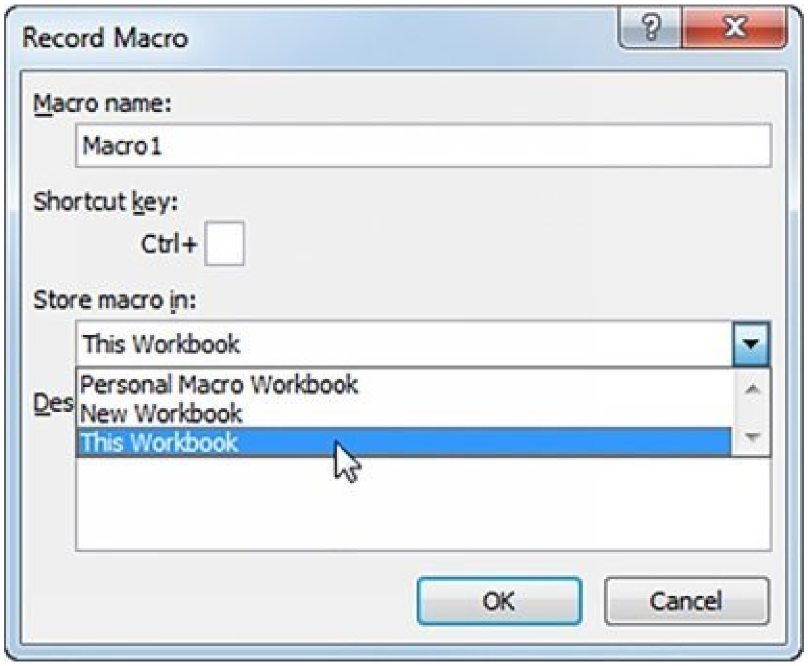


Record Macro

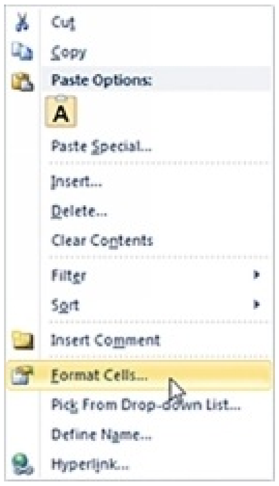
Record a macro.

Each of the commands you perform will be saved into the macro so that you can play them back again.

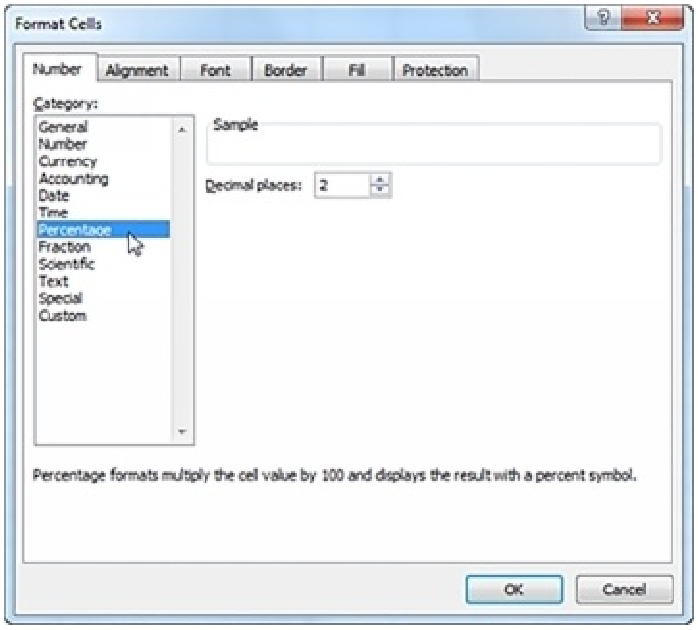
- Enter the name of the macro.
- Choose the workbook where you want to use the macro. This means that the macro can only be used in the current workbook.



- If you store the macro in a personal macro workbook, you can access the macro in all your workbooks. This is only because Excel stores the macro in a hidden workbook, which will open automatically when it starts. If you store the macro in a new workbook, you can use the macro only in the opened workbook.
- Click OK.
- Now, right click on the active cell in the worksheet. Ensure that you do not select any other cell. Click format cells.



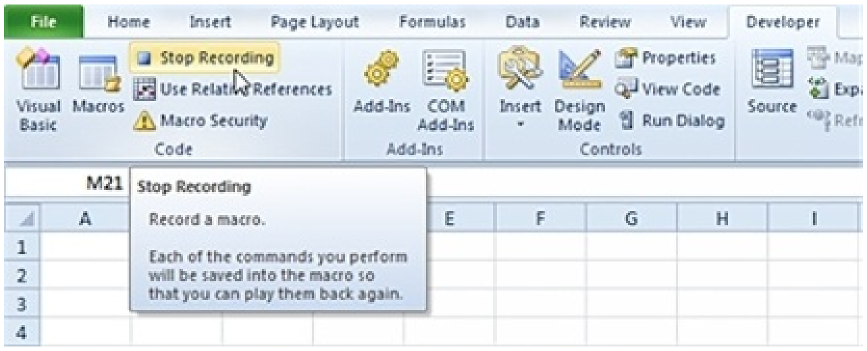
- Select the percentage.



Decimal places: 2

Percentage formats multiply the cell value by 100 and displays the result with a percent symbol.

- Click OK.
- Now, select the stop recording.



Stop Recording

Record a macro.

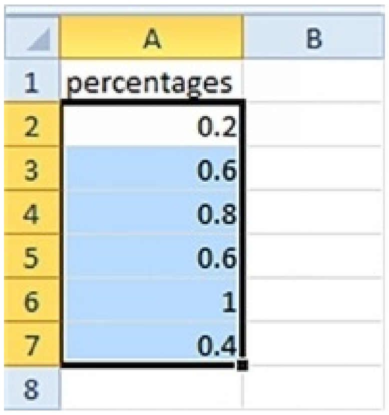
Each of the commands you perform will be saved into the macro so that you can play them back again.

You have successfully recorded your macro using the macro recorder.

Run the Recorded Macro

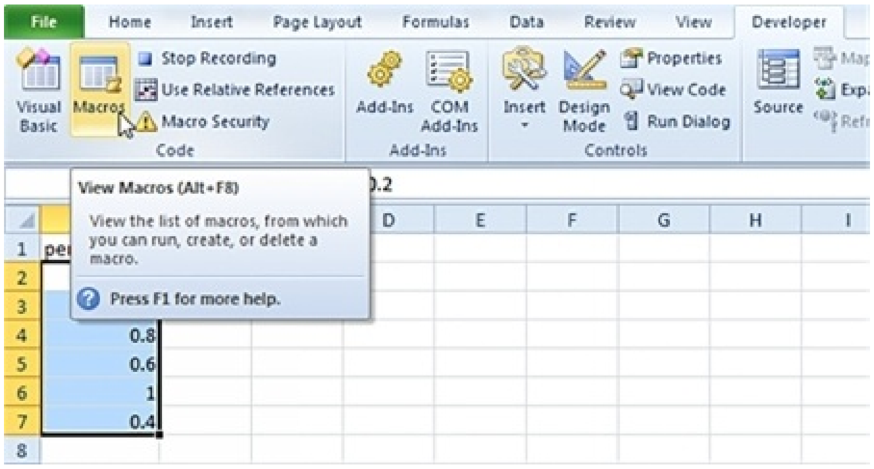
You will now need to test the macro and see if you can change the format of the numbers to percentage.

- Enter any numbers between 0 and 1 in the spreadsheet.
- Select the numbers.



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

- Move to the Developer tab, and click macros.

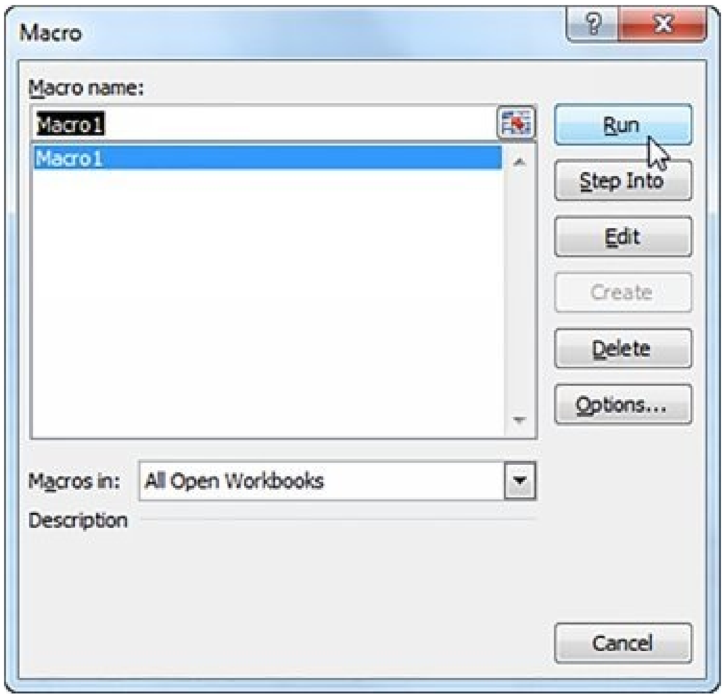


View Macros (Alt+F8)

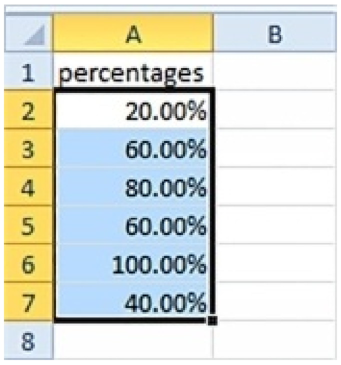
View the list of macros, from which you can run, create, or delete a macro.

Press F1 for more help.

- Now click run.



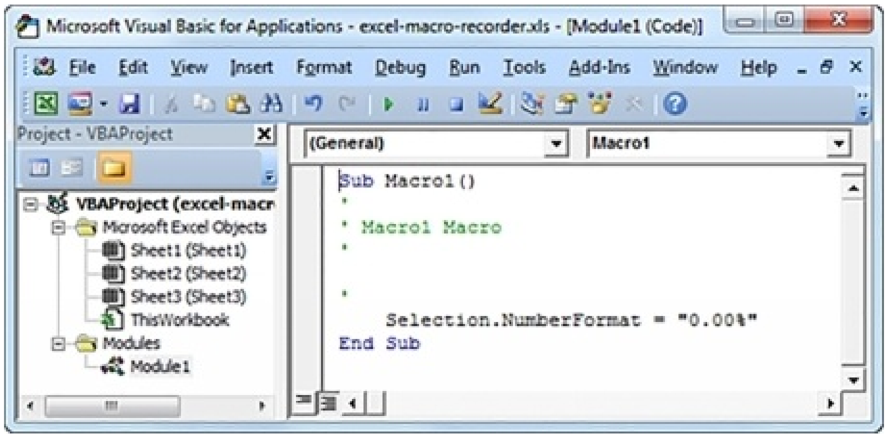
You will see the following result.



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

See The Macro

If you want to look at the macro, you should open the Visual Basic Editor.



```
Sub Macro1()
'
' Macro1 Macro
'

'
    Selection.NumberFormat = "0.00%"
End Sub
```

The macro, called Module 1, is placed in a module. The code that is placed in the module is always available to the full workbook. This means you can change the format for the numbers in all the sheets in the workbook. If you assign a macro to the command button, you should remember that the macro would only be available for that specific sheet.

Macro Storage and Security

The developers at Microsoft changed the security settings that help to prevent macros from running. This development was only made when macros were used to deliver some high-profile viruses. We covered some information about the security of macros in the first part of the book. Let us now look at how you can adjust the default settings in a macro.

You can either control the settings for a macro in some workbooks by saving the workbook in trusted locations or by adjusting the security settings globally. If you store a workbook with a macro in a folder that you label a trusted location, the macros will automatically be enabled when you open the workbook.

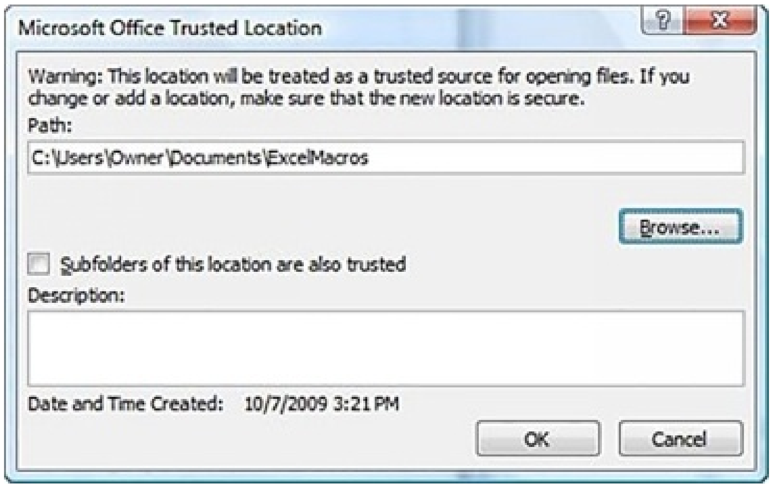
You can check the security of your macro in the Developer tab under the Macro Security icon. If you click this icon, a dialog box with the settings category will be displayed. You can access the folders that you trust by scrolling down the left navigation bar.

How to Add a Trusted Location

As mentioned earlier, you can save the workbooks with macros in a folder that you mark as a trusted location. If you save a workbook in that folder, the macros will always be enabled. The developers suggest that you should always have a trusted location in your hard drive. Remember that you can never trust the location on a network drive.

If you want to specify a trusted location, you should follow the steps given below:

1. Go to the Developer Tab and click on Macro Security.
2. Move to the left navigation pane in the Trust Center and choose the Trusted Location.
3. If you want to save the file on a network drive, you should add that location into the trusted locations.
4. Go to 'My Networks' in the Trusted Location dialog box and click the 'Add New Location' button.
5. You will see the list of Trusted Locations in a dialog box.


Microsoft Office Trusted Location

Warning: This location will be treated as a trusted source for opening files. If you change or add a location, make sure that the new location is secure.

Path: C:\Users\Owner\Documents\ExcelMacros

Subfolders of this location are also trusted

Description:

Date and Time Created: 10/7/2009 3:21 PM

6. Now click the Browse button and go to the parent folder of the folder that you want to make a trusted location. Now click on the Trusted Folder. You will not find the name of the Folder in the text box, but click OK. The correct name will come in the Browse dialog box.
7. If you want to include the subfolders in the selected folder, you should select the radio button against the 'Subfolders of this location are also trusted' option.
8. Now, click OK to add the folder to the list.

How to Enable Macros outside a Trusted Location

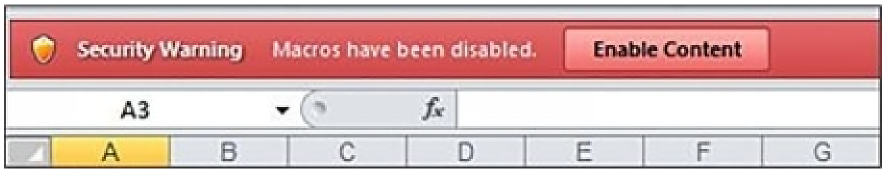
When you do not save an Excel workbook in a trusted location, Excel will always rely on the macro settings. In Excel 2003, a macro could have a low, medium, high or very high security. These settings were later renamed by the developers in Microsoft. If you want to access the macro settings, you should go to the Developers Tab and choose Macro Security. Excel will then display the Macro Settings dialog box. You should select the ‘Disable All Macros with Notification’ option. Let us look at the description of the options in the dialog box.

Disable All Macros without Notification

This setting will not allow any macro to run. If you do not always want to run the macro when you open the workbook, you should choose this setting. Since you are still learning how to use macros and work with them, you should not use this setting. This setting is equivalent to the Very High Security that is found in Excel 2003. If you choose this setting, you can only run macros if they are saved in a Trusted Location.

Disable All Macros with Notification

This setting is like the Medium security setting in Excel 2003. This is the recommended setting that you should use. If you use this setting, Excel will ask you if you want to Enable or Disable a macro when you open a workbook. You may often choose this option if you are a beginner. In Excel 2010, you will see a message in the message area, which states that the macros have been disabled. You can either choose to enable or disable the content in the workbook by choosing that option.


Security Warning Macros have been disabled.

Disable All Macros except Digitally Signed Macros

If you wish to use this setting, you should always use a digital signing tool like VeriSign or any other provider to sign your macro. If you are going to sell your macros to other parties, you should use this security option. This is a hassle if you want to write macros only for your use.

Enable All Macros

Experts suggest that you do not use this option since dangerous codes can also run on your system. This setting is equivalent to the Low security option in

Excel 2003, and is the easiest option to use. This option will open your system up to attacks from malicious viruses.

Disabling All Macros with Notification

Experts suggest that you set your macro to disable all content after it gives you a notification. If you save a workbook with a macro using this setting, you will see a security warning right above the formula bar when you open the workbook. If you know that there are macros in the workbook, all you need to do is click ‘Enable Content.’ You can click on the X on the far right of the bar if you do not want to enable any of the macros in the workbook.

If you do forget to enable the macro and then attempt to run that macro, Excel will indicate that the macro will not run since you have disabled all macros in the workbook. If this happens, you should reopen the workbook to enable the macros again.

Exercises

1. Write a program to protect all worksheets in a workbook.
2. Write a program to lock or protect cells in a worksheet.

Chapter Three

How to Manipulate Data in Excel

A macro processes code written in the Visual Basic Editor to manage and manipulate huge volumes of data. The previous chapter provides information on how you can use a macro to format certain fields or cells in Excel to meet your criteria.

The following is an example of a VBA script:

```
Sub ConfigureLogic()
    Dim qstEntries
    Dim dqstEntries
    Dim qstCnt, dqstCnt
    qstEntries = Range("QualifiedEntry").Count
    qst = qstEntries -
    WorksheetFunction.CountIf(Range("QualifiedEntry"), "")
    ReDim QualifiedEntryText(qst)
    'MsgBox (qst)
    dqstEntries = Range("DisQualifiedEntry").Count
    dqst = dqstEntries -
    WorksheetFunction.CountIf(Range("DisQualifiedEntry"), "")
    ReDim DisqualifiedEntryText(dqst)
    'MsgBox (dqst)
    For qstCnt = 1 To qst
    QualifiedEntryText(qstCnt) =
    ThisWorkbook.Worksheets("Qualifiers").Range("J" & 8 +
    qstCnt).value
    'MsgBox (QualifiedEntryText(qstCnt))
    logging ("Configured Qualified Entry entry #" & qstCnt & " as {" &
    QualifiedEntryText(qstCnt) & "}")
    Next
    For dqstCnt = 1 To dqst
```

```
        DisqualifiedEntryText(dqstCnt) =
        ThisWorkbook.Worksheets("Qualifiers").Range("M" & 8 +
        dqstCnt).value
        'MsgBox (DisqualifiedEntryText(dqstCnt))
        logging ("Configured DisQualified Entry entry #" & qstCnt & " as
        {" & DisqualifiedEntryText(dqstCnt) & "}")
        Next
        includeEntry =
        ThisWorkbook.Worksheets("Qualifiers").Range("IncludeSibling").va
        'MsgBox (includeEntry)
        logging ("Entrys included in search - " & includeEntry)
    End Sub
```

How to Analyze and Manipulate Data in a Spreadsheet

If you want to use VBA to analyze data, you should check the macro settings in Excel. Ensure that the settings as per your requirements. You should also make sure that the macro settings are activated in Excel. Now, create a worksheet and call it 'Qualifiers.' We will be using this worksheet to check the data and ensure that the data qualifies all the selections that you require. You must then set up the qualifiers based on the code you have written. You cannot cut and paste these qualifiers, but will need to enter them manually.

```
    ThisWorkbook.Worksheets("Qualifiers").Range("J" & 8 + qstCnt).value
```

How to Construct an Array and Locate The Range

In the above function, the range will start from Cell J9. The function notes 8, but the range is 9 since we have declared the qstCnt to be 1 using the following code:

```
    For qstCnt = 1 To qst
```

It is because of this statement that the list will start at 9.

If you want to construct an array using the entries in the Qualifiers worksheet, you should add random words or numbers between cells J9 and J13, including those cells. When the rows are complete, you can find and manipulate the data in Excel.

```
Private Sub CountSheets()
    Dim sheetcount
    Dim WS As Worksheet
    sheetcount = 0
    logging ("*****Starting Scrub*********")
    For Each WS In ThisWorkbook.Worksheets
    sheetcount = sheetcount + 1
    If WS.Name = "Selected" Then
    'need to log the date and time into sheet named "Logging"
    ActionCnt = ActionCnt + 1
    logging ("Calling sheet: " & WS.Name)
    scrubsheet (sheetcount)
    Else
    ActionCnt = ActionCnt + 1
    logging ("Skipped over sheet: " & WS.Name)
    End If
    Next WS
    'MsgBox ("ending")
    ActionCnt = ActionCnt + 1
    logging ("****Scrub DONE!")
    Application.ScreenUpdating = True
End Sub
```

The following example will show you how you can write a macro for a working tab counter:

```
    Dim sheetcount
    Dim WS As Worksheet
    sheetcount = 0
    logging ("*****Starting Scrub*********")
For Each WS In ThisWorkbook.Worksheets
```

```
sheetcount = sheetcount + 1
```

When you initialize the sheet count variable, you should first set it to zero before you restart the counter. You can also use the logging() subroutine to keep track of all the actions in the qualifiers tab to make the correct selections. The For loop in the above example will set up the counting variable in the Active Workbook. Once you initialize WS, it will make the worksheet that you are currently in the active worksheet. Since this module is unnamed, it will run in any workbook. If you have many workbooks open, this module may run in an incorrect workbook. If you want to avoid any errors, you should name the workbook that you want the module to run in.

When the loop runs, it will add another variable to the sheet count and keep a track of the tabs. We will then move to

```
If WS.Name = "Selected" Then
'need to log the date and time into sheet named "Logging"
ActionCnt = ActionCnt + 1
logging ("Calling sheet: " & WS.Name)
scrubsheet (sheetcount)
Else
ActionCnt = ActionCnt + 1
logging ("Skipped over sheet: " & WS.Name)
End If
```

In this section of the code, we are looking for the Selected tab. If the variable WS is the same as the Selected worksheet, you can fire up the Scrub sheet subroutine. If the variable WS is not the same as the Selected worksheet, then the sheet will be skipped and the action will be counted. The code above is an example of how you can write a macro to count the number of tabs and locate a specific tab.

The next parts of this chapter talk about the different ways you can manipulate data in Excel.

Different Ways to Manipulate Data

Count the Number of Sheets in a Workbook

```
Dim TAB
For Each TAB In ThisWorkbook.Worksheets
'some routine here
Next
```

Filter by using Advanced Criteria

```
Range("A2:Z99").Sort key1:=Range("A5"), order1:=xlAscending,
Header:=xlNo
```

Find The Last Column, Cell Or Row On A Worksheet

```
Dim cellcount
cellcount = Cells(ThisWorkbook.Worksheets("worksheet").Rows.Count,
1).End(xlUp).Row
```

Getting Values from another Worksheet

```
dim newvalue
newvalue = ThisWorkbook.Worksheets("worksheet").Range("F1").value
```

Apply Auto-Fit To A Column

```
Columns("A:A").EntireColumn.AutoFit
```

Adding Named Ranges to Specific Sheets

```
ThisWorkbook.Worksheets("worksheet").Names.Add Name:="Status",
RefersToR1C1:="=worksheet!C2"
```

Insert Rows Into A Worksheet

```
Dim Row, Column
Cells(Row, Column).EntireRow.Select
Selection.Insert
```

Copy an Entire Row for Pasting

```
ActiveSheet.Range("A1").EntireRow.Select
Selection.Copy
```

```
Delete An Entire Row
ActiveSheet.Range("A1").EntireRow.Select
Selection.Delete
```

Inserting a Column into a Worksheet

```
Dim Row, Column
Cells(Row, Column).EntireColumn.Select
Selection.Insert
```

Insert Multiple Columns into a Worksheet

```
Dim insertCnt
Dim Row, Column
For insertCnt = 1 To N
ThisWorkbook.Worksheets("worksheet").Select
Cells(Row, Column).EntireColumn.Select
Selection.Insert
Next
```

Select a Specific Sheet

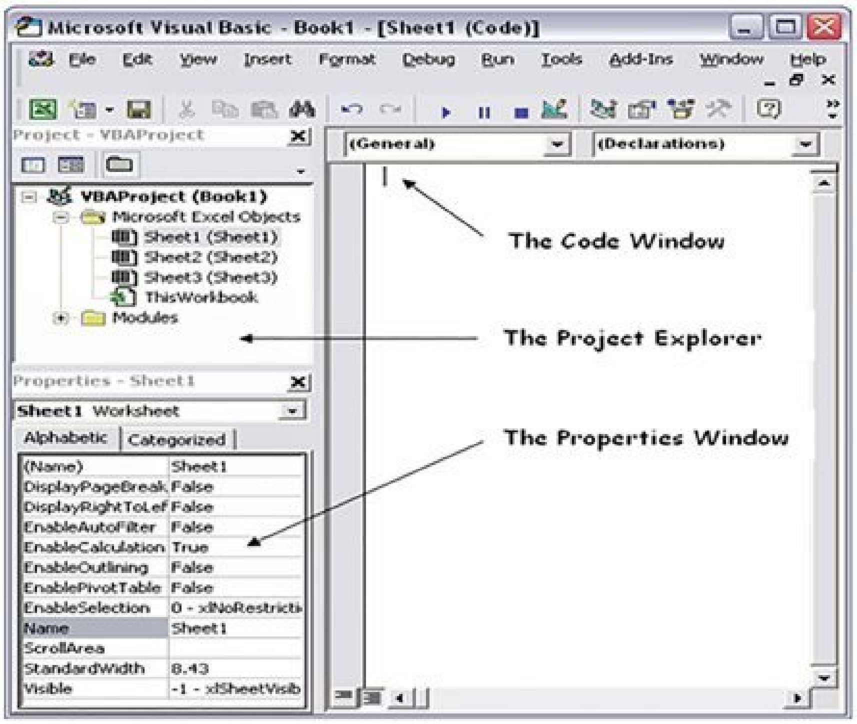
```
ThisWorkbook.Worksheets("worksheet").Select
Compare Values In A Range
Dim firstrange
Dim Logictest
Logictest = "some word or value"
If (Range(firstrange).value = Logictest) then
'some routine here
End If
```

Chapter Four

Fundamentals of VBA

VBA is a visual programming environment. That is, you see how your program will look before you run it. Its editor is very visual, using various windows to make your programming experience easy and manageable. You will notice slight differences in the appearance of the editor when you use it with Vista as compared to older versions of Windows. Regardless of which version of Windows you use or which Office Product you use, the Visual Basic Editor has the same appearance, same functionality and same items.

The IDE is like a word processor, database form or a spreadsheet. The IDE, like every other application editor has special features which makes it easy to work with data. Apart from that, the IDE can also be used to write special instructions which help with data manipulation and analysis. VBA will follow the instructions in the program. The figure below shows you the IDE Window in Excel:



(Name)	Sheet1
DisplayPageBreak	False
DisplayRightToLef	False
EnableAutoFilter	False
EnableCalculation	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestricti
Name	Sheet1
ScrollArea	
StandardWidth	8.43
Visible	-1 - xlSheetVisib

The IDE in VBA consists of a toolbar, menu system, a Properties window, a Project Explorer window and a Code window. Below is a summary of what each Window contains.

Project Explorer

This window provides a list of the items or objects that are in your project. These items contain the document elements that are present in a single file. This application exists within a file which you will see in the Project Explorer window.

Properties

When you select an object, the Properties window will give you all the information you need about that object. For instance, this window will tell you whether the object is empty or whether there are some words in it.

Code

Eventually, you will need to write some code which will make the application work. This window will contain the special words which will tell the editor what it needs to do. This space is analogous to a to-do list or an algorithm.

Looking at the VBA Toolbox

You will not have to write a program for every task that you want Excel to perform. The IDE also allows you to use forms, which are similar to the forms that you use to perform different tasks. In case of VBA, you will decide what should appear on the forms and also decide how the forms should act when a user enters some data into the forms. VBA allows you to use the toolbox to create a form. This toolbox contains controls used to create forms.

Each Toolbox button performs a unique task. For example, when you click one button, a text box may appear on the screen. If you click another button, a mathematical operation may take place.

Starting the Visual Basic Editor

One can start the Visual Basic Editor in different ways depending on the application you are using. The newer versions of the Office Product use a different approach when compared to the older versions.

Step 1: Go to Option “View” on the toolbar.

Step 2: In the drop-down list, select “Record Macro.”

Step 3: The interface will open and you can begin typing the code for the worksheet you are in.

Using Project Explorer

The Project Explorer will appear in the Project Explorer Window, and you can use this to interact with different objects that make up the project. Every project is an individual file that you can use to hold your program or at least some pieces of it. This project will reside in the Office document which you are using. Therefore, when you open the document, you also open the project. We will look at how programs and projects interact with each other in later chapters. The Project Explorer works like the left pane in Windows Explorer.

The Project Explorer lists the different objects you are using in the project. These objects depend on the type of application you are working with. For example, if you are working with Word, you see documents and document templates. Likewise, if you are working with Excel, you will come across different workbooks and worksheets. Regardless of the type of application you work with, the Project Explorer will be used in the same way.

A project can contain modules, class modules and forms. Let us look at the description of these objects:

- Forms: These contain some user interface elements that allow you to interact with a user and collect necessary information.
- Modules: These contain the nonvisual parts of your code or application. For instance, you can use a module to store some calculations.
- Class modules: These contain objects that you want to develop, and you can use a class module to create new data types.

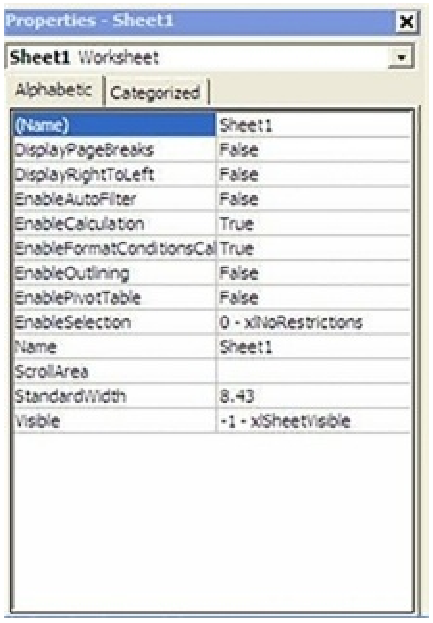
Working with special entries

You can sometimes see some special entries in the Project Explorer. For instance, when you work on a Word document, you will see a References folder which will contain the references that the Word document makes. This contains a list of templates which the document uses to format the data in the document.

In many cases, you cannot modify or manipulate the objects in the folders. This is the case when Word document objects use a Reference folder. This folder is only available to provide information. If you want to modify or develop a referenced template, you should look for the object in the Project Explorer window. We will not discuss these concepts in the book since you do not work with these often.

Using the Properties window

Most objects that you select in the IDE in VBA always have properties that describe the objects in a specific way. The "Property values are up" section talks about the properties that you have not worked with before. The following section will provide more information about the Properties Window.



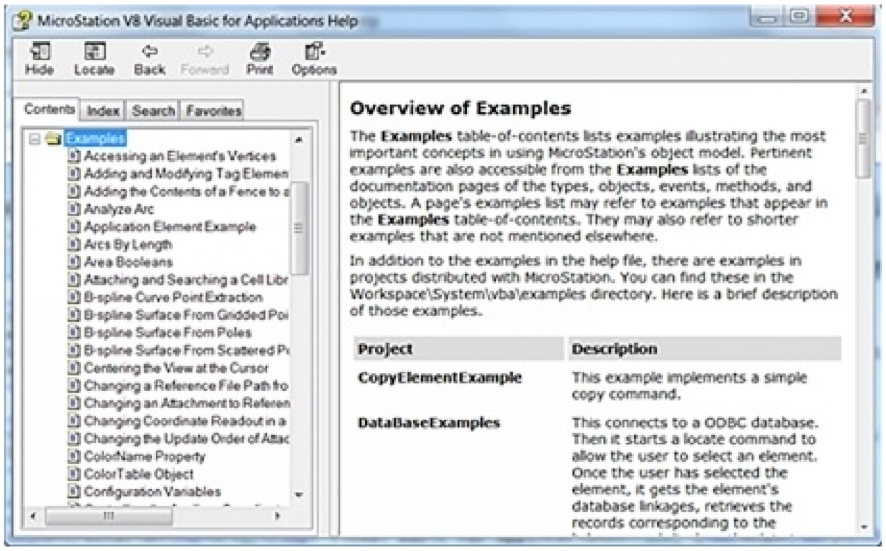
(Name)	Sheet1
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsCal	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	Sheet1
ScrollArea	
StandardWidth	8.43
Visible	-1 - xlSheetVisible

Understanding property types

A property will always describe the object. When you look at an object, you will assume something about the product depending on whether the object is red, yellow or green. In the same way, every VBA object has a specific type. One of the most common types is text. The property of every form is text, and this text appears at the top or bottom of the form when a user opens it. Another common property type is a Boolean value.

Getting help with properties

Do not expect to memorize every property for every object that VBA applications can create. Not even the gurus can do that. To determine what a property will do for your application, just highlight the property and press F1, and, in most cases, VBA displays a Help window like the image below.


MicroStation V8 Visual Basic for Applications Help
Hide
Locate
Back
Forward
Print
Options
Contents
Index
Search
Favorites
Examples
Accessing an Element's Vertices
Adding and Modifying Tag Elemen
Adding the Contents of a Fence to a
Analyze Arc
Application Element Example
Arcs By Length
Area Booleans
Attaching and Searching a Cell Libr
B-spline Curve Point Extraction
B-spline Surface From Gridded Poi
B-spline Surface From Poles
B-spline Surface From Scattered P
Centering the View at the Cursor
Changing a Reference File Path fo
Changing an Attachment to Referen
Changing Coordinate Readout in a
Changing the Update Order of Attac
ColorName Property
ColorTable Object
Configuration Variables

Overview of Examples

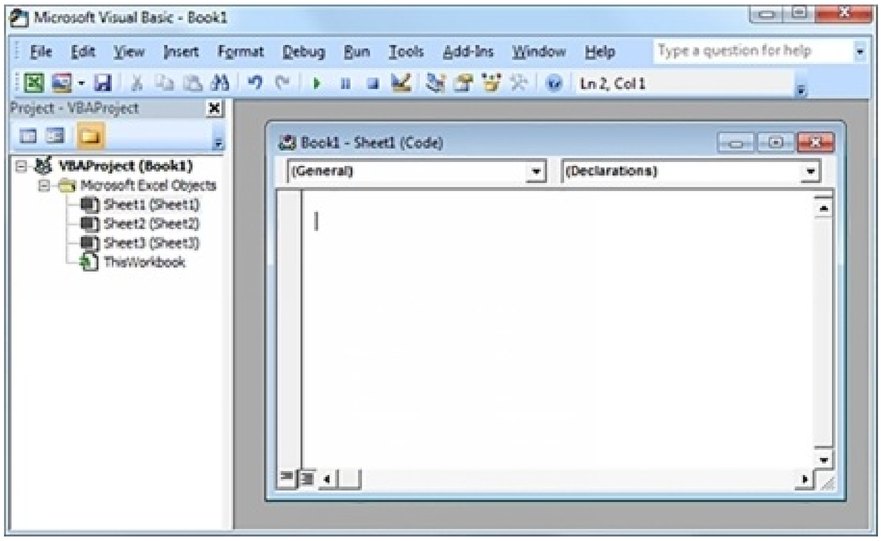
The **Examples** table-of-contents lists examples illustrating the most important concepts in using MicroStation's object model. Pertinent examples are also accessible from the **Examples** lists of the documentation pages of the types, objects, events, methods, and objects. A page's examples list may refer to examples that appear in the **Examples** table-of-contents. They may also refer to shorter examples that are not mentioned elsewhere.

In addition to the examples in the help file, there are examples in projects distributed with MicroStation. You can find these in the Workspace\System\vba\examples directory. Here is a brief description of those examples.

Project	Description
CopyElementExample	This example implements a simple copy command.
DataBaseExamples	This connects to a ODBC database. Then it starts a locate command to allow the user to select an element. Once the user has selected the element, it gets the element's database linkages, retrieves the records corresponding to the

Using the Code window

The Code Window is the space where you will write the code for your application. This window works like every other text editor that you have used, except that you type according to the syntax.



The Project Explorer window and Property window are no longer present in the image above. You can display these again by following the path: View -> Project Explorer and View ->Properties Window commands.

Opening an existing Code window

Sometimes you will not have the time to complete the code for an application and will need to work on it later. If you want to open an existing code window, you should find the module you want to work on in the Project Explorer.

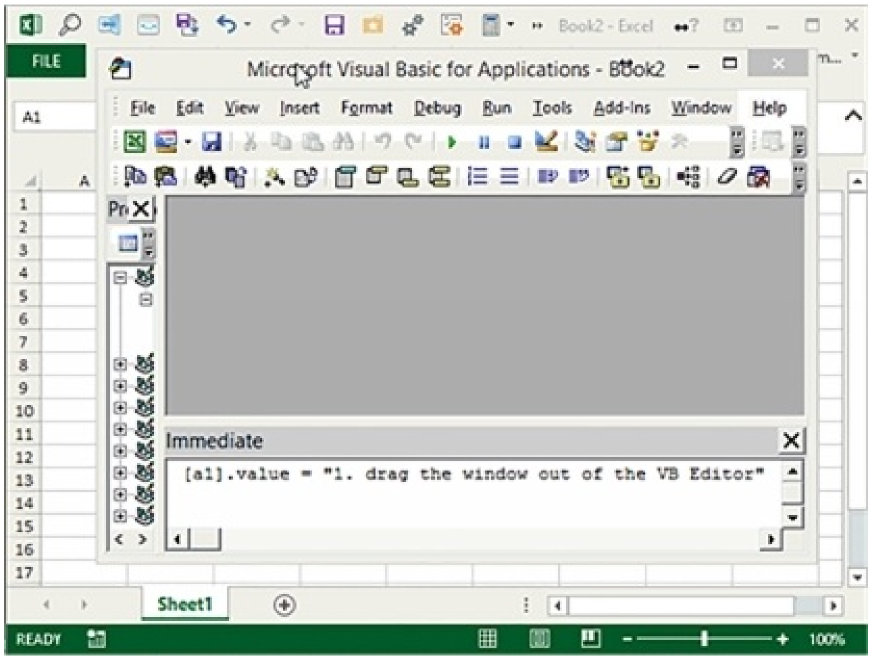
Double-click the name of the module that you want to enter. You will see the code in the IDE window. This Code window will also appear when you want to perform a variety of tasks.

Creating a new Code window

When you want to develop a new module in an existing document or template, you should open a new code window by using the following path: Insert -> Module or Insert -> Class Module command. Once you save this class module or module, it will always be in the Project Explorer with every other module that is in your project.

It is easier to execute one line of code at a time to understand where you may have made an error. You can do this by using the Immediate Window. You will always find this window at the bottom of the IDE, and it will not contain any information until you type something in it.

A developer spends a lot of time using the Immediate Window to check if there are any errors in the applications they are developing. You can use the immediate window to check with VBA if the function you have written produces the required value. To try this feature, type String1 = “Hello World” in the Immediate window and then press Enter. Now type ‘? String1’ and then press Enter. Here, you have asked the editor to create a variable called String1 and assign it a value of Hello World. You can use the ‘?’ operator to check the value assigned to the variable String1.


Book2 - Excel
FILE
A1
Microsoft Visual Basic for Applications - Book2
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Immediate
Sheet1
READY
100%

```
[a1].value = "1. drag the window out of the VB Editor"
```

Chapter Five

Working With Loops and Conditional Statements

In the first part of the book, we looked at the different types of loops that one can use in Excel VBA. This chapter covers some examples and exercises that you can use to practice.

For Loop

Most people use the For Loop in VBA. There are two forms of the For Loop – For Next and For Each In Next. The For Loop will move through a series or data in a sequence. You can use the Exit statement to end the For Loop at any point. The loop will continue to run until the condition is met. When the final condition is met, the editor will move to the next statement in the program, which is the natural direction.

Let us look at the syntax of the loop:

The For … Next loop has the following syntax:

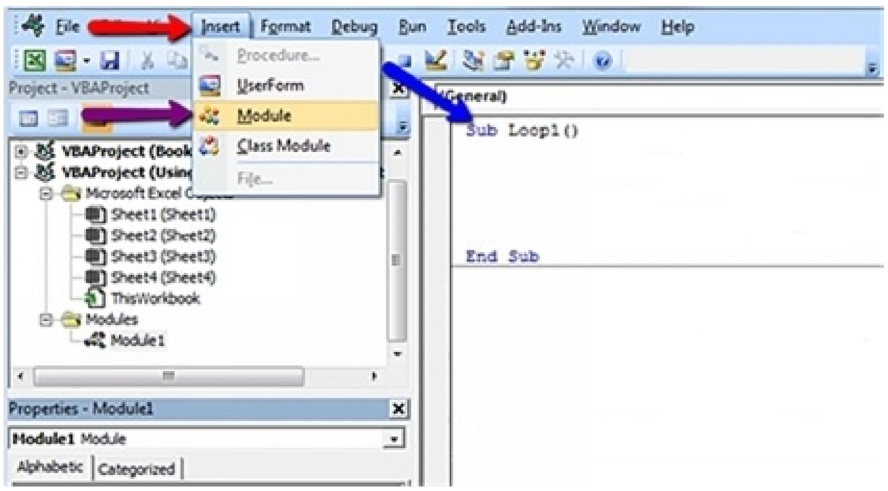
```
For counter = start_counter To end_counter
'Do something here (your code)
Next counter
```

In the syntax above, we are initializing the counter variable, which will maintain the loop. This counter variable will be set to a value that is equal to the start counter, which will be the beginning of the loop. This variable will increase in number until it meets the end condition which is the end counter variable. The loop will continue to run until the value of the counter is equal to the value of the end counter variable. This loop will execute once until the values match, after which the loop will stop.

The explanation above can be slightly confusing, therefore let us look at a few examples that you can use to understand the For Loop better. Before we look at the examples, follow the steps given below:

- Open a new workbook and save it using the .xlsm extension.
- Now, press Alt+F11 to launch the Visual Basic Editor screen.

- Now, insert a new module.

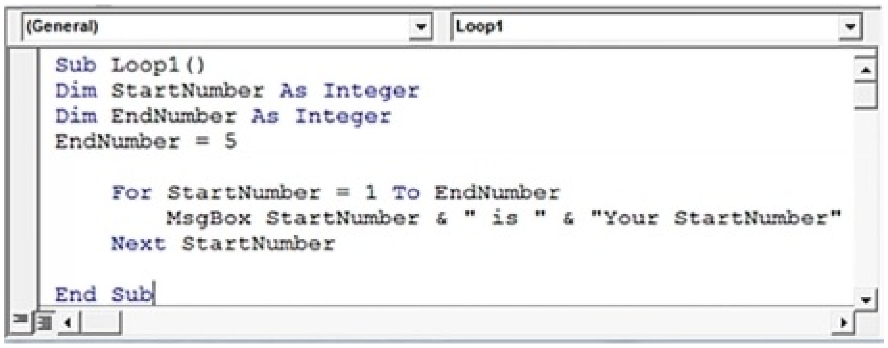


```
Sub Loop1 ()

End Sub
```

Example 1

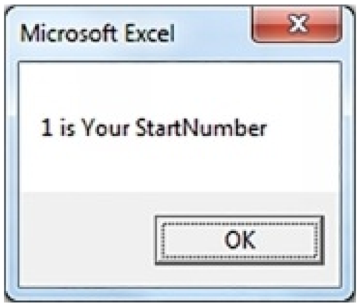
In this example, we will display a number using a message box.

```
Sub Loop1()
Dim StartNumber As Integer
Dim EndNumber As Integer
EndNumber = 5
    For StartNumber = 1 To EndNumber
        MsgBox StartNumber & " is " & "Your StartNumber"
    Next StartNumber
End Sub
```



```
Sub Loop1 ()
Dim StartNumber As Integer
Dim EndNumber As Integer
EndNumber = 5

    For StartNumber = 1 To EndNumber
        MsgBox StartNumber & " is " & "Your StartNumber"
    Next StartNumber

End Sub
```

In the above code, the StartNumber and EndNumber variables are declared as integers, and the StartNumber is the start of your loop. The values that you enter in the loop can be anywhere in between the StartNumber and EndNumber. The code will start from StartNumber, which is 1, and end at EndNumber which is 5. Once the code runs, the following message will be displayed on the screen.

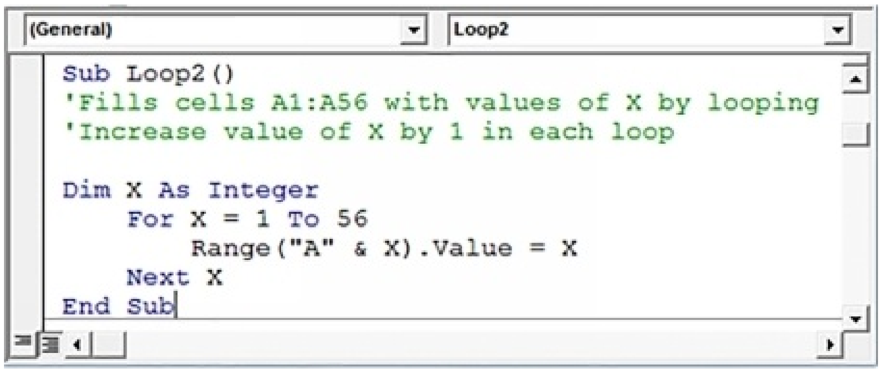

Microsoft Excel

1 is Your StartNumber

Example 2

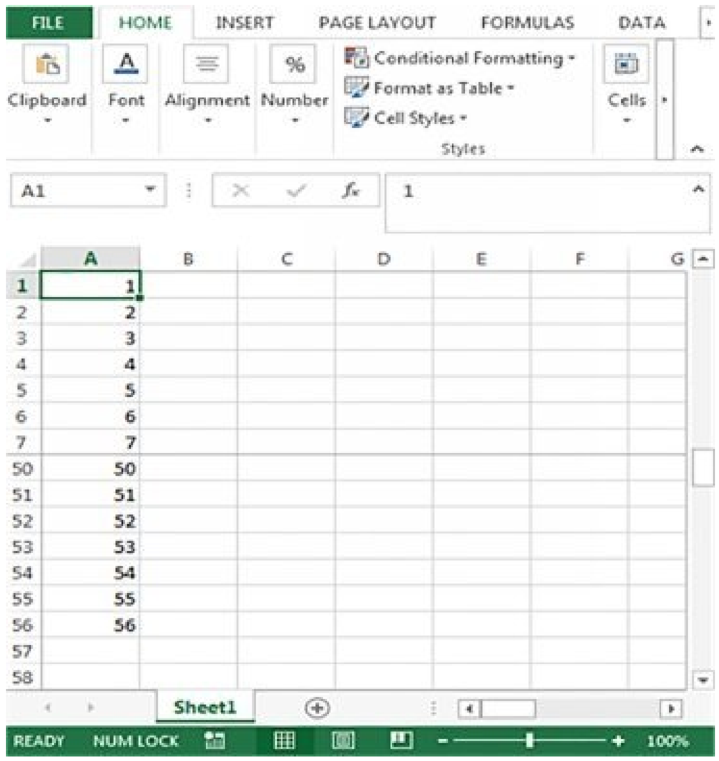
In this example, we will fill values in the Active worksheet.

```
Sub Loop2()
'Fills cells A1:A56 with values of X by looping' --- Comment
'Increase value of X by 1 in each loop' --- Comment
Dim X As Integer
    For X = 1 To 56
        Range("A" & X).Value = X
    Next X
End Sub
```



```
Sub Loop2()
'Fills cells A1:A56 with values of X by looping
'Increase value of X by 1 in each loop

Dim X As Integer
    For X = 1 To 56
        Range("A" & X).Value = X
    Next X
End Sub
```

You will see the following output.



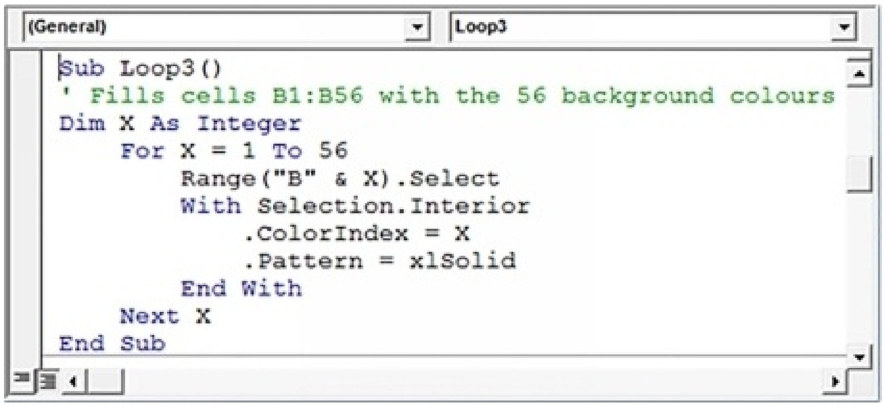
A1 | 1

	A	B	C	D	E	F	G
1	1						
2	2						
3	3						
4	4						
5	5						
6	6						
7	7						
50	50						
51	51						
52	52						
53	53						
54	54						
55	55						
56	56						
57							
58							

Example 3

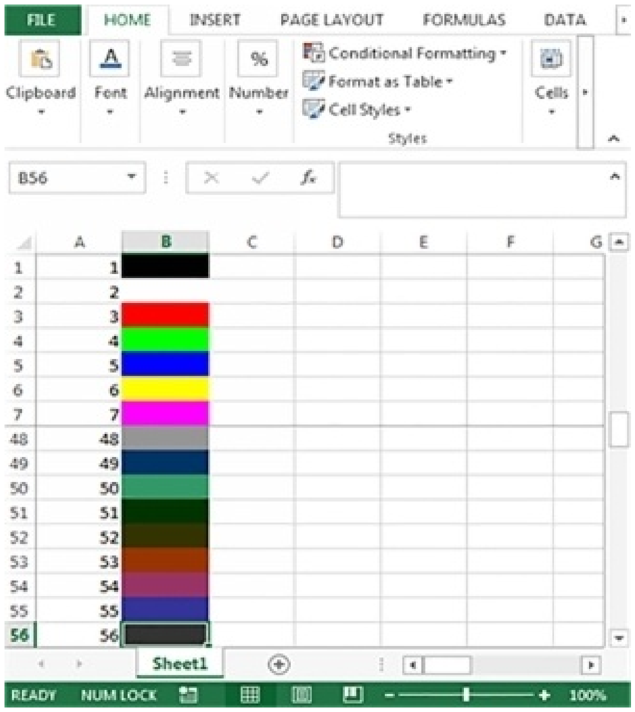
In this example, we will fill the cells in the active worksheet with a background color.

```
Sub Loop3()
' Fills cells B1:B56 with the 56 background colors'--- Comment
Dim X As Integer
    For X = 1 To 56
        Range("B" & X).Select
        With Selection.Interior
            .ColorIndex = X
            .Pattern = xlSolid
        End With
    Next X
End Sub
```



```
Sub Loop3()
' Fills cells B1:B56 with the 56 background colours
Dim X As Integer
    For X = 1 To 56
        Range("B" & X).Select
        With Selection.Interior
            .ColorIndex = X
            .Pattern = xlSolid
        End With
    Next X
End Sub
```

You will see the following output.



