Macros Examples

Using the Macro Recorder

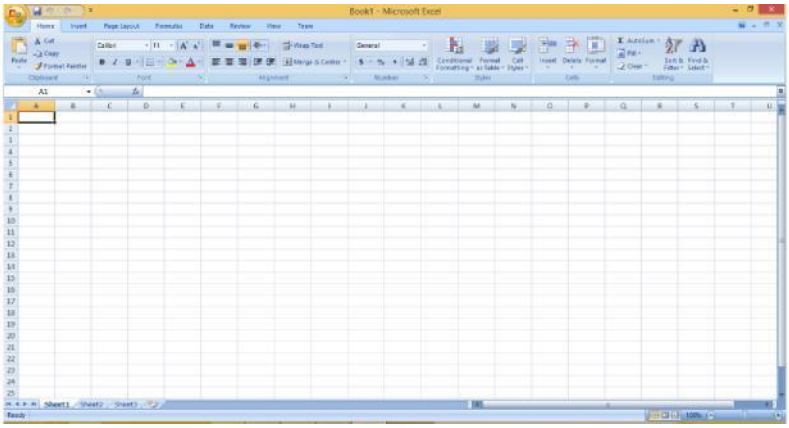
Excel's macro recorder operates very much like the recorder that stores the greeting on your telephone answering machine. To record a greeting, you first prepare yourself by rehearsing the greeting to ensure that it says what you want. Then you switch on the recorder and deliver the greeting. When you have finished, you switch off the recorder. You now have a recording that automatically plays when you leave a call unanswered.

Recording Macros

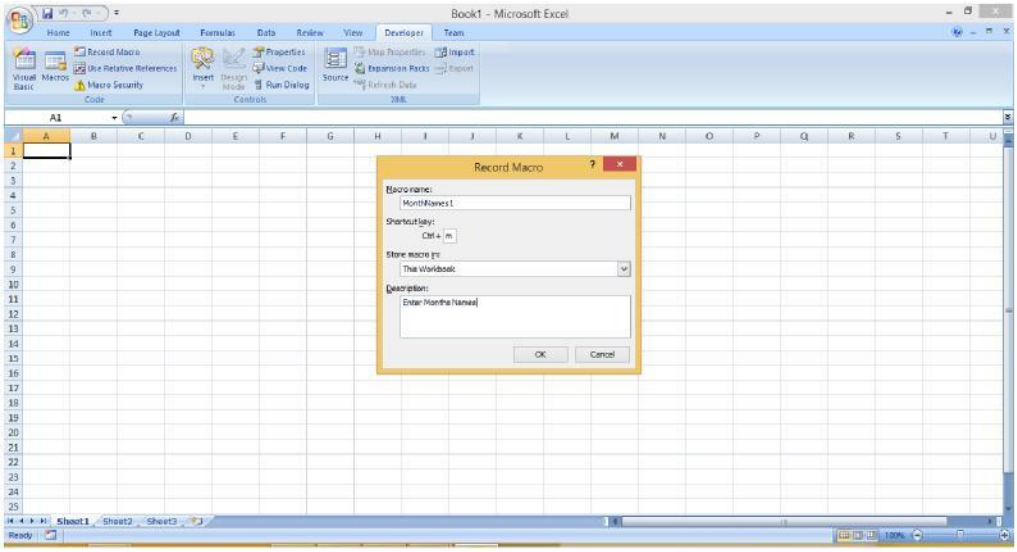
Say you want a macro that types six month names as three-letter abbreviations, Jan to Jun, across the top of your worksheet, starting in cell B1. I know this is rather a silly macro because you could do this easily with an AutoFill operation, but this example will serve to show you some important general concepts:

- First, think about how you are going to carry out this operation. In this case, it is easy—you will just type the data across the worksheet. Remember, a more complex macro might need more rehearsals before you are ready to record it. Next, think about when you want to start recording. In this case, you should include the selection of cell B1 in the recording, because you want to always have Jan in B1. If you don't select B1 at the start, you will record typing Jan into the active cell, which could be anywhere when you play back the macro.
- Next, think about when you want to stop recording. You might first want to include some formatting such as making the cells bold and italic, so you should include that in the recording. Where do you want the active cell to be after the macro runs? Do you want it to be in the same cell as Jun, or would you rather have the active cell in column A or column B, ready for your next input? Assume that you want the active cell to be A2, at the completion of the macro, so you will select A2 before turning off the recorder.
- For video training please search ebay.in as Excel Macros Video Training and you will get the all in one training drive which includes a full course step by step, from basic to advanced level. A must for every beginner and professional.
- Now you can set up your screen, ready to record.

  In this case, start with an empty worksheet with cell A1 selected. If you can't see the Developer tab above the Ribbon, you will need to click the round Microsoft Office button that you can see in the top left corner of the Excel screen shown in below Figure.



Click Excel Options at the bottom of the dialog box and select Personalize. Select the checkbox for Show Developer tab in the Ribbon and click OK. Now you can select the Developer section of the Ribbon and click Record Macro to display the Record Macro dialog box, shown in below Figure.



In the Macro name: box, replace the default entry, such as Macro1, with the name you want for your macro. The name should start with a letter and contain only letters, numbers, and the underscore character, with a maximum length of 255 characters. The macro name must not contain special characters such as exclamation points (!) or question marks (?), nor should it contain blank spaces. It is also best to use a short but descriptive name that you will recognize later. You can use the underscore character to separate words, but it is easy to just use capitalization to distinguish words.

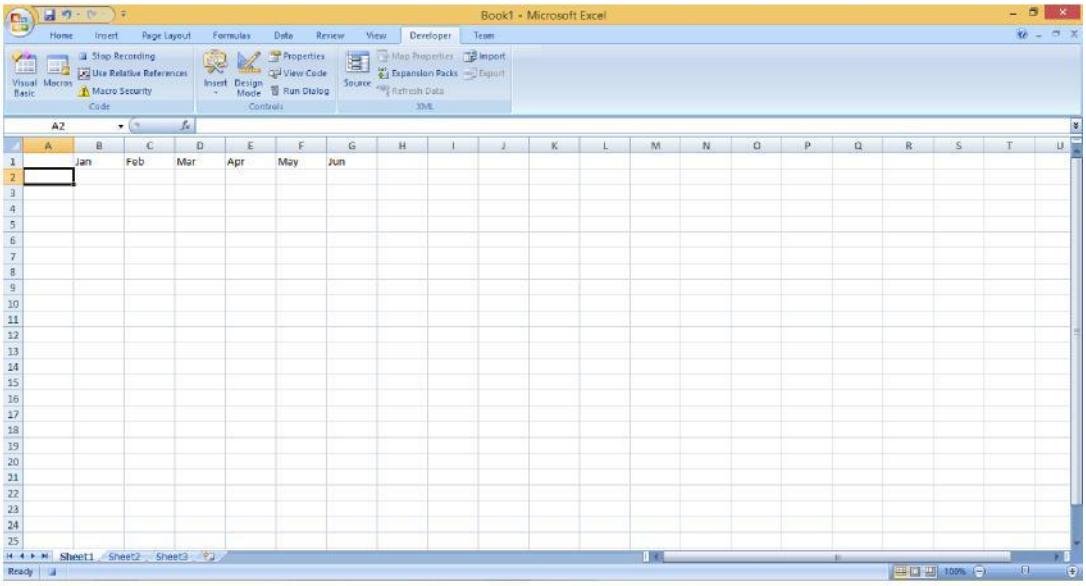
Call the macro MonthNames1, because you will create another version later. In the Shortcut key: box, you can type in a single letter. This key can be pressed later, while holding down the Ctrl key, to run the macro. Use a lowercase m. alternatively, you can use an uppercase M.