Example 4

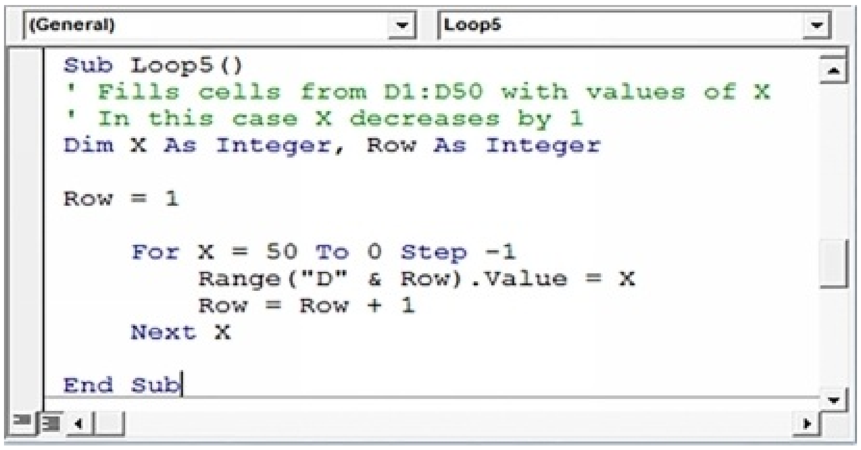
It is important to remember that a loop does not necessarily have to move from a low value to a higher value. You can use the For Loop to move from higher values to lower values using the STEP function. This example will show you how you can perform the same function.

Sub Loop5()

' Fills cells from D1:D50 with values of X' --- Comment

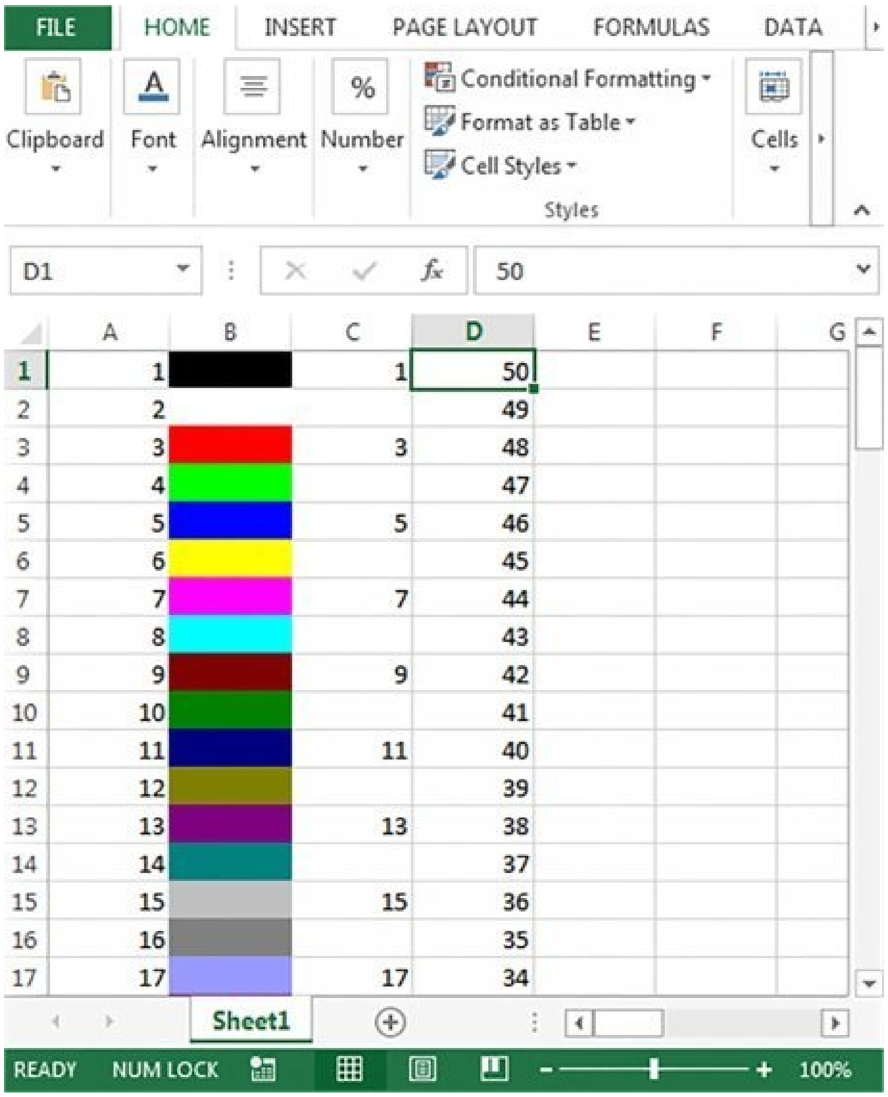
' In this case X decreases by 1' --- Comment

```
            Dim X As Integer, Row As Integer
            Row = 1
              For X = 50 To 0 Step -1
                Range("D" & Row).Value = X
                Row = Row + 1
              Next X
        End Sub
```



```
(General)                         Loop5
Sub Loop5()
' Fills cells from D1:D50 with values of X
' In this case X decreases by 1
Dim X As Integer, Row As Integer

Row = 1

    For X = 50 To 0 Step -1
        Range("D" & Row).Value = X
        Row = Row + 1
    Next X

End Sub
```

The output of the program is below:



D1 | 50

	A	B	C	D	E	F	G
1	1	⬛	1	50			
2	2			49			
3	3	🟥	3	48			
4	4	🟩		47			
5	5	🟦	5	46			
6	6	🟨		45			
7	7	🟪	7	44			
8	8	🟦		43			
9	9	🟫	9	42			
10	10	🟩		41			
11	11	🟦	11	40			
12	12	🟫		39			
13	13	🟪	13	38			
14	14	🟦		37			
15	15	⬜	15	36			
16	16	⬛		35			
17	17	🟪	17	34			

Exercises

1. Write a program to fill the values in the cells in the active worksheet with an increment of 2.
2. Using example 4, write a program to fill values in cells in the reverse order using the STEP function.
3. Write a program to fill in values in a spreadsheet from a specific cell.

The solutions to the exercises above are in the last chapter of the book. If you can write these programs, you will know how to write any kind of program using the for loop.

Do…Loop Statement

In the first part of the book, we looked at what the Do…Loop statement is and how you can use it in Excel VBA. In this section, we will look at the syntax and understand every part of the syntax. There are some examples and exercises in this section that will help you master the Do…Loop statement.

Syntax

Type 1

```
Do { While | Until } condition
   [ statements ]
   [ Continue Do ]
   [ statements ]
   [ Exit Do ]
   [ statements ]
Loop
```

Type 2

```
Do
   [ statements ]
   [ Continue Do ]
   [ statements ]
   [ Exit Do ]
   [ statements ]
Loop { While | Until } condition
```

Understanding The Parts

Term	**Definition**
Do	This term is necessary to include since this starts the Do Loop.

While	This is required unless you use UNTIL in the loop. This keyword will ensure that the editor runs the loop until the condition is false.
Until	This keyword is necessary unless you are using the WHILE keyword. This will ensure that the editor will run the loop until the condition holds true.
Condition	This is optional, but it should always be a Boolean expression. If the condition is nothing, the editor will treat it as false.
Statements	These are optional. You can add one or more statements that you want the editor to repeat until the condition holds true.
Continue Do	This is also an optional statement. If you use this statement in the loop, the editor will move to the next iteration of the loop.
Exit Do	This is optional, and if you use it, the editor will move out of the Do Loop.
Loop	This keyword is necessary since it terminates the loop.

You should use the Do…Loop structure if you want to repeat a set of statements infinitely until the condition holds true. If you want to repeat the statements in the loop for a specific number of times, you should use the For…Next statements. You can either use the Until or While keywords when you specify a condition, but you should never use both.

You can test the condition either at the start or the end of the loop. The first book mentions which structure you should use depending on when you want to test the condition. If you want to test the condition at the beginning the loop does not have to run even once. If you test the condition at the end of the loop, the statements in the body of the loop will run at least once. This condition is a Boolean value and is often a comparison of two values. These values can be of any data type that the editor can convert to Boolean.

You can nest a Do loop by adding another loop in it. You can also nest different control structures within the Do Loop. These concepts have been covered in the first book of the series.

You should remember that the Do…Loop structure is more flexible than the While…End While statement. This is because the former allows you to decide if you want to end the loop when the condition first becomes true or when it stops being true. You also have the ability to test the condition either at the start or the end of the loop.

Exit Do

You can use the Exit Do statement as an alternative way to exit the Do…Loop. This statement will transfer the control to the statements that follow the Loop statement. The Exit Do is used if you nest conditional statements within the loop. If you know that there is some condition that is unnecessary or makes it impossible for the editor to evaluate the statements within the loop. You can use this statement if you want to check for a condition that can lead to an endless loop. This statement will help you exit the loop immediately. You can use any number of Exit Do statements in the Do…Loop structure.

When you use the Exit Do statement in a nest Do loop, the editor will move from the statements within the innermost loop to the next level of nesting statements.

Example 1

In the example below, the editor will run the statements in the loop only when the index variable is greater than 10. The Until keyword will end the loop.

```
Dim index As Integer = 0

Do
```

```
    Debug.Write(index.ToString & " ")
    index += 1
Loop Until index > 10
Debug.WriteLine("")
```

The output will be,

```
0 1 2 3 4 5 6 7 8 9 10
```

Example 2

In the example below, we will use a While clause instead of the Until clause. The editor will test the condition at the start of the loop.

```
Dim index As Integer = 0
Do While index <= 10
    Debug.Write(index.ToString & " ")
    index += 1
Loop
Debug.WriteLine("")
```

The output will be,

```
0 1 2 3 4 5 6 7 8 9 10
```

Example 3

In the example below, the condition will ensure that the editor stops running the statements in the loop if the index variable is larger than 100. This example uses conditional statements within the loop, and the Exit Do statement in the program will cause the statement to stop if the value of index is greater than 10.

```
Dim index As Integer = 0
Do While index <= 100
    If index > 10 Then
        Exit Do
    End If
    Debug.Write(index.ToString & " ")
    index += 1
Loop
Debug.WriteLine("")
```

The output will be,

```
0 1 2 3 4 5 6 7 8 9 10
```

Example 4

In the example below, the editor will read every line in a text file. The OpenText method will open the text file and returns the StreamReader which will read the characters in the text. In the example below, the Peek method in the Do…Loop condition will determine whether there are additional characters present in the text.

```
Private Sub ShowText(ByVal textFilePath As String)
    If System.IO.File.Exists(textFilePath) = False Then
        Debug.WriteLine("File Not Found: " & textFilePath)
    Else
        Dim sr As System.IO.StreamReader =
System.IO.File.OpenText(textFilePath)
        Do While sr.Peek() >= 0
            Debug.WriteLine(sr.ReadLine())
        Loop
        sr.Close()
    End If
End Sub
```

Conditional Statements

Example 1

```
Module Nested
    Public Sub Main()
        ' Run the function as part of the WriteLine output.
        Console.WriteLine("Time Check is " & CheckIfTime() & ".")
    End Sub

    Private Function CheckIfTime() As Boolean
        ' Determine the current day of week and hour of day.
        Dim dayW As DayOfWeek = DateTime.Now.DayOfWeek
        Dim hour As Integer = DateTime.Now.Hour

        ' Return True if Wednesday from 2 to 3:59 P.M.,
        ' or if Thursday from noon to 12:59 P.M.
        If dayW = DayOfWeek.Wednesday Then
            If hour = 14 Or hour = 15 Then
                Return True
            Else
                Return False
            End If
        ElseIf dayW = DayOfWeek.Thursday Then
            If hour = 12 Then
                Return True
            Else
                Return False
            End If
        Else
            Return False
        End If
    End Function
End Module
'This example displays output like the following:
'Time Check is False.
```

Example 2

```
Module SingleLine
    Public Sub Main()
        'Create a Random object to seed our starting values
        Dim randomizer As New Random()
        Dim A As Integer = randomizer.Next(10, 20)
        Dim B As Integer = randomizer.Next(0, 20)
        Dim C As Integer = randomizer.Next(0, 5)
        'Let's display the initial values for comparison
        Console.WriteLine($"A value before If: {A}")
        Console.WriteLine($"B value before If: {B}")
        Console.WriteLine($"C value before If: {C}")
        ' If A > 10, execute the three colon-separated statements in the order
        ' that they appear
        If A > 10 Then A = A + 1 : B = B + A : C = C + B
        'If the condition is true, the values will be different
        Console.WriteLine($"A value after If: {A}")
        Console.WriteLine($"B value after If: {B}")
        Console.WriteLine($"C value after If: {C}")
    End Sub
End Module
'This example displays output like the following:
'A value before If: 11
'B value before If: 6
'C value before If: 3
'A value after If: 12
'B value after If: 18
'C value after If: 21
```

Chapter Six

Data Types in VBA

Data types are a way to define the different variables you use in the program to make it easier for the editor to perform the necessary calculations. The computer will always look at the data as a series of bits, but there are different types of data types that you can use in VBA. A computer can see only the binary value, 1000001b, but it does not do anything with that value. VBA will see the binary value as a letter or a number depending on the data type that you use to assign the value. The data type is important in understanding the value and working with it. Using a data type also ensures that the program follows certain rules. Otherwise, the data could become corrupted because the program could mishandle it.

Although a variable, in general, is simply a box for storing data, you can think of these data types as special boxes for storing specific kinds of data. Just as you would use a hatbox to store a hat and not a car engine, you use these special box types to store kinds of data. For example, you use a string to hold text, not logical (true/false) values. VBA supports several standard data types, including Boolean, Byte, Long, Currency, Integer, Single, Decimal, Double, String, Variant, Date and Object. In addition to using the defined data types, you can create user-defined data types so that you can mark the information as needed for your program. A user-defined data type gives you the power to extend the VBA interpretation of data. (The computer still looks at the data as binary information.) Each of the data type descriptions that follows has a different purpose, and you can work with the data type in a variety of ways.

Using strings for text

In the first book, we discussed the data type string, and you should have a clear idea of what a string is. If you do not remember it, go back to the first few chapters where we used a message box to give the user an output. When you create the message box, you will use the string as an input. The string is the most useful data type in VBA. This chapter only introduces strings. The next chapter provides information on how you can manipulate strings in VBA.

Understanding strings

A programmer uses fancy terms to describe objects that an average person will recognize. A string is a sequence of characters. The characters cannot always be printed, but they can always include some control characters that will determine what type of text will appear on the screen. A string will also include special characters like commas and other types of punctuation, or some special features like an umlaut or circumflex. A string can contain each of these elements, but the main part of a string is always text.

Adding strings together with + or &

Sometimes you will need to combine two or more strings to make a longer string. The process of adding two or more strings together is concatenation. For instance, you may want to combine the first name and last name of a person to create their full name. You may need to take this information from more than one source and combine it together to obtain new information.

Using character codes

Strings can contain several elements. In the earlier examples, we used a control character like vbCrLF. This constant is made up of two control characters: a line feed and a carriage return. The latter will send the cursor to the beginning of the sentence while the former moves the cursor to the next line. The result of using these control characters together is like pressing the Enter key on your keyboard. You can also use special functions like Chr, which will allow you to create a special character. You can combine this function with the Character Map utility to produce any character that you need for your program

Using numbers for calculations

Numbers form the basis for a lot of the information computers store. You use numbers to perform tasks in a spreadsheet, to express quantities in a database, and to show the current page in a document. Programs also use numbers to count things such as loops, to determine the position of items such as characters in a string and to check the truth value of a statement. Finally, VBA uses numbers in myriad ways, such as determining which character to display onscreen or how to interact with your code.

Understanding the numeric types

The numbers are always looked at as a single entity. Every number is simply just that, a number. The computer will view these numbers in different ways, and the reason for this diversity is that a processor works differently with numbers. Processors will either work with integer values - that is numbers without a decimal point - or with floating-point values, which are numbers with decimal points. The four basic number types include:

Integer

The integer data type does not have any decimal point in it. Integers can hold any whole numbers like 5 but never a number with a decimal, like 5.0. Essentially these numbers are the same, bu the first is an integer and the second is not.

Real

Unlike an integer, a real number contains a decimal point. The decimal section of the number does not necessarily have to contain a value, and the number 5.0 is a valid real number. VBA stores a read number in a different format when compared to the format of an integer.

Currency

A financial calculation always needs special accuracy, and a small error can cause larger problems. This numeric type always stores numbers with great precision, but it uses a lot of memory and takes a lot of time to process.

Decimal

A computer often stores information using the binary or base 2 format. Human beings use the base 10 or decimal system to store numbers when you need to work with them. Simple errors are often made when you convert numbers from one numbering system to another. These simple errors will create larger errors. The decimal numeric system will always store the number in a base 10 format which will eliminate many computing errors. This system requires more processing time and more memory than any other numeric type.

Using Boolean values to make decisions

The Boolean type is the easiest to use and understand. This type is used to indicate yes or no, true or false and on or off. You can use this type to work with any two-state information. It has commonly used to represent data values that are diametrically opposed.

Working with Operators

Operators determine how VBA works with two variables and what result it produces. The examples in this chapter use operators to add numbers and concatenate (add) strings. In both cases, your code uses the + operator to perform the task. The result differs. When you are using numbers, the result is a summation, such as 1 + 2 = 3. When you are using strings, the result is a concatenation, such as Hello + World = Hello World.

VBA groups operators into four areas:

- Arithmetic: Operators that perform math operations, such as addition (+), subtraction (-), division (/) and multiplication (*)
- Comparison: Operators such as less than (<), greater than (>) and equal (=) that compare two values and produce a Boolean result
- Concatenation: Operators such as & and + that are used to add two strings together
- Logical: Operators such as Not, And, Or and Xor that are used to perform Boolean operations on two variables

Exercises

1. Write a program to change the case of a letter in a cell.
2. Write a program to highlight misspelled words in a worksheet.
3. Write a program to obtain the numeric part of the string from a cell.

Chapter Seven

Parts of the Program

You should follow a syntax and a structure when you want to write a code in VBA to help the debugger understand what the point of your code is. This chapter formalizes the meaning for each structural element.

Defining the parts of a program

A program is the highest level of physical structure. It contains everything needed to perform a given task. A program can cross module, class module and form boundaries. The concept of a program comes from the earliest use of computers. A program acts as a container for the code used to implement a set of features required by the operating system or the user. Some people have a hard time understanding what a program is because modern software packages often define the term incorrectly. You are not creating a new program when you create a new project. A VBA project can actually contain a number of VBA programs."

Programming Blocks

A VBA program consists of building blocks. In fact, because programming is abstract, people tend to use physical examples to explain how things work. You still need to know about the abstract elements of VBA programming, or else you cannot write a program. This section explains the basic constructs of VBA programming. Every VBA code has the following elements:

Project

The project acts as a container for the modules, class modules and forms for a file. Excel users normally see just one project for the file that they have open.

Module, class module and forms

The three elements will always contain the main programming elements like procedures and class descriptions. A single project can have multiple class modules, forms and modules in it, but each of these must have a different name.

Function and Sub

The sub and function elements hold the statements or individual lines of code. A function will always return the value that the user requires while a sub does not. You can access the code using the Sub, but never through the function.

Therefore, you must always provide the VBA editor access to the program through a Sub.

Statement
Experts and other developers often call an individual line of code as a statement.

Using the Macro Recorder
The Macro Recorder will allow you to record all the actions and keystrokes that you perform when you work on Excel. You can use this to record some tasks like setting up a document or anything as simple as highlighting text.

The Macro Recorder can help you perform the following tasks:

- Create a macro based on your actions.
- Discover how Word performs certain tasks.
- Decide how to break your program into tasks.
- Help you create the basis for a more complex program.

The Macro Recorder is not a complete solution for your VBA needs. For example, you cannot use the Macro Recorder to create interactive programs without extra coding. The same holds true for programs that must change based on user input, the environment or the data you are manipulating. All these tasks require you to add more code. It is a good starting point for many structured programming tasks. You can get the basics down quickly using the Macro Recorder and then make changes as needed.

- Start the Macro Recorder.
- Perform all the steps that you normally perform to accomplish a task.
- Stop the Macro Recorder.
- Save the macro when the Office application prompts you.
- Optionally, open the resulting macro and make any required changes.

Using Subs
A sub is the simplest way to reduce the size of the code, packaging method that appears in the Macro dialog box. Consequently, the one place where you always use a Sub is the main entry point for a program unless the program is a utility that you use only for programming purposes. A second way to use a Sub is to

perform a task and not receive a direct return value. You can use a Sub to display an informational message. A Sub can modify information in several ways; it just cannot return a value. Only a function can return a value. You can, however, use arguments as a way to modify the information in a function by using a Sub. A second method relies on global variables. You can use the Sub as a way to break up large volumes of code. You can avoid creating a code that is written on many pages, and use Subs to break the code into smaller segments. This makes it easier for you and for another viewer to read the code.

Using Functions

You might not see a use for the Function after spending some time working with the Sub. Not every problem is a screw requiring the use of a screwdriver or a nail in search of a hammer. You use a Function for different problems than a Sub can answer. In most cases, there is a correct answer to using a Function or a Sub. For example, you always use a Sub when you want to access program code from within the host application, and you always use a Function when you want to perform a calculation with a return result.

A Function always returns a value, which makes it different from a Sub. For this reason, you can write functions that contain code that you plan to repeat a lot within a program. To process a list of names, you might create a Function to process each name individually and then call that Function once for each name. The Function can provide the processed information as a return value. In Chapter 5, I describe how to create repeating code using structures such as Do...Until.

You can also use a Function for public code that you do not want to list in the Macro dialog box. You normally do not see a Function listed in the Macro dialog box — this dialog box usually lists only Subs.

Comments

It is important to write comments in your code to help other users understand the purpose of the code.

Writing basic comments

Comments can take several forms. One always writes a pseudo-comment in the program since this is the most natural one to use. A developer will always add comments to the program he or she is writing. They will also provide information about who wrote the program or when it was written. These comments will also provide information on the list of updates made to the code. Some developers begin to write better comments at this point.

An important comment that you should add to your code is why you would want to write the original program. You should also explain why you chose to write it in a specific way. You cannot simply say that the program will perform a specific task. This is not enough since you can perform the same task in different ways. When you talk about why you chose that specific way to write the code, you can avoid making any errors when you update the code. You will also know when you need to update the code.

As a good programmer, you should include the mistakes made as a part of the comments if you want to help another programmer avoid making the same mistakes. These comments will always help a beginner or an expert. That way they can avoid the mistakes that one may usually make.

Knowing when to use comments

You should always use comments whenever and wherever you think you need them. You may believe that comments are difficult to include and type, but you can include one or two lines of comments to explain the program that you are writing. You are correct — writing good comments can be time-consuming and can be difficult because writing these makes you think yet again about the code. When you do not have enough comments in your code, you will find it difficult to update that program. You may also have to start writing the code from scratch because you do not know what your code is all about.

Writing Good Comments

A comment is always good if you can understand what it says. Do not use fancy terms — write everything in plain terms that you can understand. If you think you want to explain a certain line of code, feel free to do it. You can also write comments against every line of code since that will help an amateur understand your train of thought.

Chapter Eight

Arrays

Arrays will allow you to store more than one item in a single variable or container that you can use in your program. You should think of an array as a large box with a finite number of smaller boxes within it. Every small box will store some value depending on the data type of the array. You can choose the number of small boxes that you want to store in your array when you create it. You should use the array when you must store several related items of the same data type.

Structured Storage

Arrays are a list of items. A classic example of an array is a to-do list that you prepare for yourself. The piece of paper where you write your tasks form a single container. This single container will hold several strings, and each of these strings will contain a task that you need to perform. Similarly, you can create a paper in your VBA program in the form of an array. You can define an array using a variety of techniques, but each of these techniques will use similar approaches.

Example

```
‘ Tell VBA to start all arrays at 0.
Option Base 0
Public Sub SingleDimension()
‘ Define an output string.
Dim Output As String
‘ Define a variant to hold individual strings.
Dim IndividualString As Variant
‘ Define the array of strings.
Dim StringArray(5) As String
‘ Fill each array element with information.
StringArray(0) = “This”
StringArray(1) = “Is”
StringArray(2) = “An”
StringArray(3) = “Array”
StringArray(4) = “Of”
StringArray(5) = “Strings”
‘ Use the For Each...Next statement to get each array
‘ element and place it in a string.
For Each IndividualString In StringArray
‘ Create a single output string with the array
‘ array elements.
Output = Output + IndividualString + “ “
Next
‘ Display the result.
MsgBox Trim(Output), _
vbInformation Or vbOKOnly, _
“Array Content”
End Sub
```

If you look at the above code, you will notice that the code starts with the Option Base 0 statement. This statement will tell VBA if it needs to start counting the elements in the array from 0 or 1. The default setting in VBA is that it will count the elements in the array from 0. Most programming languages will use 0 as the starting point, and it is for this reason that the developers at Microsoft made the default 0 for VBA. Older versions of VBA do use 1 as the starting point.

If you want your program to work in every environment, you should include the Option Base statement. Since the array will begin at 0 and not at 1, you can only store six items in the array, although you have defined that the array has five elements. The number you include in the declaration is always at the top, but this is not one of the numbers in the element.

Array Types

One can classify an array in several ways, and the first method is the type of data that the array will hold. A string array is very different from an integer array, but the array will always ensure that the elements are unique. If you use a Variant data type, you can mix the data types in an array. You should always be careful when you use this technique since it can lead to errors which are difficult to debug.

Another method is to define the dimensions in an array. The dimension will define the number of directions in which the array will hold the information. A simple list, like the one in the earlier "Understanding array usage" section, gives an example of a single-dimensional array. A table which consists of columns and rows is a two-dimensional array. One can create an array using any number of dimensions.

Example: Adding an Element to an Array

```
Dim a As Range

Dim arr As Variant  'Just a Variant variable (i.e., don't pre-define it as an
array)

For Each a In Range.Cells

   If IsEmpty(arr) Then

      arr = Array(a.value) 'Make the Variant an array with a single
element

   Else

      ReDim Preserve arr(UBound(arr) + 1) 'Add next array element

      arr(UBound(arr)) = a.value         'Assign the array element

   End If

Next
```

VBA Array

In this section, we will look at the steps you need to follow to create an array.

Step 1 – Create a New Workbook

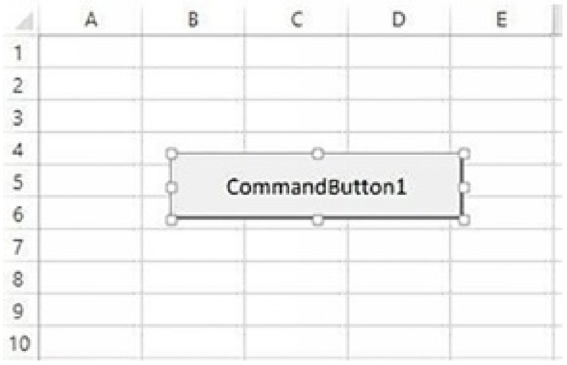
1. Open Microsoft Excel.

2. Save the excel workbook with the extension .xlsm

Step 2 – Add a Command Button

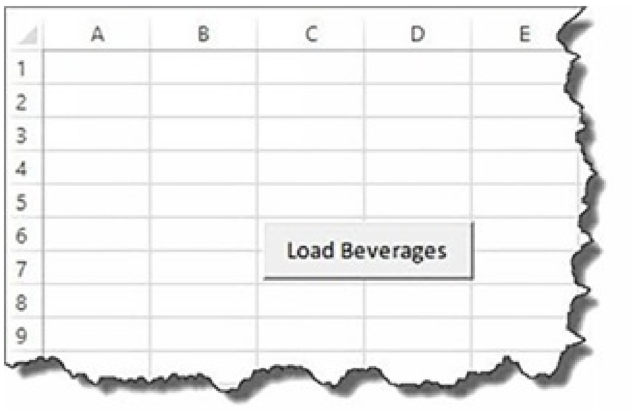
Now that you are familiar with creating an interface in a workbook. The previous chapters in the book will help you gather more information about the subroutines or subs and functions in VBA.

1. Add a command button to the active worksheet.



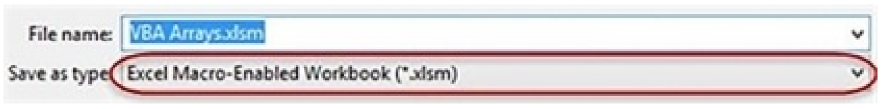
2. Set the property name to cmdLoadBeverages.
3. Now, set the Caption Property as Load Beverages.

The interface should now display the following:



Step 3 – Save the File

1. You should now click the save as button in the macro-enabled form.



File name: VBA Arrays.xlsm

Save as type: Excel Macro-Enabled Workbook (*.xlsm)

Step 4 – Write the Code

Let us now write the code for the application.

1. Right click on the button and choose to view the code.
2. Add the code in the code window.

```
Private Sub cmdLoadBeverages_Click()
   Dim Drinks(1 To 4) As String
   Drinks(1) = "Pepsi"
   Drinks(2) = "Coke"
   Drinks(3) = "Fanta"
   Drinks(4) = "Juice"
   Sheet1.Cells(1, 1).Value = "My Favorite Beverages"
   Sheet1.Cells(2, 1).Value = Drinks(1)
   Sheet1.Cells(3, 1).Value = Drinks(2)
   Sheet1.Cells(4, 1).Value = Drinks(3)
   Sheet1.Cells(5, 1).Value = Drinks(4)
End Sub
```

Example to Enter Student's Marks

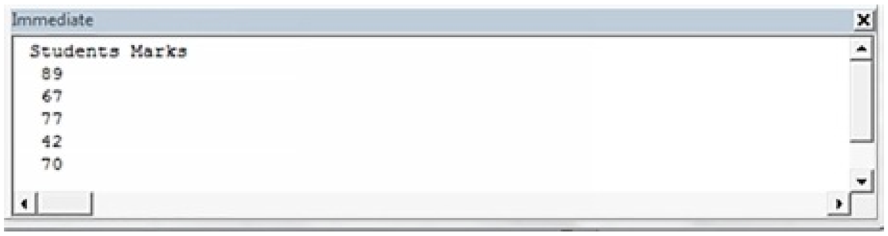
Without An Array

In the example below, we will look at how you can enter the marks for every student without using an array.

```
Public Sub StudentMarks()
   With ThisWorkbook.Worksheets("Sheet1")
      ' Declare variable for each student
      Dim Student1 As Integer
      Dim Student2 As Integer
      Dim Student3 As Integer
```

```
        Dim Student4 As Integer
        Dim Student5 As Integer
        ' Read student marks from cell
        Student1 = .Range("C2").Offset(1)
        Student2 = .Range("C2").Offset(2)
        Student3 = .Range("C2").Offset(3)
        Student4 = .Range("C2").Offset(4)
        Student5 = .Range("C2").Offset(5)
        ' Print student marks
        Debug.Print "Students Marks"
        Debug.Print Student1
        Debug.Print Student2
        Debug.Print Student3
        Debug.Print Student4
        Debug.Print Student5
    End With
End Sub
```

The output will be the following,



```
Students Marks
 89
 67
 77
 42
 70
```

Using an Array

```
Public Sub StudentMarksArr()
    With ThisWorkbook.Worksheets("Sheet1")
        ' Declare an array to hold marks for 5 students
        Dim Students(1 To 5) As Integer
        ' Read student marks from cells C3:C7 into array
        Dim i As Integer
        For i = 1 To 5
            Students(i) = .Range("C2").Offset(i)
        Next i
        ' Print student marks from the array
        Debug.Print "Students Marks"
        For i = LBound(Students) To UBound(Students)
            Debug.Print Students(i)
        Next i
    End With
End Sub
```

Notice the difference in the variables used in the two programs, and also notice the length of the program.

Example with Loops

```
Public Sub ArrayLoops()
    ' Declare  array
    Dim arrMarks(0 To 5) As Long
    ' Fill the array with random numbers
    Dim i As Long
    For i = LBound(arrMarks) To UBound(arrMarks)
        arrMarks(i) = 5 * Rnd
    Next i
    ' Print out the values in the array
    Debug.Print "Location", "Value"
    For i = LBound(arrMarks) To UBound(arrMarks)
```

```
        Debug.Print i, arrMarks(i)
    Next i
End Sub
```

Sorting an Array

```
Sub QuickSort(arr As Variant, first As Long, last As Long)
  Dim vCentreVal As Variant, vTemp As Variant
  Dim lTempLow As Long
  Dim lTempHi As Long
  lTempLow = first
  lTempHi = last
  vCentreVal = arr((first + last) \ 2)
  Do While lTempLow <= lTempHi
    Do While arr(lTempLow) < vCentreVal And lTempLow < last
      lTempLow = lTempLow + 1
    Loop
    Do While vCentreVal < arr(lTempHi) And lTempHi > first
      lTempHi = lTempHi - 1
    Loop
    If lTempLow <= lTempHi Then
        ' Swap values
        vTemp = arr(lTempLow)
        arr(lTempLow) = arr(lTempHi)
        arr(lTempHi) = vTemp
         ' Move to next positions
        lTempLow = lTempLow + 1
        lTempHi = lTempHi - 1
      End If
    Loop
    If first < lTempHi Then QuickSort arr, first, lTempHi
  If lTempLow < last Then QuickSort arr, lTempLow, last
  End Sub
```

Example for Creating a Two-Dimensional Array

```
Public Sub TwoDimArray()
    ' Declare a two dimensional array
    Dim arrMarks(0 To 3, 0 To 2) As String
    ' Fill the array with text made up of i and j values
    Dim i As Long, j As Long
    For i = LBound(arrMarks) To UBound(arrMarks)
        For j = LBound(arrMarks, 2) To UBound(arrMarks, 2)
            arrMarks(i, j) = CStr(i) & ":" & CStr(j)
        Next j
    Next i
    ' Print the values in the array to the Immediate Window
    Debug.Print "i", "j", "Value"
    For i = LBound(arrMarks) To UBound(arrMarks)
        For j = LBound(arrMarks, 2) To UBound(arrMarks, 2)
           Debug.Print i, j, arrMarks(i, j)
        Next j
    Next i
End Sub
```

Exercise

1. Write a program to sort an array using the bubble sort method.

Chapter Nine

Working with Excel Workbooks and Worksheets

The Workbook Collection

The Workbooks collection provides a list of all workbooks that you have open at a given time. You can select a single workbook from this list that you want to use in your program. The Workbook object that you select will provide general information about the file, including its name and location. You can also use this object to access other major objects in the document. These objects include standalone Chart objects and Worksheet objects.

Example:

```
Public Sub WorkbookDemo()
‘ Holds the output data.
Dim Output As String
‘ Get the test workbook.
Dim ActiveWorkbook As Workbook
Set ActiveWorkbook =
Application.Workbooks(“ExcelObjects.xls”)
‘ Get the workbook name and location.
Output = “Name: “ + ActiveWorkbook.Name + vbCrLf + _
“Full Name: “ + ActiveWorkbook.FullName + vbCrLf + _
“Path: “ + ActiveWorkbook.Path + vbCrLf + vbCrLf
‘ Holds the current sheet.
Dim CurrSheet As Worksheet
‘ Look for every sheet.
Output = “Worksheet List:” + vbCrLf
For Each CurrSheet In ActiveWorkbook.Worksheets
Output = Output + CurrSheet.Name + vbCrLf
Next
‘ Holds the current chart.
```

```
Dim CurrChart As Chart
‘ Look for every chart.
Output = Output + vbCrLf + “Chart List:” + vbCrLf
For Each CurrChart In ActiveWorkbook.Charts
Output = Output + CurrChart.Name + vbCrLf
Next
‘ Display the output.
MsgBox Output, vbInformation Or vbOKOnly, “Object List”
End Sub
```

The code starts by using the Application Workbooks collection which will allow you to retrieve a single Workbook object. You should always use the full name of the Excel file as the index in the collection. You should also include the extension of the file. The resulting workbook object will contain information about the document. This object will also provide the summary information of the document, and you can use this object to control and maintain the windows, and also add new elements like worksheets.

Once the code accesses the workbook, it will use the ActiveWorkbook object to access the worksheets in the list. The code will rely on the For Each...Next statement to access these worksheets. Alternatively, you can use an index to access the individual worksheets in the code. The Worksheet, ActiveWorksheet, contains properties and methods for manipulating any data that the Worksheet contains, including embedded objects, such as charts or even pictures. Every worksheet appears in the ActiveWorkbook object list by its object name (not the friendly name that you give it), so you can access them without using the Worksheets collection.

Only independent charts will appear when you use the ActiveWorkbook object. The same technique can be used to access any Chart object in the worksheet as a Worksheet object. The difference is that you should use Charts Collection instead of the Worksheets Collection. You should note that the Chart names will appear in the list of objects that are present in the ActiveWorkbook. This means that you can access the chart directly without having to use the Charts collection.

The Worksheet Collection

One of the easiest methods to access a worksheet in many situations is to use the Sheets collection. You do not follow the Excel object hierarchy when you want to identify the worksheet you want tot work with. If you access the worksheet at the top of the hierarchy, it means that there are no objects that exist at lower levels available either, so this technique is a tradeoff.

You can access any type of sheet, and not just a worksheet if you use the Sheets collection. A standalone Chart object that you use in any of the sheets can also come into this collection. Look at the example in the previous section, "Using the Workbooks collection," and you will notice that the charts and worksheets are treated as different objects.

Example:

```
Public Sub ListSheets()
‘ An individual entry.
Dim ThisEntry As Variant
‘ Holds the output data.
Dim Output As String
‘ Get the current number of worksheets.
Output = “Sheet Count: “ + _
CStr(Application.Sheets.Count)
‘ List each worksheet in turn.
For Each ThisEntry In Application.Sheets
‘ Verify there is a sheet to work with.
If ThisEntry.Type = XlSheetType.xlWorksheet Then
Output = Output + vbCrLf + ThisEntry.Name
End If
Next
‘ Display the result.
MsgBox Output, _
vbInformation or vbOKOnly, _
“Worksheet List”
```

```
End Sub
```

In the example above, we will create a Variant, which will hold the different sheet types. If you use a Chart or Worksheet object, the code will fail since the Sheets enumeration will return a valid type, but not the type you need. The issue with using this Variant is that the editor in VBA cannot provide automatic completion or balloon help. You have to ensure that you type the method in correctly and use the correct property names without any help.

Once the code creates the necessary variables, it will provide the number of sheets in the workbook. This number will include all the worksheets and charts in the workbook, and not just the sheets.

A For Each...Next loop will retrieve each sheet in turn. You should notice how we use the If...Then statement to compare the values of the Variant type and the XlSheetType.xlWorksheet constant. When you use this technique, you can separate the worksheet you are using from other Sheets collection types whenever necessary.

Charts Collection

The Charts collection is a way to build a custom chart whenever necessary. An advantage of creating charts by using a code is that they do not use too much space, and you can create a variety of different charts without spending too much time on the theme.

Example:

```
Public Sub BuildChart()

‘ Create a new chart.

Dim NewChart As Chart

Set NewChart = Charts.Add(After:=Charts(Charts.Count))

‘ Change the name.

NewChart.Name = “Added Chart”

‘ Create a series for the chart.

Dim TheSeries As Series
```

```
NewChart.SeriesCollection.Add _
Source:=Worksheets(“My Data Sheet”).Range(“A$3:B$8”)
Set TheSeries = NewChart.SeriesCollection(1)
‘ Change the chart type.
TheSeries.ChartType = xl3DPie
‘ Change the series title.
TheSeries.Name = “Data from My Data Sheet”
‘ Perform some data formatting.
With TheSeries
.HasDataLabels = True
.DataLabels.ShowValue = True
.DataLabels.Font.Italic = True
.DataLabels.Font.Size = 14
End With
‘ Modify the chart’s legend.
With NewChart
.HasLegend = True
.Legend.Font.Size = 14
End With
‘ Modify the 3-D view.
With NewChart
.Pie3DGroup.FirstSliceAngle = 90
.Elevation = 45
End With
‘ Format the chart title.
```

```
NewChart.ChartTitle.Font.Bold = True
NewChart.ChartTitle.Font.Size = 18
NewChart.ChartTitle.Format.Line.DashStyle _
= msoLineSolid
NewChart.ChartTitle.Format.Line.Style = msoLineSingle
NewChart.ChartTitle.Format.Line.Weight = 2
‘ Compute the optimal plot area size.
Dim Size As Integer
If NewChart.PlotArea.Height > NewChart.PlotArea.Width
Then
Size = NewChart.PlotArea.Width
Else
Size = NewChart.PlotArea.Height
End If
‘ Reduce the plot area by 10%.
Size = Size - (Size * 0.1)
‘ Format the plot area.
With NewChart.PlotArea
.Interior.Color = RGB(255, 255, 255)
.Border.LineStyle = XlLineStyle.xlLineStyleNone
.Height = Size
.Width = Size
.Top = 75
.Left = 100
End With
```

```
‘ Format the labels.

Dim ChartLabels As DataLabel

Set ChartLabels = TheSeries.DataLabels(0)

ChartLabels.Position = xlLabelPositionOutsideEnd

End Sub
```

In the above example, the code will create a new chart. This chart will appear in the workbook as the last chart, but will not appear as the last item in the workbook. A worksheet which appears after the last chart will also appear after the new chart that is created. The NewChart.Name property will change the name that appears at the bottom of the chart. This property does not change the name of the chart.

At this point, the chart is blank, and you must add a clear one series to the chart if you want to display some data on it. A pie chart will use only one series at a time, but there are other charts that allow you to use multiple data series. For instance, you can use a bubble chart to show multiple series of data. The next task of the code will create a data series based on the worksheet named My Data Sheet. You will notice that the code cannot set TheSeries variable equal to the output of the method Add in this example. Therefore, it uses an additional step to obtain the new series from the SeriesCollection collection.

You should also notice that the Range property has two columns of information. When you are working with Excel 2007, the first column defines the XValues property for the chart. The XValues property determines the entries in the legend for a pie chart. On the other hand, these values appear at the bottom of the display for a bar chart. In both cases, you want to display the labels onscreen so that you can see their effect on the overall display area.

Exercises

1. Write a program to unhide all worksheets in a workbook.
2. Write a program to hide all worksheets in the workbook except for the active worksheet.
3. Write a program to sort worksheets alphabetically using VBA.

Chapter Ten

How to Redirect the Flow

You might run into situations where the existing program flow does not work, and you must disrupt it to move somewhere else in the code. The GoTo statement provides a means of redirecting program flow. Used carefully, the GoTo statement can help you overcome specific programming problems. Unfortunately, the GoTo statement has caused more problems (such as creating hard-to-understand code and hiding programming errors) than any other programming statement because it has a great potential for misuse. Novice programmers find it easier to use the GoTo statement to overcome programming errors rather than to fix these problems. Always use the GoTo statement with extreme care. Designing your code to flow well before you write it and fixing errors when you find them are both easier than reading code with misused GoTo statements.

Using the GoTo statement correctly

The GoTo statement allows you to redirect the flow of your program. The first book of the series also provides information on this statement. Before you use the GoTo statement, you should always think of a different way to perform the redirection, like using a loop. If there is not any other way to perform the programming task efficiently, using a GoTo statement is acceptable.

You might run into situations where the existing program flow does not work, and you must disrupt it to move somewhere else in the code. The GoTo statement provides a means of redirecting program flow. Used carefully, the GoTo statement can help you overcome specific programming problems. Unfortunately, the GoTo statement has caused more problems than any other programming statement because it has a great potential for misuse. Novice programmers find it easier to use the GoTo statement to overcome programming errors rather than to fix these problems. Always use the GoTo statement with extreme care. Designing your code to flow well before you write it and fixing errors when you find them are both easier than reading code with misused GoTo statements.

Loops

You should never use the GoTo statement to replace the end statement in a loop.

The statements in your loop will always give the user Never use a GoTo statement as a loop replacement. The statements used for loops signal others about your intent. In addition, standard loop statements contain features that keep bugs, such as endless loops, to a minimum.

Exits

You should avoid using the GoTo statement when you want to exit a program. You can, however, use the End statement for the same task.

Program flow problems

If you detect any problems in the flow of your program, you should check the pseudo-code and then design the documents again. You have to ensure that you always implement the design correctly. The design might require change as well. Do not assume that the design is correct, especially if this is the first attempt.

Chapter Eleven

Error Handling

The easiest errors that you can avoid are the syntax errors, but these are some of the hardest errors to spot. The error can be because of the misuse of a punctuation, misuse of a language element or a spelling mistake. If you forget to include the End If statement in an If…Then statement, you have made a syntax error.

Typos are common syntax errors. These are especially hard to find if you make those errors in variable names. For example, the editor in VBA will view the MySpecialVariable and MySpecialVaraible as different variables, but there is a possibility that you will miss it when you begin to write the code. When you include the Adding Option Explicit at the beginning of the module, form or class module that you create, you can remove this problem. VBA can help you find a variety of typos if you add this statement to the start of your code. It is important that you use this statement in every part of your program when you write it.

There are times when you miss some of the subtle aids in locating the errors in syntax if you do not understand or view the tasks that the IDE or Integrated Development Environment performs. VBA will only display the balloon help feature when the editor in VBA can recognize the function name that you need to enter. If you do not see a balloon help button, you should understand that VBA does not know what function name you are referring to. This means that you will need to look at your code to identify the error. Unfortunately, this feature will only work where the editor in VBA will display the balloon help option. This does not work when you use property names.

Understanding compile errors

VBA uses a compiler to look for any errors that will prevent the program from functioning properly. You can create an If...Then statement and not include the End If statement in the program. The compiler will continue to run continuously and will allow you to find the mistakes in the code immediately once you make them.

VBA will use a compiler to find the syntax error in your code and then display an error message. Try the following when you write a new program. Open a new project, create a Sub using a specific name and type MsgBox(). Now, press

Enter. VBA will display a message box, which will state that it was expecting the equal to sign. If you use the parentheses after the keyword MsgBox, VBA will expect that you should include a result variable, which will hold the required result. For example, MyResult = MsgBox(“My Prompt”). As mentioned earlier, the debugger highlights the error in red.

Understanding run-time errors

A run-time error often occurs when there is an issue with something outside of your program. There are times when you type in the incorrect information and other times when the system rejects your access to the memory or disk. Your VBA code is completely correct, but the code will fail to function since there is an external error. Most companies, like Microsoft, always run a beta program to avoid any run-time errors. A beta program is a one that programmers develop to test their vendor-sponsored program before they release it into the market.

Understanding semantic errors

This is a particularly difficult error to understand and find since it is a semantic error. This error occurs when the VBA code and logic are both correct, but the meaning behind the code is incorrect. For instance, you can use the Do...Until loop in place of the Do...While loop. It may the case that the code is correct and the logic behind the code is also correct, but the result is not what you expected since the meaning of a Do...Until loop is different from the meaning of a Do...While loop.

When you write a code, the words you use in the code should match your intent. Since a good book always relies on precise terms, a good program also relies only on the precise statements that you use in VBA. These statements will help VBA understand what you want it to do. One of the best ways to avoid making any semantic errors in the application is to always plan your program in advance. You should use a pseudo-code to “pre-write” the design, and then convert that code into the actual VBA code.

Chapter Twelve

Solutions and Additional Programs

Sheet Protection

Example 1

'This code will protect all sheets in the workbook

```
Sub ProtectAllSheets()
Dim ws As Worksheet
For Each ws In Worksheets
ws.Protect
Next ws
End Sub
```

Example 2

'This macro code will lock all the cells with formulas

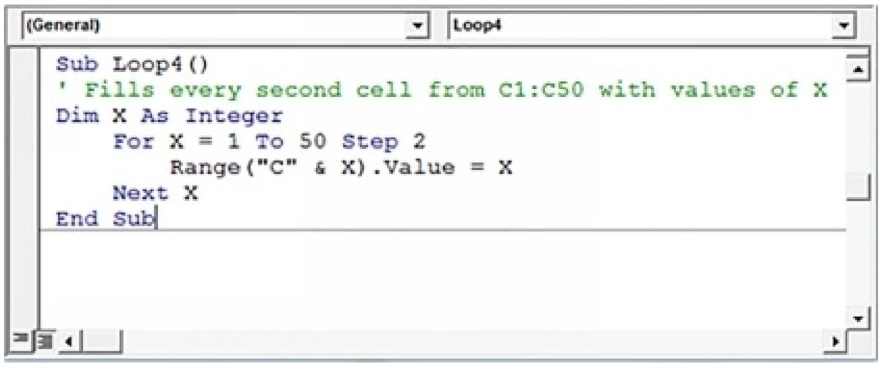
```
Sub LockCellsWithFormulas()
With ActiveSheet
   .Unprotect
   .Cells.Locked = False
   .Cells.SpecialCells(xlCellTypeFormulas).Locked = True
   .Protect AllowDeletingRows:=True
End With
End Sub
```

For Loop

Exercise 1

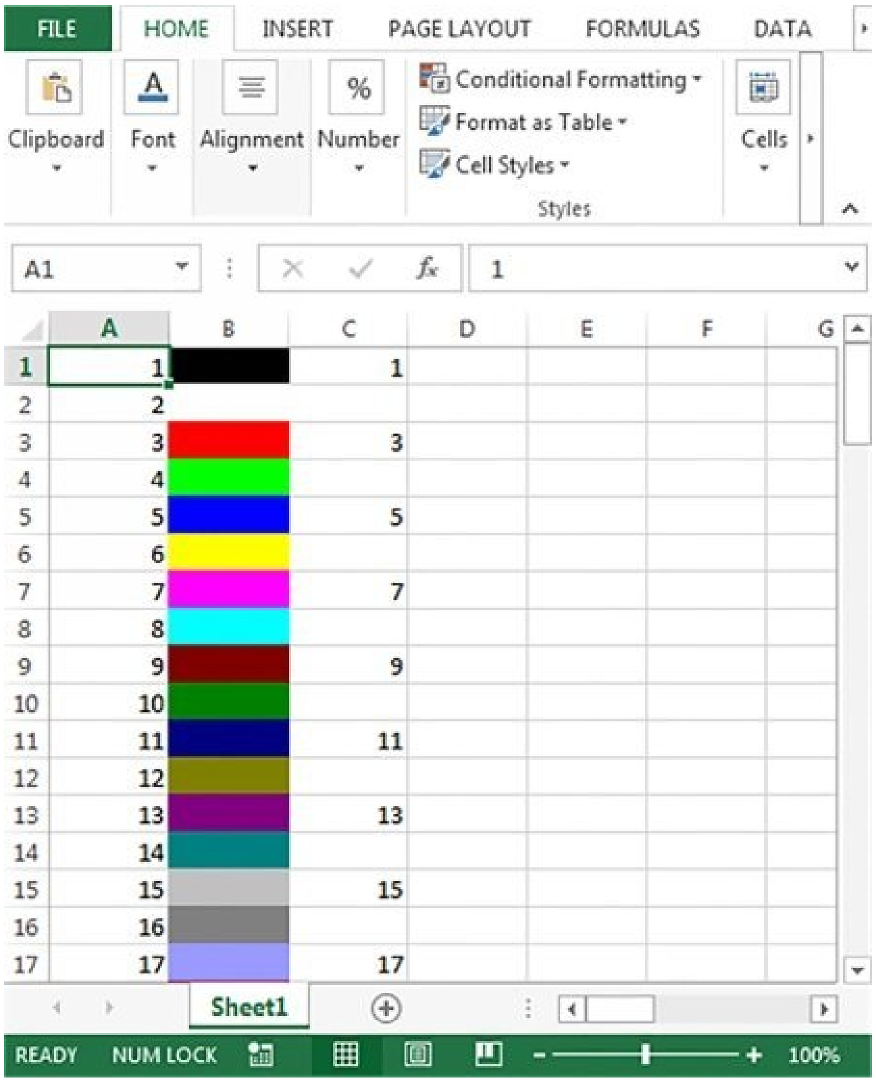
```
Sub Loop4()
```

```
' Fills every second cell from C1:C50 with values of X' --- Comment
Dim X As Integer
    For X = 1 To 50 Step 2
        Range("C" & X).Value = X
    Next X
End Sub
```



```
Sub Loop4()
' Fills every second cell from C1:C50 with values of X
Dim X As Integer
    For X = 1 To 50 Step 2
        Range("C" & X).Value = X
    Next X
End Sub
```

The output will be:



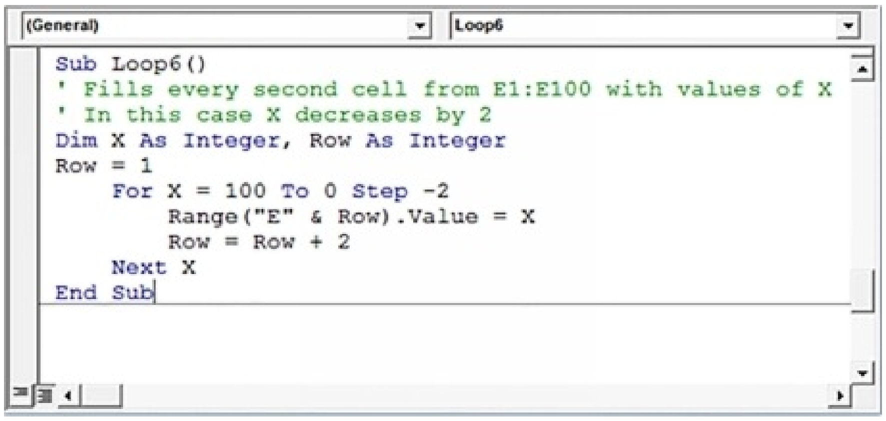
	A	B	C	D	E	F	G
1	1	⬛	1				
2	2						
3	3	🟥	3				
4	4	🟩					
5	5	🟦	5				
6	6	🟨					
7	7	🟪	7				
8	8	🟦					
9	9	🟥	9				
10	10	🟩					
11	11	🟦	11				
12	12	🟫					
13	13	🟪	13				
14	14	🟦					
15	15	⬜	15				
16	16	⬛					
17	17	🟪	17				

Exercise 2

```
Sub Loop6()
' Fills every second cell from E1:E100 with values of X' --- Comment
' In this case X decreases by 2' --- Comment
Dim X As Integer, Row As Integer
Row = 1
    For X = 100 To 0 Step -2
        Range("E" & Row).Value = X
        Row = Row + 2
```

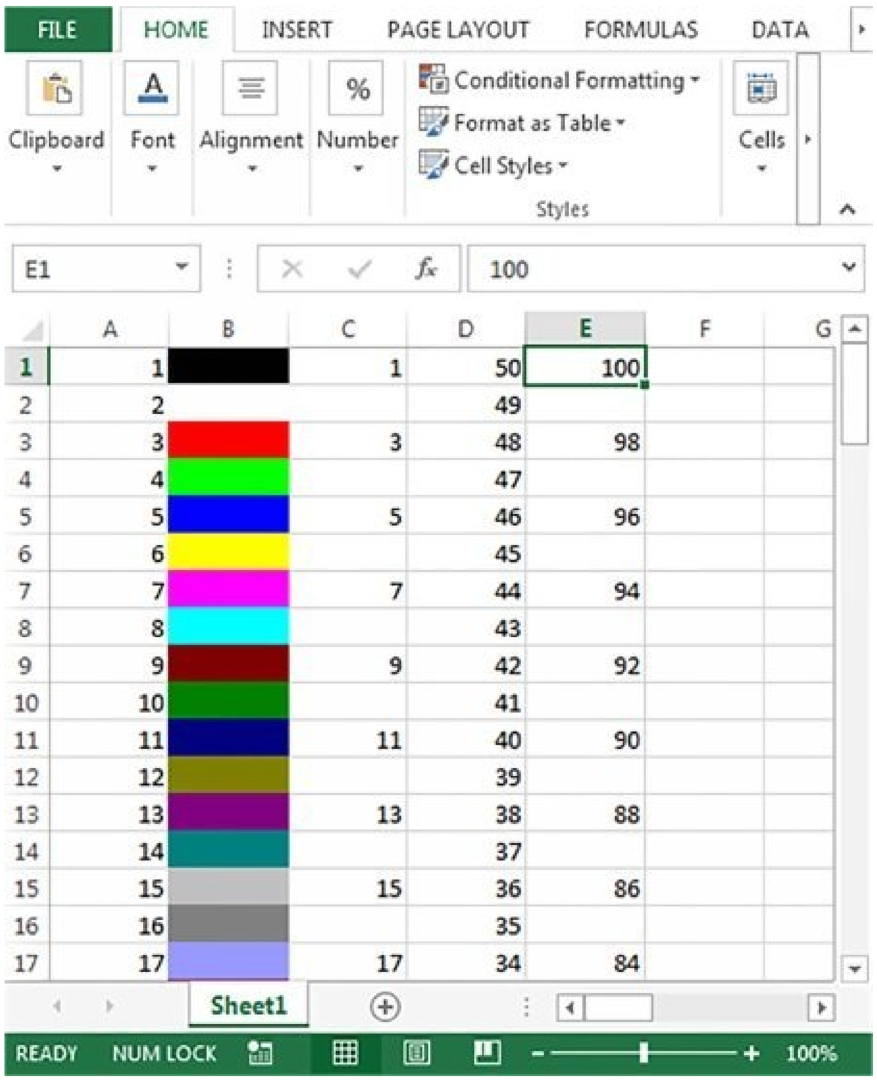
Next X

End Sub



```
Sub Loop6()
' Fills every second cell from E1:E100 with values of X
' In this case X decreases by 2
Dim X As Integer, Row As Integer
Row = 1
    For X = 100 To 0 Step -2
        Range("E" & Row).Value = X
        Row = Row + 2
    Next X
End Sub
```

The output will be,

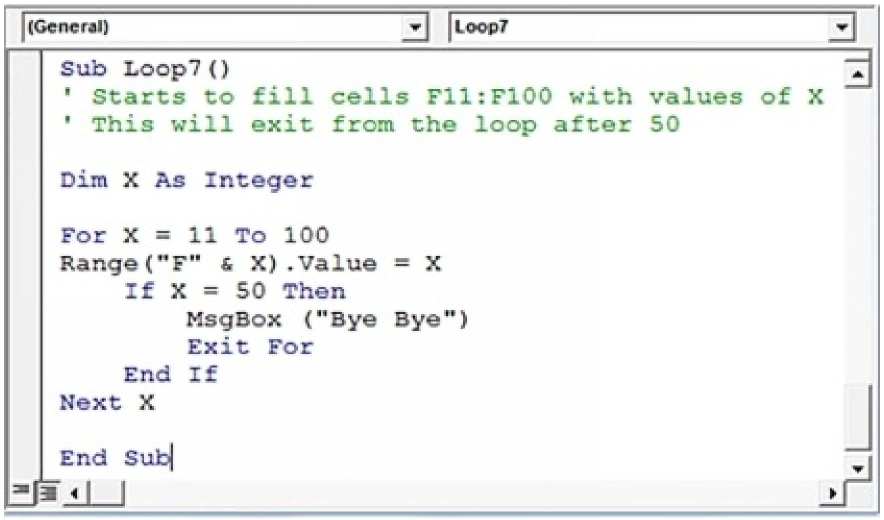


E1 | 100

	A	B	C	D	E	F	G
1	1	⬛	1	50	100		
2	2			49			
3	3	🟥	3	48	98		
4	4	🟩		47			
5	5	🟦	5	46	96		
6	6	🟨		45			
7	7	🟪	7	44	94		
8	8	🟦		43			
9	9	🟫	9	42	92		
10	10	🟩		41			
11	11	🟦	11	40	90		
12	12	🟫		39			
13	13	🟪	13	38	88		
14	14	🟦		37			
15	15	⬜	15	36	86		
16	16	⬛		35			
17	17	🟪	17	34	84		

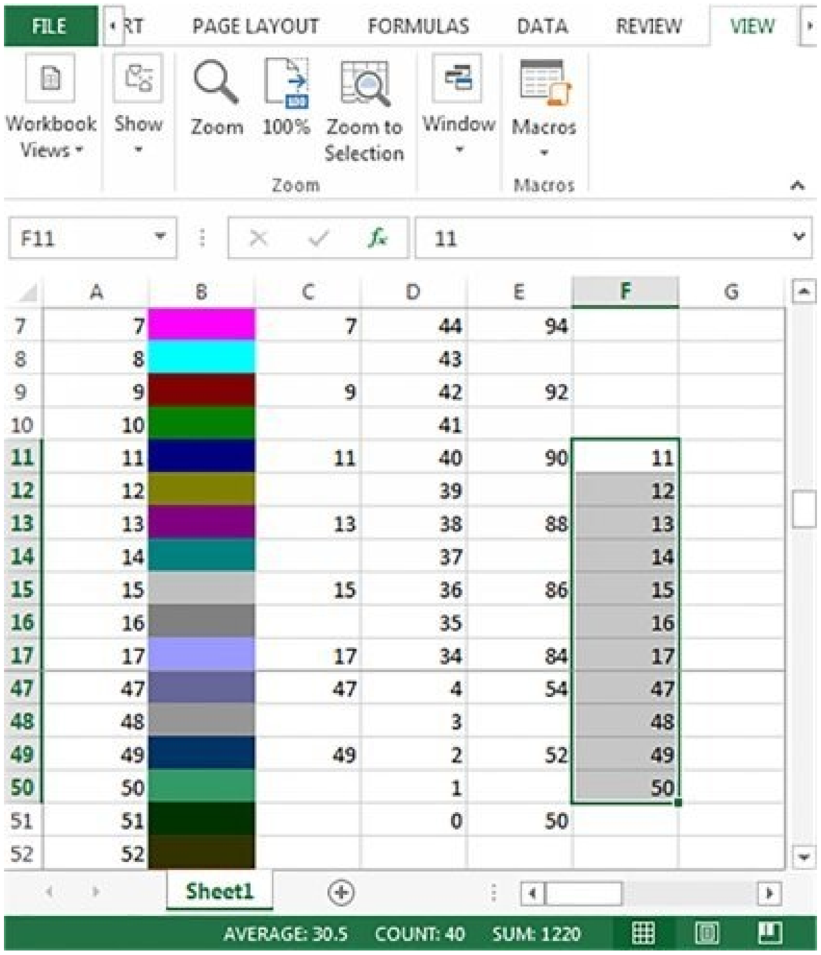
Exercise 3

```
Sub Loop7()
' Starts to fill cells F11:F100 with values of X' --- Comment
' This will exit from the loop after 50' --- Comment
Dim X As Integer
    For X = 11 To 100
        Range("F" & X).Value = X
            If X = 50 Then
                MsgBox ("Bye Bye")
                Exit For
            End If
    Next X
End Sub
```



```
Sub Loop7()
' Starts to fill cells F11:F100 with values of X
' This will exit from the loop after 50

Dim X As Integer

For X = 11 To 100
Range("F" & X).Value = X
    If X = 50 Then
        MsgBox ("Bye Bye")
        Exit For
    End If
Next X

End Sub
```

The output will be,

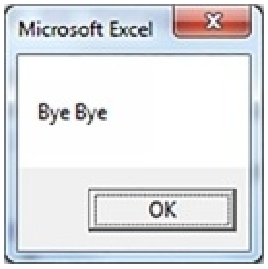


F11 | 11

	A	B	C	D	E	F	G
7	7	🟪	7	44	94		
8	8	🟦		43			
9	9	🟥	9	42	92		
10	10	🟩		41			
11	11	🟦	11	40	90	11	
12	12	🟨		39		12	
13	13	🟪	13	38	88	13	
14	14	🟩		37		14	
15	15	⬜	15	36	86	15	
16	16	▨		35		16	
17	17	🟦	17	34	84	17	
47	47	🟪	47	4	54	47	
48	48	▨		3		48	
49	49	🟦	49	2	52	49	
50	50	🟩		1		50	
51	51	🟩		0	50		
52	52	🟫					

AVERAGE: 30.5 COUNT: 40 SUM: 1220

If you enter 50 in the cell F50, the editor will display the following message box on your screen.


Microsoft Excel

Bye Bye

OK

Strings

Example 1

'This code will change the Selection to Upper Case

```
        Sub ChangeCase()
        Dim Rng As Range
        For Each Rng In Selection.Cells
        If Rng.HasFormula = False Then
        Rng.Value = UCase(Rng.Value)
        End If
        Next Rng
    End Sub
```

Example 2

'This code will highlight the cells that have misspelled words

```
    Sub HighlightMisspelledCells()
        Dim cl As Range
        For Each cl In ActiveSheet.UsedRange
        If Not Application.CheckSpelling(word:=cl.Text) Then
        cl.Interior.Color = vbRed
        End If
        Next cl
    End Sub
```

Example 3

'This VBA code will create a function to get the numeric part from a string

```
    Function GetNumeric(CellRef As String)
        Dim StringLength As Integer
        StringLength = Len(CellRef)
        For i = 1 To StringLength
        If IsNumeric(Mid(CellRef, i, 1)) Then Result = Result &
        Mid(CellRef, i, 1)
        Next i
        GetNumeric = Result
    End Function
```

Arrays

Exercise 1

```
    Sub BubbleSort(list())
    '   Sorts an array using bubble sort algorithm
        Dim First As Integer, Last As Long
        Dim i As Long, j As Long
        Dim Temp As Long
            First = LBound(list)
        Last = UBound(list)
        For i = First To Last - 1
            For j = i + 1 To Last
                If list(i) > list(j) Then
                    Temp = list(j)
                    list(j) = list(i)
                    list(i) = Temp
                End If
            Next j
        Next i
    End Sub
```

Worksheet and Workbook Methods

Exercise 1

'This code will unhide all sheets in the workbook

```
Sub UnhideAllWoksheets()
    Dim ws As Worksheet
    For Each ws In ActiveWorkbook.Worksheets
    ws.Visible = xlSheetVisible
    Next ws
End Sub
```

Exercise 2

'This macro will hide all the worksheet except the active sheet

```
Sub HideAllExceptActiveSheet()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
    If ws.Name <> ActiveSheet.Name Then ws.Visible =
    xlSheetHidden
    Next ws
End Sub
```

Exercise 3

'This code will sort the worksheets alphabetically

```
Sub SortSheetsTabName()
    Application.ScreenUpdating = False
    Dim ShCount As Integer, i As Integer, j As Integer
    ShCount = Sheets.Count
    For i = 1 To ShCount - 1
    For j = i + 1 To ShCount
    If Sheets(j).Name < Sheets(i).Name Then
    Sheets(j).Move before:=Sheets(i)
```

```
        End If
        Next j
        Next i
        Application.ScreenUpdating = True
    End Sub
```

Additional Programs

Sum of Two numbers

```
    Private Sub Calculate_Click()
         Dim a As Integer
         Dim b As Integer
         Dim c As Integer
         a = Val(`TextBox1.Text`)
         b = Val(`TextBox2.Text`)
         c = a + b
         MsgBox (c)
    End Sub
```

Concatenate Two Strings

```
    Sub ConcatenateStrings()
        Dim str1 As String, str2 As String
        str1 = "Captain"
        str2 = "America"
        'returns "Captain America":
        MsgBox str1 & str2
        MsgBox str1 & " " & str2
        'returns "Captain America":
        MsgBox str1 & " " & str2 & " in Australia"
        'returns "Captain America in Australia":
    End Sub
```

Prime Number

```
Function IsPrime(Num As Double) As Boolean
    Dim i As Double
    If Int(Num / 2) = (Num / 2) Then
        Exit Function
        Else
        For i = 3 To Sqr(Num) Step 2
            If Int(Num / i) = (Num / i) Then
                Exit Function
            End If
        Next i
    End If
    IsPrime = True
End Function
```

IIf function

```
' Test if a Supplied Integer is Positive or Negative.
Dim testVal As Integer
Dim sign1 As String
Dim sign2 As String
' First call to IIf function. The test value is negative:
testVal = -2
sign1 = IIf( testVal < 0, "negative", "positive" )
' sign1 is now equal to "negative".
' Second call to IIf function. The test value is positive:
testVal = 8
sign2 = IIf( testVal < 0, "negative", "positive" )
' sign2 is now equal to "positive".
```

Fibonacci Series

```
Private Sub Command1_Click()
    Dim x, g, n, i, sum As Integer
    n = Val(Text1.Text)
    x = 0
    y = 1
    Print x
    Print y
    For i = 3 To n
    sum = x + y
    Print sum
    x = y
    y = sum
    y = sum
    Next i
End Sub
```

For Each…Next Statement

```
Sub Unhide_First_Sheet_Exit_For()
'Unhides the first sheet that contain a specific phrase
'in the sheet name, then exits the loop.
Dim ws As Worksheet
  For Each ws In ActiveWorkbook.Worksheets
    'Find the sheet that starts with the word "Report"
     If Left(ws.Name, 6) = "Report" Then
       ws.Visible = xlSheetVisible
```

```
            'Exit the loop after the first sheet is found
             Exit For
        End If
      Next ws
End Sub
```

Conclusion

Thank you for purchasing the book. Most organizations have begun to use VBA to automate some of their processes in Excel. You can copy and paste information or create a pivot in Excel using VBA. If you want to improve some processes and gather more information about the different tools you can use in Excel, you have come to the right place. This is the third series of the book which provides more information on how you can use VBA to improve processes.

You must remember that experts also make many errors when they build applications and programs. Therefore, you should never beat yourself up if you make any mistakes. You should take some time to understand what the error is and identify a way to fix it. You have to always practice to improve your skills at programming in VBA.

I hope you gather all the information you are looking for. I believe you can automate some or all processes that you work on and impress your bosses and colleagues!

Thank you again for buying the book and if you’re finding the information valuable so far, please be sure to leave **5-star feedback on Amazon.**

Sources

ttps://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ah to-write-a-macro-in-excel%2F&psig=AOvVaw0XYs9S9qRneDdPgmz6Bmld&ust=153921http: us/dotnet/visual-basic/language-reference/statements/do-loop-statement8405542672

ttps://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/do-loop-statement

ttps://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ah to-write-a-macro-in-excel%2F&psig=AOvVaw0XYs9S9qRneDdPgmz6Bmld&ust=153921840

ttps://www.exceltip.com/vba/for-loops-with-7-examples.html

ttps://www.excel-easy.com/vba/examples/macro-recorder.html

ttps://docs.microsoft.com/en-us/office/vba/api/excel.application.automationsecurity

ttp://www.informit.com/articles/article.aspx?p=1610813&seqNum=4

ttps://chartio.com/resources/tutorials/how-to-get-values-from-another-sheet-in-excel-using-vba/

ttps://ccm.net/faq/53497-how-to-manipulate-data-in-excel-using-vba

ttp://www.databison.com/how-to-write-a-macro-in-excel/

ttp://www.la-solutions.co.uk/content/V8/MVBA/MVBA-Tips.htm

ttp://www.homeandlearn.org/the_excel_vba_editor.html

ttps://www.excelcampus.com/vba/vba-immediate-window-excel/

ttps://www.guru99.com/vba-arrays.html

ttps://www.excel-easy.com/vba/array.html ttps://excelmacromastery.com/excel-vba-array/

ttps://www.excel-easy.com/vba/if-then-statement.html

ttps://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/if-then-else-statement

ttps://bettersolutions.com/vba/arrays/sorting-bubble-sort.htm

ttps://trumpexcel.com/excel-macro-examples/#tab-con-20

Excel VBA

A Comprehensive, Step-By-Step Guide On Excel VBA Finance For Data Reporting And Business Analysis

Introduction

VBA is an amazing tool that many organizations have begun to use to perform some tasks in a few minutes or a few seconds. For instance, you can use VBA to analyze some information or extract some information from a given data set. You can also perform some special types of data analysis. When you write a macro in VBA, you can be certain that you have mastered your work.

If you want to learn how you can use VBA for finance and data analysis, you have come to the right place. Over the course of the book you will gather information on what VBA is, and what different parts of VBA you need to be aware of. You will learn more about the data types, the modules and other functions you can use to automate tasks.

Thank you for purchasing the book. I hope you gather all the information you are looking for.

If you enjoyed this title, please visit my author profile in Amazon and consider leaving a review.

Chapter One

Introduction to VBA

Visual Basic for Applications or VBA is a programming language, which is compatible with most Microsoft Office Products, including Excel. In other words, you can use VBA to develop programs in Excel. These programs will make Excel operate accurately and very fast.

What can you do with VBA?

Most people use Excel for a million different reasons. Here are a few examples:

- Forecasting and Budgeting
- Analyzing data
- Developing diagrams and charts using data
- Creating lists
- Creating forms and invoices

This list is endless, but I am sure you get the idea. In other words, you can use Excel to perform a variety of tasks, and I am sure you are reading this book because you have a set of expectations. If you want to automate the functions of Excel, you should use VBA.

For example, you may want to create a program that will help you import some data or numbers and then format that data to print a report. Once you develop the code, you can execute the macro using a command or a button. This will ensure that Excel performs the task in a few seconds or minutes.

Common Uses of VBA

You must understand why you want to use VBA. You must ensure that you can take some time out of your busy schedule to sit down and write a VBA code. You must understand the different tasks you can use VBA for. You cannot use VBA to perform your chores, but you can use it to make some tasks easier for you. This section covers some tasks that you can perform with VBA.

Automating Documents

Most people do not like to prepare documents, and if these documents contain the same information, they will not want to work on that document. You can use the Excel Ad-in called Mail Merge to automate letters, but this is not an option to use when you want to write individual letters or documents. In such situations, you can use a VBA code to create a form that will include the common information. You can include check boxes that VBA will use to write the document for you.

Word processing is not the only task you can automate using VBA. You can also automate the spreadsheet and there are numerous programs you can create for the same. For example, you can extract information or data from the Internet into a spreadsheet by clicking a button. Therefore, you can limit the time you spend on simply copying the data from the web and pasting it according to the required format in your Excel worksheet.

Customizing Application Interfaces

There are times when the features of an application will bug you, and you can turn off those features. But, that is not an option if you want to use that feature in your work. Instead of disabling that feature, you can use VBA to create a new feature that has all the functions that you need. For instance, instead of using conditional formatting every time you need to make changes in a worksheet, you can write a VBA code to do that for you.

It is easy to change the interface of an application, so it works better for you. You can customize toolbars or menu systems, and can also move some elements around in the interface to make it look presentable. Additionally, you can use multiple interfaces and use a VBA code to shift between those interfaces.

One of the most common applications of VBA is to perform a variety of calculations. You can create different equations and graphs using the data you obtain. There are times when you will need to make changes to the data so you can perform some calculations on it. If you find that an equation is complicated, you can use VBA to simplify the process. You can also use iterative functions to perform a calculation.

Sometimes the number that you create using a calculation does not mean much — it is just a number until someone makes a decision. Some decisions are easy to make yet repetitive. Smart applications save you more time for playing that game of Solitaire.

Adding new application features

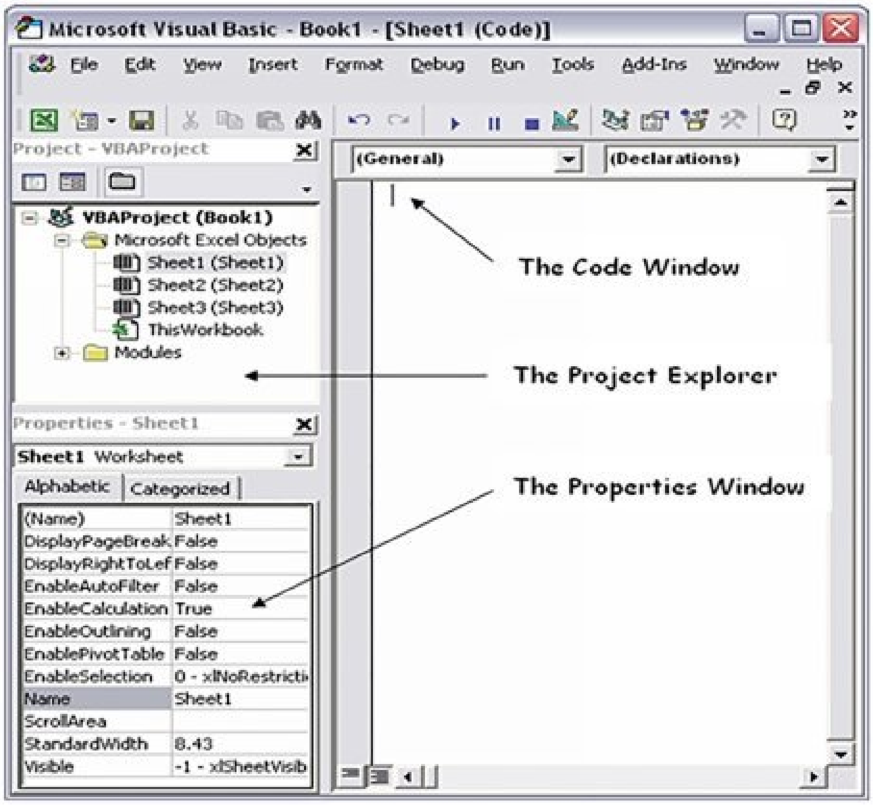
Most vendors or developers never use the applications they build. Therefore, they never update the code for their application. You can add new features to the application using VBA codes and work on developing an application. When you develop applications that complete some of your work in a few minutes, you will impress your boss and colleagues. This is an added advantage

Chapter Two

The IDE

VBA is a visual programming environment. That is, you see how your program will look before you run it. Its editor is very visual, using various windows to make your programming experience easy and manageable. You will notice slight differences in the appearance of the editor when you use it with Vista as compared to older versions of Windows. No matter which Office product and version of Windows you use, the editor has essentially the same appearance (and some small differences), the same menu items, and the same functionality.

An IDE is an editor, just like your word processor, spreadsheet, or database form. Just as application editors have special features that make them especially useful for working with data, an IDE is a programming editor with special features that make it useful for writing instructions that the application should follow. These instructions are procedural code — a set of steps. The figure below shows you the IDE Window in Excel:



(Name)	Sheet1
DisplayPageBreak	False
DisplayRightToLef	False
EnableAutoFilter	False
EnableCalculation	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestricti
Name	Sheet1
ScrollArea	
StandardWidth	8.43
Visible	-1 - xlSheetVisib

(https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjDrZTG0frdAhUIEnIKH

to-write-a-macro-in-excel%2F&psig=AOvVaw0XYs9S9qRneDdPgmz6Bmld&ust=1539218405542672)

The VBA IDE consists of a menu system, toolbars, a Project Explorer window, a Properties window, and a Code window. Below is a summary of what each Window contains.

Project Explorer

This window contains a list of the items in your project, which contains all the document elements in a single file. Your application exists within a file that appears in the Project Explorer window.

Properties

Whenever you select an object, the Properties window tells you about it. For example, this window tells you whether the object is blue or whether it has words on it.

Code

Eventually, you must write some code to make your application work. This window contains the special words that tell your application what to do. Think of it as a place to write a specialized to-do list.

Looking at the VBA Toolbox

You will not have to write code for every task in VBA. The IDE also supports forms, just like the forms that you use to perform other tasks. In this case, you decide what appears on the form and how the form acts when the user works with it. To make it easier to create forms, VBA provides the Toolbox, which contains controls used to create forms.

Each Toolbox button performs a unique task. For example, when you click one button, a text box may appear on the screen. But, if you click another button, a mathematical operation may take place.

Starting the Visual Basic Editor

How you start the Visual Basic Editor depends on the application that you are using. Newer versions of Office use a different approach than older versions.

Step 1: Go to Option “View” on the toolbar.

Step 2: In the drop-down list, select “Record Macro.”

Step 3: The interface will open, and you can begin typing the code for the worksheet you are in.

Using Project Explorer

Project Explorer appears in the Project Explorer window. You use it to interact with the objects that make up a project. A project is an individual file used to hold your program, or at least pieces of it. The project resides within the Office document that you are using, so when you open the Office document, you also open the project. See Chapter 3 for a description of how projects and programs interact. Project Explorer works much like how the left pane of Windows Explorer does.

The objects listed in Project Explorer depend on the kind of application that you are working with. For example, if you are working with Word, you see documents and document templates. Likewise, if you are working with Excel, you see worksheets and workbooks. However, no matter what kind of application you work with, the way that you use Project Explorer is the same.

A project can contain forms, modules, and class modules. Here is a description of these special objects:

- **Forms**: Contain user interface elements and help you interact with the user.
- **Modules**: Contain the nonvisual code for your application. For example, you can use a module to store a special calculation.
- **Class modules**: Contain new objects that you want to build. You can use a class module to create a new data type.

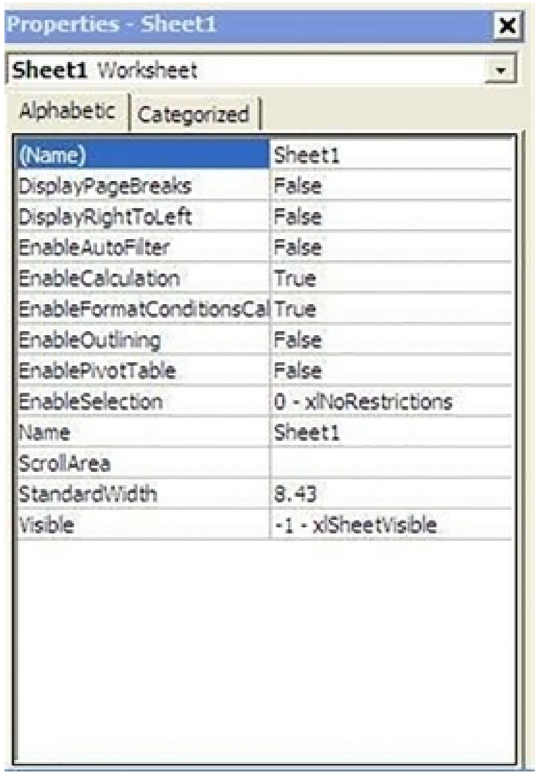
Working with special entries

Sometimes you see a special entry in Project Explorer. For example, when you work with a Word document, you might see a References folder, which contains any references that the Word document makes. Normally, it contains a list of templates that the document relies upon for formatting.

In many cases, you cannot modify the objects in the special folders. This is the case with the References folder used by Word document objects. The References folder is there for information only. To modify the referenced template, you need to find its object in Project Explorer. We will not discuss these concepts in the book since you do not work with these often.

Using the Properties window

Most of the objects that you click in the VBA IDE have properties that describe the object in some way. The earlier “Property values are up” section of this chapter tells about properties if you have not worked with them before. The following sections provide details about the Properties window.



Properties - Sheet1

Sheet1 Worksheet

(Name)	Sheet1
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsCal	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	Sheet1
ScrollArea	
StandardWidth	8.43
Visible	-1 - xlSheetVisible

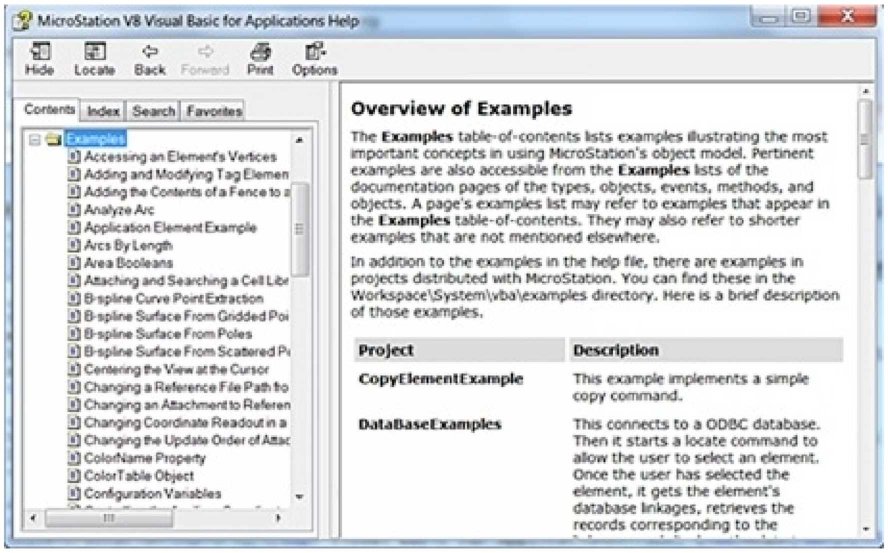
(http://www.affordsol.be/vba-prog-1-3-editor-properties.htm)

Understanding property types

A property needs to describe the object. When you look at an object, you naturally assume something about the information provided by a use red, yellow, or green. Likewise, VBA object properties have specific types. One of the most common property types is text. The Caption property of a form is text. The text appears at the top of the form when the user opens it. Another common property type is a logic, or Boolean, value.

Getting help with properties

Do not expect to memorize every property for every object that VBA applications can create. Not even the gurus can do that. To determine what a property will do for your application, just highlight the property and press F1, and, in most cases, VBA displays a Help window like the image below.


MicroStation V8 Visual Basic for Applications Help
Hide
Locate
Back
Forward
Print
Options
Contents
Index
Search
Favorites
Examples
Accessing an Element's Vertices
Adding and Modifying Tag Elemen
Adding the Contents of a Fence to a
Analyze Arc
Application Element Example
Arcs By Length
Area Booleans
Attaching and Searching a Cell Libr
B-spline Curve Point Extraction
B-spline Surface From Gridded Poi
B-spline Surface From Poles
B-spline Surface From Scattered P
Centering the View at the Cursor
Changing a Reference File Path fro
Changing an Attachment to Referen
Changing Coordinate Readout in a
Changing the Update Order of Attac
ColorName Property
ColorTable Object
Configuration Variables

Overview of Examples

The **Examples** table-of-contents lists examples illustrating the most important concepts in using MicroStation's object model. Pertinent examples are also accessible from the **Examples** lists of the documentation pages of the types, objects, events, methods, and objects. A page's examples list may refer to examples that appear in the **Examples** table-of-contents. They may also refer to shorter examples that are not mentioned elsewhere.

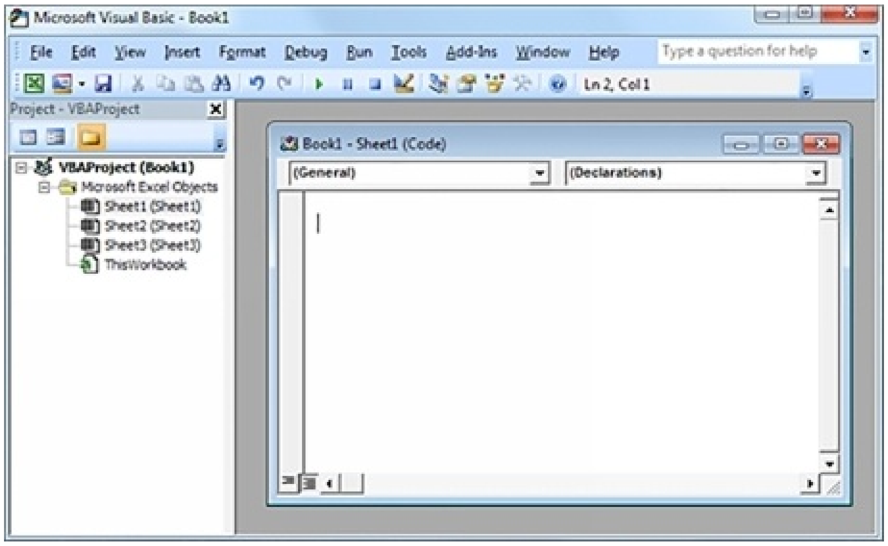
In addition to the examples in the help file, there are examples in projects distributed with MicroStation. You can find these in the Workspace\System\vba\examples directory. Here is a brief description of those examples.

Project	Description
CopyElementExample	This example implements a simple copy command.
DataBaseExamples	This connects to a ODBC database. Then it starts a locate command to allow the user to select an element. Once the user has selected the element, it gets the element's database linkages, retrieves the records corresponding to the

(http://www.la-solutions.co.uk/content/V8/MVBA/MVBA-Tips.htm)

Using the Code Window

The Code window is where you write your application code. It works like any other editor that you have used, except that you type according to the syntax.



(http://www.homeandlearn.org/the_excel_vba_editor.html)

Notice that the Project Explorer window and the Properties window are gone — you can display them again by using the View -> Project Explorer and View ->Properties Window commands.

Opening an existing Code window

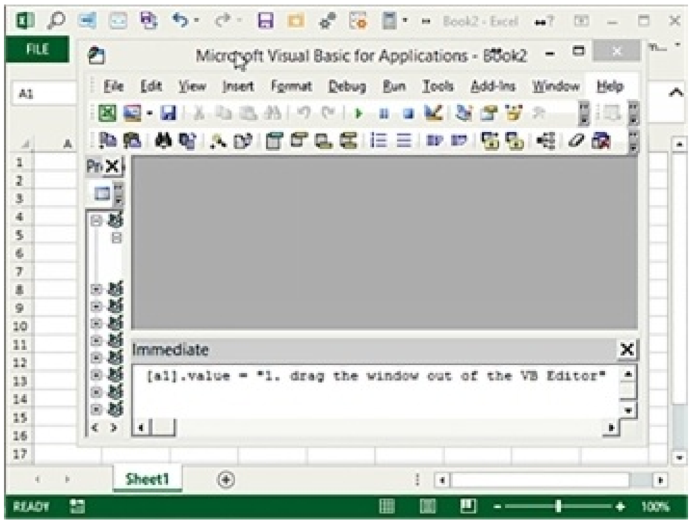
Sometimes you will not be able to complete an application and need to work on it later. To open an existing Code window, find the module that you want to open in Project Explorer. Double-click the module entry, and the IDE displays the code within it with your code loaded. The Code window also appears when you perform different tasks.

Creating a new Code window

When you start a new module within an existing document or template, open a new Code window by using either the Insert -> Module or Insert -> Class Module command. After you save this module or class module, it appears in Project Explorer with the other modules in your project.

It is easier to execute one line of code at a time to understand where you may have made an error. You can do this by using the Immediate Window. This window normally appears at the bottom of the IDE, and it will not contain any information until you type something in it.

Most developers spend their days using the Immediate window to check their applications for errors. You can use the Immediate window to ask VBA about the value of a variable, for example. To try this feature, type String1 = “Hello World” in the Immediate window and then press Enter. Now type ‘? String1’ and then press Enter. You asked VBA to create a variable named String1 and assign it a value of Hello World. You can use the ‘?’ operator to check the value assigned to the variable String1.


Book2 - Excel
FILE
Microsoft Visual Basic for Applications - Book2
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
A1
Immediate
Sheet1
READY
100%

```
[a1].value = "1. drag the window out of the VB Editor"
```

(https://www.excelcampus.com/vba/vba-immediate-window-excel/)

Chapter Three

VBA, A Primer

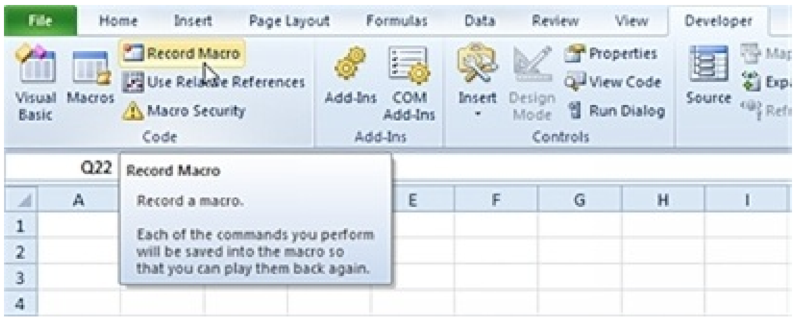
Macro Recorder

As mentioned in the previous book, the macro recorder is an important and useful tool in Excel. This tool will record every action that you perform in Excel. You only need to record a task once using the macro recorder, and you can execute that same task a million times by clicking a button. If you do not know how to program a specific task in Excel, you can use the Macro Recorder to help you understand what you need to do. You can then open the Visual Basic Editor once you have recorded the task to see how you can program it.

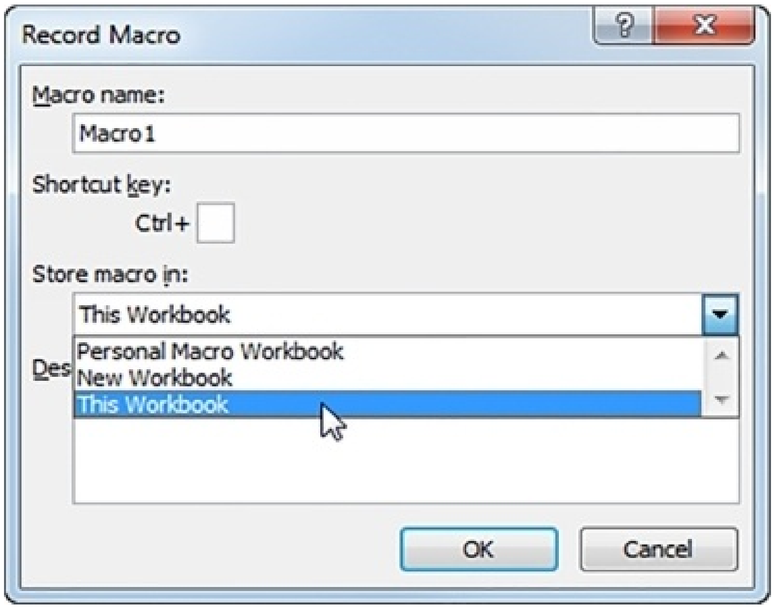
You cannot perform many tasks when you use the Macro Recorder. For instance, you cannot use the macro recorder to loop through data. The macro recorder also uses more code than you need, which will slow the process down.

Record a Macro

- Go to the Menu Bar and move the Developer Tab, and click the button to Record the Macro.



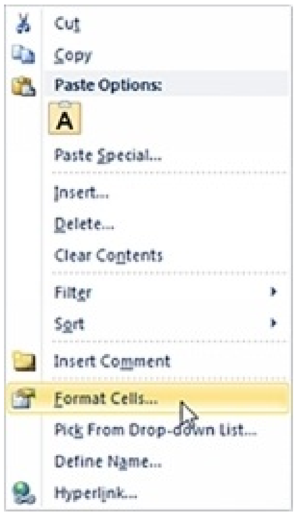
- Enter the name of the macro.
- Choose the workbook where you want to use the macro. This means that the macro can only be used in the current workbook.



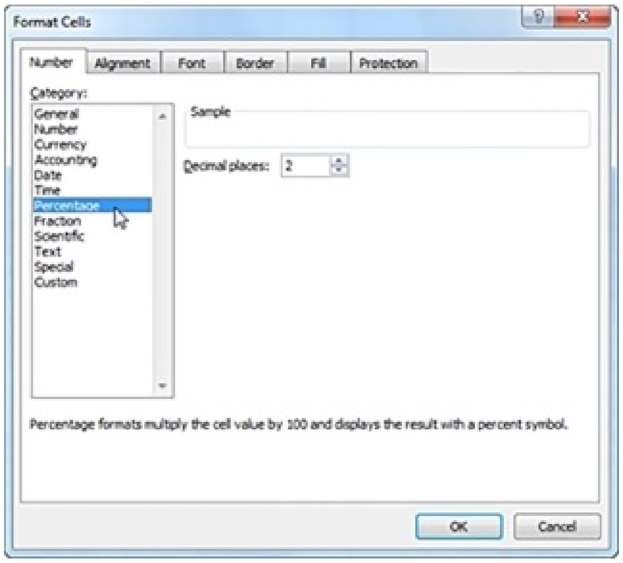
4. If you store the macro in a personal macro workbook, you can access the macro in all your workbooks. This is only because Excel stores the macro in a hidden workbook, which will open automatically when it starts. If you store the macro in a new workbook, you can use the macro only in the opened workbook.

5. Click OK.

6. Now, right click on the active cell in the worksheet. Ensure that you do not select any other cell. Click format cells.

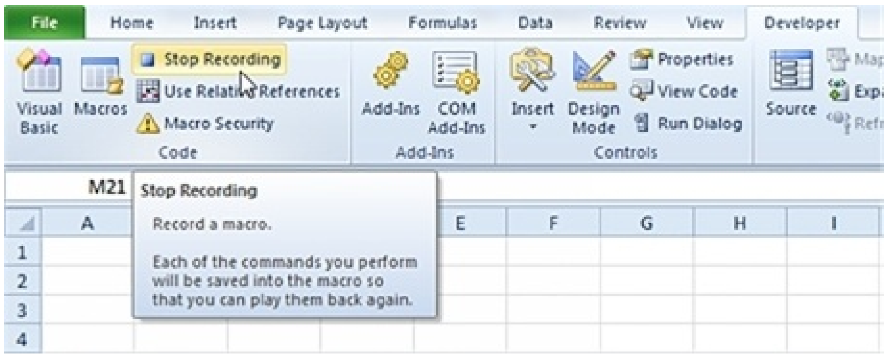


7. Select the percentage.



Percentage formats multiply the cell value by 100 and displays the result with a percent symbol.

8. Click OK.
9. Now, select the stop recording.



Stop Recording

Record a macro.

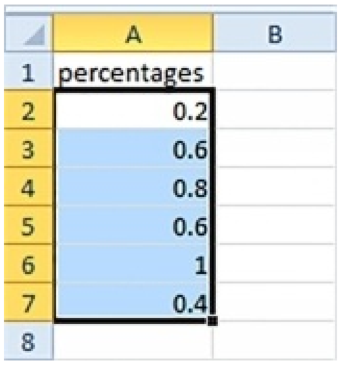
Each of the commands you perform will be saved into the macro so that you can play them back again.

You have successfully recorded your macro using the macro recorder.

Run the Recorded Macro

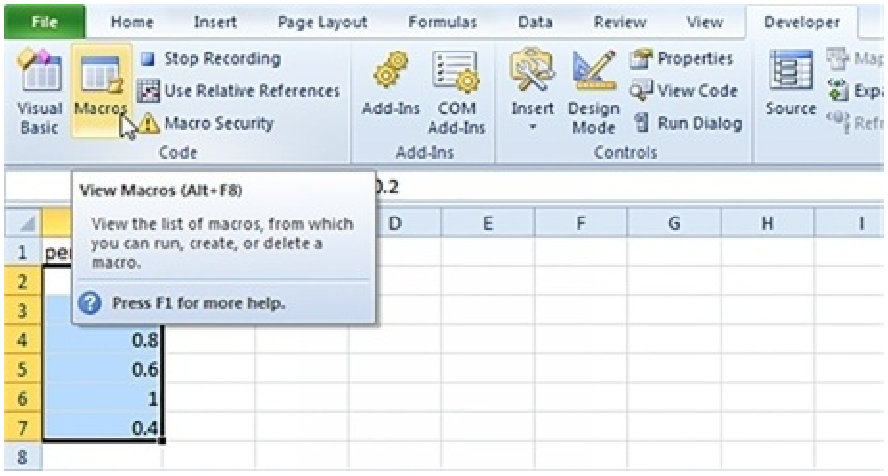
You will now need to test the macro and see if you can change the format of the numbers to percentage.