In this case, when you later want to run the macro, you need to use the keystroke combination Ctrl+Shift+M. It is not mandatory to provide a shortcut key; you can run a macro in a number of other ways, as you will see.

In the Description: box, you can add text that will be added as comments to the macro. These lines will appear at the top of your macro code. They have no significance to VBA, but provide you and others with information about the macro.

All Excel macros are stored in workbooks. You are given a choice regarding where the recorded macro will be stored. The Store macro in: combo box lists three possibilities. If you choose New Workbook, the recorder will open a new empty workbook for the macro. Personal Macro Workbook refers to a special hidden workbook, which is discussed in a moment. Choose This Workbook to store the macro in the currently active workbook.

When you have filled in the Record Macro dialog box, click the OK button. You will see a new Stop Recording button appear on the left side of the status bar at the bottom of the screen, as shown in below given Figure. You will also notice that the Start Recording button in the Ribbon has been replaced by anew Stop Recording button.



You should now click cell B1, type in Jan, and fill in the rest of the cells as shown in above Figure. Then select B1:G1 and click the Bold and Italic buttons on the Home tab of the Ribbon. Click the A2 cell and then stop the recorder. You can stop the recorder by clicking the Stop Recording button on the Ribbon or by clicking the Stop Recording button on the status bar.

You could now save the workbook, but before you do so, you should determine the file type you need and consider the security issues covered in the next section.

You can't save the workbook as the default Excel Workbook (*.xlsx) type. This file format does not allow macros to be included. You can save the workbook as an Excel Macro-Enabled Workbook (*.xlsm) type, which is in XLM format, or you can save it as an Excel Binary Workbook (*.xlsb) type, which is in a binary format. Neither of these file types is compatible with previous versions of Excel. Another alternative is to save the workbook as an Excel 97-2003 Workbook (*.xls) type, which produces a workbook compatible with Excel versions from Excel 97 through Excel 2003.

Protecting a Macro Code

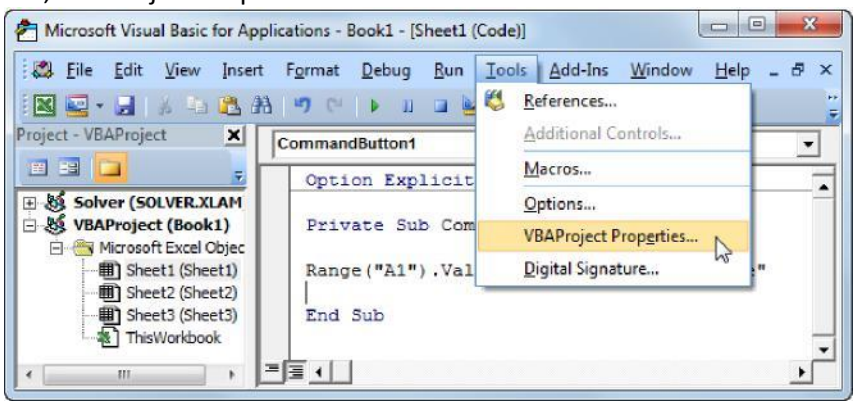
Just like you can password protect workbooks and worksheets, you can password protect a macro in Excel from being viewed (and executed).

Place a command button on your worksheet and add the following code lines:

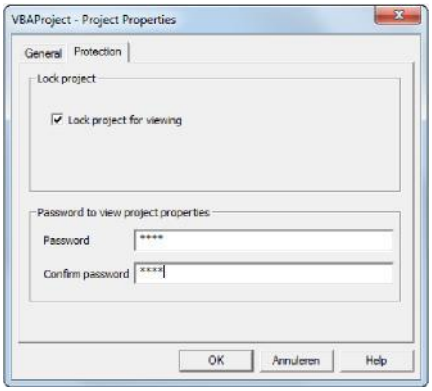
1. First, create a simple macro that you want to protect.

```
Range("A1").Value = "This is secret code"
```

2. Next, click Tools, VBAProject Properties...



3. On the Protection tab, check "Lock project for viewing" and enter a password twice.



4. Click OK.
5. Save, close and reopen the Excel file. Try to view the code.

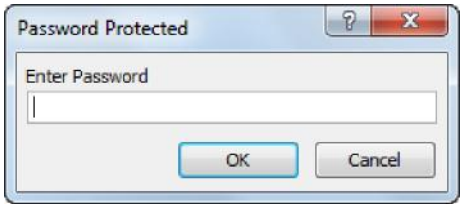
The following dialog box will appear:

You can still execute the code by clicking on the command button but you cannot view or edit the code anymore (unless you know the password). The password for the downloadable Excel file is "easy".

6. If you want to password protect the macro from being executed, add the following code lines:

```
Dim password As Variant
password = Application.InputBox("Enter Password", "Password Protected")

Select Case password
  Case Is = False
  'do nothing
  Case Is = "easy"
    Range("A1").Value = "This is secret code"
  Case Else
    MsgBox "Incorrect Password"
End Select
```

Result when you click the command button on the sheet:



Explanation: The macro uses the InputBox method of the Application object. If the users clicks Cancel, this method returns False and nothing happens (InputBox disappears). Only when the user knows the password ("easy" again), the secret code will be executed. If the entered password is incorrect, an MsgBox is displayed. Note that the user cannot take a look at the password in the Visual Basic Editor because the project is protected from being viewed.

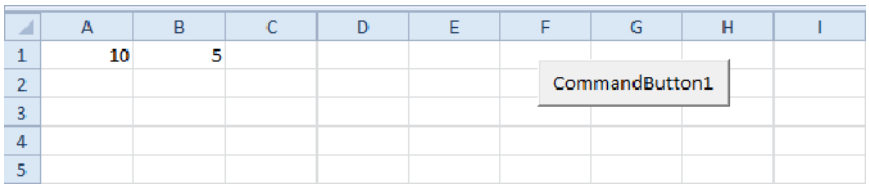
VBA Examples

Swap 2 cells Values

This example teaches you how to swap two values in Excel VBA. You will often need this structure in more complicated programs as we will see later.

Situation:

Two values on your worksheet.