10. Enter any numbers between 0 and 1 in the spreadsheet.
11. Select the numbers.



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

12. Move to the Developer tab, and click macros.

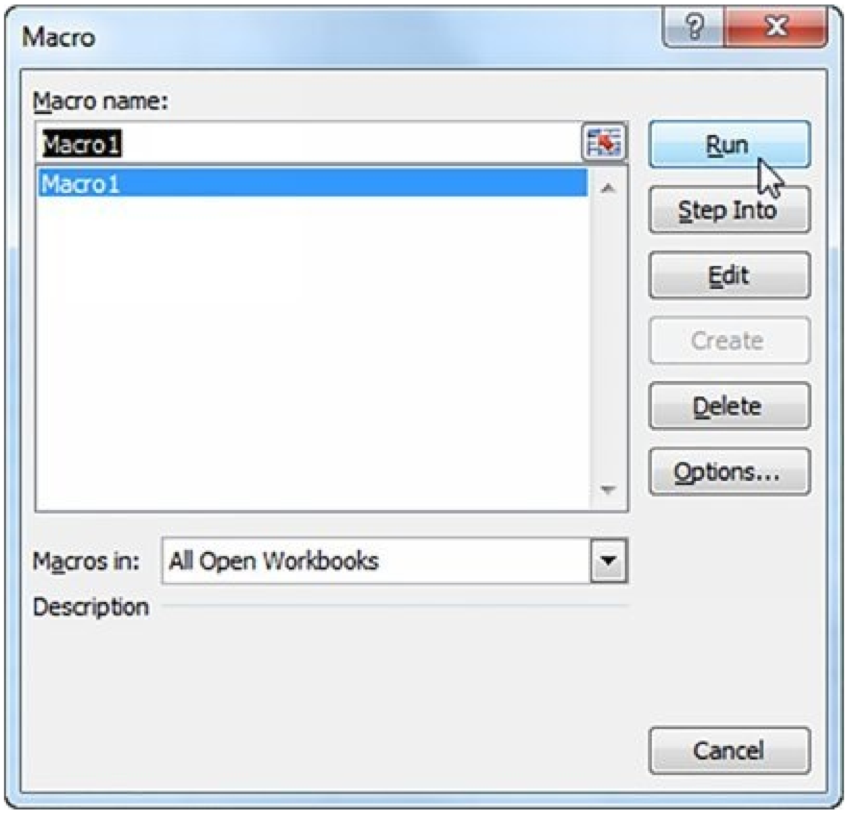


View Macros (Alt+F8)

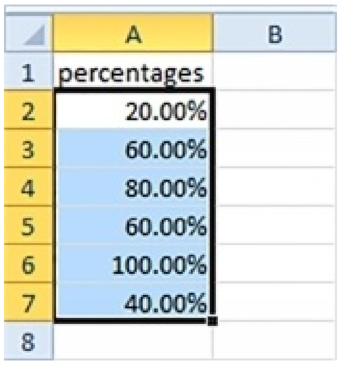
View the list of macros, from which you can run, create, or delete a macro.

Press F1 for more help.

13. Now click run.



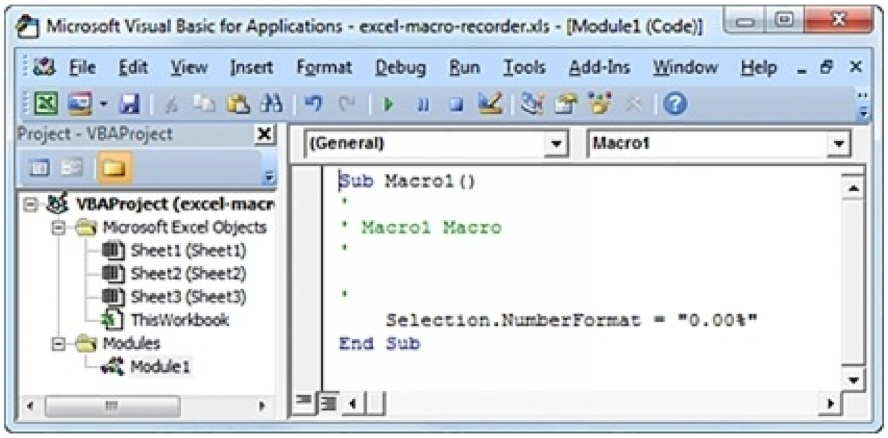
You will see the following result.



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

See the Macro

If you want to look at the macro, you should open the Visual Basic Editor.



```
Sub Macro1()
'
' Macro1 Macro
'

'
    Selection.NumberFormat = "0.00%"
End Sub
```

The macro, called Module 1, is placed in a module. The code that is placed in the module is always available to the full workbook. This means you can change the format for the numbers in all the sheets in the workbook. If you assign a macro to the command button, you should remember that the macro would only be available for that specific sheet.

Macro Storage and Security

The developers at Microsoft changed the security settings that help to prevent macros from running. This development was only made when macros were used to deliver some high-profile viruses. We covered some information about the security of macros in the first part of the book. Let us now look at how you can adjust the default settings in a macro.

You can either control the settings for a macro in some workbooks by saving the workbook in trusted locations or by adjusting the security settings globally. If you store a workbook with a macro in a folder that you label a trusted location, the macros will automatically be enabled when you open the workbook.

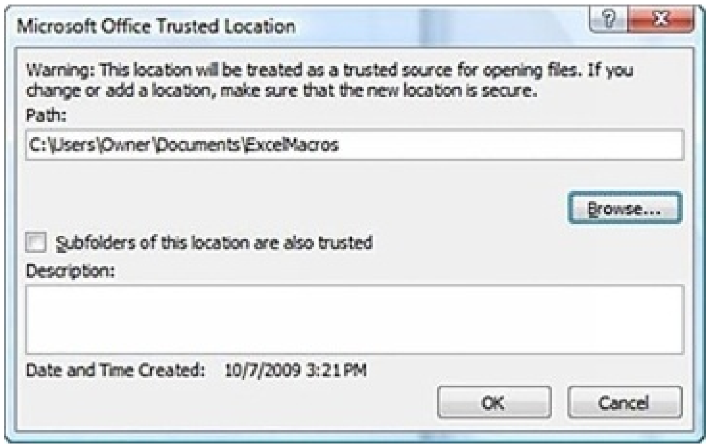
You can check the security of your macro in the Developer tab under the Macro Security icon. If you click this icon, a dialog box with the settings category will be displayed. You can access the folders that you trust by scrolling down the left navigation bar.

How to Add a Trusted Location

As mentioned earlier, you can save the workbooks with macros in a folder that you mark as a trusted location. If you save a workbook in that folder, the macros will always be enabled. The developers suggest that you should always have a trusted location in your hard drive. Remember that you can never trust the location on a network drive.

If you want to specify a trusted location, you should follow the steps given below:

1. Go to the Developer Tab and click on Macro Security.
2. Move to the left navigation pane in the Trust Center and choose the Trusted Location.
3. If you want to save the file on a network drive, you should add that location into the trusted locations.
4. Go to ‘My Networks’ in the Trusted Location dialog box and click the ‘Add New Location’ button.
5. You will see the list of Trusted Locations in a dialog box.



Warning: This location will be treated as a trusted source for opening files. If you change or add a location, make sure that the new location is secure.

C:\Users\Owner\Documents\ExcelMacros

Subfolders of this location are also trusted

Date and Time Created: 10/7/2009 3:21 PM

6. Now click the Browse button and go to the parent folder of the folder that you want to make a trusted location. Now click on the Trusted Folder. You will not find the name of the Folder in the text box, but click OK. The correct name will come in the Browse dialog box.
7. If you want to include the subfolders in the selected folder, you should select the radio button against the ‘Subfolders of this location are also trusted’ option.
8. Now, click OK to add the folder to the list.

How to Enable Macros Outside a Trusted Location

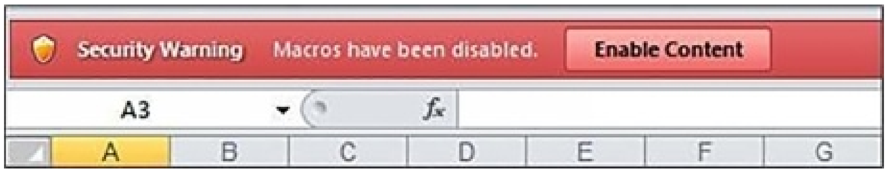
When you do not save an excel workbook in a trusted location, excel will always rely on the macro settings. In Excel 2003, a macro could have a low, medium, high or very high security. The developers of Microsoft later renamed these settings. If you want to access the macro settings, you should go to the Developers Tab and choose Macro Security. Excel will then display the Macro Settings dialog box. You should select the ‘Disable All Macros with Notification’ option. Let us look at the description of the options in the dialog box.

Disable All Macros Without Notification

This setting will not allow any macro to run. If you do not always want to run the macro when you open the workbook, you should choose this setting. Since you are still learning how to use macros and work with them, you should not use this setting. This setting is equivalent to the Very High Security that is found in Excel 2003. If you choose this setting, you can only run macros if they are saved in a Trusted Location.

Disable All Macros With Notification

This setting is like the Medium security setting in Excel 2003. This is the recommended setting that you should use. If you use this setting, Excel will ask you if you want to Enable to Disable a macro when you open a workbook. You may often choose this option if you are a beginner. In Excel 2010, you will see a message in the message area, which states that the macros have been disabled. You can either choose to enable or disable the content in the workbook by choosing that option.


Security Warning Macros have been disabled.

Disable All Macros Except Digitally Signed Macros

If you wish to use this setting, you should always use a digital signing tool like VeriSign or any other provider to sign your macro. If you are going to sell your macros to other parties, you should use this security option. This is a hassle if you want to write macros only for your use.

Enable All Macros

Experts suggest that you do not use this option since dangerous codes can also run on your system. This setting is equivalent to the Low security option in

Excel 2003, and is the easiest option to use. This option will open your system up to attacks from malicious viruses.

Disabling All Macros With Notification

Experts suggest that you set your macro to disable all content after it gives you a notification. If you save a workbook with a macro using this setting, you will see a security warning right above the formula bar when you open the workbook. If you know that there are macros in the workbook, all you need to do is click 'Enable Content.' You can click on the X on the far right of the bar if you do not want to enable any of the macros in the workbook.

If you do forget to enable the macro and then attempt to run that macro, Excel will indicate that the macro will not run since you have disabled all macros in the workbook. If this happens, you should reopen the workbook to enable the macros again.

Hi there! If you found the topic or information useful, it would be a great help if you can leave a quick review on Amazon. Thanks a lot!

Chapter Four

Data Types

Data types are a way to define the data to make it easier for the user to assign values to a variable within the program. The computer will still compute the data in the form of bits, and VBA uses different data types in different ways. The computer will see that binary value 1000001b, but it will not do anything with that value. The compiler in VBA will view this specific value as a letter or number depending on the data type that you assign to that value. It is important to understand what a data type is, and how you should use data types. You should adhere to some rules when you use data types. The data can be corrupted by another program or procedure.

Although a variable, in general, is simply a box for storing data, you can think of these data types as special boxes for storing specific kinds of data. Just as you would use a hatbox to store a hat and not a car engine, you use these special box types to store kinds of data. For example, you use a string to hold text, not logical (true/false) values. VBA supports several standard data types, including Boolean, Byte, Long, Currency, Integer, Single, Decimal, Double, Date, Object, String and Variant. Apart from these defined data types, you can also create some user-defined data types so you can mark the necessary information for your program. A user-defined data type gives you the power to extend the VBA interpretation of data. (The computer still looks at the data as binary information.) Each of the data type descriptions that follow has a different purpose, and you can work with the data type in a variety of ways.

Using strings for text

The first data type that I discuss in this chapter is one that you have already seen in the message box examples: the string. When you create a message box, you use a string as input. The string is the most useful data type in VBA. This chapter only introduces strings. The next chapter provides information on how you can manipulate strings in VBA.

Understanding strings

Programmers often use fancy terms for things that the average person easily recognizes. A string is a sequence of characters. The characters are not always printable but can include control characters that determine how the text appears on-screen. A string can also include special characters, such as punctuation, or even special features, such as a circumflex or an umlaut. Although a string can contain all these elements, the main content of a string is always text.

Adding strings together with + or &

Sometimes you will want to concatenate two or more strings to make a longer string. Concatenation is the process of adding strings together. For example, you might want to add a person's first name to their last name to create their full name. Often, you need to take information from more than one place and join it together to create a new kind of information.

Using character codes

Strings can contain several elements. In previous examples, I show you strings that contain control character constants such as vbCrLf. This constant contains two control characters: a carriage return and a line feed. The carriage return sends the cursor back to the beginning of the line; the line feed places the cursor on the next line. The result of using both control characters together is the same as pressing Enter on the keyboard. Strings can also use a special function, Chr, to create special characters. You can combine this function with the Character Map utility to produce any character that you need for your program

Using numbers for calculations

Most computers store information in the form of numbers. You can use a number to express some values in a database, perform a few tasks in spreadsheets or show the reader or user the current page in a document or webpage. In procedures, numbers are used to count iterations in loops, determine the position of values in arrays or check a specific condition. You can use numbers in VBA in different ways, like interacting with your code or determining which character you want to view on screen.

Understanding the numeric types

Numbers are always looked at as a single entity, and a number is simply just that – a number. A computer can view the number in different ways since the processor works differently with numbers. For example, you have a data type for an integer value, which is a value that does not have a decimal point and a data type for a decimal point. The four types of numeric data types are:

1. **Integer**: As mentioned earlier, these numbers do not have a decimal point. Integers can only hold whole numbers like 5, but cannot hold

numbers like 5.6.

2. **Real**: A real number can contain a decimal point, and the number after the decimal point does not necessarily have to contain a value. This means the number 5.0 is accepted as a real number. The compiler will store a real number in a different format when compared to an integer.
3. **Currency**: A financial calculation will require some accuracy since you will be using small currency values. Even the smallest issue with the program can cause an error. The currency numeric allows you to store numbers with extreme precision, but the compiler takes a lot of time to process or store this data type.
4. **Decimal**: A computer stores information in the binary or base 2 format. When you use a base 10 or decimal value when you work with numbers, the compiler will need to convert the number into binary before it can use it.

Using Boolean values to make decisions

The Boolean type is the easiest to use and understand. This type is used to indicate yes or no, true or false and on or off. You can use this type to work with any two-state information. It is commonly used to represent data values that are diametrically opposed.

Working with Operators

Operators determine how VBA works with two variables and what result it produces. The examples in this chapter use operators to add numbers and concatenate (add) strings. In both cases, your code uses the + operator to perform the task. However, the result differs. When you are using numbers, the result is a summation, such as 1 + 2 = 3. When you are using strings, the result is a concatenation, such as Hello + World = Hello World.

VBA groups operators into four areas:

1. **Arithmetic**: Operators that perform math operations, such as addition (+), subtraction (-), division (/), and multiplication (*)
2. **Comparison**: Operators such as less than (<), greater than (>), and equal (=) that compare two values and produce a Boolean result
3. **Concatenation**: Operators such as & and + that are used to add two

strings together

4. **Logical**: Operators such as Not, And, Or, and Xor that are used to perform Boolean operations on two variables

Chapter Five

Decision Making Statements

Few programs use all the statements in the program file all the time. You might want the program to perform one task when you click Yes and another task when you click No. The statements for both tasks appear in the code, but the program executes only one set of statements. To control program execution, the developer adds special statements — such as the If...Then statement — that shows the beginning and end of each task and decide which task to execute. You might think that letting the computer decide which task to execute would cause the developer to lose control of the program. However, the developer has not lost control of the program, because the decision-making process is predefined as part of the program design.

If...Then Statement

Most programs require decision-making code. When you need to make the same decision every time that you perform a task and the outcome of the decision is always the same, then making the decision is something that you can tell VBA to do for you by using the If...Then statement. Decision-making code has several benefits:

1. **Consistency**: The decision is made by using the same criteria and in the same manner every time.
2. **Speed**: A computer can make static decisions faster than humans can. However, the decision must be the same every time, and the decision must have the same answer set every time.
3. **Complexity**: Requesting that the computer make static decisions can reduce program complexity. Fewer decisions translate into ease of use for most people.

Example:

```
Public Sub IfThenTest()

        ‘ Create a variable for the selected text.

        Dim TestText As String

        ‘ Get the current selection.
        TestText = ActiveWindow.Selection.Text
        ‘ Test the selection for “Hello.”
        If TestText = “Hello” Then
        ‘ Modify the selected text to show it’s correct.
        TestText = “Correct!” + vbCrLf + “Hello”
    End If
```

If...Then...Else statement

The If...Then...Else statement makes one of two choices. If the expression controlling the statement is true, VBA executes the first set of statements. On the other hand, if the expression is false, VBA executes the second set of statements.

If...Then...ElseIf statement

When making multiple comparisons, you can use the If...Then...ElseIf statement to make the code easier to read. Using this format can also reduce the number of decisions that VBA must make, which ensures that your code runs as quickly as possible.

Using the IIf function

You might need to decide in a single line of code instead of the three lines (minimum) that other decision-making techniques require. The IIf function is a good choice when you need to make simple and concise decisions in your program. It has the advantage of providing decision-making capability in a single line of code.

Chapter Six

Loops

Many tasks that you perform require more than one check, change, or data manipulation. You do not change just one entry in a worksheet; you change all the affected entries. Likewise, you do not change just one word in a document; you might change all occurrences based on certain criteria. Databases require multiple changes for almost any task.

Loops provide a method for performing tasks for more than one time. You can use loops to save code-writing time. Simply write the code to perform the repetitive task once and then tell VBA to perform the task multiple times. When using loops, you decide how the code determines when to stop. You can tell the loop to execute a specific number of times or to continue executing until the program meets a certain condition.

Do While...Loop statement

A Do While...Loop statement keeps performing a task until a certain condition is true. The loop checks the expression first and then executes the code within the structure if the expression is true. You use this loop to perform processing zero or more times. A Do While...Loop works especially well if you cannot determine the number of times that the loop should execute when you design your program.

Do...Loop While statement

The Do...Loop While statement works the same as the Do While...Loop statement. The difference is that this statement always executes once because the expression used to verify a need to loop appears at the end of the structure. Even if the expression is false, this statement still executes at least one time. You can use this statement when you want to ensure that a task is always completed at least one time.

Do Until...Loop statement

The Do Until...Loop statement continues processing information until the expression is false. You can view the Do While...Loop statement as a loop that continues while a task is incomplete. The Do Until...Loop statement continues until the task is finished. The subtle difference between the two statements points out something interesting: They rely on your perspective of the task to complete. These two statement types are completely interchangeable. The big difference is how you define the expression used to signal the end of the looping sequence.

Do...Loop Until statement

The Do...Loop Until statement is the counterpart of the Do Until...Loop statement. This statement examines the expression at the end of the loop, so it always executes at least once even if the expression is false.

For...Next statement

The For...Next statement is very handy for performing a task a specific number of times. If you can determine how many times to do something in advance, this is the best looping option to use because there is less chance of creating an infinite loop. You can create absurdly large loops, but they eventually end.

For Each...Next statement

The For Each...Next statement is like the For...Next statement in operation. However, this statement does not rely on an external counter. The statement uses an object index as a counter. The advantage of using this statement is that you do not have to figure out how many times to perform the loop — the object provides this information. The disadvantage of using this statement is that you lose a little control over how the loop executes because the counter is no longer under your control.

Found this title interesting or useful? Then a review on Amazon will be highly appreciated!

Chapter Seven

Arrays

Arrays provide a way for your programs to store more than one item in a single container. Think of the array as a large box with a bunch of small boxes inside. Each small box can store a single value. You decide how many small boxes the array can hold when you create the array. Use arrays when you need to store several related items of the same data type.

Structured Storage

An array is a list of items. When you write a list of tasks to perform for the day, you create an array. The piece of paper is a single container that holds several strings, each of which is a task that you must perform. Likewise, you can create a single piece of paper in your VBA program — an array — and use that array to hold multiple items. You can define arrays by using several techniques. However, all these techniques use the same basic approach.

Example:

```
‘ Tell VBA to start all arrays at 0.
Option Base 0
Public Sub SingleDimension()
‘ Define an output string.
Dim Output As String
‘ Define a variant to hold individual strings.
Dim IndividualString As Variant
‘ Define the array of strings.
Dim StringArray(5) As String
‘ Fill each array element with information.
StringArray(0) = “This”
StringArray(1) = “Is”
StringArray(2) = “An”
StringArray(3) = “Array”
StringArray(4) = “Of”
StringArray(5) = “Strings”
‘ Use the For Each...Next statement to get each array
‘ element and place it in a string.
For Each IndividualString In StringArray
‘ Create a single output string with the array
‘ array elements.
Output = Output + IndividualString + “ “
Next
‘ Display the result.
MsgBox Trim(Output), _
vbInformation Or vbOKOnly, _
“Array Content”
End Sub
```

Notice that the code begins with an Option Base 0 statement. This statement tells VBA whether you want to start counting array elements at 0 or 1. The default setting is 0. Most programming languages use 0 as the starting point, which is why Microsoft made 0 the default for VBA. However, older versions of Visual Basic (including VBA) use 1 as the starting point. When you want to ensure that your program works in every environment, include the Option Base statement.

Because the array begins at 0 and not at 1, you can store six items in an array that is defined as having five elements. The number that you include in the declaration is always the top element number of the array and not the actual number of elements.

Array Types

One can classify an array in several ways, and the first method is the type of data that the array will hold. A string array is very different from an integer array, but the array will always ensure that the elements are unique. If you use a Variant data type, you can mix the data types in an array. You should always be careful when you use this technique since it can lead to errors which are difficult to debug.

Another method is to define the dimensions in an array. The dimension will define the number of directions in which the array will hold the information. A simple list, like the one in the earlier "Understanding array usage" section gives an example of a single-dimensional array. A table which consists of columns and rows is a two-dimensional array. One can create an array using any number of dimensions.

Example: Adding an Element to an Array

```
Dim a As Range

Dim arr As Variant  'Just a Variant variable (i.e. don't pre-define it as an array)

For Each a In Range.Cells

        If IsEmpty(arr) Then

            arr = Array(a.value) 'Make the Variant an array with a single
    element

        Else

            ReDim Preserve arr(UBound(arr) + 1) 'Add next array element

            arr(UBound(arr)) = a.value          'Assign the array element

        End If

    Next
```

VBA Array

In this section, we will look at the steps you need to follow to create an array.

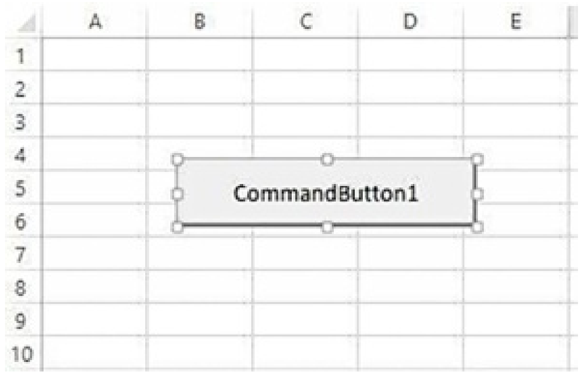
Step 1 – Create A New Workbook

1. Open Microsoft Excel.
2. Save the excel workbook with the extension .xlsm

Step 2 – Add A Command Button

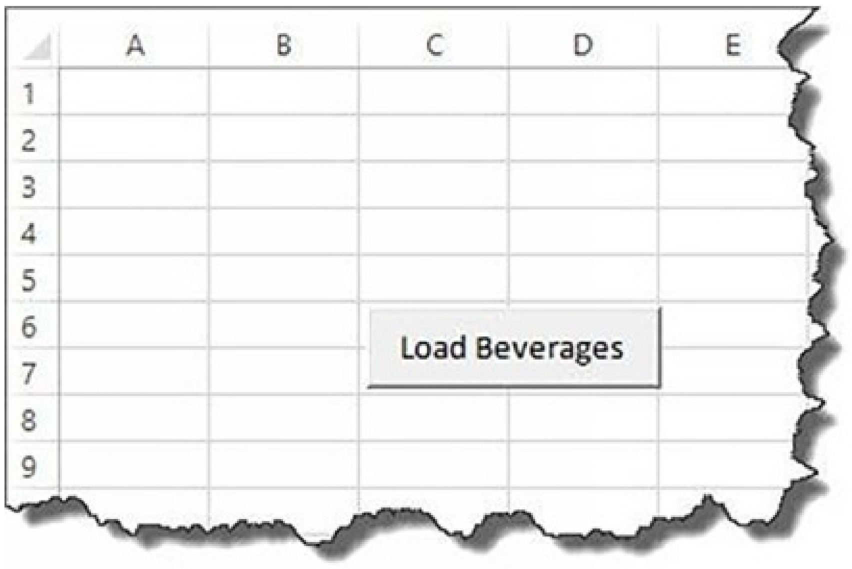
Now that you are familiar with creating an interface in a workbook. The previous chapters in the book will help you gather more information about the subroutines or subs and functions in VBA.

1. Add a command button to the active worksheet.



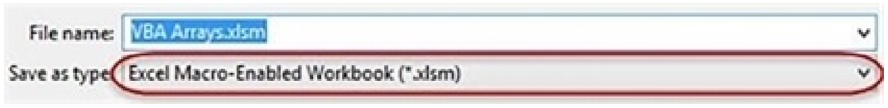
2. Set the property name to cmdLoadBeverages.
3. Now, set the Caption Property as Load Beverages.

The interface should now display the following:



Step 3 – Save the File

1. You should now click the save as button in the macro-enabled form.



File name: VBA Arrays.xlsm
Save as type: Excel Macro-Enabled Workbook (*.xlsm)

Step 4 – Write the Code

Let us now write the code for the application.

1. Right click on the button and choose to view the code.
2. Add the code in the code window.

```
Private Sub cmdLoadBeverages_Click()
    Dim Drinks(1 To 4) As String
    Drinks(1) = "Pepsi"
    Drinks(2) = "Coke"
    Drinks(3) = "Fanta"
    Drinks(4) = "Juice"
    Sheet1.Cells(1, 1).Value = "My Favorite Beverages"
    Sheet1.Cells(2, 1).Value = Drinks(1)
    Sheet1.Cells(3, 1).Value = Drinks(2)
    Sheet1.Cells(4, 1).Value = Drinks(3)
    Sheet1.Cells(5, 1).Value = Drinks(4)
End Sub
```

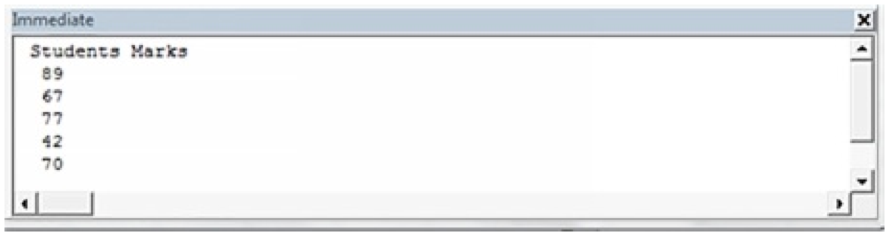
Example to Enter Student's Marks

Without An Array

In the example below, we will look at how you can enter the marks for every student without using an array.

```
Public Sub StudentMarks()
    With ThisWorkbook.Worksheets("Sheet1")
        ' Declare variable for each student
        Dim Student1 As Integer
        Dim Student2 As Integer
        Dim Student3 As Integer
        Dim Student4 As Integer
        Dim Student5 As Integer
        ' Read student marks from cell
        Student1 = .Range("C2").Offset(1)
        Student2 = .Range("C2").Offset(2)
        Student3 = .Range("C2").Offset(3)
        Student4 = .Range("C2").Offset(4)
        Student5 = .Range("C2").Offset(5)
        ' Print student marks
        Debug.Print "Students Marks"
        Debug.Print Student1
        Debug.Print Student2
        Debug.Print Student3
        Debug.Print Student4
        Debug.Print Student5
    End With
End Sub
```

The output will be the following,



```
Students Marks
 89
 67
 77
 42
 70
```

Using an Array

```
Public Sub StudentMarksArr()
    With ThisWorkbook.Worksheets("Sheet1")
        ' Declare an array to hold marks for 5 students
        Dim Students(1 To 5) As Integer
        ' Read student marks from cells C3:C7 into array
        Dim i As Integer
        For i = 1 To 5
            Students(i) = .Range("C2").Offset(i)
        Next i
        ' Print student marks from the array
        Debug.Print "Students Marks"
        For i = LBound(Students) To UBound(Students)
            Debug.Print Students(i)
        Next i
    End With
End Sub
```

Notice the difference in the variables used in the two programs, and also notice the length of the program.

Example with Loops

```
Public Sub ArrayLoops()
    ' Declare  array
    Dim arrMarks(0 To 5) As Long
    ' Fill the array with random numbers
    Dim i As Long
    For i = LBound(arrMarks) To UBound(arrMarks)
        arrMarks(i) = 5 * Rnd
    Next i
    ' Print out the values in the array
    Debug.Print "Location", "Value"
    For i = LBound(arrMarks) To UBound(arrMarks)
        Debug.Print i, arrMarks(i)
    Next i
End Sub
```

Sorting an Array

```
Sub QuickSort(arr As Variant, first As Long, last As Long)
```

```
Dim vCentreVal As Variant, vTemp As Variant
Dim lTempLow As Long
Dim lTempHi As Long
lTempLow = first
lTempHi = last
vCentreVal = arr((first + last) \ 2)
Do While lTempLow <= lTempHi
  Do While arr(lTempLow) < vCentreVal And lTempLow < last
    lTempLow = lTempLow + 1
  Loop
  Do While vCentreVal < arr(lTempHi) And lTempHi > first
    lTempHi = lTempHi - 1
  Loop
  If lTempLow <= lTempHi Then
      ' Swap values
      vTemp = arr(lTempLow)
      arr(lTempLow) = arr(lTempHi)
      arr(lTempHi) = vTemp
       ' Move to next positions
      lTempLow = lTempLow + 1
      lTempHi = lTempHi - 1
    End If
  Loop
  If first < lTempHi Then QuickSort arr, first, lTempHi
If lTempLow < last Then QuickSort arr, lTempLow, last
```

```
  End Sub
```

Example for Creating a Two-Dimensional Array

```
Public Sub TwoDimArray()
    ' Declare a two dimensional array
    Dim arrMarks(0 To 3, 0 To 2) As String
    ' Fill the array with text made up of i and j values
    Dim i As Long, j As Long
    For i = LBound(arrMarks) To UBound(arrMarks)
        For j = LBound(arrMarks, 2) To UBound(arrMarks, 2)
            arrMarks(i, j) = CStr(i) & ":" & CStr(j)
        Next j
    Next i
    ' Print the values in the array to the Immediate Window
    Debug.Print "i", "j", "Value"
    For i = LBound(arrMarks) To UBound(arrMarks)
        For j = LBound(arrMarks, 2) To UBound(arrMarks, 2)
           Debug.Print i, j, arrMarks(i, j)
        Next j
    Next i
End Sub
```

Chapter Eight

How to Manipulate Data in Excel

A macro processes code written in the Visual Basic Editor to manage and manipulate huge volumes of data. The previous chapter provides information on how you can use a macro to format certain fields or cells in Excel to meet your criteria.

The following is an example of a VBA script:

```
Sub ConfigureLogic()
    Dim qstEntries
    Dim dqstEntries
    Dim qstCnt, dqstCnt
    qstEntries = Range("QualifiedEntry").Count
    qst = qstEntries - WorksheetFunction.CountIf(Range("QualifiedEntry"),
    "")
    ReDim QualifiedEntryText(qst)
    'MsgBox (qst)
    dqstEntries = Range("DisQualifiedEntry").Count
    dqst = dqstEntries -
    WorksheetFunction.CountIf(Range("DisQualifiedEntry"), "")
    ReDim DisqualifiedEntryText(dqst)
    'MsgBox (dqst)
    For qstCnt = 1 To qst
    QualifiedEntryText(qstCnt) =
    ThisWorkbook.Worksheets("Qualifiers").Range("J" & 8 + qstCnt).value
    'MsgBox (QualifiedEntryText(qstCnt))
    logging ("Configured Qualified Entry entry #" & qstCnt & " as {" &
    QualifiedEntryText(qstCnt) & "}")
    Next
    For dqstCnt = 1 To dqst
```

```
DisqualifiedEntryText(dqstCnt) =
ThisWorkbook.Worksheets("Qualifiers").Range("M" & 8 +
dqstCnt).value

'MsgBox (DisqualifiedEntryText(dqstCnt))

logging ("Configured DisQualified Entry entry #" & qstCnt & " as {" &
DisqualifiedEntryText(dqstCnt) & "}")

Next

includeEntry =
ThisWorkbook.Worksheets("Qualifiers").Range("IncludeSibling").value

'MsgBox (includeEntry)

logging ("Entrys included in search - " & includeEntry)

End Sub
```

How to Analyze and Manipulate Data on a Spreadsheet

If you want to use VBA to analyze data, you should check the macro settings in Excel. Ensure that the settings as per your requirements. You should also make sure that the macro settings are activated in Excel. Now, create a worksheet and call it ‘Qualifiers.’ We will be using this worksheet to check the data and ensure that the data qualifies all the selections that you require. You must then set up the qualifiers based on the code you have written. You cannot cut and paste these qualifiers but will need to enter them manually.

```
ThisWorkbook.Worksheets("Qualifiers").Range("J" & 8 + qstCnt).value
```

How to Construct an Array and Locate the Range

In the above function, the range will start from Cell J9. The function notes 8, but the range is 9 since we have declared the qstCnt to be 1 using the following code:

```
For qstCnt = 1 To qst
```

It is because of this statement that the list will start at 9.

If you want to construct an array using the entries in the Qualifiers worksheet, you should add random words or numbers between cells J9 and J13, including those cells. When the rows are complete, you can find and manipulate the data in Excel.

```
Private Sub CountSheets()
    Dim sheetcount
    Dim WS As Worksheet
    sheetcount = 0
    logging ("*****Starting Scrub*********")
    For Each WS In ThisWorkbook.Worksheets
    sheetcount = sheetcount + 1
    If WS.Name = "Selected" Then
    'need to log the date and time into sheet named "Logging"
    ActionCnt = ActionCnt + 1
    logging ("Calling sheet: " & WS.Name)
    scrubsheet (sheetcount)
    Else
    ActionCnt = ActionCnt + 1
    logging ("Skipped over sheet: " & WS.Name)
    End If
    Next WS
    'MsgBox ("ending")
    ActionCnt = ActionCnt + 1
    logging ("****Scrub DONE!")
    Application.ScreenUpdating = True
End Sub
```

The following example will show you how you can write a macro for a working tab counter.

```
Dim sheetcount
Dim WS As Worksheet
sheetcount = 0
logging ("*****Starting Scrub*********")
For Each WS In ThisWorkbook.Worksheets
sheetcount = sheetcount + 1
```

When you initialize the sheet count variable, you should first set it to zero before you restart the counter. You can also use the logging() subroutine to keep track of all the actions in the qualifiers tab to make the correct selections. The For loop in the above example will set up the counting variable in the Active Workbook. Once you initialize WS, it will make the worksheet that you are currently in the active worksheet. Since this module is unnamed, it will run in any workbook. If you have many workbooks open, this module may run in an incorrect workbook. If you want to avoid any errors, you should name the workbook that you want the module to run in.

When the loop runs, it will add another variable to the sheet count and keep a track of the tabs. We will then move to

```
If WS.Name = "Selected" Then
'need to log the date and time into sheet named "Logging"
ActionCnt = ActionCnt + 1
logging ("Calling sheet: " & WS.Name)
scrubsheet (sheetcount)
Else
ActionCnt = ActionCnt + 1
logging ("Skipped over sheet: " & WS.Name)
End If
```

In this section of the code, we are looking for the Selected tab. If the variable WS is the same as the Selected worksheet, you can fire the Scrub sheet subroutine. If the variable WS is not the same as the Selected worksheet, then the sheet will be skipped, and the action will be counted. The code above is an example of how you can write a macro to count the number of tabs and locate a specific tab.

The next parts of this chapter talk about the different ways you can manipulate data in Excel.

Different Ways to Manipulate Data

Count the Number of Sheets in a Workbook

```
Dim TAB
For Each TAB In ThisWorkbook.Worksheets
'some routine here
Next
```

Filter by Using Advanced Criteria

```
Range("A2:Z99").Sort key1:=Range("A5"), order1:=xlAscending, Header:=xlNo
```

Find The Last Column, Cell Or Row On A Worksheet

```
Dim cellcount
cellcount = Cells(ThisWorkbook.Worksheets("worksheet").Rows.Count, 1).End(xlUp).Row
```

Getting Values from Another Worksheet

```
dim newvalue
newvalue = ThisWorkbook.Worksheets("worksheet").Range("F1").value
```

Apply Auto-Fit To A Column

```
Columns("A:A").EntireColumn.AutoFit
```

Adding Named Ranges to Specific Sheets

```
ThisWorkbook.Worksheets("worksheet").Names.Add Name:="Status", RefersToR1C1:="=worksheet!C2"
```

Insert Rows Into A Worksheet

```
Dim Row, Column
Cells(Row, Column).EntireRow.Select
Selection.Insert
```

Copy an Entire Row for Pasting

```
ActiveSheet.Range("A1").EntireRow.Select
Selection.Copy
Delete An Entire Row
ActiveSheet.Range("A1").EntireRow.Select
Selection.Delete
```

Inserting a Column into a Worksheet

```
Dim Row, Column
Cells(Row, Column).EntireColumn.Select
Selection.Insert
```

Insert Multiple Columns into a Worksheet

```
Dim insertCnt
Dim Row, Column
For insertCnt = 1 To N
ThisWorkbook.Worksheets("worksheet").Select
Cells(Row, Column).EntireColumn.Select
Selection.Insert
Next
```

Select a Specific Sheet

```
ThisWorkbook.Worksheets("worksheet").Select
```

Compare Values In A Range

```
Dim firstrange
Dim Logictest
Logictest = "some word or value"
If (Range(firstrange).value = Logictest) then
'some routine here
End If
```

Chapter Nine

Working with Excel Workbooks and Worksheets

The Workbook Collection

The Workbooks collection provides a list of all workbooks that you have open at a given time. You can select a single workbook from this list that you want to use in your program. The Workbook object that you select will provide general information about the file, including its name and location. You can also use this object to access other major objects in the document. These objects include standalone Chart objects and Worksheet objects.

Example:

```
Public Sub WorkbookDemo()
‘ Holds the output data.
Dim Output As String
‘ Get the test workbook.
Dim ActiveWorkbook As Workbook
Set ActiveWorkbook =
Application.Workbooks(“ExcelObjects.xls”)
‘ Get the workbook name and location.
Output = “Name: “ + ActiveWorkbook.Name + vbCrLf + _
“Full Name: “ + ActiveWorkbook.FullName + vbCrLf + _
“Path: “ + ActiveWorkbook.Path + vbCrLf + vbCrLf
‘ Holds the current sheet.
Dim CurrSheet As Worksheet
‘ Look for every sheet.
Output = “Worksheet List:” + vbCrLf
For Each CurrSheet In ActiveWorkbook.Worksheets
Output = Output + CurrSheet.Name + vbCrLf
Next
```

```
‘ Holds the current chart.
Dim CurrChart As Chart
‘ Look for every chart.
Output = Output + vbCrLf + “Chart List:” + vbCrLf
For Each CurrChart In ActiveWorkbook.Charts
Output = Output + CurrChart.Name + vbCrLf
Next
‘ Display the output.
MsgBox Output, vbInformation Or vbOKOnly, “Object List”
End Sub
```

The code starts by using the Application.Workbooks collection which will allow you to retrieve a single Workbook object. You should always use the full name of the Excel file as the index in the collection. You should also include the extension of the file. The resulting workbook object will contain information about the document. This object will also provide the summary information of the document, and you can use this object to control and maintain the windows, and also add new elements like worksheets.

Once the code accesses the workbook, it will use the ActiveWorkbook object to access the worksheets in the list. The code will rely on the For Each...Next statement to access these worksheets. Alternatively, you can use an index to access the individual worksheets in the code. The Worksheet, ActiveWorksheet, contains properties and methods for manipulating any data that the Worksheet contains, including embedded objects, such as charts or even pictures. Every worksheet appears in the ActiveWorkbook object list by its object name (not the friendly name that you give it), so you can access them without using the Worksheets collection.

Only independent charts will appear when you use the ActiveWorkbook object. The same technique can be used to access any Chart object in the worksheet as a Worksheet object. The difference is that you should use Charts Collection instead of the Worksheets Collection. You should note that the Chart names will appear in the list of objects that are present in the ActiveWorkbook. This means that you can access the chart directly without having to use the Charts collection.

The Worksheet Collection

One of the easiest methods to access a worksheet in many situations is to use the Sheets collection. You do not follow the Excel object hierarchy when you want to identify the worksheet you want tot work with. If you access the worksheet at the top of the hierarchy, it means that there are no objects that exist at lower levels available either, so this technique is a tradeoff.

You can access any type of sheet, and not just a worksheet if you use the Sheets collection. A standalone Chart object that you use in any of the sheets can also come into this collection. Look at the example in the previous section, "Using the Workbooks collection," and you will notice that the charts and worksheets are treated as different objects.

Example:

```
Public Sub ListSheets()
‘ An individual entry.
Dim ThisEntry As Variant
‘ Holds the output data.
Dim Output As String
‘ Get the current number of worksheets.
Output = “Sheet Count: “ + _
CStr(Application.Sheets.Count)
‘ List each worksheet in turn.
For Each ThisEntry In Application.Sheets
‘ Verify there is a sheet to work with.
If ThisEntry.Type = XlSheetType.xlWorksheet Then
Output = Output + vbCrLf + ThisEntry.Name
End If
Next
‘ Display the result.
MsgBox Output, _
vbInformation or vbOKOnly, _
```

```
“Worksheet List”

End Sub
```

In the example above, we will create a Variant which will hold the different sheet types. If you use a Chart or Worksheet object, the code will fail since the Sheets enumeration will return a valid type, but not the type you need. The issue with using this Variant is that the editor in VBA cannot provide automatic completion or balloon help. You have to ensure that you type the method in correctly and use the correct property names without any help.

Once the code creates the necessary variables, it will provide the number of sheets in the workbook. This number will include all the worksheets and charts in the workbook, and not just the sheets.

A For Each...Next loop will retrieve each sheet in turn. You should notice how we use the If...Then statement to compare the values of the Variant type and the XlSheetType.xlWorksheet constant. When you use this technique, you can separate the worksheet you are using from other Sheets collection types whenever necessary.

Charts Collection

The Charts collection is a way to build a custom chart whenever necessary. An advantage of creating charts by using a code is that they do not use too much space, and you can create a variety of different charts without spending too much time on the theme.

Example:

```
Public Sub BuildChart()

‘ Create a new chart.

Dim NewChart As Chart

Set NewChart = Charts.Add(After:=Charts(Charts.Count))

‘ Change the name.

NewChart.Name = “Added Chart”

‘ Create a series for the chart.

Dim TheSeries As Series
```

```
NewChart.SeriesCollection.Add _
Source:=Worksheets(“My Data Sheet”).Range(“A$3:B$8”)
Set TheSeries = NewChart.SeriesCollection(1)
‘ Change the chart type.
TheSeries.ChartType = xl3DPie
‘ Change the series title.
TheSeries.Name = “Data from My Data Sheet”
‘ Perform some data formatting.
With TheSeries
.HasDataLabels = True
.DataLabels.ShowValue = True
.DataLabels.Font.Italic = True
.DataLabels.Font.Size = 14
End With
‘ Modify the chart’s legend.
With NewChart
.HasLegend = True
.Legend.Font.Size = 14
End With
‘ Modify the 3-D view.
With NewChart
.Pie3DGroup.FirstSliceAngle = 90
.Elevation = 45
End With
‘ Format the chart title.
```

```
NewChart.ChartTitle.Font.Bold = True
NewChart.ChartTitle.Font.Size = 18
NewChart.ChartTitle.Format.Line.DashStyle _
= msoLineSolid
NewChart.ChartTitle.Format.Line.Style = msoLineSingle
NewChart.ChartTitle.Format.Line.Weight = 2
‘ Compute the optimal plot area size.
Dim Size As Integer
If NewChart.PlotArea.Height > NewChart.PlotArea.Width
Then
Size = NewChart.PlotArea.Width
Else
Size = NewChart.PlotArea.Height
End If
‘ Reduce the plot area by 10%.
Size = Size - (Size * 0.1)
‘ Format the plot area.
With NewChart.PlotArea
.Interior.Color = RGB(255, 255, 255)
.Border.LineStyle = XlLineStyle.xlLineStyleNone
.Height = Size
.Width = Size
.Top = 75
.Left = 100
End With
```

```
‘ Format the labels.
Dim ChartLabels As DataLabel
Set ChartLabels = TheSeries.DataLabels(0)
ChartLabels.Position = xlLabelPositionOutsideEnd
End Sub
```

In the above example, the code will create a new chart. This chart will appear in the workbook as the last chart but will not appear as the last item in the workbook. A worksheet which appears after the last chart will also appear after the new chart that is created. The NewChart.Name property will change the name that appears at the bottom of the chart. This property does not change the name of the chart.

At this point, the chart is blank, and you must add at lear one series to the chart if you want to display some data on it. A pie chart will use only one series at a time, but there are other charts that allow you to use multiple data series. For instance, you can use a bubble chart to show multiple series of data. The next task of the code will create a data series based on the worksheet named My Data Sheet. You will notice that the code cannot set TheSeries variable equal to the output of the method Add in this example. Therefore, it uses an additional step to obtain the new series from the SeriesCollection collection.

You should also notice that the Range property has two columns of information. When you are working with Excel 2007, the first column defines the XValues property for the chart. The XValues property determines the entries in the legend for a pie chart. On the other hand, these values appear at the bottom of the display for a bar chart. In both cases, you want to display the labels onscreen so that you can see their effect on the overall display area.

Chapter Ten

Automating Processes Using VBA

You can use a visual basic application (VBA) to automate any process that you want to in any MS Office product. All you need to ensure is that you have a basic understanding of what VBA is. This chapter will provide some real examples of how you can use VBA to automate processes in Excel. You can see how you can transform an entire business process into the click of a button.

We will look at how you can create a composite key and identify all the records in a file that will match the master source file that you are using. We will also see how you can analyze the records that do not match the source data by using a pivot table to recognize or identify patterns.

Before we delve into learning how to write the code, we will need to ensure that you develop the right mindset to ensure that VBA works for you. You can save enough time when you use VBA to automate the processes. This will also make you a hero at work, but you must ensure that you do use it wisely. There is a fundamental shift that will happen when you begin to work with VBA, and you need to learn how to make that shift work for you and for the company.

The Macro Mindset

When you have automated most or all the processes that are related to your job, you will have adopted a new outlook on how you can use the different MS Office products, especially Excel. You will develop this mindset once you have a good understanding of the object hierarchy of the applications. You will also gain this mindset when you learn how to use the macro recorder effectively.

When you understand how a macro works, you will look for different ways to automate other processes and potentially save yourself a lot of time. A macro is like any other program – you will need to map some input parameters to output parameters. This is often done to automate work. When you do this, you can say that your mindset has shifted from the fundamental mindset to the macro mindset.

This mindset will cause problems if you do not know how you should use it, or in what context you should use it in. You certainly can automate every process that your firm does, but what is the cost of doing that?

You need to keep two points in mind:

1. How much time will I take to automate the process?
2. Will I ever have an issue with any overlapping in the functionality of other processes or applications?

Let us now look at how you should decide on when you should use VBA to automate processes.

Understand the Context of your Automation Project

As with every project, you should first understand the parts of the process that you want to automate. You should define the timeline and define your deadlines and the life expectancy of every solution that you develop.

You should also try to understand what the risks are of using VBA are, and also look at other alternatives. For instance, if the department you are in is investing in some new business intelligence tool which will solve the issue, you should focus on testing the tool before you write any VBA code.

You should always look at the timeline and understand how long it will take you to finish writing the code. You should compare that with the time it will take you to finish that task manually. Also try to understand how long you will need to perform this task for. If you know that your process will change in a few months, you should see whether it is required for you to automate the process.

Changes made to the decisions in resource allocation and IT procurement will reduce the life expectancy or use of the code that you write. This means that you must understand that there is a probability that you may be wasting your time developing a solution using VBA which can be replaced by any other tool. The code you write can also become obsolete because of some changes in the business. For example, you may have developed the process for a client who your company no longer works with.

That being said, a business intelligence tool will cost a company thousands of dollars, and there are many companies that use Microsoft Office tools by default. Therefore, there is very little risk in starting the VBA project, identifying that the project is no longer useful and then removing the process altogether. You can instead investigate the tool and see how it can be used to improve the process.

You can then spend some hours training yourself on the tool, and then identify that the tool cannot be used in the company because of some issues with the compatibility.

You can use VBA to automate processes in small companies, small departments in large companies and companies that are downsizing. This is because they operate on a small budget and would use any tool that they have access to. In other words, departments and companies that are trying to save a lot of money will find it easier to use VBA to automate processes when compared to business intelligence tools.

You should keep the following rules in mind when you choose to use VBA:

- If you want to save money, and that is a high priority, you should use VBA.
- A VBA solution is highly flexible.
- A VBA solution is best when it is maintained, and there are only some users who use the process.
- Most VBA solutions are written in Excel.
- A small company can use VBA more when compared to larger companies.
- A VBA solution is robust if you know how to word the program correctly.

One of the main objectives of VBA and macros is to save you some time. This will also add to the functionality, but the focus here is always time. When you know you are saving yourself some time, the macro mindset will work best for you.

Let us now look at some actual code.

Product Code Lookup Procedure

If you are new to VBA, you should understand that a procedure is a block of code that can be executed independently. The idea behind this procedure is to understand whether the products in the ProductReport file do contain the product specification codes for new item catalogue categories. This means that the product specifications do not exist in the master database. If this is the case, the code will identify the items and also give you a count of the number of items. In this example, we will be using a text file called ProductReport, which you can download from GitHub. This report is a third-party report, and the base file is an Excel workbook which stores the historical information. It would be a good idea to scan the “ProdCodeLookup2” in GitHub in full. This chapter will break it down into smaller sections.

```
Sub ProdCodeLookup2()
'Reformats the Product Report (ProdReport), creates the ProdCombo for all items, _
and then uses VLookup to find matches
Application.ScreenUpdating = False

Dim rngX As Range, rngY As Range, rngZ As Range
Dim ICC As String
Dim ItemNum As Variant
Dim Prod1 As String, Prod2 As String, Prod3 As String, Prod4 As String, Prod5 As String
Dim ProdCombo As Variant
Dim FinalRow As Long, StartTime As Long
Dim SecondsElapsed As Double
Dim PRwb As Workbook, BLwb As Workbook
Dim PRsht As Worksheet, BLsht As Worksheet
Dim LVal As Range, VLtable As Range
Dim VLOutput As Variant
Dim Count As Integer
Dim ProdMatchTotal As Variant
Set PRwb = Application.Workbooks("ProductReport")
Set BLwb = Application.Workbooks("Baseline")
Set PRsht = PRwb.Worksheets("Export Worksheet")
Set BLsht = BLwb.Worksheets("Static Table")

StartTime = Timer
'Abbreviate two important header names:
ICC = "Item Catalog Category"
PC = "ProdCombo"

PRsht.Activate
'Hide all columns except those that comprise the ProdCombo, _
format the headers, and add the headers for ProdCombo and ProdCode:
    Columns("C:D").EntireColumn.Hidden = True
    Columns("G:N").EntireColumn.Hidden = True
    Columns("Q:AK").EntireColumn.Hidden = True
    Columns("AM").EntireColumn.Hidden = True
    Columns("AO:AU").EntireColumn.Hidden = True
    Columns("AW:BM").EntireColumn.Hidden = True
    Range("BN1").Value = "ProdCombo"
    Range("BO1").Value = "ProdCode"
    Range("1:1").Select
        Selection.Font.Bold = True

'Filter by Label:
    ActiveSheet.Range("1:1").AutoFilter
```

```
        'Find the correct header to filter by:
          Set rngX = ActiveSheet.Range("1:1").Find(ICC, LookAt:=xlWhole)
            If Not rngX Is Nothing Then
            End If


        'Filter Item Catalog Category by "Labels":
          ActiveSheet.Range(rngX.Address, Selection.End(xlDown)).AutoFilter
Field:=rngX.Column, Criteria1:="Label"
        'Create the ProdCombo:
          FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
            For Count = 0 To FinalRow - 2
        'List the first cell here in the primary key column and ensure that there are
no blanks:
              ItemNum = Range("B2").Offset(Count, 0).Value
        'Now define the products to ensure they're aligned with the first Item
Number:
              Prod1 = Cells(Count + 2, 48).Value
              Prod2 = Cells(Count + 2, 40).Value
              Prod3 = Cells(Count + 2, 15).Value
              Prod4 = Cells(Count + 2, 16).Value
              Prod5 = Cells(Count + 2, 38).Value
        'Now concatenate them in the correct order with spaces:
              ProdCombo = Prod1 & " " & Prod2 & " " & Prod3 & " " & Prod4
& " " _
              & Prod5
        'Now specify where you want the ProdCombo to appear:
                Range("BN2").Offset(Count, 0) = ProdCombo
            Next Count
        'Delete blanks from ProdCombo:
          Set rngY = ActiveSheet.Range("1:1").Find(PC, LookAt:=xlWhole)
            If Not rngY Is Nothing Then
            End If
          ActiveSheet.Range(rngY.Address, Selection.End(xlDown)).AutoFilter
Field:=rngY.Column, Criteria1:="="
          ActiveCell.Offset(1, 0).Rows("1:1").EntireRow.Select
          Range(Selection, Selection.End(xlDown)).Delete
          ActiveSheet.ShowAllData
          ActiveSheet.Range(rngX.Address, Selection.End(xlDown)).AutoFilter
Field:=rngX.Column, Criteria1:="Label"

        'Now run the ProdCodeLookup2 sub procedure
        FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
        Set VLtable = BLsht.Range("$AY$1:$BA$29741")
        On Error Resume Next
        For Count = 0 To FinalRow

          Set LVal = SRsht.Range("BN2").Offset(Count, 0)

        VLOutput = Application.VLookup(LVal, VLtable, 3, False)

        If IsError(VLOutput) Then
            VLOutput = 0
            Else
                PRsht.Range("BO2").Offset(Count, 0) = VLOutput
            End If
        Next Count
        On Error GoTo 0

        Set rngZ = ActiveSheet.Range("1:1").Find("ProdCode", LookAt:=xlPart)
                If Not rngZ Is Nothing Then
                End If
        ActiveSheet.Range(rngZ.Address, Selection.End(xlDown)).AutoFilter
Field:=rngZ.Column, Criteria1:="<>"
        ProdMatchTotal = Application.Count(Range(rngZ.Offset(1),
rngZ.Offset(FinalRow)))
        MsgBox ("There are " & ProdMatchTotal & " baseline product matches")

        Application.ScreenUpdating = True
        SecondsElapsed = Round(Timer - StartTime, 2)
        MsgBox "This code ran successfully in " & SecondsElapsed & " seconds",
vbInformation
        End Sub
```

ProdCodeLookup2 Breakdown

The most important part of every process is to break the problem down into smaller bits and tackle each of those individually. One of the best ways to do this is to write every block of code in the order in which you will perform your task as if you were doing it manually. You can then refine the process once you have a better understanding of the process. You can also rewrite the code to improve readability and performance.

Let us look at the first piece of code that is manageable:

```
Sub ProdCodeLookup2()
'Reformats the Product Report (ProdReport), creates the ProdCombo for all items, _
and then uses VLookup to find matches

Application.ScreenUpdating = False
```

You should also notice that the name of every procedure follows the Pascal Code. We also end the name with a number which is a good way to write code. It is painful when you realize that you are troubleshooting the incorrect procedure. You should always look for a way to update your code and create some working copies of the procedure.

The first statement in line 5 has been written to increase or improve the performance of the macro. ScreenUpdating is a property that controls whether the compiler needs to update the screen when the macro runs. When you turn off this property, you will see that the changes are only made once the full code has run. It is always a good idea to set this property to true at the end of the code.

```
Dim rngX As Range, rngY As Range, rngZ As Range
Dim ICC As String
Dim ItemNum As Variant
Dim Prod1 As String, Prod2 As String, Prod3 As String, Prod4 As String, Prod5 As String
Dim ProdCombo As Variant
Dim FinalRow As Long, StartTime As Long
Dim SecondsElapsed As Double
Dim PRwb As Workbook, BLwb As Workbook
Dim PRsht As Worksheet, BLsht As Worksheet
Dim LVal As Range, VLtable As Range
Dim VLOutput As Variant
Dim Count As Integer

Dim ProdMatchTotal As Variant
```

The names used in the block of code above are not creative, and you do not want to get creative with the names since it becomes harder to debug the code. If there is something wrong with the code, a simple name will help you identify where the issue is.

It is easier to work with variables and manage them by grouping them based on their function and data type. For instance, you may use the keyword rng to describe the range variables. You can then use them to find the columns or the parts of the table that you will need to work with.

You should also notice that the lookup variables are all defined in the same space. When you write a longer procedure, it will be easier to add comments to these lines, and also declare a variable in a new line using the “Dim” keyword.

If you make this a habit, it will make it easier for you to update the code or troubleshoot any errors. For instance, the above code has been rewritten in the style described above.

```
Dim rngX As Range 'Range X, ICC header

Dim rngY As Range 'Range Y, Product Combo

Dim rngZ As Range 'Range Z, Product Code

Dim ICC As String 'Item Catalogue Category

Dim ItemNum As Variant 'Item Number; used to eliminate blanks

Dim Prod1 As String 'Product 1, the first attribute in the composite key

Dim Prod2 As String 'Product 2, the second attribute in the key

Dim Prod3 As String 'Product 3

Dim Prod4 As String 'Product 4

Dim Prod5 As String 'Product 5

Dim ProdCombo As Variant 'Product Combo

Dim FinalRow As Long 'FinalRow formula variable
Dim StartTime As Long 'Used to calculate total runtime
Dim SecondsElapsed As Double 'Used to calculate total runtime
Dim PRwb As Workbook 'ProductReport workbook
Dim BLwb As Workbook 'Baseline workbook
Dim PRsht As Worksheet 'ProductReport worksheet
Dim BLsht As Worksheet 'Baseline worksheet
```

I am sure you get the idea now. All you need to do is write the name of the variable using a short phrase and describe how you will use it. This is a straightforward method, and it is important that you understand it.

You should now assign the workbooks and worksheets to different object variables. You can benefit in many ways through this action. You will also improve readability and performance. When you master writing code in VBA, you can always avoid doing this. The goal is to ensure that your code is easy for you to understand, thereby making it easy for you to troubleshoot any errors.

You should now assign the worksheet and workbook variables to different objects. You should first assign workbook variables since they are the parent objects.

```
Set PRwb = Application.Workbooks("WorkbookName1")
Set BLwb = Application.Workbooks("WorkbookName2")
Set PRsht = PRwb.Worksheets("Export Worksheet")
Set BLsht = BLwb.Worksheets("Static Table")
```

Let’s check out the next part (lines 25-30):

```
StartTime = Timer
'Abbreviate two important header names:
ICC = "Item Catalog Category"
PC = "ProdCombo"

PRsht.Activate
```

We will use the StartTime variable to hold the initial value of the timer and use the variable to calculate the time it takes to run the application. You can also create shorter variables if you want to refer to those variables multiple times in the code. You can ensure that you do not make any mistakes.

The last line in this section will define the scope for the block of code below. You should never expect that the compiler will guess which worksheet or workbook it needs to refer to. Therefore, you must ensure that you specify this.

You will need to prepare the data and format it before you can create the “ProdCombo” composite key. You will first need to hide all the columns in the baseline workbook and remove any columns that are irrelevant for us. You should then create the new columns using the headers that we are going to use. To ensure that the labels are readable, you can make them bold.

```
'Hide all columns except those that comprise the ProdCombo, _

format the headers, and add the headers for ProdCombo and ProdCode:

Columns("C:D").EntireColumn.Hidden = True

Columns("G:N").EntireColumn.Hidden = True

Columns("Q:AK").EntireColumn.Hidden = True

Columns("AM").EntireColumn.Hidden = True

Columns("AO:AU").EntireColumn.Hidden = True

Columns("AW:BM").EntireColumn.Hidden = True

Range("BN1").Value = "ProdCombo"

Range("BO1").Value = "ProdCode"

Range("1:1").Select

Selection.Font.Bold = True
```

In the lines 44 through 52, we are turning the filters on for every header, and are removing all the values except for the labels. Let us make the assumption that we are only looking at the product specifications.

We are creating a ProdCombo in the lines 50 through 69 using a loop. We are also concatenating or combining the values in the columns using a space delimiter.

It is always a good idea to delete a blank product, and this is what the lines 70 to 78 do. In this example, we only want to look at those products that have some value in one of the five fields. We do not need to look at any values that have blanks in all the labels.

You must always look at the comments. You will notice that the comment in line 80 talks about running the ProdCodeLookup2 procedure.

If you are using VLOOKUP for the first time in a macro, you should write two different procedures – one for every other process and one just to execute the lookup. Experts recommend this method since it is easier to develop the code. The ProdCodeLookup2 will perform a lookup in every cell that is in a specified column and will loop all the records in the table until it reaches the end.

Before we look at the next section where we perform some analyses, let us look at some ways to deal with issues that crop up with using VBA.

Tips to Deal with VLookups in VBA

Errors and blanks will often return an error statement when you use a vlookup function. This may cause the entire procedure to crash. If you do choose to use vlookup in your code, you will need to include an error handling option to ensure that the code does not crash. We cover this in the next chapter.

There are many examples for the same on Stack Overflow. You can learn more about how to handle errors with VLOOKUP through those examples. You can solve the problem by wrapping the loop up in the lines 83 and 96 in the code above. This should help to solve the problem in a simple manner.

You should then perform the following tasks in the last lines of the code:

1. Use the column name to define the location of the ProdCode column header.
2. Count the values under the ProdCode column header and assign the number to a variable using the workbook function “Count.”
3. You can then return the result using a message box.
4. Turn the ScreenUpdating on.
5. You can also return the time it took for the application to run. Display

this in a message box.

This will conclude the first VBA script. You may probably be wondering where the data analysis is. Let us look at the next section to understand more about how to analyze data.

Data Analysis

This information is certainly useful, but what should you do about the remaining products? How will the results of those compare to the remaining categories or the labels that you use? What types of comparison should you investigate or analyze? What about those items or products that do not match your requirement? What is it that you can learn about those products? Let us now add a status flag to every product. This will make things interesting and slightly complicated. This status will represent whether the product specification code that you are looking at is active.

To recap quickly, the dimensions of the analysis can be summarized into “active/inactive status,” “product category” and “match/no match status.” Now that we have these details in mind, you can use a pivot table to help you visualize the different user cases. The data analysis is in the procedure written below:

```
    Sub ProdFlag_v3()

    'Run this on a ProductReport to find records without any specs and copy
all of the records _

    without specs onto a separate worksheet (only copy columns A-G).
Analysis tab added with pivot tables.

    Application.ScreenUpdating = False

    Dim PRtable As Range, rngX As Range, SpecHeader As Variant

    Dim PRTableRows As Long, PRTableColumns As Long

    Dim PRsht As Worksheet, nsPRsht As Worksheet

    Dim FinalRow As Long, i As Long

    Dim IstatActive As Variant, IstatInactive As Variant
```

```
        Istat = "Item Status"

        Set PRsht = Worksheets("Export Worksheet")

        PRTableRows = PRsht.Cells.Find("*", searchorder:=xlByRows,
searchdirection:=xlPrevious).Row
        PRTableColumns = PRsht.Cells.Find("*", searchorder:=xlByColumns,
searchdirection:=xlPrevious).Column

        Set PRtable = PRsht.Range("A1", Cells(PRTableRows, PRTableColumns))

        Sheets.Add After:=ActiveSheet
        Sheets("Sheet1").Name = "NoSpecs"
        Set nsPRsht = Worksheets("NoSpecs")
        FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
        PRsht.Range("1:1").AutoFilter
          For Each SpecHeader In PRsht.Range("H1:BM1").Cells
            Range(SpecHeader.Offset(1), SpecHeader.Offset(FinalRow)) _
            .AutoFilter Field:=SpecHeader.Column, Criteria1:="="
          Next SpecHeader

        PRtable.Resize(PRTableRows, 7).Copy _
        nsPRsht.Range("A1")
        Application.CutCopyMode = False
        PRsht.ShowAllData
        nsPRsht.Range("A1").Select
```

```
FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
For i = 2 To FinalRow
    If Cells(i, 1) = "Active" Then
        Cells(i, 1).Resize(, 5).Font.ColorIndex = 25
    ElseIf Cells(i, 1) = "Inactive" Then
        Cells(i, 1).Resize(, 5).Font.ColorIndex = 3
    Else
        With Cells(i, 1).Resize(, 5).Font
            .Name = "TimesNewRoman"
            .Bold = True
        End With
    End If
Next i

ActiveSheet.Range("1:1").AutoFilter
Set rngX = ActiveSheet.Range("1:1").Find(Istat, LookAt:=xlPart)
        If Not rngX Is Nothing Then
        End If

IstatActive = Application.CountIf(Range(rngX.Offset(1),
rngX.Offset(FinalRow)), "Active")
IstatInactive = Application.CountIf(Range(rngX.Offset(1),
rngX.Offset(FinalRow)), "Inactive")
FinalRow = Cells(Rows.Count, 2).End(xlUp).Row
```

```
        'Create the NoSpecs_CategoryAnalysis tab with pivot tables

        Sheets.Add

        ActiveSheet.Name = "NoSpecs_Analysis"

            ActiveWorkbook.PivotCaches.Create(SourceType:=xlDatabase,
SourceData:= _

                "NoSpecs!R1C1:R23405C7", Version:=6).CreatePivotTable
TableDestination:= _

                "NoSpecs_Analysis!R3C1", TableName:="PivotTable1",
DefaultVersion:=6

            With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category" _

                )

                .Orientation = xlRowField

                .Position = 1

            End With

            ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _

                "PivotTable1").PivotFields("Item"), "Item Count", xlCount

            With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item
Status")

                .Orientation = xlColumnField

                .Position = 1

            End With

            ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category"). _

                AutoSort xlDescending, "Item Count"
                ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _

                "PivotTable1").PivotFields("Item"), "Percent", xlCount

            With ActiveSheet.PivotTables("PivotTable1").PivotFields("Percent")

                .Calculation = xlPercentOfTotal

                .NumberFormat = "0.00%"

            End With


        ActiveWorkbook.Worksheets("NoSpecs_Analysis").PivotTables("PivotTab
_

                PivotCache.CreatePivotTable
TableDestination:="NoSpecs_Analysis!R13C1", _

                TableName:="PivotTable2", DefaultVersion:=6

            With ActiveSheet.PivotTables("PivotTable2").PivotFields("Item
Status")

                .Orientation = xlRowField

                .Position = 1

            End With

            ActiveSheet.PivotTables("PivotTable2").AddDataField
ActiveSheet.PivotTables( _

                "PivotTable2").PivotFields("Item"), "Item Count", xlCount

            ActiveSheet.PivotTables("PivotTable2").AddDataField
ActiveSheet.PivotTables( _

                "PivotTable2").PivotFields("Item"), "Percent", xlCount

            Range("C13").Select

            With ActiveSheet.PivotTables("PivotTable2").PivotFields("Percent")

                .Calculation = xlPercentOfTotal
```

```
            .NumberFormat = "0.00%"
        End With

    Sheets.Add
    ActiveSheet.Name = "SpecAnalysis"

        ActiveWorkbook.PivotCaches.Create(SourceType:=xlDatabase,
SourceData:= _
            "Export Worksheet!R1C1:R26363C65",
Version:=6).CreatePivotTable TableDestination:= _
            "SpecAnalysis!R3C1", TableName:="PivotTable3",
DefaultVersion:=6
        With ActiveSheet.PivotTables("PivotTable3").PivotFields("Item Catalog
Category" _
            )
            .Orientation = xlRowField
            .Position = 1
        End With
        ActiveSheet.PivotTables("PivotTable3").AddDataField
ActiveSheet.PivotTables( _
            "PivotTable3").PivotFields("Item"), "Item Count", xlCount
        With ActiveSheet.PivotTables("PivotTable3").PivotFields("Item
Status")
            .Orientation = xlColumnField
            .Position = 1
        End With
        ActiveSheet.PivotTables("PivotTable3").PivotFields("Item Catalog
Category"). _
        AutoSort xlDescending, "Item Count"


        ActiveSheet.PivotTables("PivotTable3").AddDataField
ActiveSheet.PivotTables( _
        "PivotTable3").PivotFields("Item"), "Percent", xlCount
      With ActiveSheet.PivotTables("PivotTable3").PivotFields("Percent")
        .Calculation = xlPercentOfTotal
        .NumberFormat = "0.00%"
      End With


    Sheets.Add
      ActiveSheet.Name = "Label_Material"
    ActiveWorkbook.Worksheets("SpecAnalysis").PivotTables("PivotTable3").
_
        .CreatePivotTable TableDestination:="Label_Material!R3C1",
TableName:="PivotTable1" _
        , DefaultVersion:=6
      ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _
        "PivotTable1").PivotFields("Item"), "Item Count", xlCount
      ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _
        "PivotTable1").PivotFields("Item"), "Percent", xlCount
      With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category" _
        )
```

```
            .Orientation = xlPageField
            .Position = 1
        End With
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Label
Material")
            .Orientation = xlRowField
            .Position = 1
        End With
        ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category"). _
            CurrentPage = "(All)"
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category")
            .PivotItems("Bar Wrapper").Visible = False
            .PivotItems("IFC & Inner Tray").Visible = False
            .PivotItems("Printed Pouches & Packets").Visible = False
            .PivotItems("Shrink Sleeve").Visible = False
        End With
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Percent")
            .Calculation = xlPercentOfTotal
            .NumberFormat = "0.00%"
        End With
        ActiveSheet.PivotTables("PivotTable1").PivotFields("Label
Material").AutoSort _
            xlDescending, "Percent"
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Label
Material")
```

```
            .PivotItems("(blank)").Visible = False
        End With


    Sheets.Add
        ActiveSheet.Name = "COATING"
    ActiveWorkbook.Worksheets("SpecAnalysis").PivotTables("PivotTable3").
_
            .CreatePivotTable TableDestination:="COATING!R3C1",
TableName:="PivotTable1" _
            , DefaultVersion:=6
        ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _
            "PivotTable1").PivotFields("Item"), "Item Count", xlCount
        ActiveSheet.PivotTables("PivotTable1").AddDataField
ActiveSheet.PivotTables( _
            "PivotTable1").PivotFields("Item"), "Percent", xlCount
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category" _
            )
            .Orientation = xlPageField
            .Position = 1
        End With
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("COATING")
            .Orientation = xlRowField
            .Position = 1
        End With
        ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category"). _
            CurrentPage = "(All)"
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Item Catalog
Category")
            .PivotItems("Bar Wrapper").Visible = False
            .PivotItems("IFC & Inner Tray").Visible = False
            .PivotItems("Printed Pouches & Packets").Visible = False
            .PivotItems("Shrink Sleeve").Visible = False
        End With
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("Percent")
            .Calculation = xlPercentOfTotal
            .NumberFormat = "0.00%"
        End With

ActiveSheet.PivotTables("PivotTable1").PivotFields("COATING").AutoSort _
            xlDescending, "Percent"
        With ActiveSheet.PivotTables("PivotTable1").PivotFields("COATING")
            .PivotItems("(blank)").Visible = False
        End With


        MsgBox ("There are " & FinalRow & " records with no specifications"
& vbNewLine & _
    vbNewLine & "Number of Active records without specs: " & IstatActive
& vbNewLine _
    & "Number of Inactive records without specs: " & IstatInactive)
```

```
        Application.ScreenUpdating = True


    End Sub
```

This procedure does not use any pivot tables. You should keep the following points in mind when you write your own procedure:

Using Pivot Tables in VBA

There are some prerequisites to using a pivot table in VBA. You must ensure that the data source that you use has valid, high-quality and normalized data. The same holds true in the example above. You should create the test data for any product, its specifications and categories. In this section, you will learn how you should automate one of the most used and advanced features in Excel.

4. Copy the lines that were used in the lines 63 and 64 in the code above.
5. Now, navigate to the workbook and choose the option “Record Macro” in the Developer Tab.
6. Choose the cell you want to use to as the data for the pivot and click on “Ctrl + A” to select the cells.
7. Now, press “Alt + N + V + Enter.”
8. You should now create the pivot table manually. Drag the columns that you want to use, filter, group, and format the values as necessary.
9. Now, press the “Record Macro” button.
10. You now need to click on “Alt + F11” to get back to the editor window. Find the code that Excel recorded.
11. Paste this code into the project in the worksheet that you have created. Read the code and see how you can shorten it.
12. You can write the same code for any number of pivots in different worksheets. You will need to be careful about the filters you will be using.

Learn to Use Cell Referencing

Most people are not used to using the R1C1 style of cell referencing, but this is what the macro recorder will use. It is important that you familiarize yourself with this type of referencing to ensure that you save time when you work with pivot tables. This way you will have very little to do when you are editing the code that is generated by the macro recorder.

You can now decide which pivot table object or feature you want to use when you are learning to use VBA. You can always automate a file that has more than 20 pivot tables in a few minutes. You can only do this once you understand the keyboard shortcuts, set up the tables in your file properly and know how you should prepare the code.

Efficiently Allocating Your Time

If you want to automate the processes that you use to analyze information in Excel effectively, you must keep the following points in mind:

- You should spend most of your time in preparing and cleaning the data. You must ensure that the quality of the information you are using is top-notch, and you should think about any keys, flags, reconciliation, validation and any other simple calculation that you may want to use.
- You should always visualize the end result when you are building and refining the table.

You may be surprised that we focused very little on writing the code to automate a pivot table or any function in Excel. You should remember to always prepare the data set well and ensure that there are no issues with the data. This will make it easier to automate the process. You can save enough time and effort if you know how to use VBA efficiently.

Chapter Eleven

Error Handling

The easiest errors that you can avoid are the syntax errors, but these are some of the hardest errors to spot. The error can be because of the misuse of a punctuation, misuse of a language element or a spelling mistake. If you forget to include the End If statement in an If…Then statement, you have made a syntax error.

Typos are common syntax errors. These are especially hard to find if you make those errors in variable names. For example, the editor in VBA will view the MySpecialVariable and MySpecialVaraible as different variables, but there is a possibility that you will miss it when you begin to write the code. When you include the Adding Option Explicit at the beginning of the module, form or class module that you create, you can remove this problem. VBA can help you find a variety of typos if you add this statement to the start of your code. It is important that you use this statement in every part of your program when you write it.

There are times when you miss some of the subtle aids in locating the errors in syntax if you do not understand or view the tasks that the IDE or Integrated Development Environment performs. VBA will only display the balloon help feature when the editor in VBA can recognize the function name that you need to enter. If you do not see a balloon help button, you should understand that VBA does not know what function name you are referring to. This means that you will need to look at your code to identify the error. Unfortunately, this feature will only work where the editor in VBA will display the balloon help option. This does not work when you use property names.

Understanding compile errors

VBA uses a compiler to look for any errors that will prevent the program from functioning properly. You can create an If...Then statement and not include the End If statement in the program. The compiler will continue to run continuously and will allow you to find the mistakes in the code immediately once you make them.

VBA will use a compiler to find the syntax error in your code and then display an error message. Try the following when you write a new program. Open a new project, create a Sub using a specific name and type MsgBox(). Now, press

Enter. VBA will display a message box, which will state that it was expecting the equal to sign. If you use the parentheses after the keyword MsgBox, VBA will expect that you should include a result variable, which will hold the required result. For example, MyResult = MsgBox(“My Prompt”). As mentioned earlier, the debugger highlights the error in red.

Understanding run-time errors

A run-time error often occurs when there is an issue with something outside of your program. There are times when you type in the incorrect information and other times when the system rejects your access to the memory or disk. Your VBA code is completely correct, but the code will fail to function since there is an external error. Most companies, like Microsoft, always run a beta program to avoid any run-time errors. A beta program is a one that programmers develop to test their vendor-sponsored program before they release it into the market.

Understanding semantic errors

This is a particularly difficult error to understand and find since it is a semantic error. This error occurs when the VBA code and logic are both correct, but the meaning behind the code is incorrect. For instance, you can use the Do...Until loop in place of the Do...While loop. It may be the case that the code is correct and the logic behind the code is also correct, but the result is not what you expected since the meaning of a Do...Until loop is different from the meaning of a Do...While loop.

When you write a code, the words you use in the code should match your intent. Since a good book always relies on precise terms, a good program also relies only on the precise statements that you use in VBA. These statements will help VBA understand what you want it to do. One of the best ways to avoid making any semantic errors in the application is to always plan your program in advance. You should use a pseudo-code to “pre-write” the design, and then convert that code into the actual VBA code.

Conclusion

Thank you for purchasing the book. Most organizations have begun to use VBA to automate some of their processes in Excel. You can copy and paste information or create a pivot in Excel using VBA. If you want to learn VBA coding to improve processes at your workplace, you can use this book as your guide.

I hope you gather all the information you are looking for. I hope you can use the information in the book to perform data analysis.

Will You Help Me?

Hi there, avid reader! If you have extra time on your hands, I would really, really appreciate it if you could take a moment to click my author profile in Amazon. In there, you will find all the titles I authored and who knows, you might find more interesting topics to read and learn!

If it's not too much to ask, you can also leave and write a review for all the titles that you have read – whether it's a positive or negative review. An honest and constructive review of my titles is always welcome and appreciated since it will only help me moving forward in creating these books. There will always be room to add or improve, or sometimes even subtract certain topics, that is why these reviews are always important for us. They will also assist other avid readers, professionals who are looking to sharpen their knowledge, or even newbies to any topic, in their search for the book that caters to their needs the most.

If you don't want to leave a review yourself, you can also vote on the existing reviews by voting Helpful (Thumbs Up) or Unhelpful (Thumbs Down), especially on the top 10 or so reviews.

If you want to go directly to the vote or review process for my titles, just visit on any of the below titles:

Again, I truly appreciate the time and effort that you will be putting in leaving a review for my titles or even just for voting. This will only inspire me to create more quality content and titles in the future.

Thank you and have a great day!

Peter Bradley

Sources

ttps://www.excel-pratique.com/en/vba/introduction.php

ttp://www.easyexcelvba.com/introduction.html

ttps://www.tutorialspoint.com/excel_vba_online_training/excel_vba_introduction

ttps://www.thespreadsheetguru.com/getting-started-with-vba/

ttps://www.tutorialspoint.com/vba/vba_strings.htm

ttps://www.excel-easy.com/vba/string-manipulation.html

ttps://www.guru99.com/vba-data-types-variables-constant.html

ttps://corporatefinanceinstitute.com/resources/excel/study/vba-variables-dim/

ttps://powerspreadsheets.com/vba-data-types/

ttps://www.tutorialspoint.com/vba/vba_loops.htm

ttps://www.excel-easy.com/vba/loop.html

ttps://www.excelfunctions.net/vba-loops.html

ttps://powerspreadsheets.com/excel-vba-loops/

ttps://www.excelfunctions.net/vba-conditional-statements.html

ttps://analysistabs.com/excel-vba/conditional-statements/

ttps://www.techonthenet.com/excel/formulas/if_then.php

ttp://www.cpearson.com/excel/errorhandling.htm

ttps://excelmacromastery.com/vba-error-handling/

ttps://docs.microsoft.com/en-us/dotnet/visual-basic/language-
reference/statements/on-error-statement

ttps://simpleprogrammer.com/vba-data-analysis-automation/

Excel VBA

A Step-by-Step Comprehensive Guide on Excel VBA Programming Tips and Tricks for Effective Strategies

Introduction

I want to thank you for choosing this book, *Excel VBA - A Step-by-Step Comprehensive Guide on Excel VBA Programming Tips and Tricks for Effective Strategies'* and hope you find the book informative to learn Excel VBA.

It is difficult for a person to become an expert in VBA within a matter of days. It takes patience, time and practice to master coding in VBA. The first few books of the series provided information on different parts of Excel VBA. You learnt about the different data types, functions and methods you can use in Excel VBA. You also covered information on how you can handle errors and exceptions in VBA using the compiler and the visual basic environment.

There are still some topics that you will need to familiarize yourself with if you want to master coding in VBA. This book covers some of those topics. As a programmer, you will use the words procedures and sub procedures numerous times. This is because you will only be working on building and writing code for sub procedures. This book will provide information on what a sub procedure is, and how you can develop one for your project. You will also gather information on some tips and tricks that you can use to improve the project that you develop.

There are times when your system will slow down because of the volume of data, or code. In such situations, you can use some of the tips mentioned in this book to improve the performance of your project.

Thank you for purchasing the book. I hope you gathered all the information you were looking for.

Chapter One

Facts about VBA

Unlike the usual programming languages with only take code to build a program, we can record actions in VBA using the macro recorder. This has been covered extensively in the previous books. This chapter covers some important facts about macros and VBA.

Making macros available on every MS Excel Worksheet

When you begin to record a macro in VBA, Excel will prompt you to save the macro in a specific location. You can choose from the following locations: the workbook you are writing the macro in or the common workbook. If you save the macro in the current workbook, it will not be available for any other user to use in a different workbook. If you want to use a single macro or procedure across different workbooks, you should save it in personal.xlsb.

The personal.xlsb workbook is a hidden file, and cannot be seen or edited unless you unhide the file. You can view the file when you choose to look at all files in an Excel window.

Types Of Codes Found Across The Internet

You will come across three types of macros across the internet. The first type is the sub() macro. This is a macro that will run and execute a block of code or statements. This is the most common macro type available on the internet.

The second type is a function. This function is like the function that you use in excel, but unlike Excel functions, it is a user defined function. This function will also use VBA code.

The third type of macro is an event procedure that will work only when a certain event occurs. A macro which runs the moment you open your workbook is a classic example of this this type of macro.

You can choose a macro depending on what you want to another. For example, you can record a macro and use a shortcut key to call it when you want to format some cells. Alternatively, you can define a formula within a macro and store it in your files.

Where To Use The Code You Find On The Internet

You must remember to place the coffee write or source in the right place. If you do not save it correctly, it will not work. You can add functions and subs to the modules in your workbook. If you want to insert a module, go to Insert -> Module. Select the module that you want to insert, click it to open it and paste the code. You can include multiple codes in the same module. For some events, you will need to place a macro in the same sheet.

Saving A Workbook

You should always remember to save a workbook as a macro enabled workbook. A sheet that contains a macro will have different properties when compared to a sheet that does not have a macro. If you need to save the workbook as a macro enabled workbook, you should choose to save the workbook in the xlsm format.

Chapter Two

Resources for VBA Help

You cannot expect to become a VBA expert in a day. It is a journey and you will need to practice a lot before you become an expert. The best part about coding in Excel VBA is that there are many resources that you can use to improve your knowledge in Excel. This chapter covers some of the best places you can visit and some of the best resources you can use if you need a push in the right direction.

Allow Excel to Write the Code for You

If you have read the previous books, you know that you can use the macro recorder to help you with understanding your code. When you record any macro or the steps you want to automate using a record macro, Excel will write the underlying code for you. Once you record the code, you can review it and see what the recorder has done. You can then convert the code that the recorder has written into something that will suit your needs.

For instance, if you need to write a macro to refresh a pivot table or all pivot tables in your workbook and clear all the filters in the pivot table, it will get difficult to write the code from scratch. You can instead start recording the macro, and refresh every pivot table and remove all the filters yourself. When you stop recording the macro, you can review it and make the necessary changes to the code.

For a new Excel user, it would seem that the Help system is an add-in that always returns a list of topics that do not have anything to do with the topic you are looking for. The truth is that when you learn how to use the Help System correctly, it is the easiest and the fastest way to obtain more information about a topic. There are two basic tenets that you must keep in mind:

The Location Matters When You Ask For Help

There are two Help Systems in Excel – one that provides help on the different features in Excel and the other that provides information on some VBA programming topics. Excel will not perform a global search but will throw the criteria against the Help system, which is in your current location. This means that you will receive the help that you need depending on which area of Excel you are working in. If you want help on VBA and macros, you need to be in the Visual Basic Environment (VBE) when you look for information. This will ensure that the keyword search is performed on the correct help system.

Choose Online Help over Offline Help

When you look for some information on a topic, Excel will see if you are connected to the internet. If your system is connected to the internet, Excel will return results using some online content on Microsoft’s website. Otherwise, Excel will use the help files that are stored offline in Microsoft office. It is always good to choose online help since the content is more detailed. It also includes updated information and the links to other resources that you can use.

Using Code for Excel VBA from the Internet

The secret about coding or programming is that there is no necessity to build original code. The macro syntax that you need to use is always available on the internet. This proves that there is no need to create or develop code from scratch. You can use some existing code on the internet and then apply the code to a specific scenario.

If you are stuck with creating or writing a macro for a specific task, all you need to do is describe the task you want to accomplish using Google Search. All you need to do is add the words “Excel VBA” before you describe your requirement.

For instance, if you want to write a macro to delete every blank row in a worksheet, you should look for, “How to delete blank rows in a worksheet in Excel VBA?” You can bet a whole years’ worth of salary that a programmer has already developed code for the same problem. There is probably an example that is available on the internet, which will give you an idea of what you need to do. This way you can simply build your own macro.

Making Use of Excel VBA Forums

If you find yourself in a spot, and are unsure of what to do, you should post a question on a forum. The experts on these forums will guide you based on your requirement. A user forum is an online community that revolves around specific topics. You can ask numerous questions in these forums and get advice from experts on how you should solve some problems. The people answering your questions are volunteers who are passionate about helping the Excel community solve some real-world problems.

Many forums were built or developed to helping people with Excel. If you want to find such a forum, you should type “Excel Forum” in Google Search. Let us look at some tips you can use to get the most out of the user form.

You should always read the forum and follow all the rules before you begin. These rules will often include some advice on how you should post your questions and also the etiquette you should follow.

Always check if the question you want to ask has already been answered. You should try to save some time by looking at the archives. Now, take a moment to look at the forum and verify if any of the questions you want answers to have already been asked.

You should use accurate and concise titles for any of your questions. You should never create a forum question using an abstract title like “Please Help” or “Need advice.”

You should always ensure that the scope of your question is narrow. You should never ask vague questions like “Can I build an accounting tool in Excel?”

You should always be patient, and remember that the people who are trying to answer your question also have a day job. You should always give the experts time to answer the questions.

You should always check often when you post your questions. You will probably receive some information when they will ask you to provide some information about your question. Therefore, you should always return to your post to either respond to some questions or review the answer.

You should always thank the person who has answered your question. If the answer helps you, you should let the expert know the same.

Visiting Excel VBA Expert Blogs

Some Excel Gurus have shared their knowledge about VBA on their blogs. These blogs have a large number of tips and tricks that you can use to improve your VBA skills. They have some information that you can use to build your skills. The best part of using these blogs is that they are free to use.

These blogs do not necessarily answer your specific questions, but they offer many articles that you can use to advance your knowledge of VBA and Excel. These blogs can also provide some general guidance on how you can apply

Excel in different situations. Let us look at a few popular Excel blogs:

ExcelGuru

ExcelGuru is a blog that was set up by Ken Puls. He is an Excel expert who shares all his knowledge on his blog. Apart from the blog, Ken also offers many learning resources you can use to improve your knowledge in Excel.

Org

Org is a blog that was set up by Purna Chandoo Duggirala. He is an Excel expert from India who joined the scene in 2007. His blog offers innovative solutions and some free templates that will make you an Excel expert.

Contextures

Debra Dalgleish is the owner of a popular Excel website and is great with Microsoft Excel. Se has included close to 350 topics on her website, and there will definitely be something that you can read.

DailyDose

The DailyDose is a blog that is owned by Dick Kusleika. It is the longest running Excel blog, and Dick is an expert at Excel VBA. He has written articles and blogs for over ten years.

MrExcel

Bill Jelen always uses Excel to solve any problems he has at work. He offers a large archive of training resources and over thousands of free videos.

Mining YouTube for Some Excel VBA Training Videos

If you know that there are some training videos that are available on the internet, and these sessions are better than articles, you should look for those videos. There are many channels that are run by amazing experts that are passionate for sharing knowledge. You will be pleasantly surprised to see the quality of those videos.

Attending a Live Online Excel VBA Training Class

Live training sessions are a great way to absorb good Excel knowledge form a diverse set of people. The instructor is providing some information on different techniques, but the discussions held after the class will leave you with a wealth of ideas and tips. You may have never thought of these ideas ever before. If you can survive these classes, you should always consider attending more of these sessions. Here are some websites that you can use for such sessions:

1. Org
2. ExcelHero
3. ExcelJet
4. Learning From The Microsoft Office Developer Center For Help With VBA

You should use the Microsoft Office Dev Center to get some help on how to start programming in Office products. The website is slightly difficult to navigate, but it is worth it to look at the sample code, free resources, step-by-step instructions, tools and much more.

Dissecting Other Excel Files in Your Organization

Previous employees or current employees may have created files that already answer some of your questions. You should try to open different Excel files that contain the right macros, and also look at how these macros function. Then see how other employees in the organization develop macros for different applications. You should try not to go through the macro line-by-line but should look for some new techniques that may have been used.

You can also try to identify new tricks that you may have never thought of. You will probably also stumble upon some large chunks of code that you can implement or copy into your workbooks.

Ask the Local Excel Guru

Is there an excel genius in your department, company, community or organization? If yes, you should become friends with that person now. That person will become your own personal guru. Excel experts love to share their knowledge, so you should never be afraid to approach an expert if you have any questions or want to seek advice on how you can solve some problems.

Chapter Three

How to Improve the Performance of Macros

There are times when VBA will run very slowly, and this is certainly frustrating. The good news is that there are some steps that you can take to improve the performance of the macro. This chapter will provide some information on the different steps you should take to improve the speed and performance of a macro. Regardless of whether you are an IT administrator, end user or a developer, you can use these tips to your benefit.

Close Everything Except for the VBA Essentials

The first thing to do to improve the performance of VBA is to turn off all the unnecessary features like screen updating, animation, automatic events and calculations when the macro runs. All these features will always add an extra overhead, which will slow the macro down. This always happens when the macro needs to modify or change many cells and trigger a lot of recalculations or screen updates.

The code below will show you how you can enable or disable the following:

1. Animations
2. Screen updates
3. Manual Calculations

```
Option Explicit
        Dim lCalcSave As Long
        Dim bScreenUpdate As Boolean
        Sub SwitchOff(bSwitchOff As Boolean)
          Dim ws As Worksheet
          With Application
            If bSwitchOff Then
```

```
     ' OFF
      lCalcSave = .Calculation
    bScreenUpdate = .ScreenUpdating
      .Calculation = xlCalculationManual
      .ScreenUpdating = False
      .EnableAnimations = False
      '
      ' switch off display pagebreaks for all worksheets
      '
      For Each ws In ActiveWorkbook.Worksheets
        ws.DisplayPageBreaks = False
      Next ws
    Else
      ' ON
       If .Calculation <> lCalcSave And lCalcSave <> 0 Then .Calculation
= lCalcSave
      .ScreenUpdating = bScreenUpdate
      .EnableAnimations = True
   End If
  End With
End Sub
Sub Main()
  SwitchOff(True) ‘ turn off these features
  MyFunction() ‘ do your processing here
  SwitchOff(False) ‘ turn these features back on
```

```
End Sub
```

Disabling All The Animations Using System Settings

You can disable animations through the Ease of Access center in Windows. You can use this center to disable some specific features in Excel by going to the Ease of Access or Advanced Tabs on the menu. For more information, please use the following link: https://support.office.com/en-us/article/turn-off-office-animations-9ee5c4d2-d144-4fd2-b670-22cef9fa

Disabling Office Animations Using Registry Settings

You can always disable office animations on different computers by changing the appropriate registry key using a group policy setting.

HIVE: HKEY_CURRENT_USER

Key Path: Software\Microsoft\Office\16.0\Common\Graphics

Key Name: DisableAnimations

Value type: REG_DWORD

Value data: 0x00000001 (1)

If you use the Registry Editor incorrectly, you can cause some serious problems across the system. You may need to re-install Windows to use the editor correctly. Microsoft will help you solve the problems of a Registry Editor, but you should use this tool if you are willing to take the risk.

Removing Unnecessary Selects

Most people use the select method in the VBA code, but they add it in places where it is not necessary to use them. This keyword will trigger some cell events like conditional formatting and animations, which will hinder the performance of the macro. If you remove all the unnecessary selects, you can improve the performance of the macro. The following example will show you the code before and after you make a change to remove all the extra selects.

Before

```
Sheets("Order Details").Select

Columns("AC:AH").Select

Selection.ClearContents
```

After

```
Sheets("Order Details").Columns("AC:AH").ClearContents
```

Using the With Statement to Read Object Properties

When you work with objects, you should the With statement to decrease the number of times that the compiler reads the properties of the object. In the example below, see how the code changes when you use the With statement.

Before

```
Range("A1").Value = “Hello”

Range("A1").Font.Name = “Calibri”

Range("A1").Font.Bold = True

Range("A1").HorizontalAlignment = xlCenter
```

After

```
With Range("A1")

  .Value2 = “Hello”

  .HorizontalAlignment = xlCenter

    With .Font

      .Name = “Calibri”

      .Bold = True

    End With

End With
```

Using Arrays And Ranges

It is expensive to read and write to cells every time in Excel using VBA. You incur an overhead every time there is some movement of data between Excel and VBA. This means that you should always reduce the number of times the data moves between Excel and VBA. It is at such a time that ranges are useful. Instead of writing or reading the data individually to every cell within a loop, you can simply read the entire range into an array, and use that array in the loop.

The example below will show you how you can use a range to read and write the values at once without having to read each cell individually.

```
Dim vArray As Variant

Dim iRow As Integer

Dim iCol As Integer

Dim dValue As Double

vArray = Range("A1:C10000").Value2 ‘ read all the values at once from the Excel cells, put into an array

For iRow = LBound(vArray, 1) To UBound(vArray, 1)

  For iCol = LBound(vArray, 2) To UBound(vArray, 2)

    dValue = vArray (iRow, iCol)

    If dValue > 0 Then

      dValue=dValue*dValue ‘ Change the values in the array, not the cells

    vArray(iRow, iCol) = dValue

  End If

Next iCol

Next iRow

Range("A1:C10000").Value2 = vArray ‘ writes all the results back to the range at once
```

Use .Value2 Instead Of .Text or .Value

You can retrieve your values in different ways from a cell. The property you use to retrieve that information will have an impact on the performance of your code.

.Text

Most programmers use the .Text value to retrieve only the information from a cell. The property will return the formatted value of the cell. It takes a lot of processing time to retrieve a cell value along with its format, and it is for this reason that this property is slow.

.Value
The .Value keyword is a slight improvement over the previous keyword since it does not return a value with its format. Regardless of whether a cell has been formatted with a date or currency, this keyword will only return the VBA date and VBA currency, and the values for these outputs are truncated at decimal places.

.Value2
The .Value2 keyword only returns the underlying value of the cell. This keyword does not take any formatting into account and works faster than the .Text and .Value keywords. This keyword works faster if you use a variant array.

If you want to learn more about how these keywords work, please read the following post: https://fastexcel.wordpress.com/2011/11/30/text-vs-value-vs-value2-slow-text-and-how-to-avoid-it/

Avoid Using Copy and Paste
When you use the macro recorder to record any operations, including copy and paste that you perform in Excel, the code that the recorder writes will use these methods as default operations. It is always a good idea to avoid using the copy and paste operations, and use some in-built VBA functions to perform these operations. You can also use the in-built functions to copy formulae or formatting across a block of cells. The following example will give you an idea about how you should use the in-built VBA operations and functions as opposed to manual copy and paste operations.

Before

```
Range("A1").Select

Selection.Copy

Range("A2").Select

ActiveSheet.Paste
```

After

```
‘ Approach 1: copy everything (formulas, values and formatting
Range("A1").Copy Destination:=Range("A2")
‘ Approach 2: copy values only
Range("A2").Value2 = Range("A1").Value2
‘ Approach 3: copy formulas only
Range("A2").Formula = Range("A1").Formula
```

If you think that the code is still functioning slowly, you can use the following fix: https://support.microsoft.com/en-in/help/2817672/macro-takes-longer-than-expected-to-execute-many-in

Use The Option Explicit Keyword To Catch Undeclared Variables

Option Explicit is one of the many Module directives that you can use in VBA. This directive will instruct VBA about how it should treat a code within a specific module. If you use Option Explicit, you should ensure that all the variables in the code are declared. If there is any variable that is not declared, it will throw a compile error. This will help you catch any variables that have been named incorrectly. It will also help to improve the performance of the macro where variables are defined at different times. You can set this by typing “Option Explicit” at the top of every module you write. Alternatively, you can check the “Require Variable Declaration” in the VBA editor under “Tools -> Options.”

Chapter Four

Some Problems with Spreadsheets and How to Overcome Them

Most people use Excel to make a repository. This is because it is easy to make a list of small items for yourself or your colleagues in Excel. You may perhaps want to use some formulae to create something sophisticated. You may also want to use macros to automate the process of collecting and processing data. You can do this by typing an equal to sign in the cell before you write the formula. Excel will be your guide. There are some problems that everybody will face when it comes to using Excel, and that is its simplicity. You may start with a small project in Excel, and this project will grow until it becomes a daunting task. At this point, you may also face some issues with stability and speed, or some development problem that you cannot solve.

This chapter examines some of the common issues that people come across when they use spreadsheets, and also provides some solutions to tackle those problems. It will also tell you when you should switch to a database instead of sticking to Excel.

Multi-User Editing

When an Excel system begins to grow, you will quickly run into a problem where only one user can open the workbook at a time and make changes to it. Any other person who wants to open the workbook will be notified that someone already has the book open and that they can view the workbook as a read-only version or wait until the file is closed by the first user. Excel does promise to let you know when the first user has closed the file, but this is a hollow promise since Excel does not always check the status, and there are times when it may never give you an update. Even if it does give you an update, someone may already have opened the file before you.

You can get around this in the following ways:

1. You should use Excel Online. This application is a web-based and abridged version of Microsoft Excel.
2. Turn on the feature that will allow you to share the workbook.
3. Split the workbook into smaller workbooks. This will allow different

users to access different workbooks without causing any hindrances in the work.

Shared Workbooks

If you use Excel online, you can allow multiple users to edit the workbook at the same time. There is so much functionality that goes missing, which makes it a contender only for simple tasks. The shared workbook features in Excel will allow you to share the workbook between multiple users, but there are many restrictions. For instance, you cannot delete a group of cells or create a table in a shared workbook.

It is easy to walk around some restrictions, but for others, it is a matter of changing the structure of the entire workbook instead of using a workbook that has already been set up. These workarounds can, however, get in the way. As a result of this, it is impossible to use a workbook that is shared in the same way that you may use a single user workbook.

Any changes made in a shared workbook will be synchronized between the users every time the workbook is saved. These changes can be saved on a time schedule, meaning that a workbook can be saved or force saved every few minutes. The overhead of regular checking and savings every share user change is quite large. The size of the workbook can increase which will put a strain on your network thereby slowing down every other system.

A shared workbook is prone to corruption. Microsoft office knows that this is the problem, but there is nothing much you can do about the issue. The alternative to this situation is to use Excel online since you can have multiple users working on the same workbook. Not many users will switch to excel online until Microsoft will remove all the restrictions on a shared workbook, and extend a multi-authoring tool to the Excel offline application.

Linked Workbooks

If you want to overcome the issue of multi-user editing, you should try to split the data across multiple workbooks. It is likely that these workbooks must be linked so that any value entered in one can be used in another. The links between workbooks also help to separate data using a logical method instead of using separate worksheets in one workbook.

Unfortunately, these links lead to instability and frustration. This is because the links need to be absolute or relative. In the case of absolute links, you will need to include the full path resource workbook while in the case of relative links, you only need to include the difference between the destination and source paths. This may sound sensible until you come across the rules the Excel decides to employ on when you can use each type of link, and when you can change them.

These rules are governed by numerous options. Some of these rules are dependent on whether the workbook was saved and whether it was saved before every link was inserted. There are times when Excel will automatically change the link when you open a workbook and use the save as option to copy the file. Excel may also change the links when you simply save the workbook down. One of the main disadvantages of using this option is that the links can break easily, and it is difficult to recover all the broken links. This is also a time-consuming affair since you cannot use the files that are affected by the broken links.

The linked data will only be updated when all the underlying files are open unless you edit links and update values. It is because of this that you may need to open 3 or 4 workbooks to ensure that all the information is flowing through in the right order. If you made it changed it the value in the first workbook but open only 3rd workbook, you will not see any changes because this is a second workbook still does not have the updated values.

It is logical to create a change a data, but this will increase the likelihood that the data is incorrect or and when you open a workbook somebody else is already editing the underlying work. You can avoid the use of link workbooks, but there is a chance that you will end up entering the same data in more than one workbook. The danger with this is that you may type the data differently each time.