Place a command button on your worksheet and add the following code lines:

1. First, we declare a variable called temp of type Double.

```
Dim temp As Double
```

2. We initialize the variable temp with the value of cell A1.

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

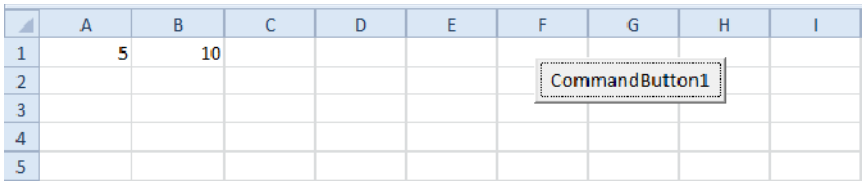
3. Now we can safely write the value of cell B1 to cell A1 (we have stored the value of cell A1 to temp so we will not lose it).

```
Range("A1").Value = Range("B1").Value
```

4. Finally, we write the value of cell A1 (written to temp) to cell B1.

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

5. Click the command button two times.
   Result:



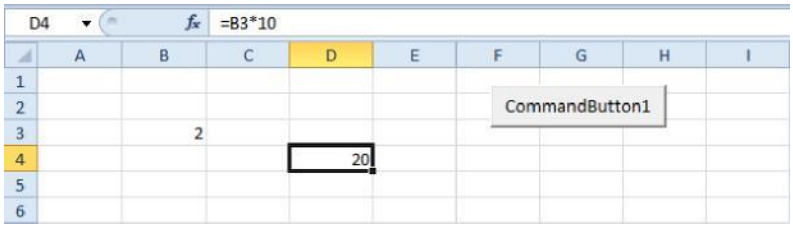
Use of FormulaR1C1

This example illustrates the difference between A1, R1C1 and R[1]C[1] style in Excel VBA.

1. Place a command button on your worksheet and add the following code line (A1 style):

```
Range("D4").Formula = "=B3*10"
```

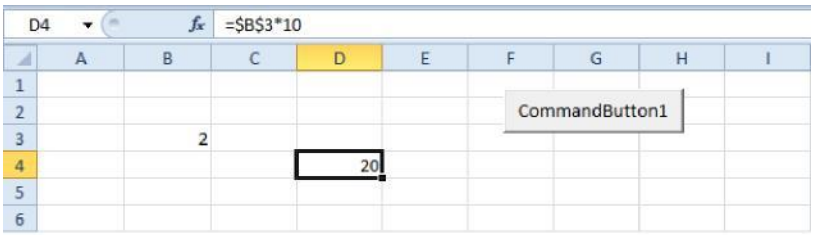
Result:



2. Add the following code line (R1C1 style):

```
Range("D4").FormulaR1C1 = "=R3C2*10"
```

Result:

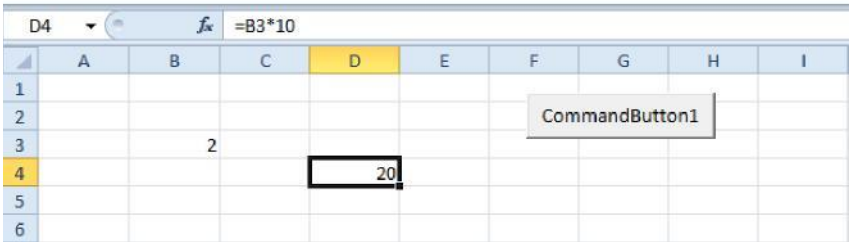


Explanation: cell D4 references cell B3 (row 3, column 2). This is an absolute reference ($ symbol in front of the row number and column letter).

3. Add the following code line (R[1]C[1] style):

```
Range("D4").FormulaR1C1 = "=R[-1]C[-2]*10"
```

Result:



Explanation: cell D4 references cell B3 (one row above and 2 columns to the left). This is a relative reference. This code line gives the exact same result as the code line used at step 1.

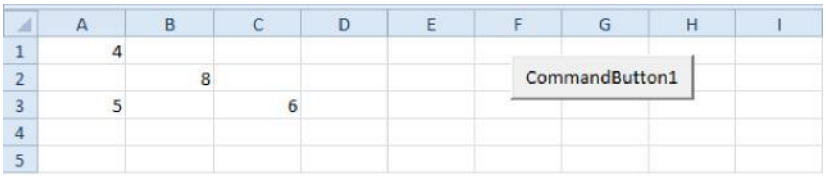
4. Why learning about this? Because the Macro Recorder uses the FormulaR1C1 property (R[1]C[1] style). The Macro Recorder creates the following code lines if you enter the formula =B3*10 into cell D4.

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

'
    Range("D4").Select
    ActiveCell.FormulaR1C1 = "=R[-1]C[-2]*10"
    Range("D5").Select
End Sub
```

Explanation: you can see that this is the exact same code line used at step 3.

Use of CurrentRegion Property

This example illustrates the CurrentRegion property in Excel VBA. The current region is a range bounded by any combination of blank rows and blank columns. Can you find the current region of cell A1?

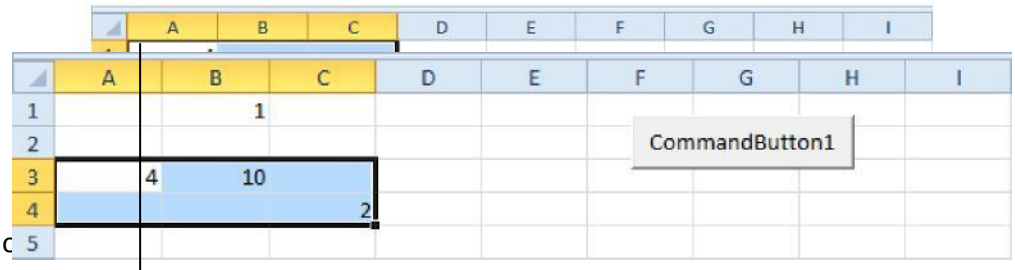


Place a command button on your worksheet and add the following code line:

```
Range("A1").CurrentRegion.Select

Range("B3").CurrentRegion.Select
```

Result when you click the command button on the sheet:

Result:



Here is anc

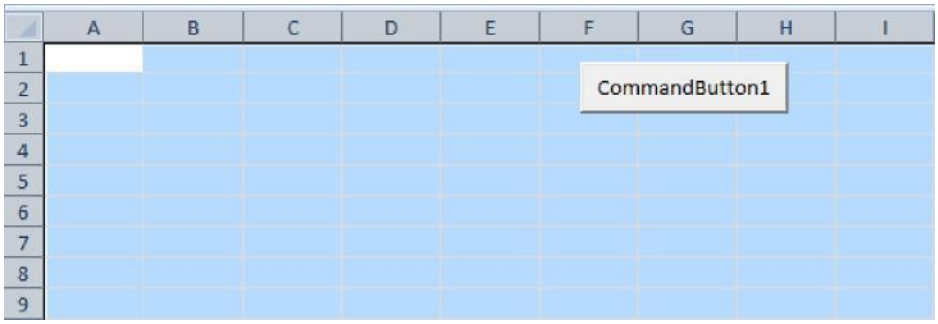
Selection of Entire Rows and Columns

]

This example teaches you how to select entire rows and columns in Excel VBA. Are you ready?
Place a command button on your worksheet and add the following code lines:

1. The following code line selects the entire sheet.

```
Cells.Select
```

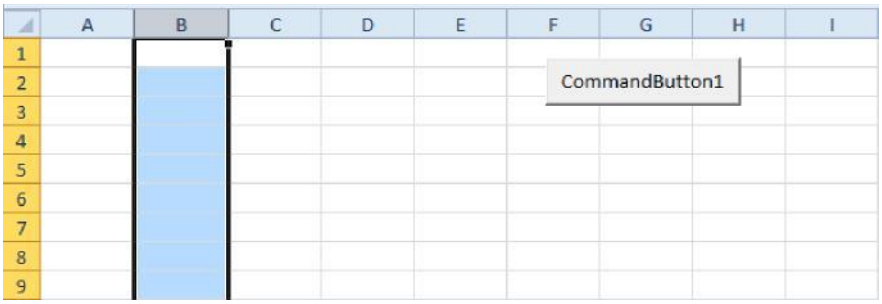


Note: because we placed our command button on the first worksheet, this code line selects the entire first sheet. To select cells on another worksheet, you have to activate this sheet first. For example, the following code lines select the entire second worksheet.

```
Worksheets(2).Activate
Worksheets(2).Cells.Select
```

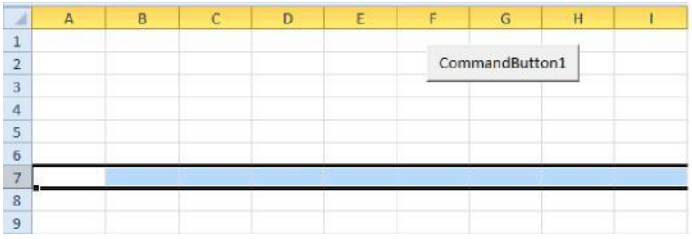
2. The following code line selects the second column.

```
Columns(2).Select
```



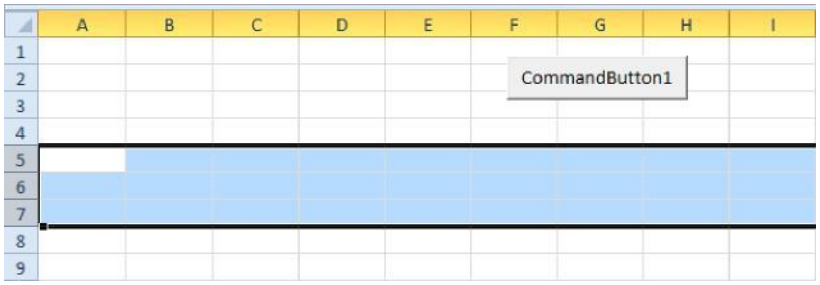
3. The following code line selects the seventh row.

```
Rows(7).Select
```



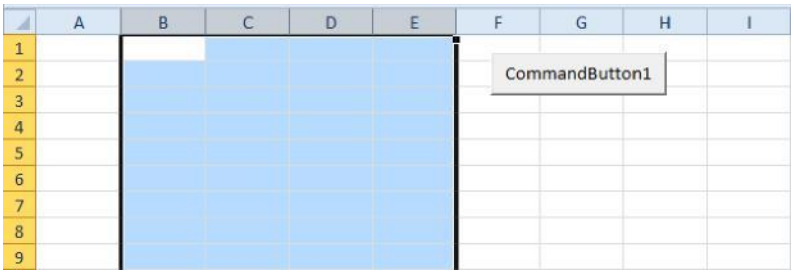
4. To select multiple rows, add a code line like this:

```
Rows("5:7").Select
```



5. To select multiple columns, add a code line like this:

```
Columns("B:E").Select
```



6. Be careful not to mix up the Rows and Columns properties with the Row and Column properties. The Rows and Columns properties return a Range object. The Row and Column properties return a single value.

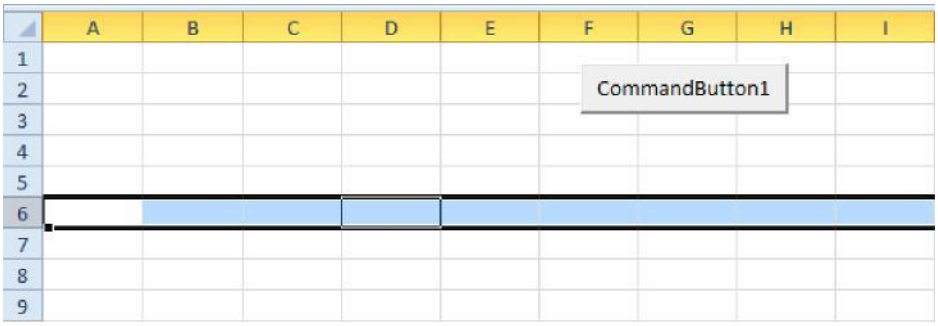
Code line:

```
MsgBox Cells(5, 2).Row
```

Result:

7. Select cell D6. The following code line selects the entire row of the active cell.

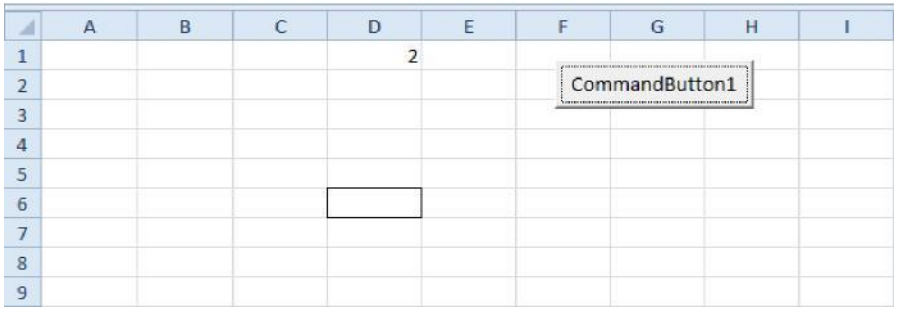
```
ActiveCell.EntireRow.Select
```



Note: border for illustration only.

8. Select cell D6. The following code line enters the value 2 into the first cell of the column that contains the active cell.

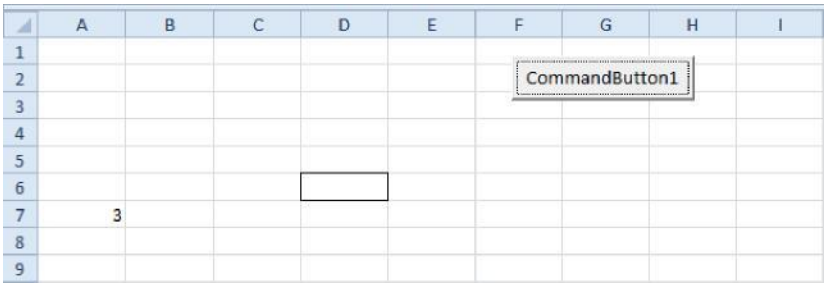
```
ActiveCell.EntireColumn.Cells(1).Value = 2
```



Note: border for illustration only.

9. Select cell D6. The following code line enters the value 3 into the first cell of the row below the row that contains the active cell.