Data Validation

You must remember that any user can enter data on any computer system. People can transpose digits in numbers or mistype words with monotonous regularity. You must ensure that you check the data when it is entered or you will have a problem in the end.

Excel will always accept whatever any user types. Therefore, it is possible to set up a validation using lists, but it is impossible to maintain this list especially if that field is used in multiple places. For example, if a user should enter a customer reference number or a document ID they can enter the wrong record.

To avoid this, it is always good to have some checks across the workbook. If there is no Data integrity, the system will be fatally compromised, which will affect the analysis.

You may already be suffering from this problem without having realized what the root cause is. Let us consider a situation where there is a list of invoices that you have entered in Excel Find the user has typed the name of every customer differently on every invoice. You got invoices to John limited, John Ltd and John. You are aware that these invoices point to the same company or customer, but Excel is not aware of this. This means that any analysis that you made using this data will always give you multiple results when they should only be one.

Navigation Issues

It is difficult to navigate through large workbooks. The number of sheet tabs in the bottom of the window is difficult to use and is a terrible way to find your way around the workbook. If there are many sheets in the workbook, and you cannot see all of them on the screen, it will be difficult for you to find what you are looking for. You can always click on the arrow to the left of your active sheet, but you will only see the first twenty sheets in that window. You cannot sort or group the list of sheets in any order.

Security Issues

You can add a lot of security features to an Excel workbook, but it is still going to have many problems. It is more important to work toward protecting the structure of the workbook, instead of worrying about the data. You can always lock some sheets and cells in the workbook to prevent some users from making any changes to the data or formulae. Regardless of whether you protect the sheet or not, if someone can see the data, they can make changes to it. You can avoid this by using some clever macro skills.

Speed Issues

You must remember that Excel is not the fastest application there is, and the programming language we use in Excel, VBA is slow and slightly sluggish when compared to the more professional languages like C and C#. This is because of the intended use of Excel and its flexibility. You should remember that Excel is a spreadsheet engine alone, and it can only be used to manage large volumes of data. This does not mean that you must always use Excel for this type of work.

There are many other applications that you can use to perform such tasks since those applications were designed to perform these functions.

Enter the database

If you are facing any of the issues that have been listed above, you should not ignore them. The answer or solution to these problems is to store the data in a structured manner. This means that we will need to start saving data in a database. This will allow you to think about your data in a logical manner. You have the ability to see how the data welding together and how you will need to interact with it to analyze the information.

You must, however, take heed. If you move from spreadsheets to databases, you should not duplicate the design of a spreadsheet. Instead, you should find a way to make the design better. There are some general database applications, listed below with which you can construct a simple solution. Alternatively, you can also use specialist database applications that allow you to switch from spreadsheet to databases within a few minutes point these applications are a better fit to big data.

For example, if you have a list of customers, their details and any interaction you have had with these customers you should consider using a customer relationship management system. Customer relationship management system is a specialized database. Similarly, you can save accounts on packages like Sage and QuickBooks. The may be times when you cannot find an existing application to suit your needs. As such times you may need to build a database by yourself or request see IT department or any consultant to build the database for you.

The relational database is the most common type of database used in today's world. This database stores information or data in the form of tables, which consists of columns and rows of data. Every row data will hold a separate item and every column will describe a different attribute of that item. For example, if the rows hold customer information, the columns can describe attributes like customer name and customer ID. All you need to do is enter the data once, and then you can use the same data to print on every invoice.

Every table in a relational database has a relationship between them. You can take the relationship between an invoice and the customer ID. Here you can always find an invoice that is related to a specific customer using the customer ID. Alternatively, you can also retrieve customer information from the invoice if necessary. All you need to do is enter the customer data of one in the database to create a record, and you can use that information across different invoices without having to type the data again. To use or create a database, you must define the tables and the relationships between those tables, and then define the type of layout you want to use to edit or list the data.

There are over a dozen applications that you can choose from. Some of the applications are easy to use and do the job for you. These applications will allow you to define the table, the data screen, and the reports. There are other applications that are more useful in specific areas but will require other tools to perform the job.

For example, some applications may be very powerful when comes defining a table and the relationship that table shares with the database and other tables, and it may also have some excellent analysis and reporting features. This application can, however, lack a tool, which will allow you to define the data entry screen. An obvious example of such an application is Microsoft SQL. As is the case with large database systems, the SQL server will only take care of the back-end annual expect you to use, and other tools like visual studio to develop or maintain the front-end.

Choosing the Right Database

Access

Microsoft Access is one of the oldest databases available. This is easy to use and is extremely easy to abuse. You can design screens, reports, and tables from scratch or use an existing template. Some of the templates in Access do not teach you some good practices, but they will help you get started quickly. The programming and screen features and options are sophisticated, and you can deploy the application on the intranet without having to rely on sharing the files with users.

SharePoint

SharePoint is a document storage application and a database. This application can be used to compile and link simple lists. You can use the form designer to customize your dashboard, but it is important to remember that it is not a sophisticated application to use. SharePoint has the ability to suck the information from Excel and put it into a custom list. This makes it a useful application since everybody in your network will have access to the list. You can choose to add some security features, which will restrict the access for some people. SharePoint can also send you an alert email when someone makes a change – adds, deletes or edits – to a record. You can also synchronize the information with Outlook if you have some data that concerns a person, calendar or task.

Zoho Creator

There is a database application that you can use in the Zoho office services available on the internet. You can drag and drop the required layout in an easy way. This will also help you decide how the work should flow and what the interaction can be like. Since this is a web application, the data you use and the applications you develop can be found anywhere. Therefore, you should use the simple security features that this application provides to keep your data private. Zoho charges you per month but will allow you to store only some records depending on the price you choose to pay. If you want to use advanced features like email integration, you will need to pay an additional amount of money.

Chapter Five

Sub Procedures

If you have read Excel tutorials that talk about VBA and macros, you would have come across the term procedure at least a hundred times. If you are unsure of what these are, you should learn this now. There are many good reasons why this is important.

If you want to become an expert at writing macros and using VBA, you should understand what a procedure is, the different types of procedures and how you should work with them. This is one of the most essential tools to learn if you want to become a VBA expert. This chapter will provide all the information you need about procedures and will dig deeper into the concepts of sub procedures. Let us first begin with an introduction to sub procedures.

What Is A Sub Procedure?

If you have written programs or code in VBA, you will know that a procedure is a block of code or statements that are enclosed between a declaration statement and an End statement. The purpose of the procedure is to perform a specific action or task. All the instructions that you want to give the compiler are within a procedure. If you want to master coding in VBA, you should fully understand this concept. There are two types of procedures in VBA – Function procedures and sub procedures. This chapter will focus only on Sub procedures.

The following are the differences between a VBA sub procedure and a function procedure:

1. A VBA sub procedure will perform some function or action with Excel. This means that when you execute a sub procedure, Excel will do something. The changes or functions that happen in Excel depend on what the code says.

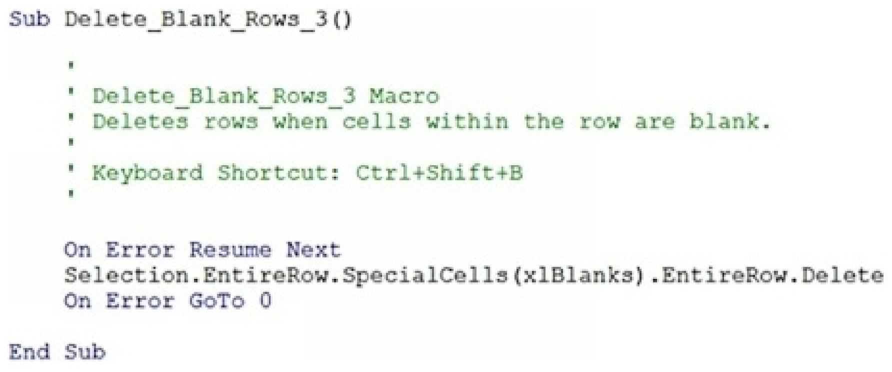
2. A function procedure will perform some calculations and return a value. The value returned can either be an array, number or string. If you have worked regularly with functions in Excel, you already know how the function procedure will work. This is because they work in the same way as Excel functions. These procedures will perform some function on the data in Excel and return a value.

Experts say that most macros that people write are sub procedures. If you always use the macro recorder to create your macro, you are creating a sub procedure. From the above comments, it is clear that you will work a lot with Excel VBA Sub procedures.

How Does The VBA Sub Procedure Look?

The image below will show you how a VBA sub procedure looks. You should notice that this procedure has the following features:

1. It begins with the statement “Sub.” This is the declaration statement.
2. There is an End declaration statement.
3. There is a block of code that is enclosed between the declaration statements.



```
Sub Delete_Blank_Rows_3()

    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

The purpose of this VBA sub procedure is to delete some rows in the worksheet where there are blank cells. Before we move on, let us take a look at the first statement in the sub procedure. There are three sections to look at:

1. The sub keyword, also the declaration statement, which tells the compiler that the sub procedure has started.
2. The name of the sub procedure. We will cover the rules that must be followed when working with sub procedures in the following sections.
3. The parentheses in the sub procedure are where you will need to add the arguments that you will be using from other procedures. You should separate these by using a comma. You can always create a VBA sub procedure that does not use any arguments. You should, however, include an empty set of parentheses when you name the sub procedure.

The following are the four elements that you should include in a VBA sub procedure:

- Sub statement
- Name
- Parentheses
- End Sub keyword

You can include two optional elements in a sub procedure:

1. A list of arguments that you will need to include in the parentheses.
2. Valid instructions that are included in the declaration statements in the code.

Apart from these, you can include three optional items in a VBA sub procedure. These items are optional, but they are important to consider. Before we look at them, let us look at how this procedure will be structured.

```
[Private/Public] [Static] Sub name ([Argument list])

[Instructions]

[Exit Sub]

[Instructions]

End Sub
```

To learn more about the Sub statement, you should read the articles found in the Microsoft Dev Center. Let us take a look at the optional elements that are present in the above structure. These items are written in square brackets to indicate that they are optional.

Element #1: [Private/Public]

The keywords private and public are called access modifiers. If you type private before the declaration statement in a sub procedure, it implies that only the procedures or codes written within the same module can access that sub procedure.

If you choose to use the keyword public, the sub procedure will not have any access restrictions. Despite this, if you were to use the keywords option private statement at the beginning of the sub procedure, any procedure outside the relevant project cannot refer to or use the sub procedure. We will talk a little more about the scope of the project in the sections below.

Element #2: [Static]

If you use the keyword static at the beginning of the sub procedure, any variable, which is the part of the sub procedure will be preserved even when the module ends.

Element #3: [Exit Sub]

The exit sub statement is the final declaration statement, which is used to immediately exit A sub procedure. This means that any statements within the sub procedure not run once the compiler reaches this decoration statement.

How to Name A VBA Sub Procedure

You must always name a procedure. The rules that you need to follow when naming A sub procedure are given below:

1. You should always use a letter as the first character.
2. The remaining characters in the name can be numbers, letters or some punctuation characters. For example, the following characters can't be used: #, $, %, &, @, ^, * and !
3. You should avoid using spaces and periods.
4. Since VBA is not a case sensitive language, cannot distinguish between lowercase and uppercase letters.
5. Any sub procedure can have a name, which has a maximum of 255 characters.

Experts suggest that we be a procedure name should always:

6. Describe what the purpose of the sub procedure is or what it is supposed to do.
7. Have some meaning.
8. Be a combination of a noun and a verb.

There are some programmers who choose useful sentences to name the sub procedures. There is one advantage and disadvantage of doing this:

1. A sentence will definitely let any other user of programmer know what the purpose of the sub procedure is. This is because a sentence is very unambiguous and descriptive.
2. When you type of full sentence you will use more time. This means that you will take longer to finish your macro.

I believe that you should always use a name that is descriptive, unambiguous and meaningful. It is at this point that you should choose what your style is, and always stick to something that is comfortable and will help you achieve your goals.

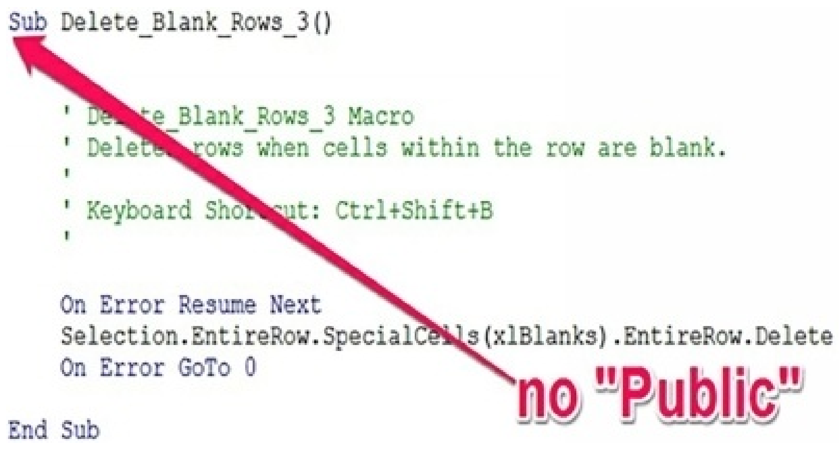
How to Determine the Scope of A VBA Sub Procedure

The scope will define how you are when you should call upon a VBA sub procedure. When you create a BBA procedure you have the option to determine which other procedure can call it. You can do this by using the keywords public and private which were introduced in the above section as an optional element of a procedure.

Let us now take a look at what the meaning of these keywords is and how you can determine whether a specific VBA procedure is public or private.

Public VBA Sub Procedures

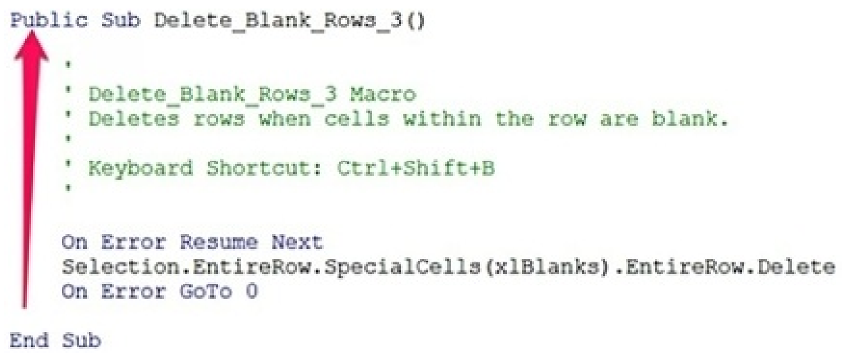
Every sub procedure is public by default. If a specific procedure is public there is no access restriction. Since the default option is that a procedure is public, you do not have to include the public keyword at the beginning of the name. For instance, the following procedure delete_blank_rows_3 is a public procedure although we did not use the keyword public.



```
Sub Delete_Blank_Rows_3()

    ' Delete_Blank_Rows_3 Macro
    ' Delete rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

If you want to make the courts near, you should try to include the public keyword in the procedure. Most programmers do follow this practice. In the image below, you will see that the keyword public has been included in the delete_blank_rows_3 macro.



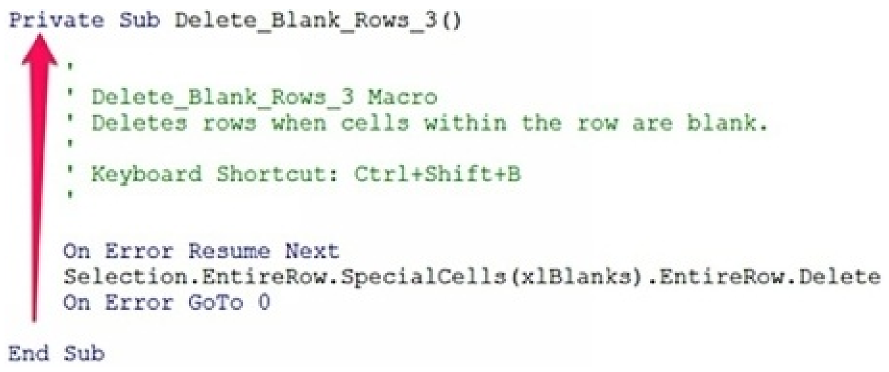
```
Public Sub Delete_Blank_Rows_3()
    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

In both cases, the sub procedure is public. In simple words both the macros, with and without the public keyword are the same.

Private VBA Sub Procedures

When you use the private keyword ahead of the sub procedure, the content of the statements within the sub procedure can only be accessed by other procedures within the same module. If there is any other procedure or module that wants to access this sir procedure it cannot, even if the module is in the same Excel workbook. For instance, if you need the delete_blank_rows_3 macro a private macro you will need to follow the syntax given in the image below.



```
Private Sub Delete_Blank_Rows_3()
    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

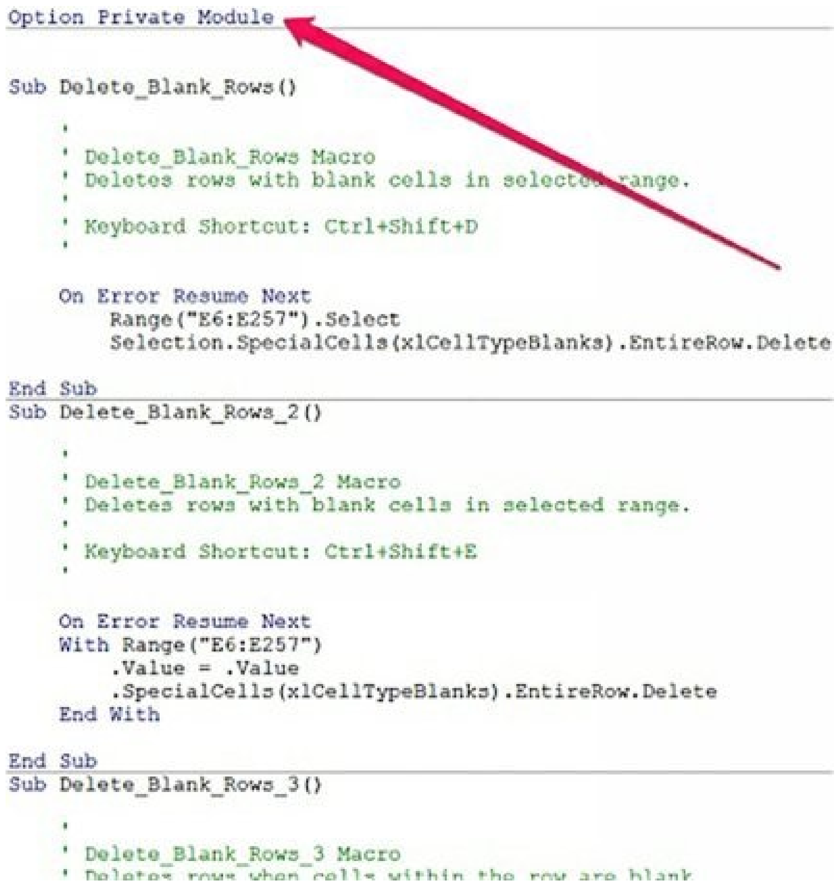
How to Make All VBA Sub Procedures in a Module Private to A VBA

Project

A person using the public in the private keywords, you can also make a sub procedure accessible to other modules in different BBA project by using the option private statement. To use the option private statement, you must include the keywords “option private module” before the sub procedure. If you are certain that you want to use this statement ensure that the keywords are ahead of the declaration statement in the sub procedure.

The image below shows how VBA uses three different methods to delete a row based on whether the row has an empty cell or not. The third macro or sub procedure in the image below is the Delete_Blank_Rows_3 macro. This macro does not appear fully in the image.

Found this title interesting or useful? Then a review on Amazon will be highly appreciated!



```
Option Private Module

Sub Delete_Blank_Rows()

    '
    ' Delete_Blank_Rows Macro
    ' Deletes rows with blank cells in selected range.
    '
    ' Keyboard Shortcut: Ctrl+Shift+D
    '

    On Error Resume Next
        Range("E6:E257").Select
        Selection.SpecialCells(xlCellTypeBlanks).EntireRow.Delete

End Sub
Sub Delete_Blank_Rows_2()

    '
    ' Delete_Blank_Rows_2 Macro
    ' Deletes rows with blank cells in selected range.
    '
    ' Keyboard Shortcut: Ctrl+Shift+E
    '

    On Error Resume Next
    With Range("E6:E257")
        .Value = .Value
        .SpecialCells(xlCellTypeBlanks).EntireRow.Delete
    End With

End Sub
Sub Delete_Blank_Rows_3()

    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank
```

(https://powerspreadsheets.com/vba-sub-procedures/)

All the sub procedure is in the above image can only be accessed referenced by module or the procedures in the VBA project that contains them.

When to Make VBA Sub Procedures Private: An Example

You can always execute A sub procedure by using another procedure to call it. Most programmers use this method to run or execute a procedure. In some cases, you may also have procedures that are designed to be called by other sub procedures. If you have any procedures in a specific workbook, it is always good to make them private. When you do this the sub procedure will no longer be listed in the macro dialog box. The macro dialog box is one of the easiest and fastest ways to execute A sub procedure.

If you do not understand how this works, do not worry. I will explain how you can call VBA sub procedure from other procedure or by using the macro dialogue box in the section below.

How to Execute / Run / Call a VBA Sub Procedure

When you work with macros you will often use the terms run execute or call interchangeably. These words refer to the action of executing the statements in the sub procedure. You can use these words interchangeably.

You can execute run a call A sub procedure in many ways. This section provides 9 different ways in which you can execute or run the statements in a procedure. There is a 10th option, which you can use but this is out of the scope of this book. This in this option you will need to execute the macro or the statement using a customized context menu. This section does not talk about using a context menu customization to run a block of code since it is a separate topic that we need to be covered extensively.

We will use the delete_blank_rows_3 macro as an example in all the options in this section.

Option One: How to Execute A VBA Sub Procedure Directly From the Visual Basic Editor

Experts state that this is the best and the fastest way to execute the block of code in a sub procedure. In this method, you will run the procedure directly in the visual basic editor using the module in which it is written.

Describe the factor this is one of the fastest methods, most people do not use it often. In practice, people often execute the macro only when they are in excel and not in the visual basic editor. There are some other options that are listed in this section, which will allow you to do this.

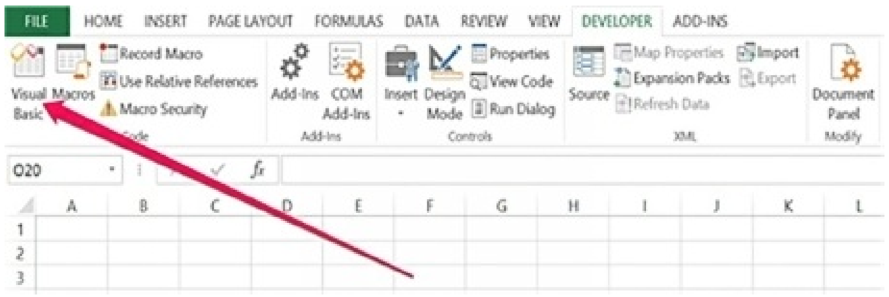
This method will only work when a specific sub procedure that you want to run does not require any arguments from other procedures and macros. The reason is that this option does not allow you to use any arguments are inputs from other procedures.

If you ever want to run a sub procedure, which contains arguments, you can only do it by calling it from another procedure. The procedure used to call the sub procedure will need to pass the arguments that the sub procedure required to execute the block of code.

If you do choose to use this method to call or run the code in the sub procedure, you should follow the following steps:

Step One: Open the Visual Basic Editor

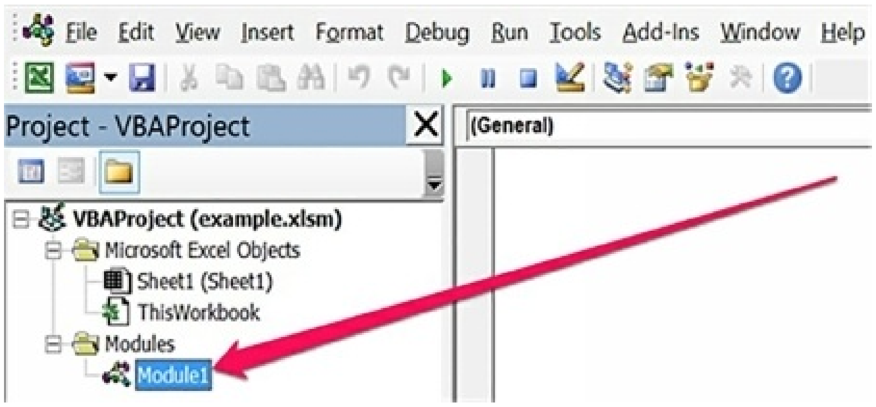
You can open the visual basic editor using the keyboard shortcut Alt + F11. Alternatively, you can go to the developer tab in the ribbon and choose the visual basic icon.



(https://powerspreadsheets.com/vba-sub-procedures/)

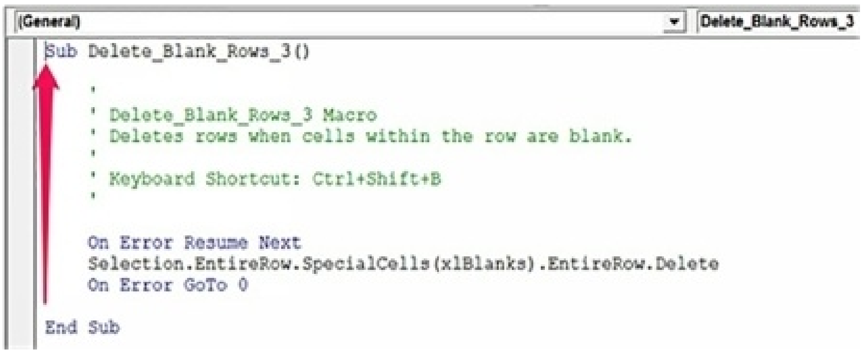
Step Two: Open The VBA Module That Contains The VBA Sub Procedure You Want To Execute.

You now want the visual basic editor to give you the code that is in the sub procedure that you are calling. This can be done in several ways. One of the easiest methods is to double click on the relevant module or procedure. For instance, if the sub procedure that you want to call is within module 1, you will simply need to click module 1 in the project explorer in the visual basic editor.



(https://powerspreadsheets.com/vba-sub-procedures/)

As a result of this, the visual basic editor will display the relevant code in the programming window in the visual basic environment.