```
ActiveCell.EntireRow.Offset(1, 0).Cells(1).Value = 3
```



Note: border for illustration only.

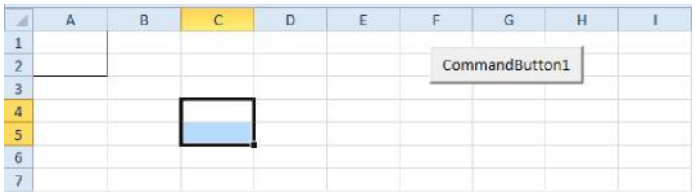
Use of Offset Property

The Offset property in Excel VBA takes the range which is a particular number of rows and columns away from a certain range (border below for illustration only).

Place a command button on your worksheet and add the following code lines:

```
Dim example As Range
Set example = Range("A1:A2")

example.Offset(3, 2).Select
```

Result when you click the command button on the sheet:



Explanation: these code lines select the range which is 3 rows below and 2 columns to the right of Range("A1:A2"). The Offset property always takes the top left cell of a range as the starting point.

Retrieve Path and FullName

The Path property in Excel VBA returns the complete, saved path to the workbook (Excel file). The FullName property in Excel VBA returns the complete, saved path, including the name of the workbook.

Place a command button on your worksheet and add the following code lines:

1. The following code line returns the complete path to path-fullname.xls.

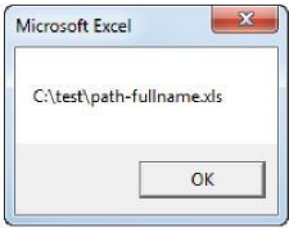
```
MsgBox Workbooks("path-fullname.xls").Path
```

Result:

2. The following code line returns the complete path, including the name of the active workbook.

```
MsgBox ActiveWorkbook.FullName
```

Result:



For a practical example of the FullName property, see our example program Create a Footer Before Printing.

Close and Open workbook

The Close and Open Method in Excel VBA can be used to close and open workbooks. Remember, the Workbooks collection contains all the Workbook objects that are currently open.

Place a command button on your worksheet and add the following code lines:

1. The code line below closes close-open-workbooks.xls.

```
Workbooks("close-open-workbooks.xls").Close
```

2. The code line below closes the first opened/created workbook.

```
Workbooks(1).Close
```

3. The code line below closes the active workbook.

```
ActiveWorkbook.Close
```

4. The code line below closes all workbooks that are currently open.

```
Workbooks.Close
```

5. The code line below opens sales.xls.

```
Workbooks.Open ("sales.xls")
```

Note: you can only open sales.xls without specifying the file's path if it's stored in your default file location. The default file location is the folder you see when you open or save a file.

6. You can also use the GetOpenFilename method of the Application object to display the standard open Dialog box and select the file (without actually opening the file).

```
Dim MyFile As String
MyFile = Application.GetOpenFilename()
```

Result:

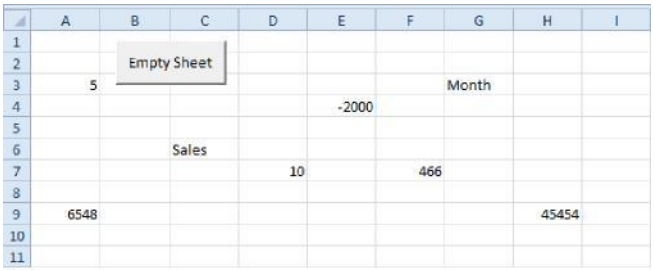
7. Next, you can open the workbook as usual.

```
Workbooks.Open (MyFile)
```

Use of MsgBox Function

The MsgBox function in Excel VBA can return a result while a simple MsgBox cannot.

Situation:



Place a command button on your worksheet and add the following code lines:

1. First, we declare a variable called answer of type Integer.

```
Dim answer As Integer
```

2. We use the MsgBox function to initialize the variable answer with the input from the user.

The MsgBox function, when using parentheses, has three arguments. The first part is used for the message in the message box. Use the second part to specify which buttons and icons you want to appear in the message box. The third part is displayed in the title bar of the message box.

```
answer = MsgBox("Are you sure you want to empty the sheet?", vbYesNo + vbQuestion, "Empty Sheet")
```

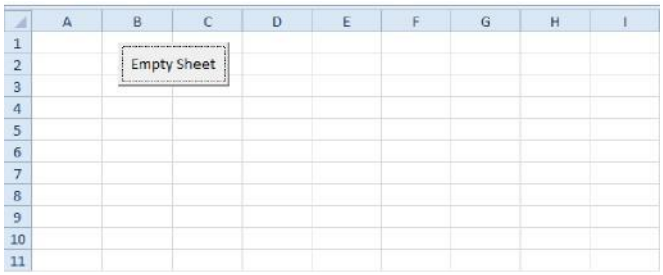
Note: Place your cursor on vbYesNo in the Visual Basic Editor and click F1 to see which other buttons and icons you can use. Instead of the constants vbYesNo and vbQuestion, you can also use the corresponding values 4 and 32.

3. If the user clicks the Yes button, Excel VBA empties the sheet. If the user clicks the No button, nothing happens. Add the following code lines to achieve this.

```
If answer = vbYes Then
 Cells.ClearContents
Else
 'do nothing
End If
```

4. Click the command button on the sheet.
5. Click Yes.

Result:



InputBox Function

You can use the InputBox function in Excel VBA to prompt the user to enter a value. Place a command button on your worksheet and add the following code lines:

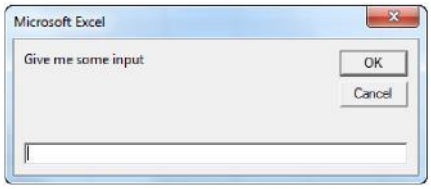
1. First, declare the variable myValue of type Variant.

```
Dim myValue As Variant
```

Note: we use a variable of type Variant here because a Variant variable can hold any type of value. This way the user can enter text, numbers, etc.

2. Add the following code line to show the input box.

```
myValue = InputBox("Give me some input")
```



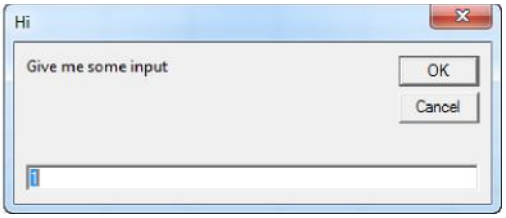
3. Write the value of myValue to cell A1.

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

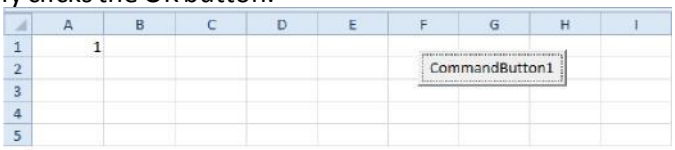
Result when the user enters the value 5 and clicks the OK button.

4. The InputBox function has more optional arguments. The following code line shows an input box with a title displayed in the title bar and has a default value. The default value will be used if no other input is provided.

```
myValue = InputBox("Give me some input", "Hi", 1)
```



Result when the user only clicks the OK button.



Note: Place your cursor on InputBox in the Visual Basic Editor and click F1 for help on the other optional arguments.

Loop through Workbooks and Worksheets

Below we will look at a program in Excel VBA that loops through all open workbooks and worksheets, and displays all the names.

Situation:

Add the following code lines to the command button:

1. First, we declare two objects and one variable. One object of type Workbook we call book, one object of type Worksheet we call sheet, and a variable of type String we call text.

```
Dim book As Workbook, sheet As Worksheet, text As String
```

2. We want to loop through all open workbooks. To achieve this, add the following code line:

```
For Each book In Workbooks
```

3. We write the text "Workbook: ", the name of the workbook, and the text "Worksheets: "" to the variable text.

```
text = text & "Workbook: " & book.Name & vbNewLine & "Worksheets: " & vbNewLine
```

Note: you can use “&”operator to concatenate (join) elements. To start a new line, you can use vbNewLine.

4. To loop through all the worksheets of a workbook, add the following code line:

```
For Each sheet In book.Worksheets
```

5. We write the names of the worksheets of a workbook to the variable text.

```
text = text & sheet.Name & vbNewLine
```

6. Close the second loop.

```
Next sheet
```

7. Add a white line.

```
text = text & vbNewLine
```

8. Don't forget to close the first loop.

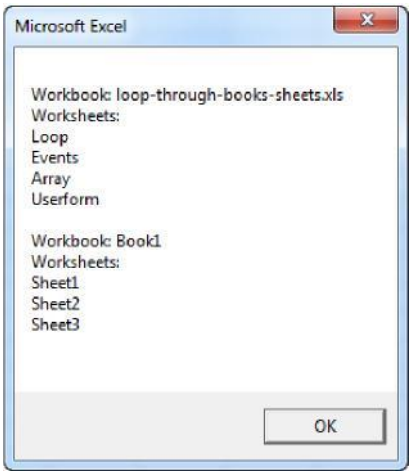
```
Next book
```

9. Finally, we display the variable text using a MsgBox.

```
MsgBox text
```

10. Test the program. Before you click on the command button, give your worksheets some descriptive names and open another blank workbook.

Result:



Sales Calculator

Below we will look at a program in Excel VBA that calculates the total sales of each employee over a period of three years.

Situation:

The other two sheets have the same setup, but with different combinations of months and employees, and different sales numbers. There are several ways to calculate the total sales of each employee in Excel, but we will see that it can be done in Excel VBA very easily.

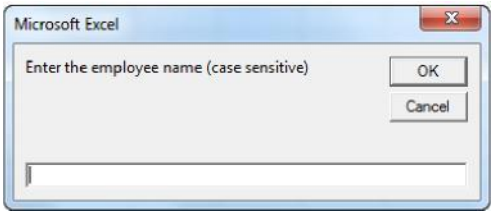
Place a command button on your worksheet and add the following code lines:

1. First, we declare three variables and one Worksheet object. One variable of type String we call employee, one variable of type Integer we call total, one Worksheet object we call sheet, and one variable of type Integer we call i.

```
Dim employee As String, total As Integer, sheet As Worksheet, i As Integer
```

2. We initialize two variables. We initialize the variable total with value 0. We use the InputBox function to get the employee name from the user.

```
total = 0
employee = InputBox("Enter the employee name (case sensitive)")
```



3. After the user has entered an employee name, we want to calculate the total sales of this employee. The workbook consists of three sheets. We want a program that can still be used if sheets are added in the future. Therefore we use the following code line:

```
For Each sheet In Worksheets
```

4. We start another For Next loop.

```
For i = 2 To 13
```

5. If the entered employee name matches with the employee name in column B, Excel VBA adds the sales number to the variable total. Add the following code lines:

```
If sheet.Cells(i, 2).Value = employee Then
total = total + sheet.Cells(i, 3).Value
End If
```

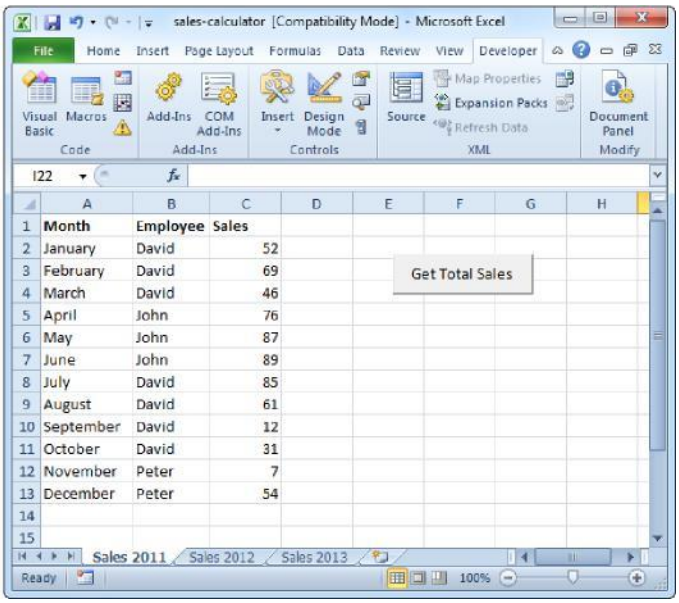
6. Don't forget to close both loops.

```
  Next i
Next sheet
```

7. Finally, we display the total sales of the employee using an msgbox.

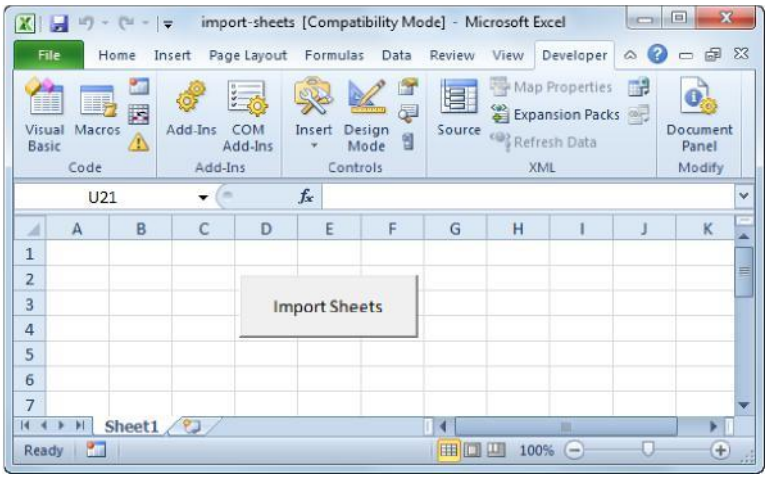
```
MsgBox "Total sales of "& employee & " is " & total
```

8. Test the program.
Result:



Import Sheets

Below we will look at a program in Excel VBA that imports sheets from other Excel files into one Excel file.
Situation:



Add the following code lines to the command button:

1. First, we declare two variables of type String, a Worksheet object and one variable of type Integer.