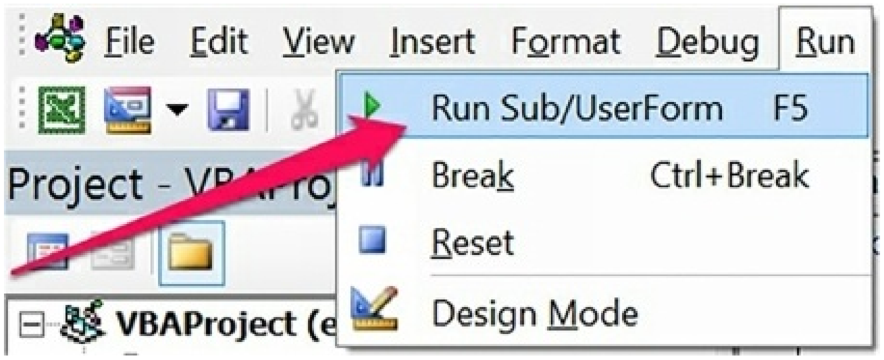
```
Sub Delete_Blank_Rows_3()

    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

Step Three: Run The VBA Sub Procedure.

If you want to call A sub procedure directly using the relevant module in the visual basic editor, you must use the following methods:

1. Go to the run menu and click on the option “Run Sub/UserForm.”



(https://powerspreadsheets.com/vba-sub-procedures/)

2. Click on F5, which is the keyboard shortcut.

Option Two: How to Execute A VBA Sub Procedure Using the Macro Dialog

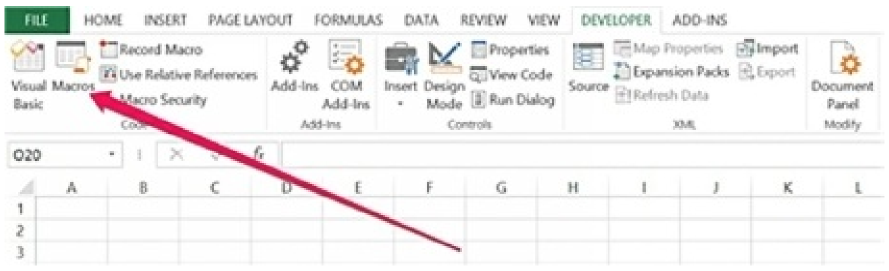
This method, as the first method, will only work when the sub procedure does not require any arguments. It is because of the same reason mentioned earlier - you cannot specify the arguments.

Regardless of what the arguments are, this is an option that most programmers used to execute sub procedures. When you use this method you can run the sub procedure in two steps. Let us look at them.

Step One: Open the Macro dialog.

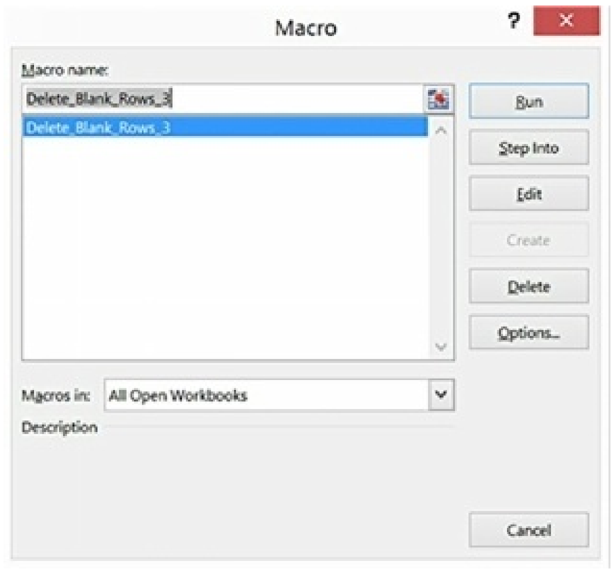
You should first instruct Excel to open the macro dialog box using the following methods:

1. Click on Alt + F8, which is the shortcut key.
2. Go to the developer tab in the ribbon, and click on the macros option.



(https://powerspreadsheets.com/vba-sub-procedures/)

Excel window display the macro dialog box which will look as follows:



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Two: Select The Macro You Want To Execute And Execute It.

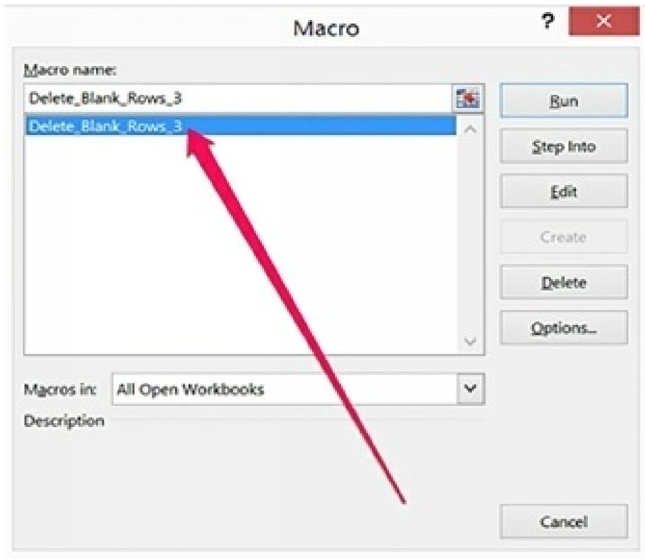
In the above image, you will notice that there is only one macro open in every Excel workbook. This macro is the Delete_Blank_Rows_3 macro. Since this is the only method that is listed, we will only be running or executing the code within that macro.

You are already aware that when you use this method you cannot use any arguments in the sub procedure. Therefore any sab procedure that requires arguments will not appear in the macro dialog box.

It is also important to remember that the macro dialog box will only show procedures that are public. You can still execute A sub procedure, which is private. For this, you should fill the relevant sub procedure name in the macro name field, which appears in the image below. The macro dialogue box does not show any sub procedure, which contains and adding. In this case, you can execute a macro by typing in the relevant macro in the name field.

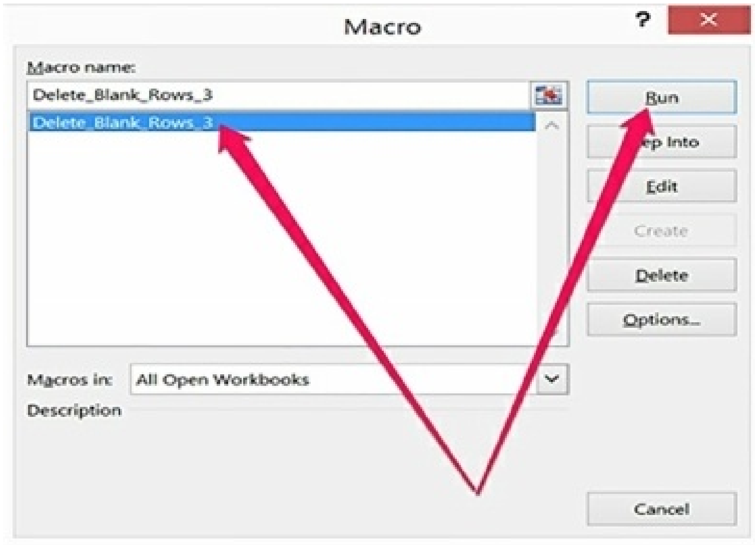
The rule to select and run or execute macro is the same regardless of whether you have one or multiple macros in the open or active Excel workbook. You can always select the matter that you want to run in the following ways:

Double click the name of the matter that you want to execute. For example, we want to run the Delete_Blank_Rows_3 macro. For this, we will need to double click on the name of the macro.



(https://powerspreadsheets.com/vba-sub-procedures/)

Click on the name of the macro you want to run, and hit run button on the top right corner.



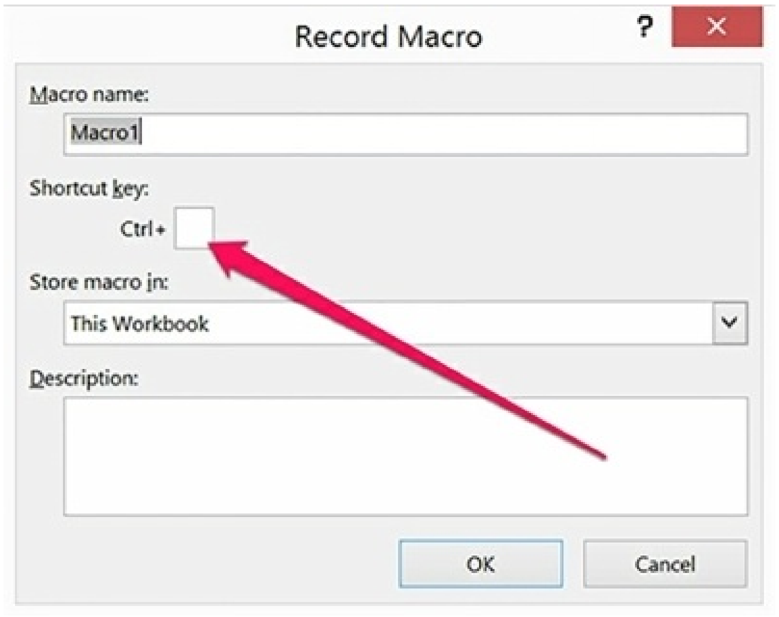
(https://powerspreadsheets.com/vba-sub-procedures/)

Option Three: How to Execute A VBA Sub Procedure Using a Keyboard Shortcut

You can also execute or run a sub procedure using keyboard shortcuts. If you want to run a sub procedure using this method you have to select a press the relevant key combination. It is important to remember that this does not work for macros that require arguments for the same reason as mentioned above.

You may now be wondering how you can assign keyboard shortcuts to a macro. This can be done in two ways:

In this method, you will need to assign a keyboard shortcut to the macro when you are in the macro recorder. When you use the macro recorder to record the process, you will encounter a record macro dialog box. In this dialog box, you can determine whether you want to call a macro using a keyboard shortcut and also determine which keys will compose that shortcut.

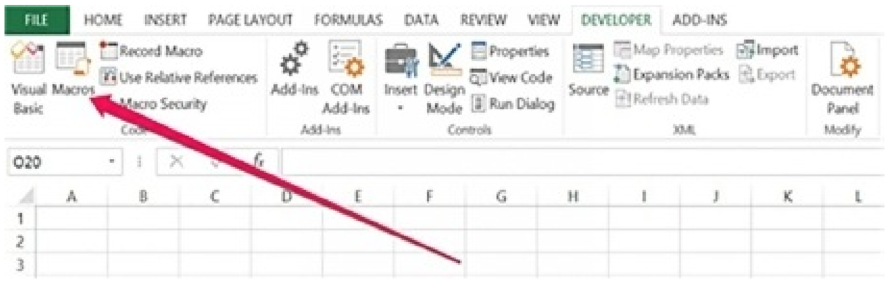


(https://powerspreadsheets.com/vba-sub-procedures/)

This method is more interesting when compared to the previous method. In this method you can assign or edit a keyboard shortcut to any macro in the following method:

Step One: Open The Macro Dialog.

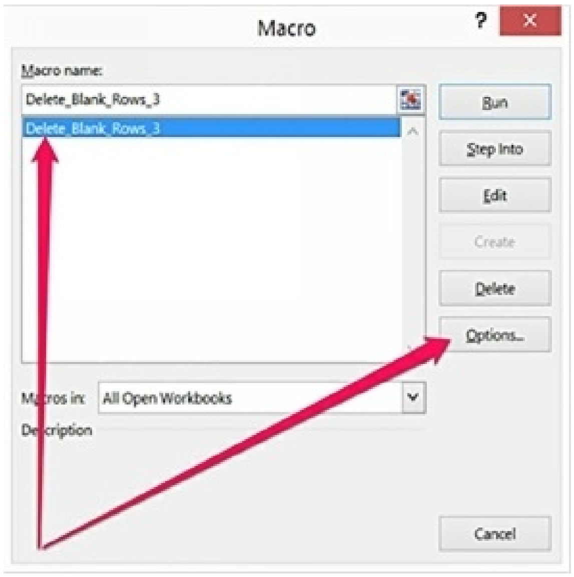
You can use the shortcut Alt + F8 to access the microbe dialogue box. Alternatively, you can go to the developer tab in the ribbon and click on the icon of the macro.



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Two: Select The Macro You Want To Assign A Keyboard Shortcut To.

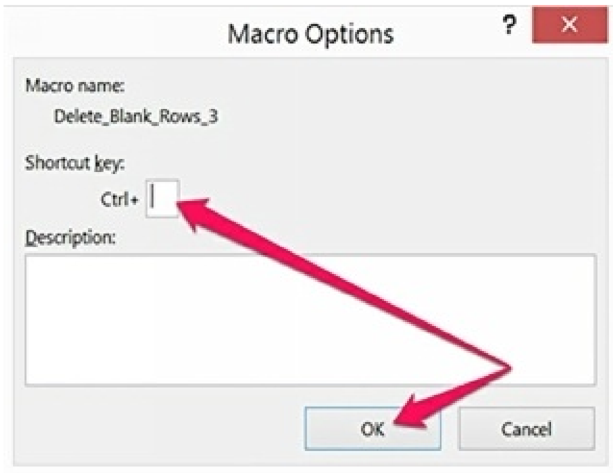
Now you should select the sub procedure you want to assign a macro to all shortcuts to and click on the options button on the bottom right corner of the dialog box. For example, we have selected the Delete_Blank_Rows_3 macro in the image below.



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Three: Assign A Keyboard Shortcut.

When you see the macro option dialog box open, you can assign a keyboard shortcut and click the ok button at the bottom of the box.



(https://powerspreadsheets.com/vba-sub-procedures/)

It is important to remember that keyboard shortcuts will always take the form of control + letter or control + shift + letter.

When you select the keyboard shortcut, you have to be careful that you are not assigning a combination all shortcuts that already exist in VBA. If you choose as an existing of built-in shortcut, you will be disabling the latter.

For instance, the control + B shortcut is a built-in shortcut for bold. If you assign the same shortcuts to any other Microsoft procedure, you cannot use it to make text bold.

It is always a better idea to use the control + shift + letter form of a shortcut since it reduces the risk of disabling a pre-existing shortcut. Regardless of what the situation is, you have to be careful about what combination you assign a sub procedure.

Option Four: How to Execute A VBA Sub Procedure Using a Button or Other Object

The idea behind using this method is that you can always attach a macro to a specific object. Here I am not referring to a specific object in the macro, but I am referring to the type of object that Excel will allow you to use in a worksheet. Experts have classified these objects into the following classes:

1. ActiveX controls
2. Form controls

3. Inserted objects, like as shapes, text boxes, clip art, SmartArt, WordArt, charts and pictures.

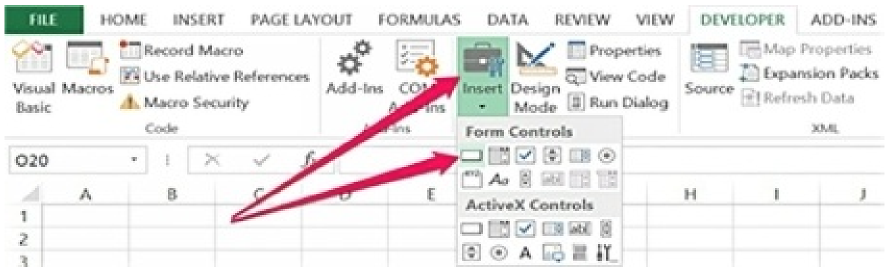
In this section, we will see how you can attach a macro to a button using the form controls option or to any other inserted object in the workbook.

How to Assign a Macro to a Form Control Button

You can attach any macro sub procedure to a form control button using the following four steps:

Step One: Insert a Button

You should first go to the ribbon and navigate to the developer tab. now move to the insert and choose the button form of control. The image below will show you exactly what needs to be done in the step.

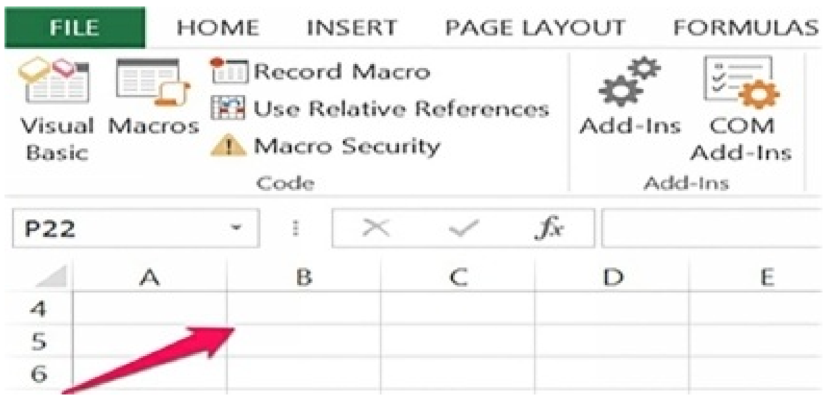


(https://powerspreadsheets.com/vba-sub-procedures/)

Step Two: Create the Button

Now that you have created the button form control, you will need to create the button in the Excel worksheet point you can create the Spartan of places button in any section of the worksheet where you want it to appear.

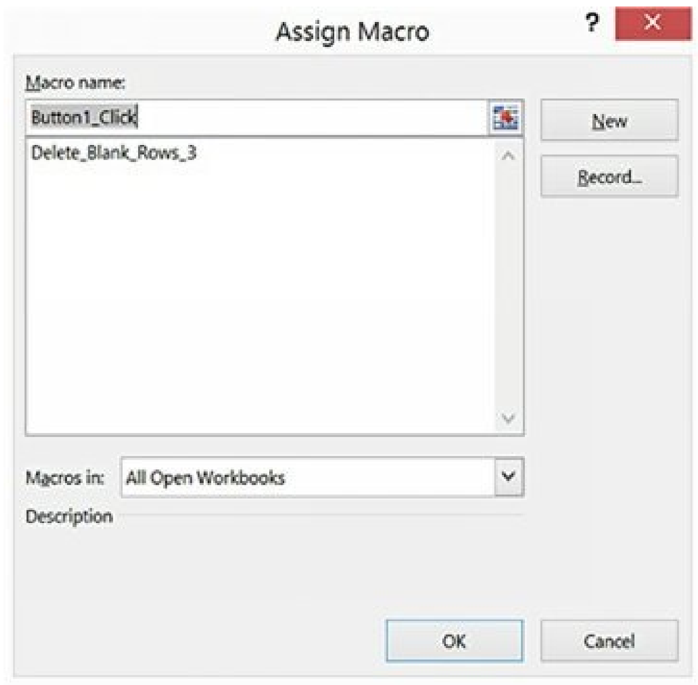
For example, if I want the button to be in Cell B5, I will click on the top left corner of that cell.



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Three: Assign a Macro to the Button

Once you have selected the location where you want the button to be, Excel will display the assign macro dialog box.



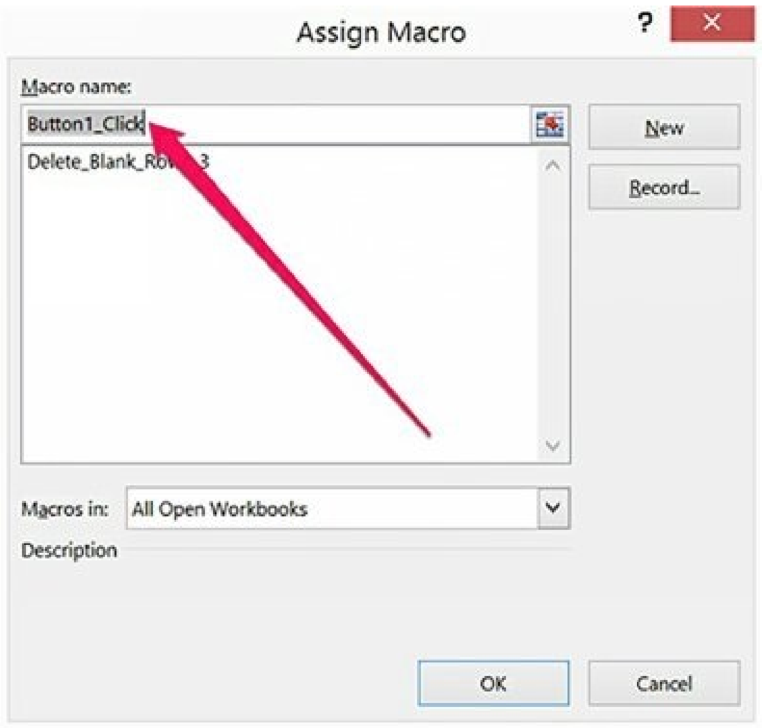
Macro name: Button1_Click

Delete_Blank_Rows_3

Macros in: All Open Workbooks

(https://powerspreadsheets.com/vba-sub-procedures/)

Based on what the buttons name is, excel will suggest a macro that you can assign to that button. In the example below, we have named the button “Button1_Click.”

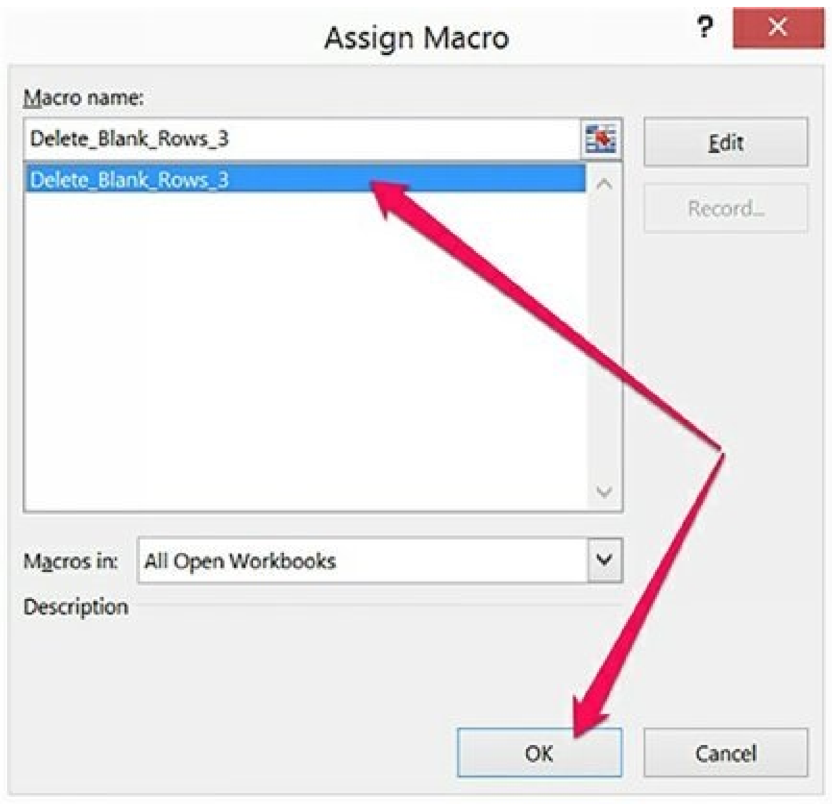


Button1_Click

Delete_Blank_Rows_3

(https://powerspreadsheets.com/vba-sub-procedures/)

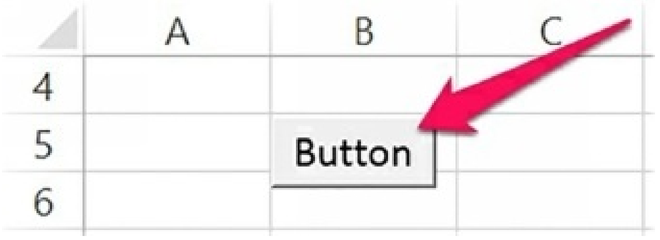
In most cases, the suggestion excel gives will not match what you want. Therefore, you have to select a method that you want to assign to the button and then click on ok at the bottom right corner of the dialog box. In the example below, we will be using the Delete_Blank_Rows_3 macro and assign a button to that macro.



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Four: Edit Button (Optional)

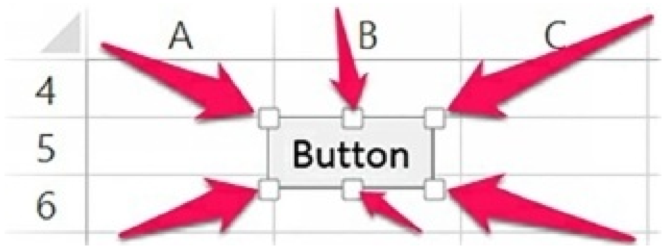
When you have completed the steps described above, Excel will create the button. Now, you only need to execute the relevant sub procedure by clicking on that button.

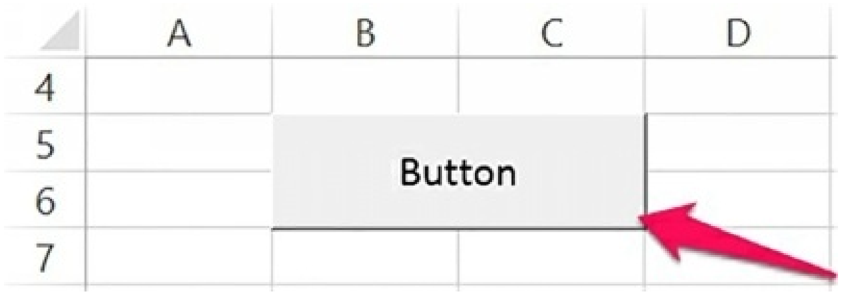


(https://powerspreadsheets.com/vba-sub-procedures/)

Once the button is in place, you can edit it in some ways. You the following are the four main aspects of the button that you can change:

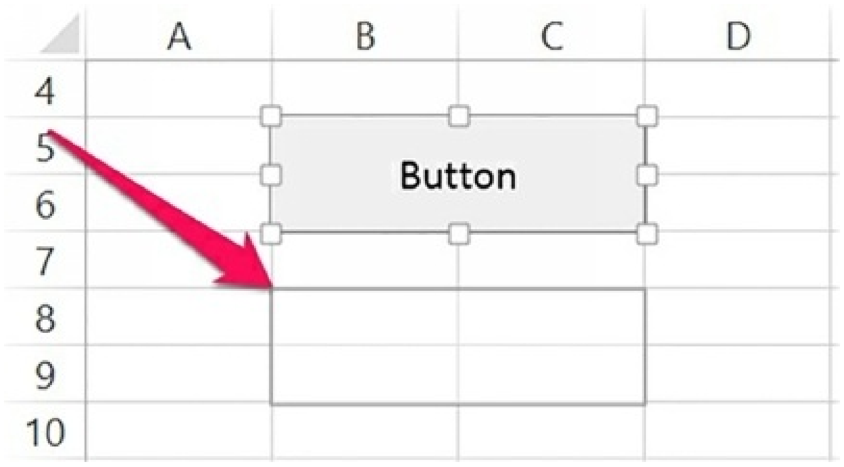
Size: Every button that you create an Excel has a default size. You can always adjust the size by joining the handles of the button with your mouse. For example, if you want to increase the size of a button so that it covers at least four cells, you can drag the bottom right handle as required. If you cannot find the handles, you can right click on the button or press the left mouse button at the same time as the control key to view the handles.

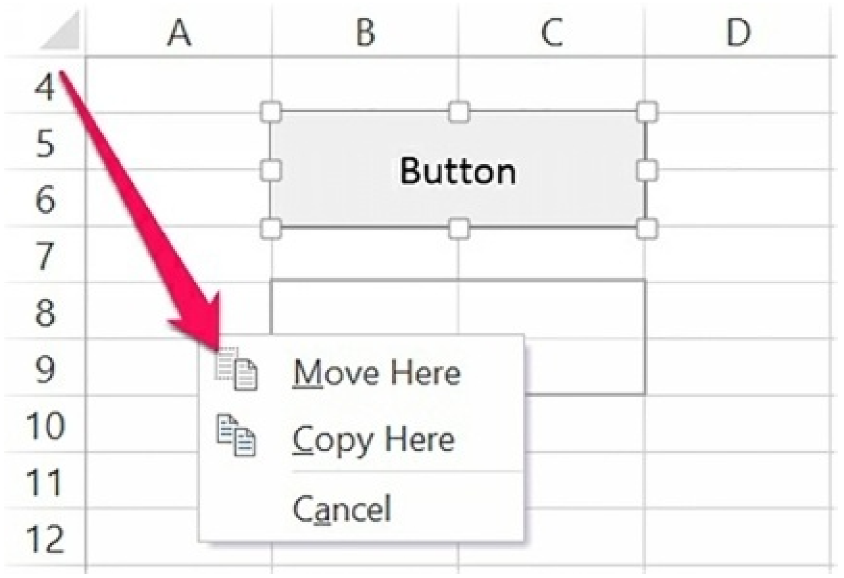




(https://powerspreadsheets.com/vba-sub-procedures/)

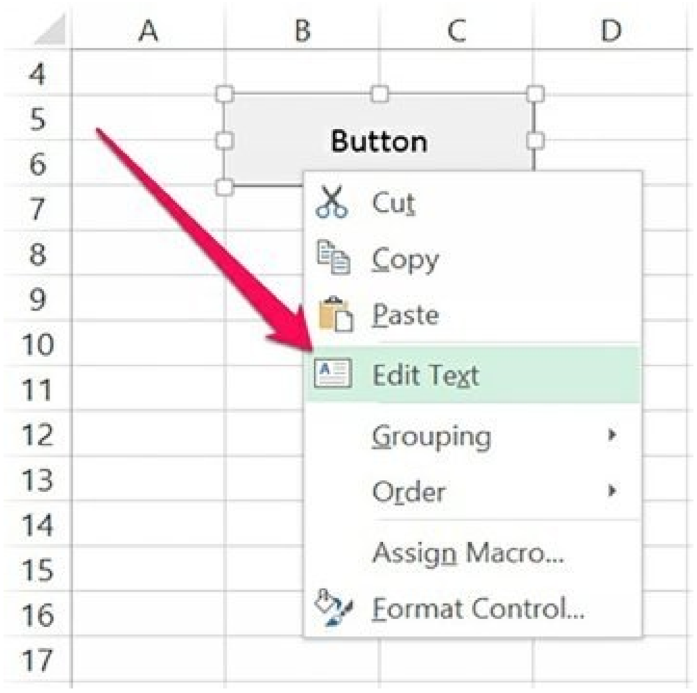
Location: You can always modify the location of a button by dragging it with your mouth. You can only track the button using the left button of your mouth is the handles are visible. If they are not visible, you will need to right click on the button and then attempt to drag it to a different location. Alternatively, you can simply change the position of the button by dragging it using the right mouse button. For example, if you want to move the button to cover a couple of cells down, that is it should cover the cells B8, B9, C8 and C9, you should drag the button until the desired point. When you drive the button excel will show you a Shadow in the new location but will leave the button and its original spot. When you let go or remove your hand from the right now button, accessories display the contextual menu. You should choose the option to move the button by selecting “move here.”

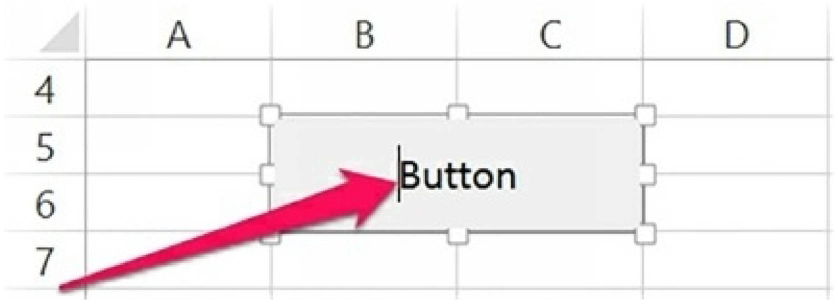


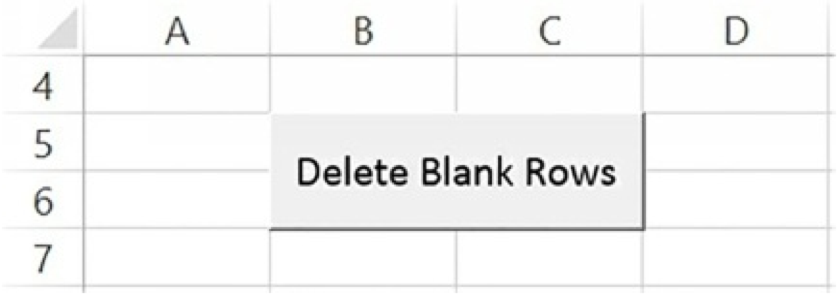


(https://powerspreadsheets.com/vba-sub-procedures/)

Text: If you want to edit the text on the button, you should I click on the button. Excel will then display a contextual menu where you can choose to edit text. Excel word and place the cursor inside the button, so you can modify the text. When you are done click outside of the button to confirm the changes made. Since we are using the Delete_Blank_Rows_3 macro as an example, we will rename the button to delete blank rows in that is more appropriate.

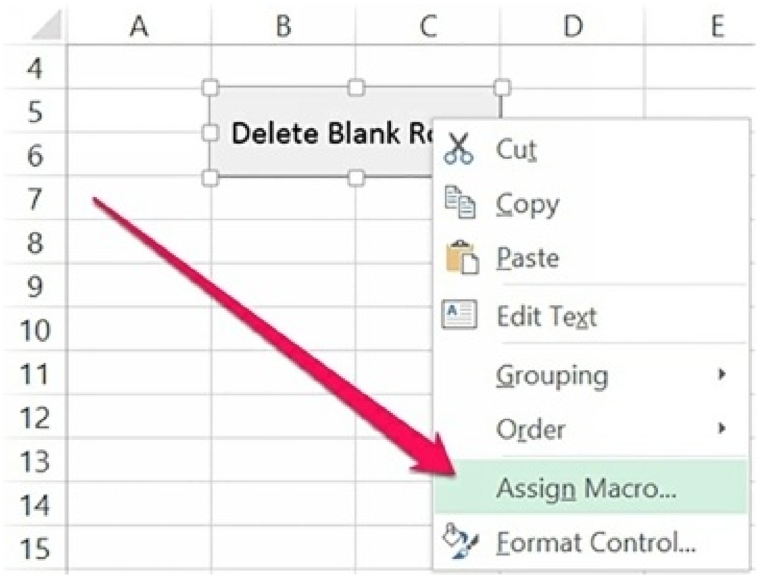


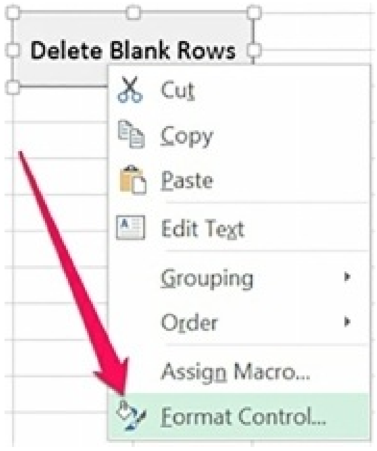


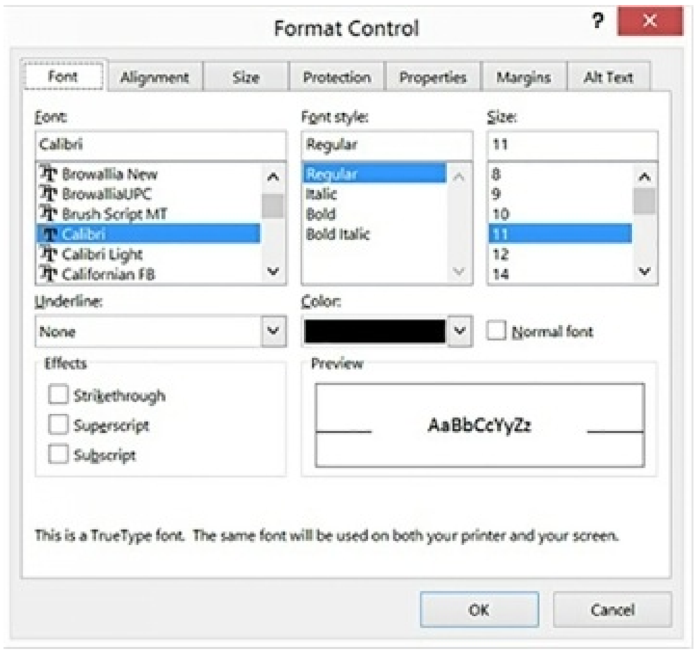


(https://powerspreadsheets.com/vba-sub-procedures/)

Assigned macro: If there is a necessity, you can always change the VBA sub procedure, and assign it to a different macro by right-clicking on the button. In this case, Excel will take you back to the macro dialogue box where you can select which sub procedure you want to assign to the button. You are already familiar with this process since it has been described above. In addition to this, you can also edit many other aspects of the button by right-clicking on the button and choosing the option to format control. Excel will now display the format control dialogue box. By using the options in this box, you can determine or make changes to many settings of the macro button.







Font: Calibri

Font style: Regular

Size: 11

Underline: None

Color:

This is a TrueType font. The same font will be used on both your printer and your screen.

(https://powerspreadsheets.com/vba-sub-procedures/)

Some of the settings that you can change during the format control dialog box are:

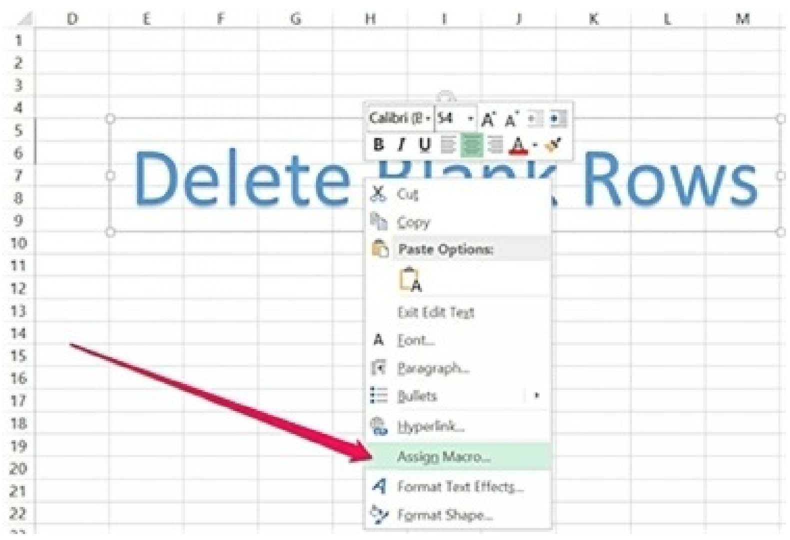
1. Font, including typeface, style, size, underline, color and effects
2. Text alignment and orientation
3. Internal margins
4. Size
5. Whether the button moves and/or sizes with cells

How to Assign a Macro to another Object

In addition to assigning macros to form control buttons, x I will also allow you to assign a macro to other objects. As explain the bug these objects can include text boxes, shapes, SmartArt, WordArt, text boxes, pictures or charts. It is extremely easy to attach a sub procedure to an object in excel. Let us see how you can do this in the case of a word art object which reads delete blank rows.

Step One: Open the Assign Macro Dialog

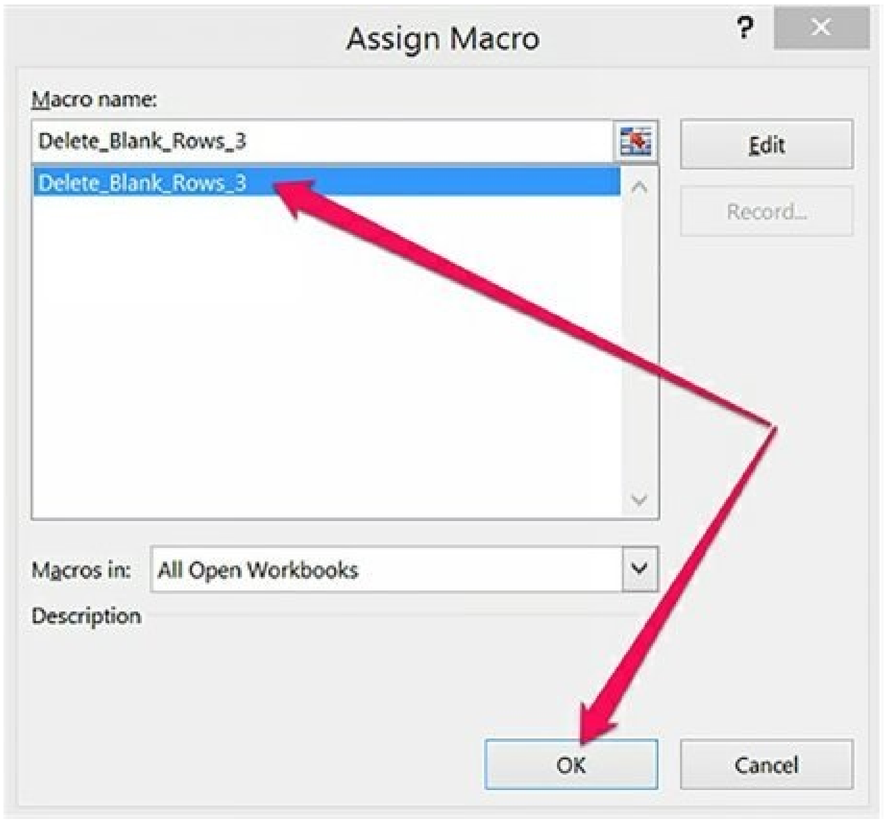
Right click on the object, and select the assign macro option. This will open the assigned macro dialog box.



(https://powerspreadsheets.com/vba-sub-procedures/)

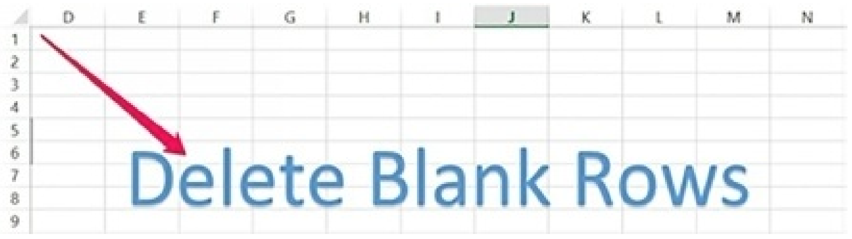
Step Two: Select Macro to Assign

Once you complete the step above, Excel will display the macro dialogue box. You are already familiar with this box. To assign a sub procedure to the object in the image above, you should select the matter you want to assign and click ok. In the example below, we will be using the Delete_Blank_Rows_3 macro.



(https://powerspreadsheets.com/vba-sub-procedures/)

When you have finished the steps above, you can always execute the procedure by clicking on the relevant object.



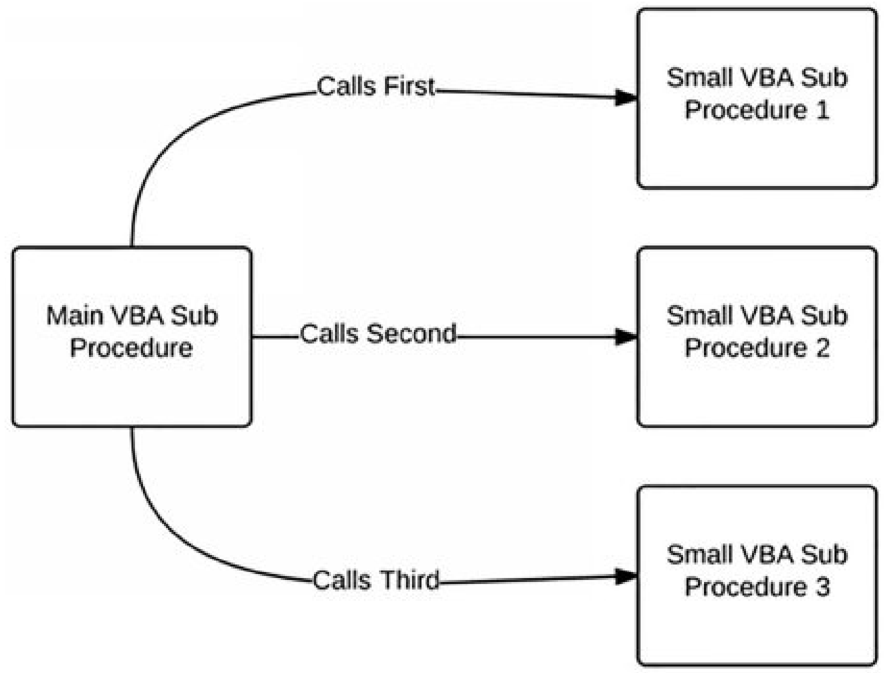
(https://powerspreadsheets.com/vba-sub-procedures/)

Option Five: How to Execute A VBA Sub Procedure from another Procedure

Experts mention that most programmers use an existing procedure to execute a sub procedure. This process is called calling code since you are running a procedure to run an existing sub procedure point it is only when you call this procedure that the cold within the sub procedure is invoked. The calling code will always specify the correct irrelevant sub procedure and will transfer the control to that procedure. When the sub procedure has run the control will go back to the calling code or the main procedure.

Experts also say that there are many reasons why one should not call other procedures when running a sub procedure including the following:

This will help to reduce the size of the code and will also ensure that the code is crisp and clear for any other user to understand. It is also easier for you to debug, maintain or modify the code. Generally, it is a good idea to use this method to maintain several different procedures. You can use to create long procedures, but experts suggest that you avoid this. You should instead follow the suggestions of expert and create several small procedures, and write the main procedure to call all the small procedures. In the diagram below, you will see how it is easy to call upon several sub procedures using the main procedure. The main procedure is on the left side of the image while the sub procedures are on the right.


Main VBA Sub Procedure
Calls First
Calls Second
Calls Third
Small VBA Sub Procedure 1
Small VBA Sub Procedure 2
Small VBA Sub Procedure 3

(https://powerspreadsheets.com/vba-sub-procedures/)

This will also help you avoid redundancies and repetition in the data. There are times when you will need to create a macro, which will carry out the same action in multiple places. In such cases, you can either create a sub procedure, which you will call in all those instances, or create a piece of code every time you need to call upon it. I am sure you know that it is easier and faster to use the same sub procedure across different applications of parts of the workbook.

If there are some procedures that you use often, you can store them in a module that you frequently used. When you do this you can import that sub procedure into every VBA project. When the module is imported you can call on the macros whenever required. The alternative to this is to copy and paste the code into a new VBA sub procedure point you will probably want to use the first option since it is easier and faster to implement in your work.

You can use any of the methods mentioned below to call A sub procedure from any other procedure or module letters look at these three methods in detail.

Method One: Use VBA Sub Procedure's Name

When you use this method, you will need to enter the following two things in the BBA cold where you are calling the sub procedure:

6. The name of the procedure that you will be calling in the sub procedure.
7. The argument that you will be using in the procedure will it be separated by commas.

In other words, the syntax that you will need to apply when I use this method is “Procedure_Name Arguments”.

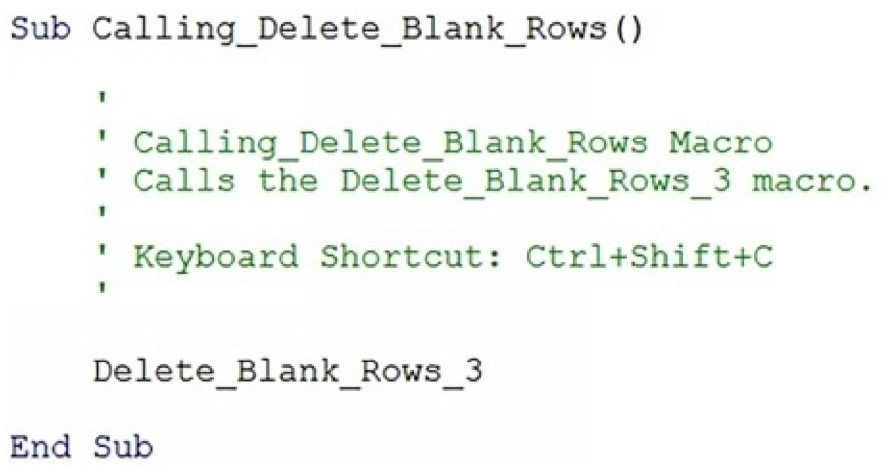
Latest assume that you will create a VBS the procedure, which will only call the Delete_Blank_Rows_3 macro.

The macro that we have written does not make sense because you can execute the Delete_Blank_Rows_3 macro directly. Since the structure is simple, we will use it as an example to see how the method works.

We have not developed a new sub procedure call calling Delete_Blank_Rows_3. This macro will only contain the following statements:

```
Sub Calling_Delete_Blank_Rows()

Delete_Blank_Rows_3

End Sub
```

The image below will show you have the BBA code will appear in the visual basic editor environment.



```
Sub Calling_Delete_Blank_Rows()
    '
    ' Calling_Delete_Blank_Rows Macro
    ' Calls the Delete_Blank_Rows_3 macro.
    '
    ' Keyboard Shortcut: Ctrl+Shift+C
    '

    Delete_Blank_Rows_3

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

You can always as a statement to the sub procedure to make it more useful and realistic.

Method Two: Use Call Statement

If you want to apply this method, you should proceed in the same fashion as method 1. In this case, you will also enter the name and the arguments of the procedure, which you will be calling within the VBA sub procedure.

There are two main differences between the methods 1 and 2:

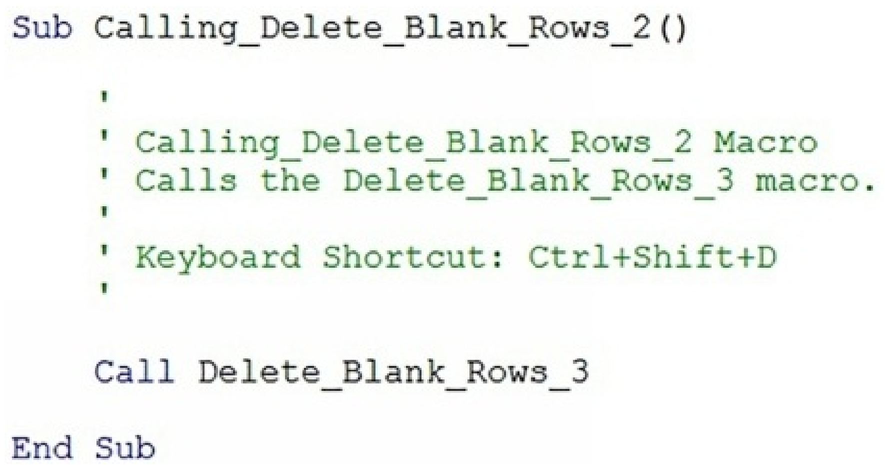
3. In this method, you will need to use the call statement. This keyword will always be written ahead of the procedure you want to call.
4. In this method, arguments will always be enclosed in the parentheses.

In other words, if you use the second method you will need to apply the syntax “Call Procedure_Name (Arguments)”.

Latest now locate how this will look in practice. We will create a simple VB A sub procedure and the sole purpose of this procedure is to call the Delete_Blank_Rows_3 macro. Latest call this new macro Delete_Blank_Rows_2. The syntax for this matter is given below:

```
Sub Calling_Delete_Blank_Rows_2()

Call Delete_Blank_Rows_3

End Sub
```

The sub procedure will look as follows in the visual basic editor environment.



```
Sub Calling_Delete_Blank_Rows_2()

    '
    ' Calling_Delete_Blank_Rows_2 Macro
    ' Calls the Delete_Blank_Rows_3 macro.
    '
    ' Keyboard Shortcut: Ctrl+Shift+D
    '

    Call Delete_Blank_Rows_3

End Sub
```

(https://powerspreadsheets.com/vba-sub-procedures/)

You may wonder why you would need to use a method where you should use the call keyword when you can use the previous method, which does not require the use of any keyword. One of the main reasons for using this method is that it provides clarity. Experts say that some programmers used the call keyword although it is optional to ensure that another procedure is being called whenever necessary.

Describe the above reasons expert suggest that you do not use the call keyword when running a sub procedure. According to the information found at the Microsoft development center, call statement is often used when a sub procedure does not begin with a specific identifier.

Method Three: Use The Application.Run Method

You should use the application.run method to execute the VBA sub procedure.

Experts suggest that you use this method if you want to call A sub procedure which has a name assigned to another variable. When you use the application.run method you can run the block of code in the sub procedure because you are passing the variable as an argument in the run method.

An example of this can be found in the book titled Excel 2013 power programming with VBA.

How to Call A VBA Sub Procedure from a Different Module

If you want to refer to A VBA sub procedure from other procedures, you will need to follow the process given below:

The search will first be carried out in the same module. If you do not find the VBA sub procedure in a module, you should look at the accessible procedures in different modules in the same workbook. If you want to call a procedure, which is private, both the procedures should be within the same module.

There will be cases where you have different procedures with the same name but in different modules. When you try to call one of the sub procedures by stating its name you will see that an error message is displayed.

This does not mean that you can always ask Excel to execute a procedure, which you want. To be more precise, you call a procedure, which is in a different module you have to clarify the following:

4. You should always state the name of the relevant module before you name the procedure.
5. You should always use a dot to separate the name of the sub procedure from the module.

You must use the following syntax in these cases: "Module_Name.Procedure_Name".

Now that you know how you should handle the cases where you have to call A sub procedure in a different module, you can choose to run the module in a different Excel workbook. Therefore we should now take a look at how to call a VBA sub procedure that is present in a different workbook.

How to Call A VBA Sub Procedure from a Different Excel Workbook

Experts say that there are two different ways in which excel will execute or run a sub procedure, which is stored in a different Excel workbook.

Build or establish a reference to different workbooks.

Specify the name of the workbook you explicitly want to refer to when you run the method.

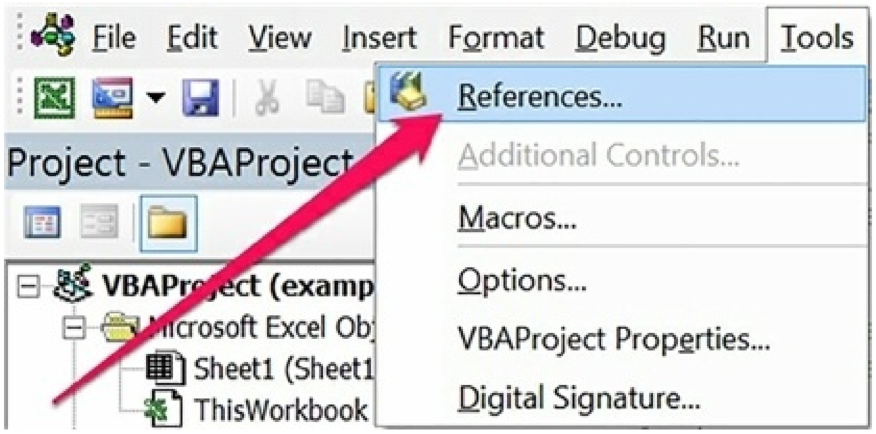
Let us now look at how you can use either method for this purpose:

Method One: Establish A Reference To Another Excel Workbook.

You can create a reference to an Excel workbook using the following steps:

Step One: Open The References Dialog.

You should now go to the tools menu in the visual basic editor and select references.



(https://powerspreadsheets.com/vba-sub-procedures/)

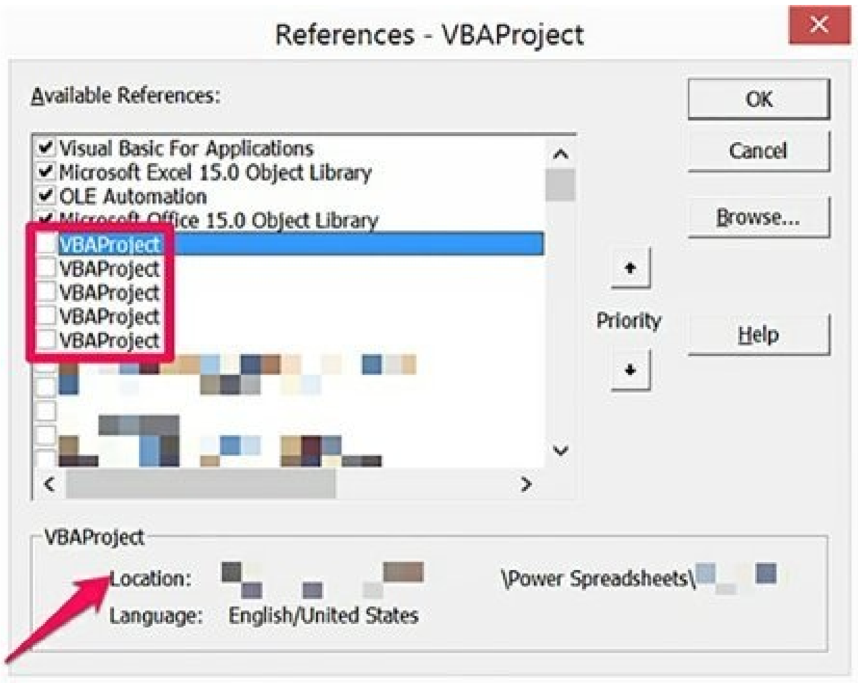
Step Two: Select The Excel Workbook To Add As Reference.

When you have completed the first step, Excel will display the reference dialog box.

This dialog box and provide all the references that you can use. The workbooks that are currently open are listed in that box. For example, look at the image below to see which Excel workbooks appear on the list.

In this case, every Excel workbook is not listed using its regular name. Instead, they will appear under the visual basic editor as their project names. Since every project name is VBAProject by default, the situation below is not very uncommon.

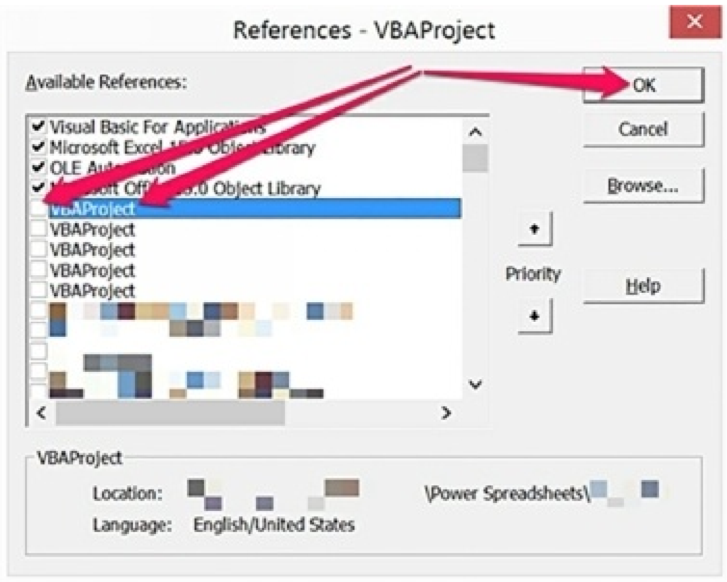
If you want to identify which VBA project you want to use as a reference, you can use the location data, which appears, at the bottom of the dialog box. Alternatively, you can always go back to the visual basic editor and change the name of the relevant project. If you want to add in Excel workbook, which is currently open, you should double click the name and select it. Then click on the ok button.



(https://powerspreadsheets.com/vba-sub-procedures/)

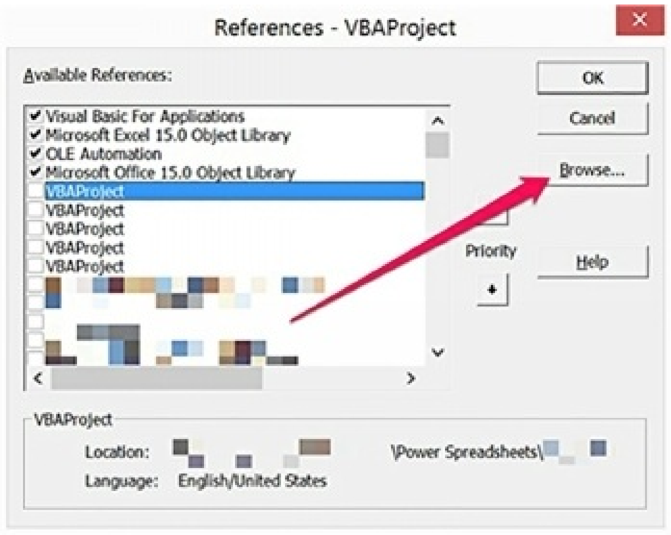
The references dialogue box only lists the Excel workbooks, which are open currently. You can also create a list of the references to workbooks, which are not currently open. To do this you will first need to click on the browse button on the right side in the references dialog box.

You see that the add reference dialog box is displayed. This box looks like every other dialogue box that you have used before. You should use the add reference dialog box to move to the folder where you have the relevant Excel workbook, selected workbook and then open it.



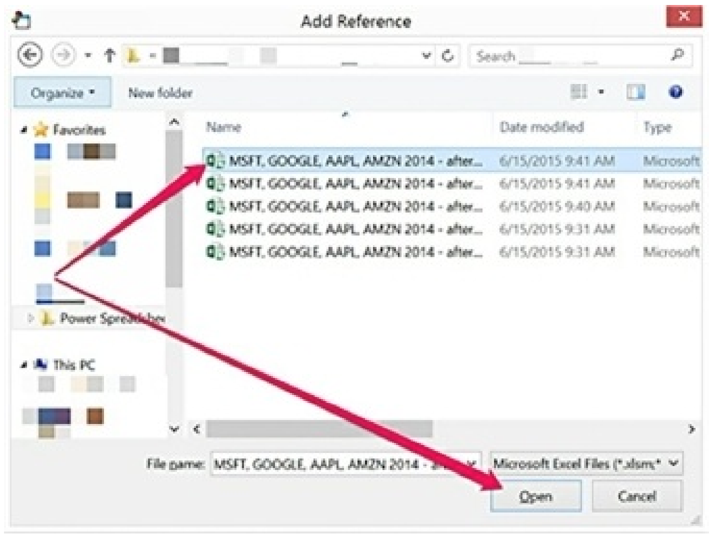
(https://powerspreadsheets.com/vba-sub-procedures/)

In this example below, we will add one sample Excel workbook for the purpose of this section.



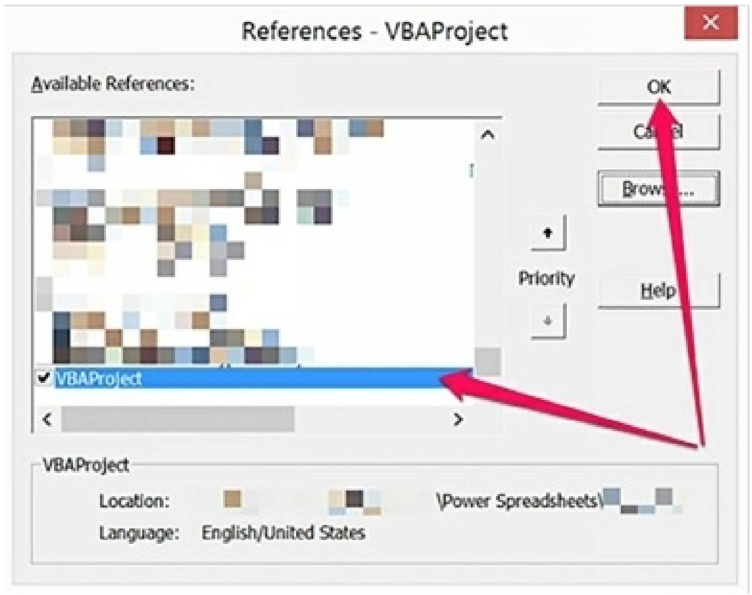
(https://powerspreadsheets.com/vba-sub-procedures/)

When you have completed the above step you will see that the relevant workbook is now I added to the bottom of the list of available references. You can then select the relevant reference and click ok. The references dialogue box will allow you to use this as the reference.



Name	Date modified	Type
MSFT, GOOGLE, AAPL, AMZN 2014 - after...	6/15/2015 9:41 AM	Microsoft
MSFT, GOOGLE, AAPL, AMZN 2014 - after...	6/15/2015 9:41 AM	Microsoft
MSFT, GOOGLE, AAPL, AMZN 2014 - after...	6/15/2015 9:40 AM	Microsoft
MSFT, GOOGLE, AAPL, AMZN 2014 - after...	6/15/2015 9:31 AM	Microsoft
MSFT, GOOGLE, AAPL, AMZN 2014 - after...	6/15/2015 9:31 AM	Microsoft

File name: MSFT, GOOGLE, AAPL, AMZN 2014 - a

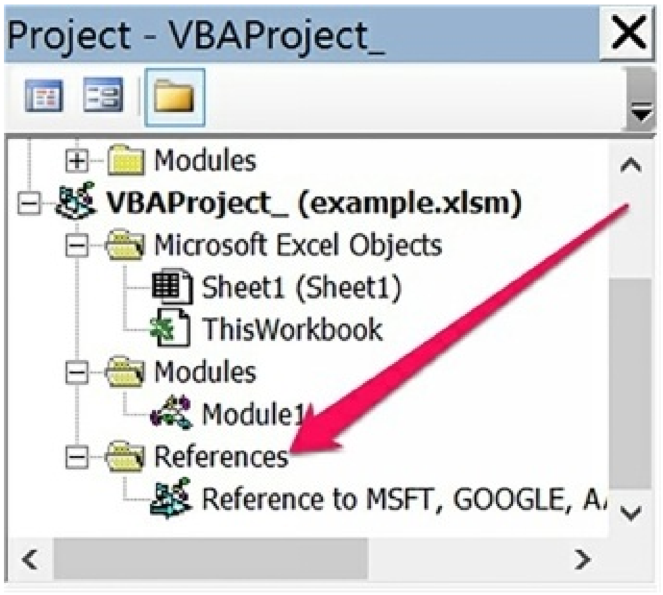


☑ VBAProject

VBAProject
Location: \Power Spreadsheets\
Language: English/United States

(https://powerspreadsheets.com/vba-sub-procedures/)

You are officially done. When you carry the steps mentioned above, the new reference that you have included will be listed in the project window in the visual basic editor. You can find this information in the references node. You can now prefer a call any procedures in a reference workbook as a divided the same workbook where you have written the sub procedure. This can be done by using a call keyword or the sub procedure’s name.



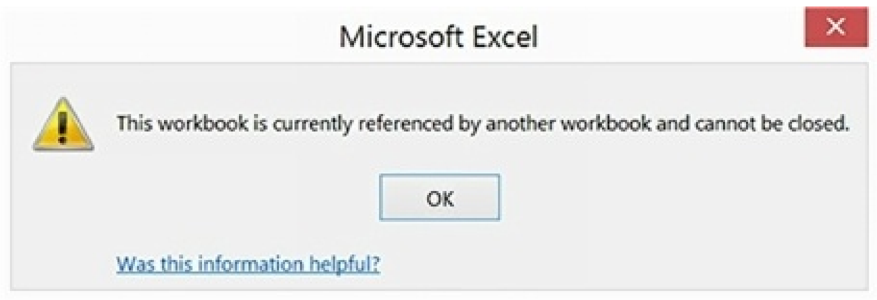
(https://powerspreadsheets.com/vba-sub-procedures/)

In the book titled Excel 2013 power programming with VB a, the author says that you should use the following syntax if you want to identify a procedure within another Excel workbook:

Project_Name.Module_Name.Procedure_Name

In simple words, experts suggest that you should first specify the name of your project, the name of the module and then the name of the actual sub procedure you want to use.

You will notice that when you open an Excel workbook, it will reference another workbook, which will be open automatically. Additional e you cannot close the referenced workbook without closing the originally opened workbook. If you try to do this excel will send you a warning message that this workbook is being referenced by another workbook and therefore cannot be closed.


Microsoft Excel

This workbook is currently referenced by another workbook and cannot be closed.

(https://powerspreadsheets.com/vba-sub-procedures/)

Method Two: Use The Application.Run Method.

You can use the application.run method to execute any VBA sub procedure. If you want to use this method, you do not have to create a reference as explained in the previous section. You must, however, have the Excel workbook, which contains a sub procedure open. To see an example of how you can use this method, please refer to the book titled 'Excel 2013 Power Programming with VBA'.

Option Six: How to Execute A VBA Sub Procedure Using the Ribbon

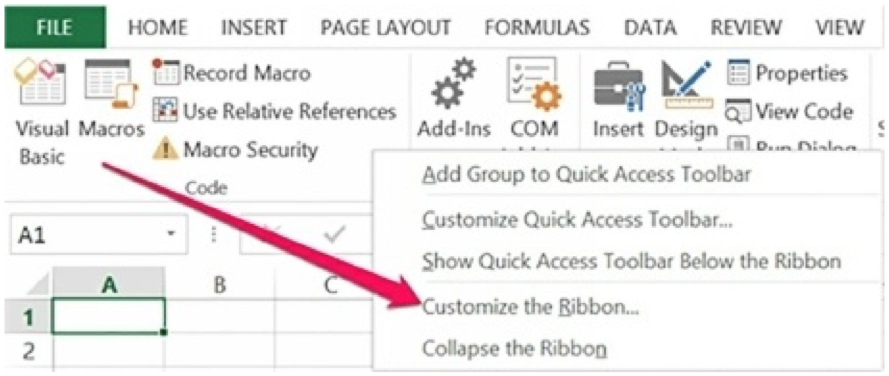
If you want to include a button to the ribbon which points to a relevant sub procedure, you should follow the steps below. You can execute the macro by clicking on the button in the Ribbon.

In this section, we will look at how you can add a button to the ribbon, and what you should do to run the macro or the block of code within that sub procedure. We will continue to use the Delete_Blank_Rows_3 macro for our example. This is the most appropriate method that one can use for a macro that is present in the personal workbook.

The personal workbook is where you will store the relevant macros that you can use in an Excel workbook. In simple words, the macros that are stored in the personal workbook can be called upon or used regardless of which excel workbook you are working on. Let us look at five simple steps that you will need to follow to add the button to the Ribbon.

Step One: Access The Excel Options Dialog.

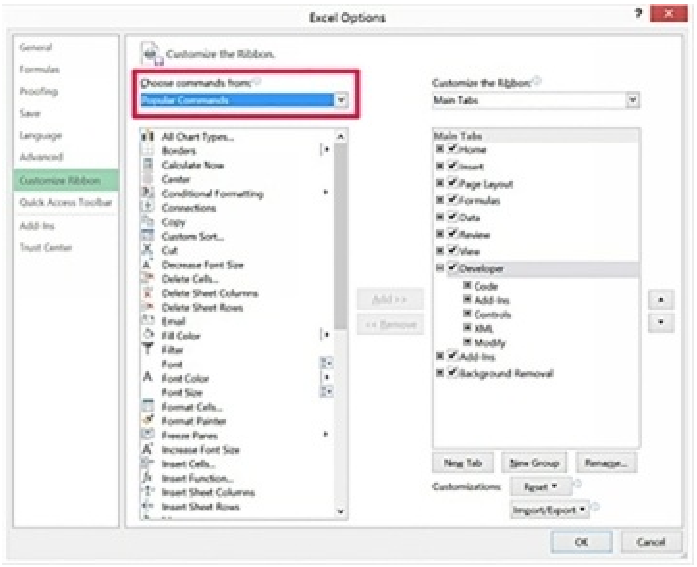
Right click on the ribbon and choose the option "Customize the Ribbon" to display the context menu.



(https://powerspreadsheets.com/vba-sub-procedures/)

Step Two: Choose To Work With Macros.

In the Excel Options dialog box, you can choose the commands you want to include to the ribbon from the drop-down menu in the customize ribbon tab.



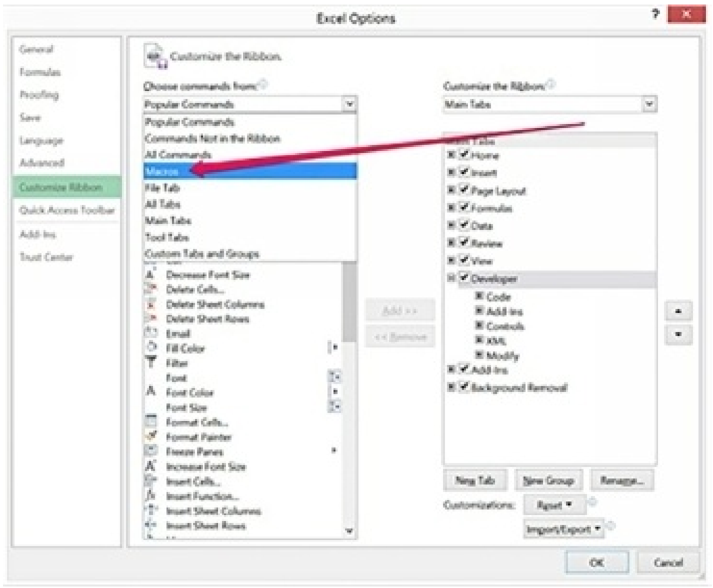
Choose commands from: Popular Commands

Customize the Ribbon: Main Tabs

Customizations: Reset

(https://powerspreadsheets.com/vba-sub-procedures/)

You can browse through different commands before you add them to the Ribbon. All you need to do is click the command, and then select “Macros.”



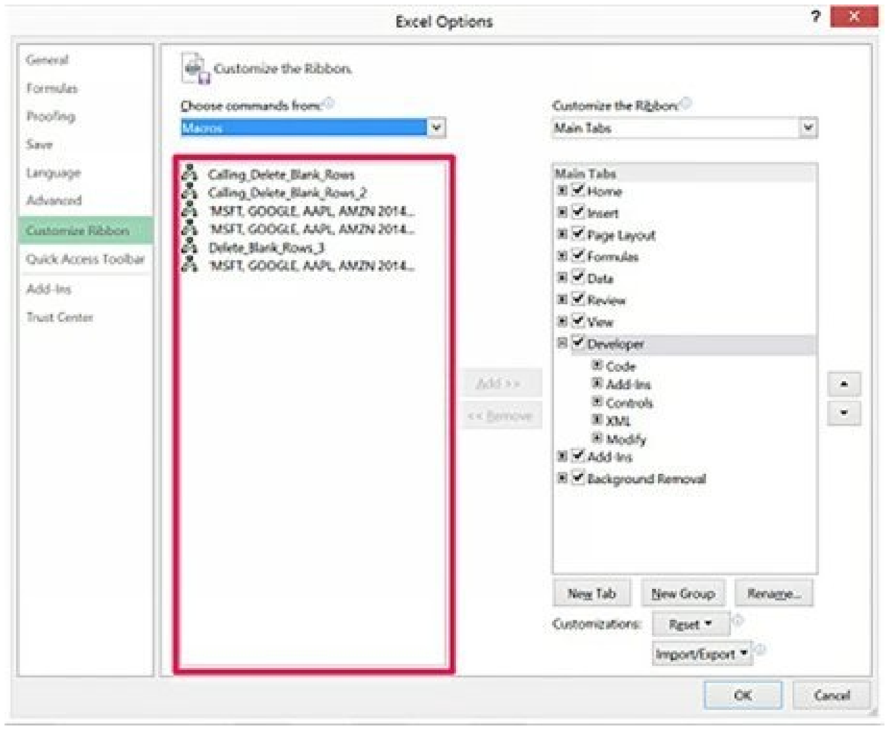
Choose commands from: Popular Commands

Customize the Ribbon: Main Tabs

Customizations: Reset

(https://powerspreadsheets.com/vba-sub-procedures/)

When you have done this, you will see a list of all the macros that you can include in the Ribbon. This list will be found in the Choose commands list, and will appear below the choose commands drop down menu. Please see the image below for a better understanding.