```
Dim directory As String, fileName As String, sheet As Worksheet, total As Integer
```

2. Turn off screen updating and displaying alerts.

```
Application.ScreenUpdating = False
Application.DisplayAlerts = False
```

3. Initialize the variable directory. We use the Dir function to find the first *.xl?? file stored in this directory.

```
directory = "c:\test\"
fileName = Dir(directory & "*.xl??")
```

Note: The Dir function supports the use of multiple character (*) and single character (?) wildcards to search for all different type of Excel files.

4. The variable fileName now holds the name of the first Excel file found in the directory. Add a Do While Loop.

```
Do While fileName <> ""

Loop
```

Add the following code lines (at 5, 6, 7 and 8) to the loop.

5. There is no simple way to copy worksheets from closed Excel files. Therefore we open the Excel file.

```
Workbooks.Open (directory & fileName)
```

6. Import the sheets from the Excel file into import-sheet.xls.

```
For Each sheet In Workbooks(fileName).Worksheets
total = Workbooks("import-sheets.xls").Worksheets.count
Workbooks(fileName).Worksheets(sheet.Name).Copy _
after:=Workbooks("import-sheets.xls").Worksheets(total)
Next sheet
```

Explanation: the variable total holds track of the total number of worksheets of import-sheet.xls. We use the Copy method of the Worksheet object to copy each worksheet and paste it after the last worksheet of import-sheets.xls.

7. Close the Excel file.

```
Workbooks(fileName).Close
```

8. The Dir function is a special function. To get the other Excel files, you can use the Dir function again with no arguments.

```
fileName = Dir()
```

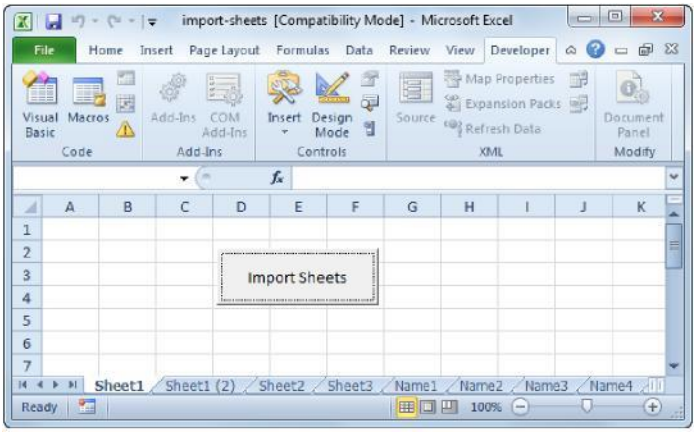
Note: When no more file names match, the Dir function returns a zero-length string (""). As a result, Excel VBA will leave the Do While loop.

9. Turn on screen updating and displaying alerts again (outside the loop).

```
Application.ScreenUpdating = True
Application.DisplayAlerts = True
```

10. Test the program.

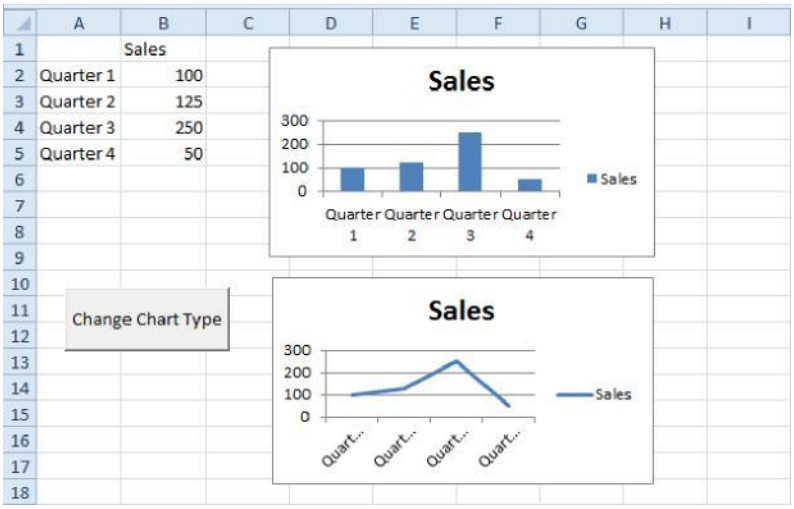
Result:



Charts Programming

Below we will look at two programs in Excel VBA. One program loops through all charts on a sheet and changes each chart to a pie chart. The other program changes some properties of the first chart.

1. Create some charts



Place a command button on the worksheet and add the following code lines:

1. First, we need to declare a ChartObject object. The ChartObject object acts as a container for a Chart object. We call the ChartObject cht but you can use any name.

```
Dim cht As ChartObject
```

2. The ChartObjects collection contains all the embedded charts on a single sheet. We want to loop through all charts on the first sheet. Add the following For Each Next loop.

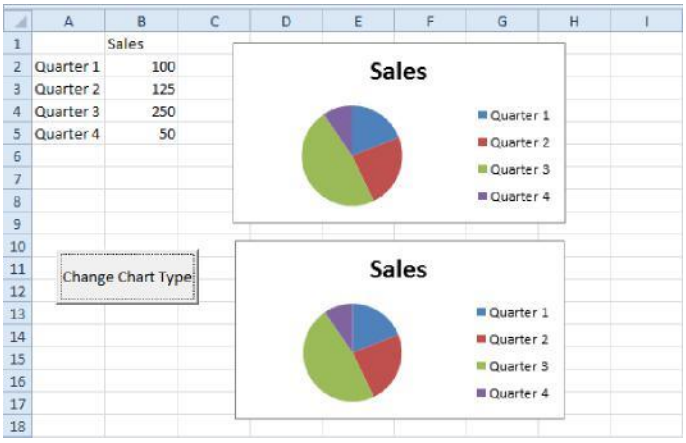
```
For Each cht In Worksheets(1).ChartObjects

Next cht
```

3. The Chart object represents a chart in a workbook. Add the following code line to the For Each Next loop to change each chart to a pie chart.