Choose commands from: Macros

- Calling_Delete_Blank_Rows
- Calling_Delete_Blank_Rows_2
- 'MSFT, GOOGLE, AAPL, AMZN 2014...
- 'MSFT, GOOGLE, AAPL, AMZN 2014...
- Delete_Blank_Rows_3
- 'MSFT, GOOGLE, AAPL, AMZN 2014...

Customize the Ribbon: Main Tabs

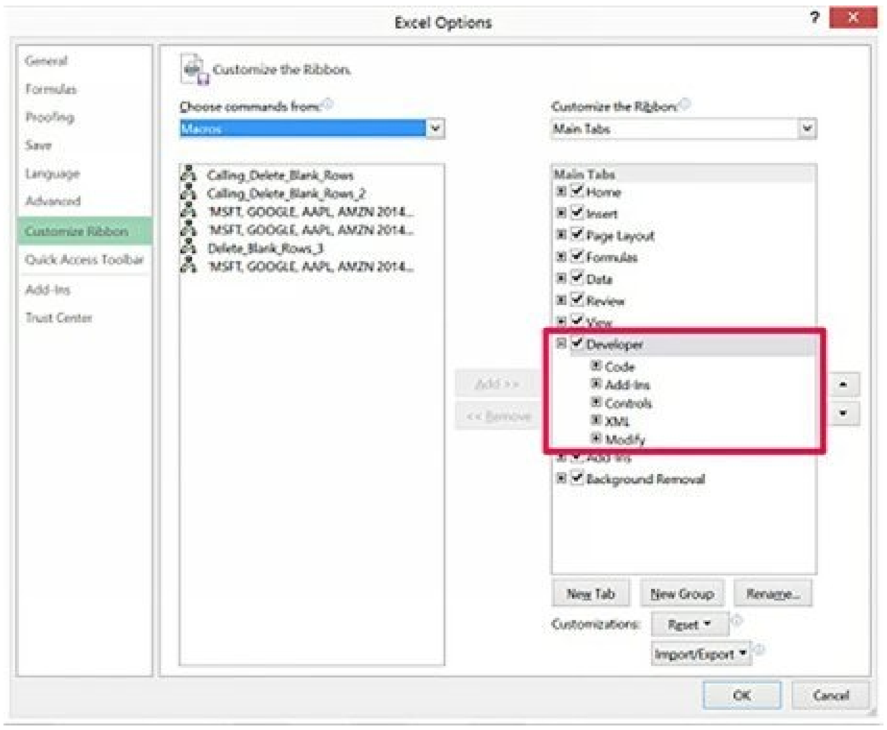
Main Tabs
- ✓ Home
- ✓ Insert
- ✓ Page Layout
- ✓ Formulas
- ✓ Data
- ✓ Review
- ✓ View
- ✓ Developer
  - Code
  - Add-Ins
  - Controls
  - XML
  - Modify
- ✓ Add-Ins
- ✓ Background Removal

(https://powerspreadsheets.com/vba-sub-procedures/)

Step Three: Select The Tab And Group Of Commands To Which You Want To Add The Macro.

You can find the Customize the Ribbon list on the right side in the Excel Options dialog box. This is where you will find the list of all the commands that can be found in the Ribbon. You will notice that these commands are organized by groups of commands and tabs.

The image below will show you how there are five groups of commands, namely Add-Ins, Code, XML, Modify and Controls in the Developer Tab. You can choose where to add the macro button to the ribbon in the "Customize the Ribbon" List. You can also expand or contract a tab in Excel or any group of commands using the plus and minus signs that appear in the list on the left side. You can now include new command groups or tabs to the Excel workbook using the buttons in the Customize Ribbon list dialog box.



Choose commands from: Macros

- Calling_Delete_Blank_Rows
- Calling_Delete_Blank_Rows_2
- 'MSFT, GOOGLE, AAPL, AMZN 2014...
- 'MSFT, GOOGLE, AAPL, AMZN 2014...
- Delete_Blank_Rows_3
- 'MSFT, GOOGLE, AAPL, AMZN 2014...

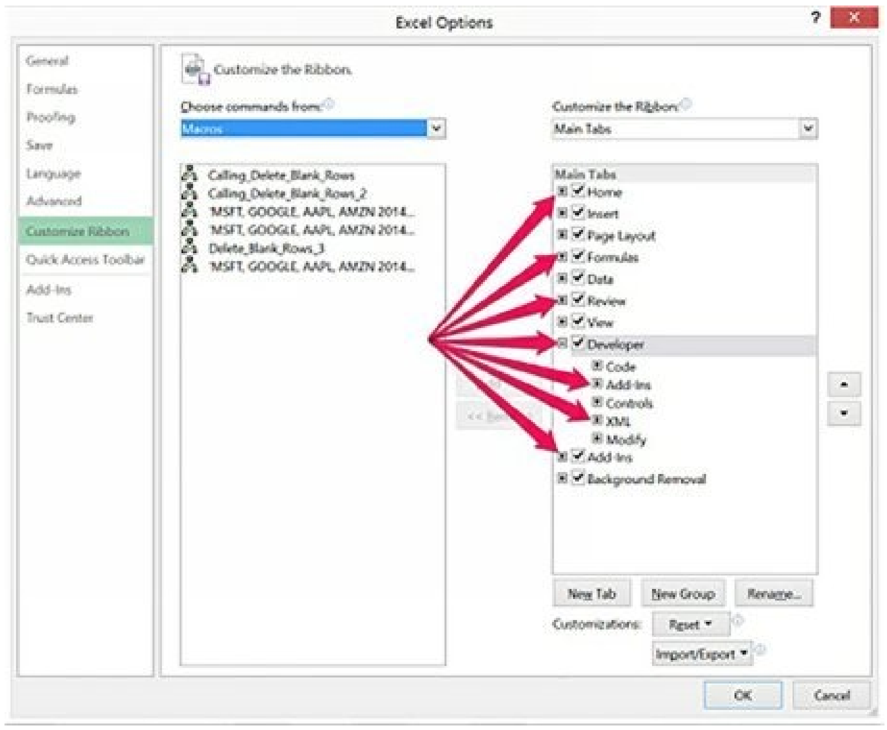
Customize the Ribbon: Main Tabs

Main Tabs
- ✓ Home
- ✓ Insert
- ✓ Page Layout
- ✓ Formulas
- ✓ Data
- ✓ Review
- ✓ View
- ✓ Developer
  - Code
  - Add-Ins
  - Controls
  - XML
  - Modify
- ✓ Add-Ins
- ✓ Background Removal

Customizations: Reset ▾

(https://powerspreadsheets.com/vba-sub-procedures/)

You can now choose to create a new command group or choose an existing command group when you want to add a macro to the Ribbon. In the section below, we will talk about how you can add a new group of commands and tab to the ribbon.



Choose commands from: Macros

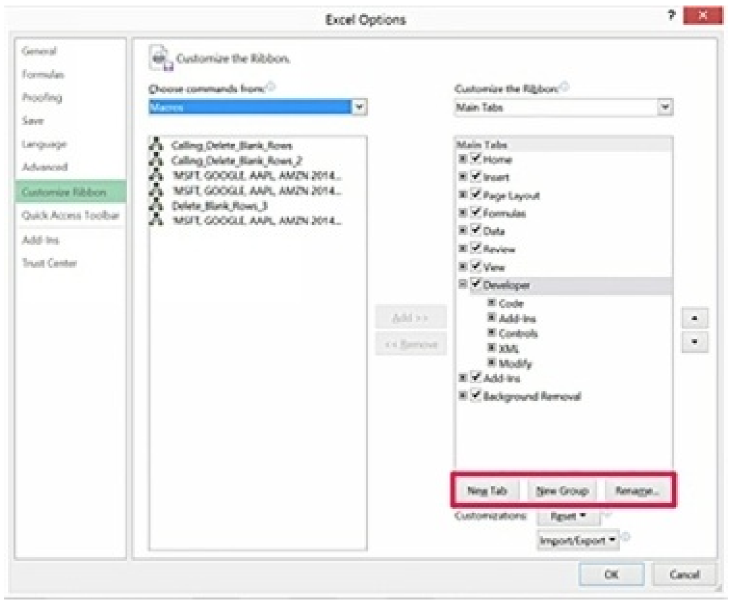
Customize the Ribbon: Main Tabs

Calling_Delete_Blank_Rows
Calling_Delete_Blank_Rows_2
'MSFT, GOOGLE, AAPL, AMZN 2014...
'MSFT, GOOGLE, AAPL, AMZN 2014...
Delete_Blank_Rows_3
'MSFT, GOOGLE, AAPL, AMZN 2014...

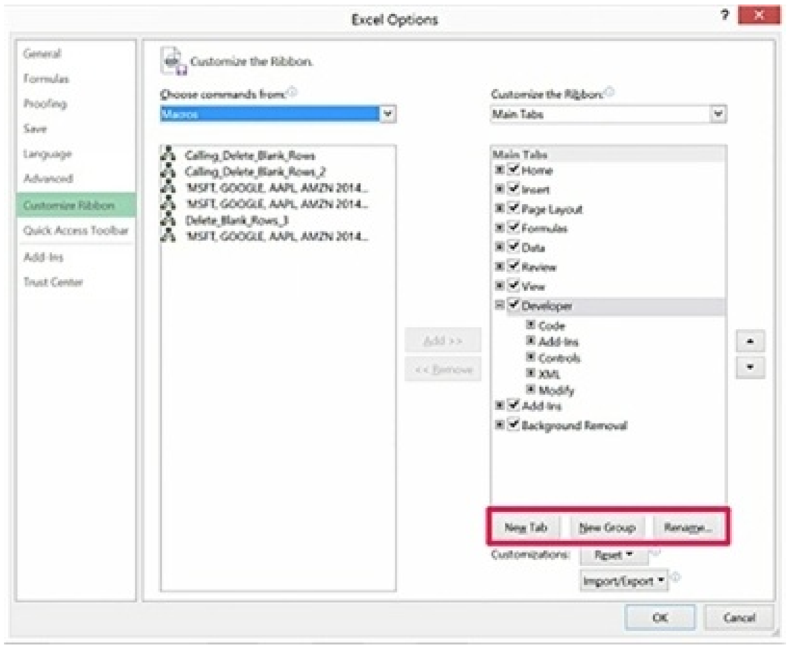
(https://powerspreadsheets.com/vba-sub-procedures/)

In the example below, we will add a new tab immediately after the Developer tab in Excel. To do this, you should go to the “Developer” Tab and click the “New Tab” button.

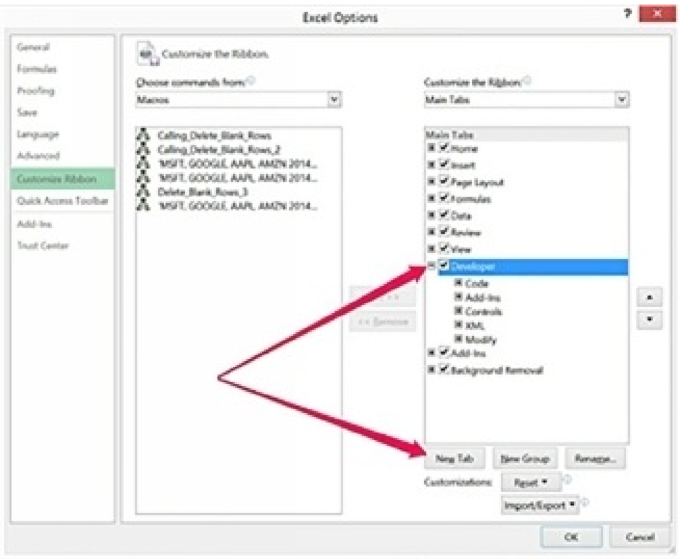
You can choose to rename the newly included tab by clicking the Rename button. Excel will now display the Rename dialog box. You should enter the name of the collection and click OK. You must repeat this process for the command group. You should first select the “New Group (Custom)” option and then rename the button.



(https://powerspreadsheets.com/vba-sub-procedures/)

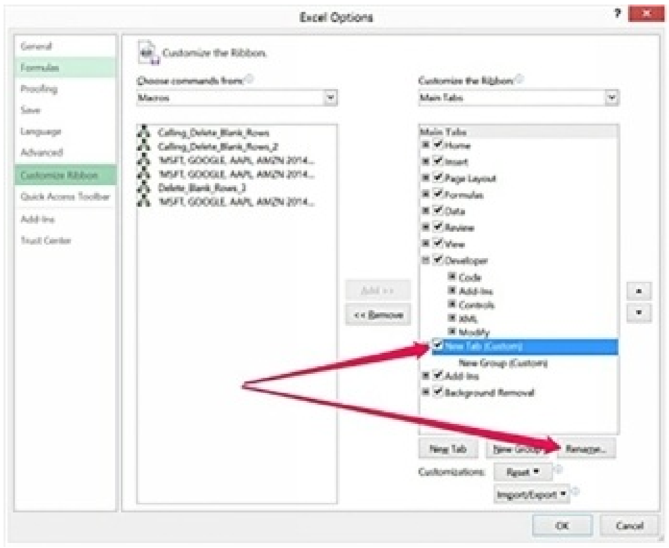


(https://powerspreadsheets.com/vba-sub-procedures/)



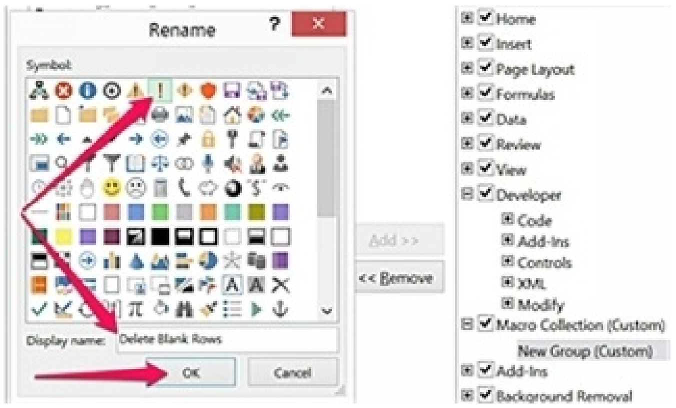
(https://powerspreadsheets.com/vba-sub-procedures/)

Excel will display a different rename box, which will allow you to choose the name or a symbol that will represent the group of command.



(https://powerspreadsheets.com/vba-sub-procedures/)

You can choose an icon if you want. In this example, we will choose an icon and then enter the name for the command group. Click OK once you are sure of the changes.

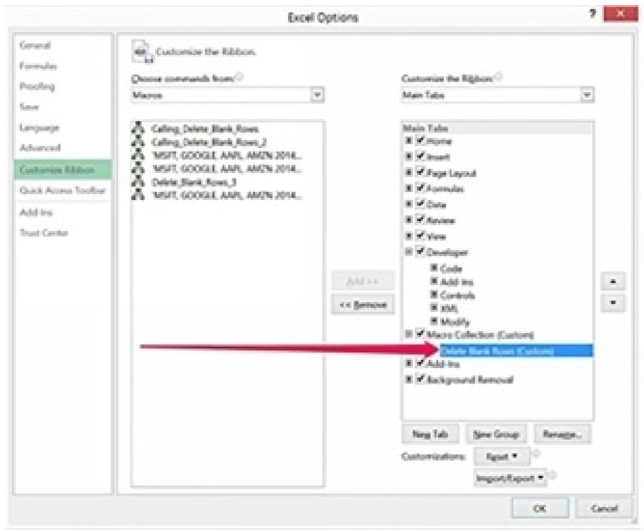

Rename

Symbol:

Display name: Delete Blank Rows

(https://powerspreadsheets.com/vba-sub-procedures/)

When all the steps are in order, you should choose the group of commands that you wish to include in the macro. In the example below, the Delete Blank Rows group is the command group that has just been created.


Excel Options

Customize the Ribbon.

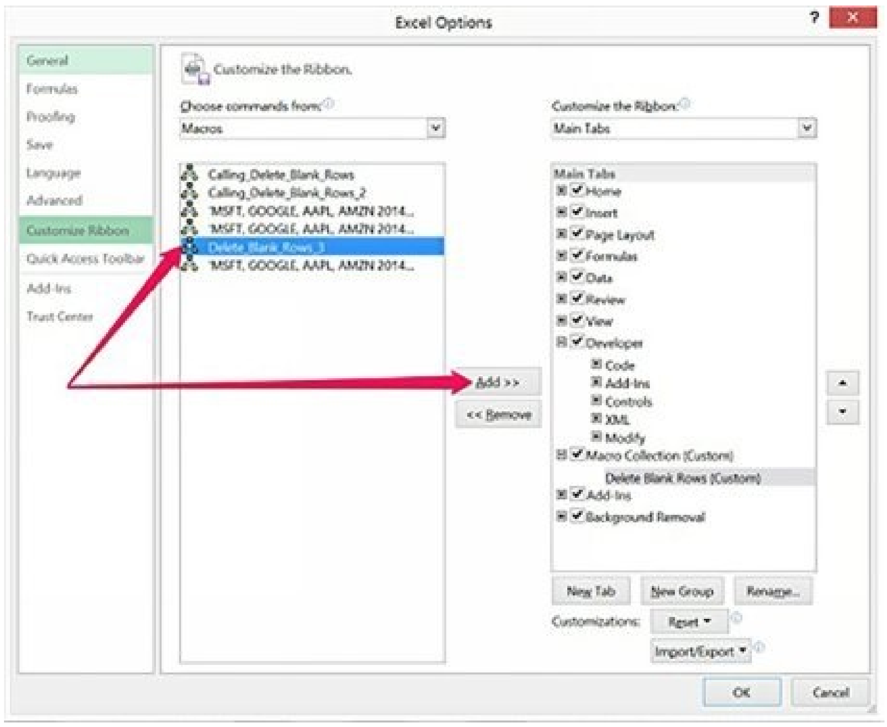
Choose commands from: Macros

Customize the Ribbon: Main Tabs

(https://powerspreadsheets.com/vba-sub-procedures/)

Step Four: Add VBA Sub Procedure To The Ribbon.

When you want to add a macro button to the ribbon, you should select the relevant macro from choose commands list and hit the Add button, which is present in the center of the dialog box. The image below will show you how you can add the Delete_Blank_Rows_3 macro to the ribbon.

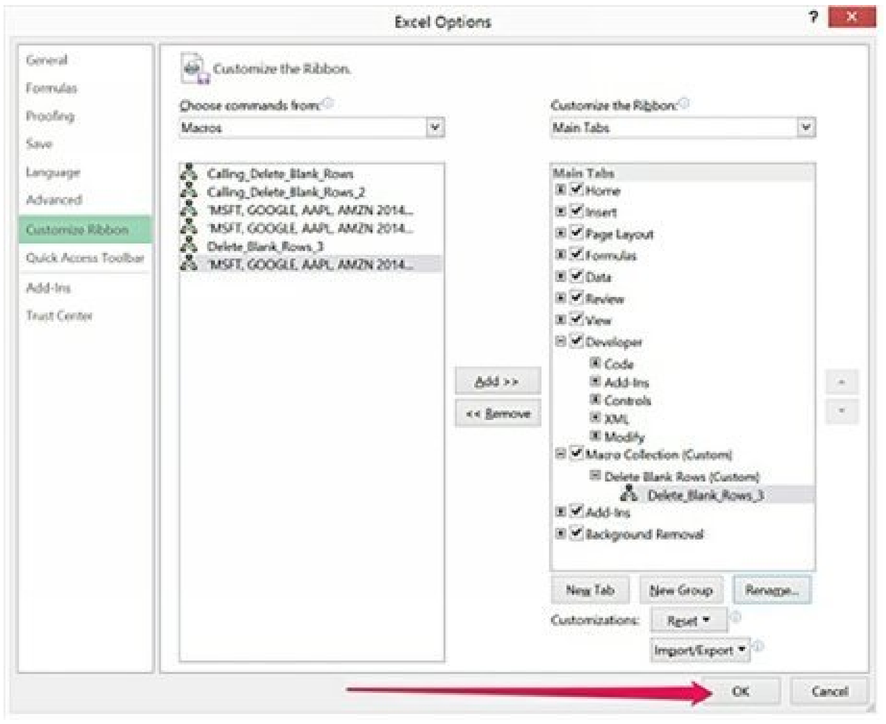

Choose commands from: Macros

Customize the Ribbon: Main Tabs

(https://powerspreadsheets.com/vba-sub-procedures/)

Step Five: Finish The Process.

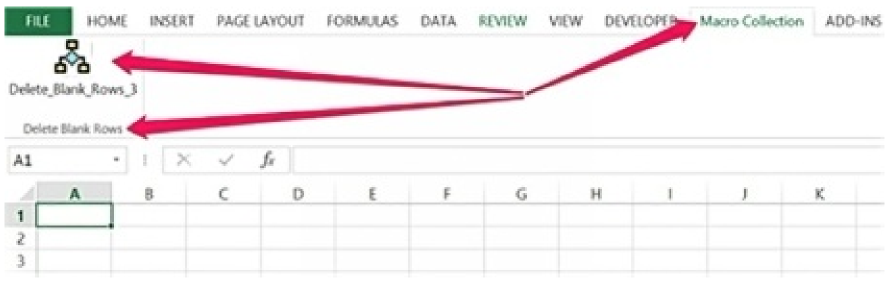
To complete the process, you should click OK which is found at the bottom right corner of the dialog box.


Choose commands from: Macros

Customize the Ribbon: Main Tabs

(https://powerspreadsheets.com/vba-sub-procedures/)

Excel will now close the Excel Options dialog box and make the necessary changes. You will notice that in the case of this example, there is already a new tab, called Macro Collections that has been included. You will also see that a group of commands and a button have been included to the ribbon.

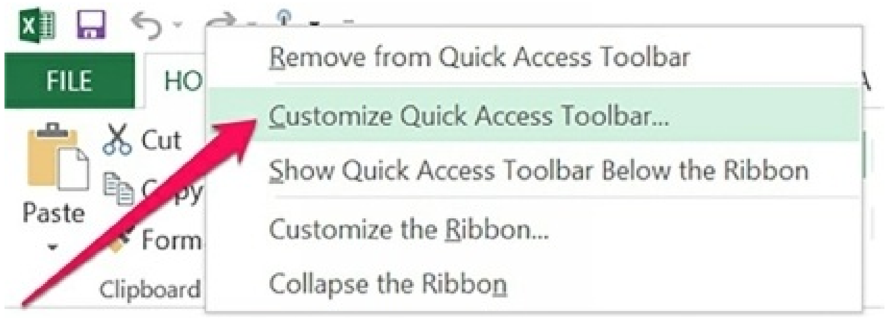


(https://powerspreadsheets.com/vba-sub-procedures/)

When you complete this process, you can execute the macro or the sub procedure by simply clicking on the correct button in the Ribbon. Excel will enable this icon even if the workbook that has the macro in it is closed. If the relevant workbook is closed, Excel will open that workbook with the macro in it, before it executes the code.

Option Seven: How to Execute A VBA Sub Procedure Using the Quick Access Toolbar

You can find the Quick Access Toolbar on the upper left corner in your Excel workbook.


FILE
HO
Cut
Copy
Form
Paste
Clipboard
Remove from Quick Access Toolbar
Customize Quick Access Toolbar...
Show Quick Access Toolbar Below the Ribbon
Customize the Ribbon...
Collapse the Ribbon

(https://powerspreadsheets.com/vba-sub-procedures/)

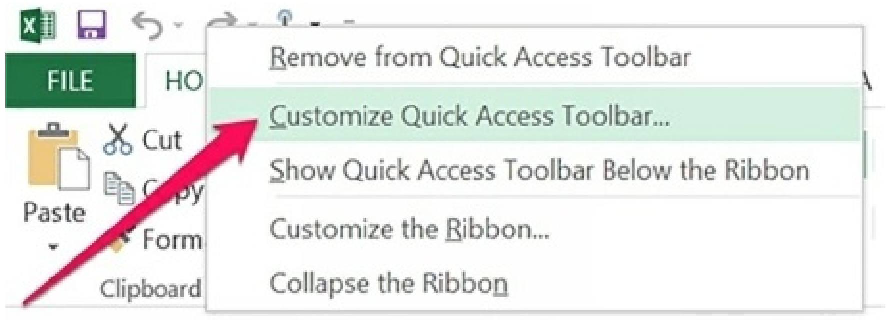
Just like we did with Ribbon, you can make some changes to the Quick Access Toolbar to include a button that is assigned to the sub procedure. This means that you can execute the code in the procedure by simply clicking that button.

You should use this method only when the macro you are including to the quick access toolbar is found in your personal workbook. It is the same as the method of using the ribbon to execute the sub procedure. You should also remember that if you include the macro to the Quick Access toolbar, you can also ensure that this button only appears in the Excel workbook that has the macro written in it.

We will look at how you can use the quick access toolbar and how you can customize it for this purpose. Let us first look at how you can add a macro button to the toolbar in five simple steps.

Step One: Access the Excel Options Dialog

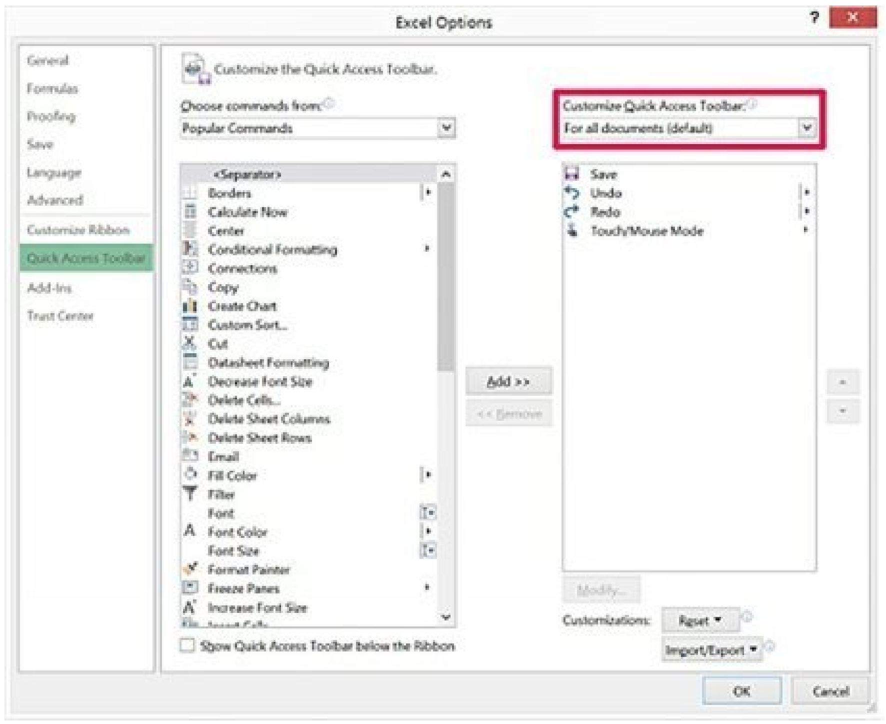
In the previous sections, you have learned how you can access the Options dialog box, and this section provides some additional information that you can use. For this specific example, it is always a good idea to access the Quick Access Toolbar tab using the Excel Options dialog box. All you need to do is right-click on the Quick Access toolbar and select the option to “Customize Quick Access Toolbar.” When you complete this step, you will see the Options Dialog box open in front of you.


FILE
HO
Cut
Copy
Format
Paste
Clipboard
Remove from Quick Access Toolbar
Customize Quick Access Toolbar...
Show Quick Access Toolbar Below the Ribbon
Customize the Ribbon...
Collapse the Ribbon

(https://powerspreadsheets.com/vba-sub-procedures/)

Step Two: Choose For Which Excel Workbooks The Customized Quick Access Toolbar Applies.

You will see a drop-down menu in the top right corner of the excel options dialog box. This drop-down menu is for the Quick Access toolbar. You should navigate to the section where you can make some changes to which workbooks should reflect the change you make. Choose your preferred option from the drop-down menu in the Customize Quick Access dialog box.



Choose commands from: Popular Commands

Customize Quick Access Toolbar: For all documents (default)

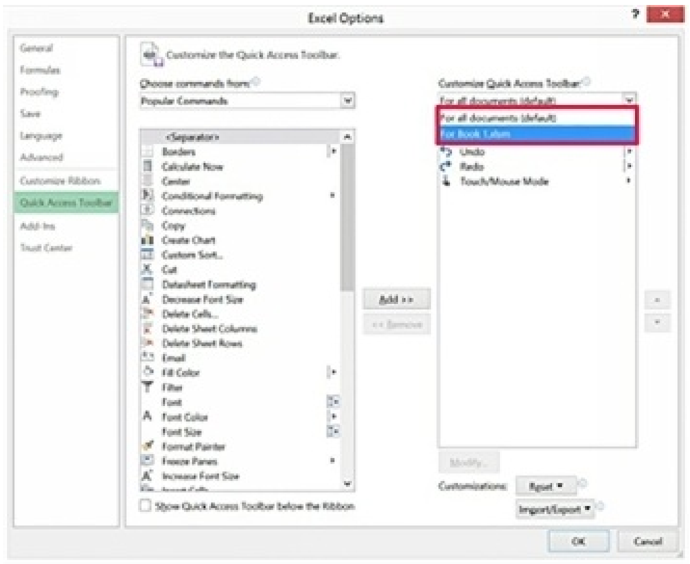
Show Quick Access Toolbar below the Ribbon

Customizations: Reset

(https://powerspreadsheets.com/vba-sub-procedures/)

If you want this button to appear in every Excel workbook, you should choose the option “For all documents (default). This is the default setting that will be applied to every workbook.

If you want the button to appear only in one workbook, you should choose the name of the workbook. The image below shows a section of the options that you can find in the drop-down menu in the Customize Quick Access Toolbar section.



Choose commands from: Popular Commands

Customize Quick Access Toolbar: For all documents (default)

For all documents (default)

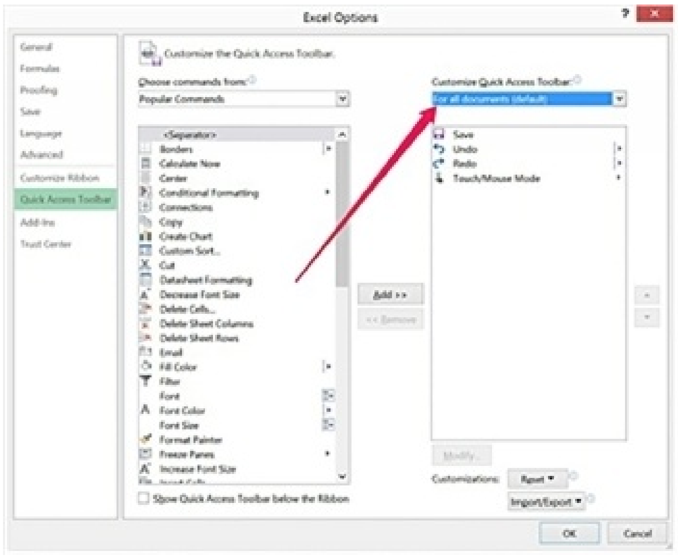
For Book 1.xlsm

Show Quick Access Toolbar below the Ribbon

Customizations: Reset

(https://powerspreadsheets.com/vba-sub-procedures/)

In this example, we only have the workbook named “Book 1” open on the system. In this example, we will use the default option. This customization will apply to ever workbook that you open.



Choose commands from: Popular Commands

Customize Quick Access Toolbar: For all documents (default)

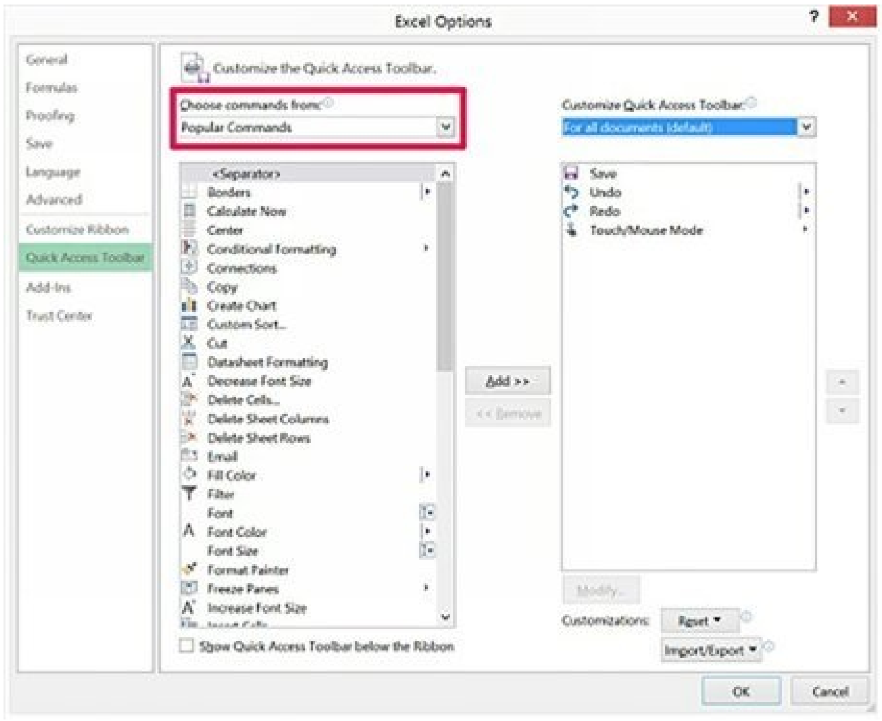
(https://powerspreadsheets.com/vba-sub-procedures/)

Step Three: Choose To Work With Macros.

In the Excel Options dialog box, you should navigate to the top left corner in the Quick Access toolbar. Here, you will find the choose commands option in the drop-down menu. Select “Macros” from the drop-down list.

Step Four: Add Macro To Quick Access Toolbar.

Once you complete the step above, you will see a list of all the macros in your Excel workbook in the options dialog box. These are the macros that you can include to the Quick Access Toolbar. These macros will be in the choose Commands list on the left side of the Quick Acess toolbar tab in the dialog box.

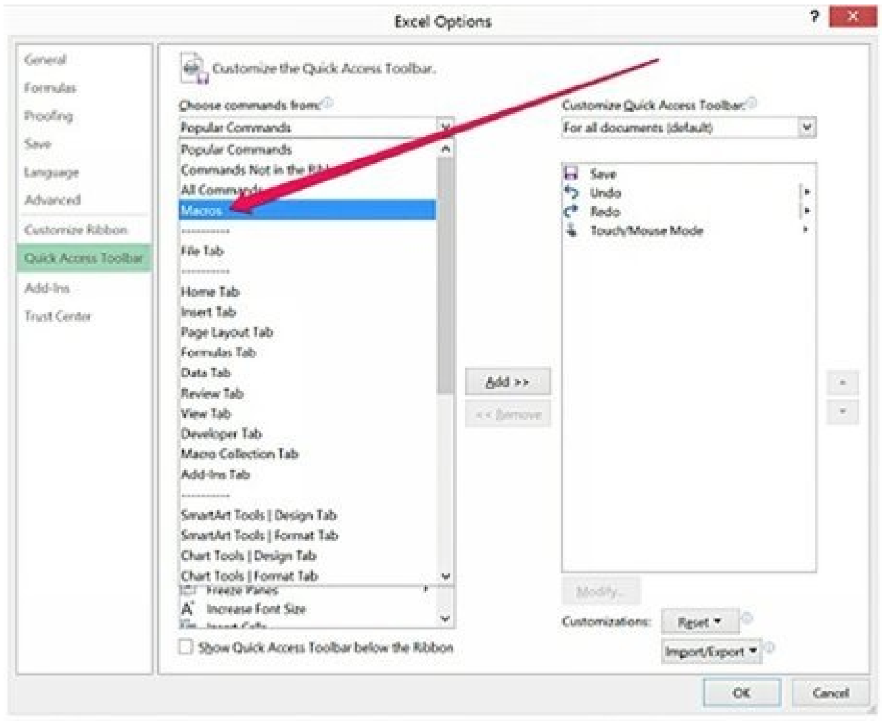


Choose commands from: Popular Commands

Customize Quick Access Toolbar: For all documents (default)

(https://powerspreadsheets.com/vba-sub-procedures/)

You should now choose the macro that you want to include from the Choose Commands option in the list box. Click on Add button, which will appear in the center of the Options dialog box. The image below will show you how you should include the Delete_Blank_Rows_3 macro.



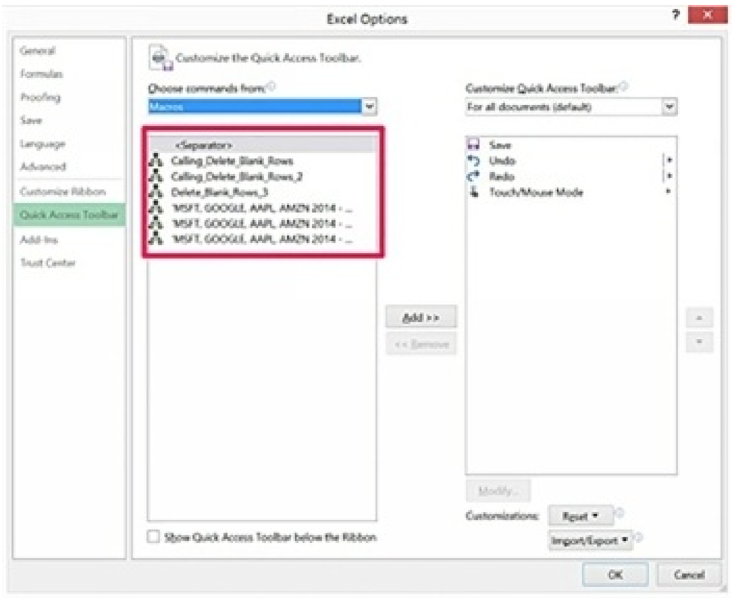
Choose commands from: Popular Commands

Customize Quick Access Toolbar: For all documents (default)

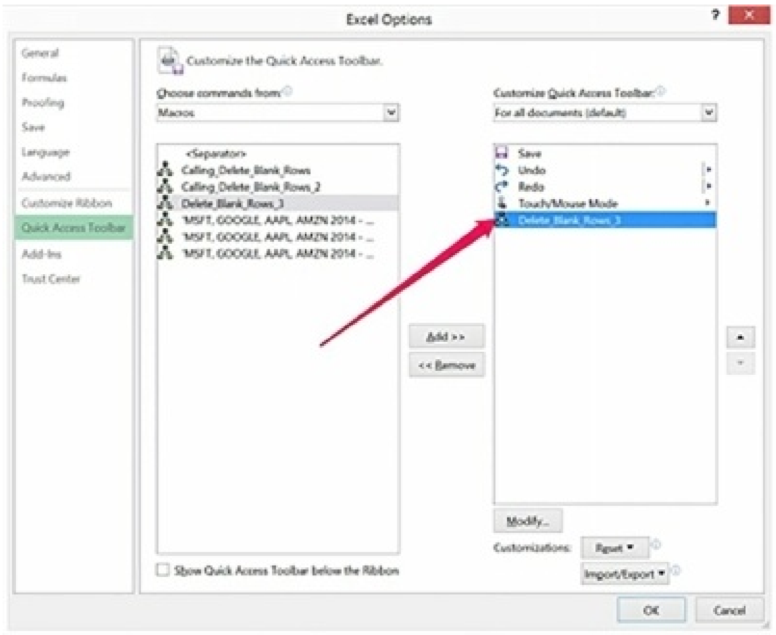
(https://powerspreadsheets.com/vba-sub-procedures/)

Step Five: Click The OK Button.

Once you run the first four steps, you will see the macro button in the Quick Access Toolbar. You will see that this button is found in the Customize Quick Access Toolbar list. This list is found on the right side of the options dialog box. You should now press the OK button at the bottom right corner of the options dialog box to complete the process. This will implement the necessary changes.


Excel Options
General
Formulas
Proofing
Save
Language
Advanced
Customize Ribbon
Quick Access Toolbar
Add-Ins
Trust Center
Customize the Quick Access Toolbar.
Choose commands from:
Macros
<Separator>
Calling_Delete_Blank_Rows
Calling_Delete_Blank_Rows_2
Delete_Blank_Rows_3
'MSFT, GOOGLE, AAPL, AMZN 2014 - ...
'MSFT, GOOGLE, AAPL, AMZN 2014 - ...
'MSFT, GOOGLE, AAPL, AMZN 2014 - ...
Add >>
<< Remove
Customize Quick Access Toolbar:
For all documents (default)
Save
Undo
Redo
Touch/Mouse Mode
Modify...
Customizations:
Reset ▾
Import/Export ▾
Show Quick Access Toolbar below the Ribbon
OK
Cancel

(https://powerspreadsheets.com/vba-sub-procedures/)

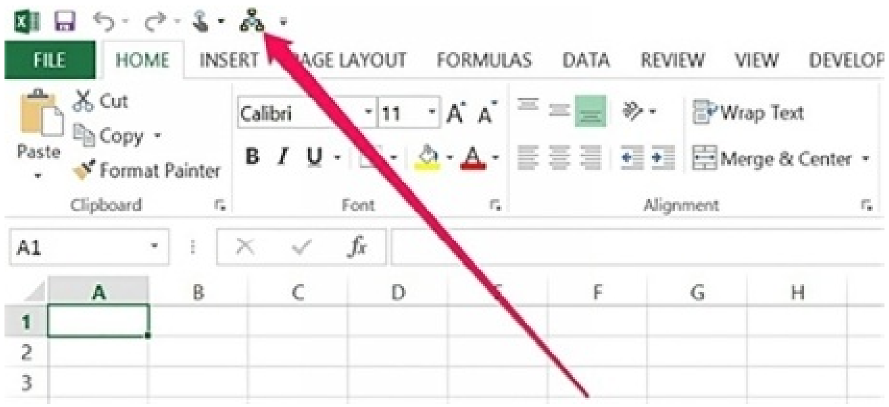


Choose commands from: Macros

Customize Quick Access Toolbar: For all documents (default)

(https://powerspreadsheets.com/vba-sub-procedures/)

When you move back to Excel, you will see that the necessary macro button has been included in the Quick Access Toolbar. To run or execute the sub procedure, you can click on the button that you added to the Quick Access toolbar.



(https://powerspreadsheets.com/vba-sub-procedures/)

Option Eight: How to Execute A VBA Sub Procedure When a Particular Event Occurs

You can run a specific VBA Sub procedure in Excel even when an event occurs. Excel will check with you if the sub procedure should run, and you can either choose to ignore the procedure or ignore the event. In the book titled, ‘Excel 2013 Power Programming With VBA,’ the author talks about several events where you may come across this issue:

1. Entering information in a worksheet
2. Opening a workbook
3. Clicking a command button
4. Saving a file

A VBA Sub procedure that is executed even when an event occurs is called an event handler procedure. This type of procedure has two characteristics that separate it from the other Sub procedures in VBA.

The name of this sub procedure will always have a different structure. Their name will need to stick to the following syntax – “object_EventName.” The names of such procedures will have three elements:

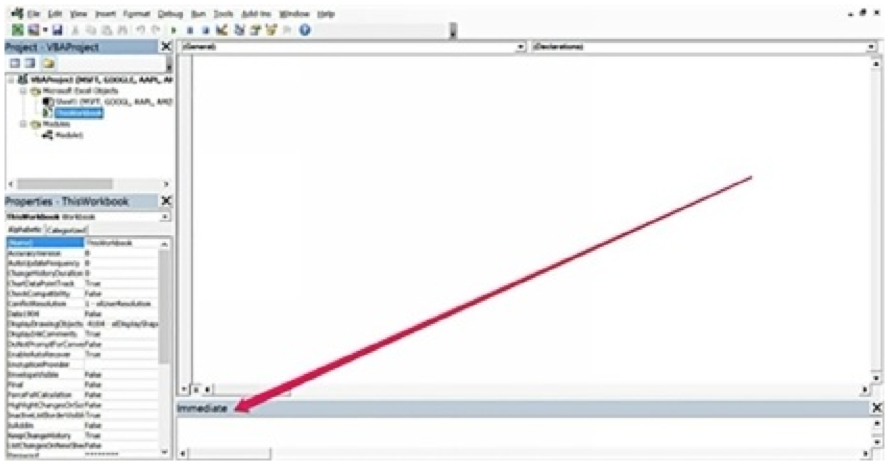
1. Objects
2. Underscore
3. Name of the event

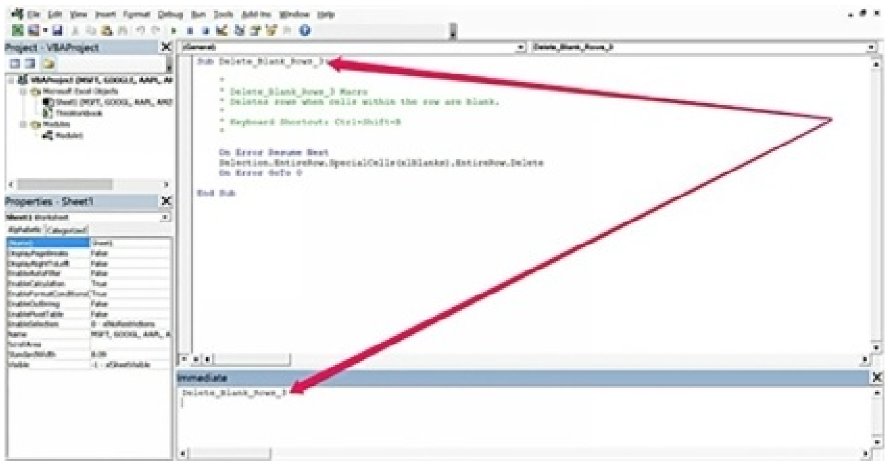
The module for the object in the name is the VBA module in which this sub procedure is written or stored. It is important to learn more about event handler procedures, and this is an extensive topic to cover. To learn more about this topic, you should refer to chapter seventeen in the book titled ‘Excel 2013 Power Programming With VBA.’

Option Nine: Executing the VBA Sub Procedure Using the Immediate Window

It is a good idea to execute a sub procedure in VBA using the immediate window in the environment. It is always a good idea to do this if you are building an application in the visual basic environment. The Immediate Window can be found at the bottom section of the Visual Basic Editor.

*If you’re finding the information valuable so far, please be sure to leave **5-star feedback on Amazon***





```
Sub Delete_Blank_Rows_3()
    '
    ' Delete_Blank_Rows_3 Macro
    ' Deletes rows when cells within the row are blank.
    '
    ' Keyboard Shortcut: Ctrl+Shift+B
    '

    On Error Resume Next
    Selection.EntireRow.SpecialCells(xlBlanks).EntireRow.Delete
    On Error GoTo 0

End Sub
```

```
Delete_Blank_Rows_3
```

(https://powerspreadsheets.com/vba-sub-procedures/)

To learn more about the Immediate Window, please read the first few books of the series. You can also learn more about the IDE in VBA in those books. If you want to run a sub procedure in VBA in the Immediate Window, you should type the name of that procedure in the window and click Enter. To conclude, the concept of procedures and sub procedures is used frequently in most blogs and books that cover VBA and macros. It is important for you to understand what these terms mean if you want to become an expert in VBA.

Conclusion

Thank you for purchasing the book. If you want to master VBA, there are some concepts that you should know well. You should also have some tricks and tips up your sleeve to help you overcome any problems you may have with VBA. This book will help you master some of the concepts, and also leave you with some tips that you can use to troubleshoot and handle any errors and exceptions.

I hope the information in the book will help you improve your VBA programming skills.

P.S. If you don’t mind, please drop a short review of my title on Amazon and feel free to tell me what you think! Thanks a lot!

Will You Help Me?

Hi there, avid reader! If you have extra time on your hands, I would really, really appreciate it if you could take a moment to click my author profile in Amazon. In there, you will find all the titles I authored and who knows, you might find more interesting topics to read and learn!

If it's not too much to ask, you can also leave and write a review for all the titles that you have read – whether it's a positive or negative review. An honest and constructive review of my titles is always welcome and appreciated since it will only help me moving forward in creating these books. There will always be room to add or improve, or sometimes even subtract certain topics, that is why these reviews are always important for us. They will also assist other avid readers, professionals who are looking to sharpen their knowledge, or even newbies to any topic, in their search for the book that caters to their needs the most.

If you don't want to leave a review yourself, you can also vote on the existing reviews by voting Helpful (Thumbs Up) or Unhelpful (Thumbs Down), especially on the top 10 or so reviews.

If you want to go directly to the vote or review process, please visit my author file page in Amazon for my below titles:

www.audible.com

Excel VBA: A Step-By-Step Comprehensive Guide on Advanced Excel VBA Programming Techniques and Strategies

Again, I truly appreciate the time and effort that you will be putting in leaving a review for my titles or even just for voting. This will only inspire me to create more quality content and titles in the future.

Thank you and have a great day!

Peter Bradley

Sources

ttps://www.dummies.com/software/microsoft-office/excel/10-resources-for-excel-vba-help/

ttps://techcommunity.microsoft.com/t5/Excel/9-quick-tips-to-improve-your-VBA-macro-performance/td-p/173687

ttp://what-when-how.com/excel-vba/ten-vba-tips-and-tricks/

ttps://www.tutorialspoint.com/vba/vba_sub_procedure.htm

ttps://powerspreadsheets.com/vba-sub-procedures/

ttps://docs.microsoft.com/en-us/office/vba/language/concepts/getting-started/calling-sub-and-function-procedures

ttps://www.excelfunctions.net/vba-functions-and-subroutines.html

ttps://powerspreadsheets.com/vba-sub-procedures/