```
cht.Chart.ChartType = xlPie
```

Note: again, cht acts as a container for the Chart object. We use the ChartType property to change the chart type. We use the built-in constant xlPie to change each chart to a pie chart.

4. Result when you click the command button on the sheet:



Now we will change some properties of the first chart.

Place another command button on the worksheet and add the following code lines:

5. The ChartObjects collection contains all the embedded charts on a single sheet. Add the following code line to activate the first chart:

```
Worksheets(1).ChartObjects(1).Activate
```

We can now refer to this chart as the ActiveChart.

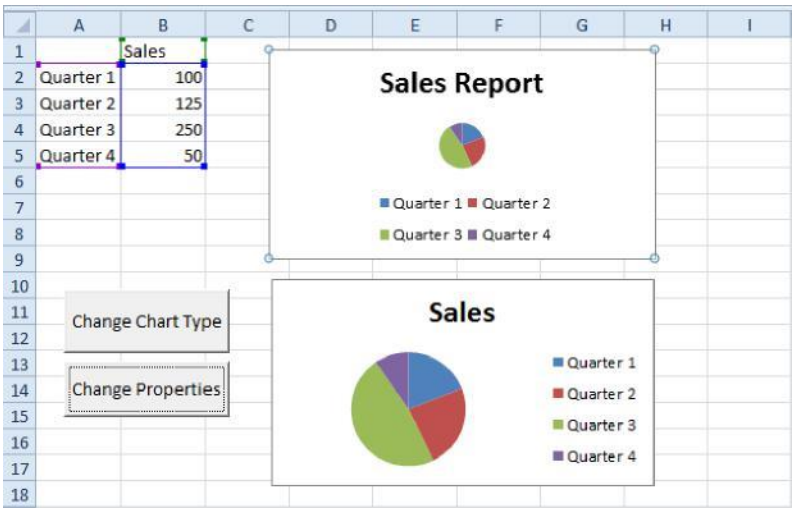
6. Add the following code line to change the Chart title.

```
ActiveChart.ChartTitle.Text = "Sales Report"
```

7. Add the following code line to move the legend to the bottom of the chart:

```
ActiveChart.Legend.Position = xlBottom
```

8. Result when you click the command button on the sheet:

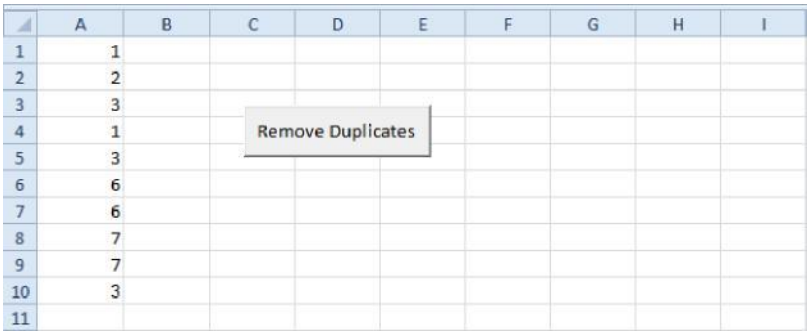


Remove Duplicate Values

Below we will look at a program in Excel VBA that removes duplicates.

Situation:

In column A we have 10 numbers. We want to remove the duplicates from these numbers and place the unique numbers in column B.



1. First, we declare four variables. toAdd of type Boolean, uniqueNumbers of type Integer, i of type Integer, and j of type Integer.

```
Dim toAdd As Boolean, uniqueNumbers As Integer, i As Integer, j As Integer
```

2. Next, we write the first number of column A to column B since the first number is always 'unique'.

```
Cells(1, 2).Value = Cells(1, 1).Value
```

3. We initialize two variables. We've just added one number to column B, so we initialize uniqueNumbers with the value 1. We set toAdd to True assuming that the next number needs to be added as well (this is not necessarily true of course).

```
uniqueNumbers = 1
toAdd = True
```

We need to determine whether the second number is 'unique' or not. This can be done in a very easy way. Only if the number is not already in column B, the second number needs to be added to column B.

4. We also need to check this for the third number, fourth number, and so on. We start a For Next loop for this.

```
For i = 2 To 10
```

5. Now comes the most important part of the program. If the second number is equal to one of the numbers in column B (so far we only have one unique number), we set toAdd to False because in this case we don't want to add this number! (it is not 'unique'). At the moment uniqueNumbers is still equal to 1, but uniqueNumbers can be a whole list. To check this whole list, we need another For Next loop. Again: if the number we want to add is equal to one of the numbers in this list, toAdd will be set to False and the number will not be added. Add the following code lines:

```
For j = 1 To uniqueNumbers
If Cells(i, 1).Value = Cells(j, 2).Value Then
    toAdd = False
End If
Next j
```

6. Only if toAdd is still True and not set to False, Excel VBA needs to add the number to column B. At the same time, we increment uniqueNumbers by 1 because we have one unique number more now. The following code lines get the job done:

```
If toAdd = True Then
Cells(uniqueNumbers + 1, 2).Value = Cells(i, 1).Value
uniqueNumbers = uniqueNumbers + 1
End If
```

7. Finally, we set toAdd to True assuming the next number (third number) needs to be added. Again this is not necessarily true.

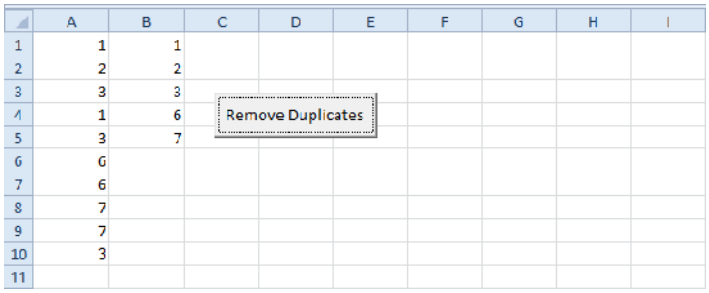
```
toAdd = True
```

8. Don't forget to close the loop.

```
Next i
```

9. Place your macro in a command button and test it.

Result:

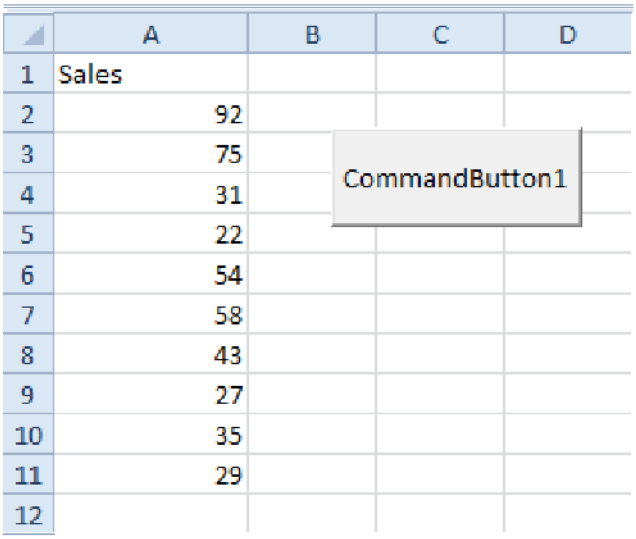


Selection From Active Cell to Last cell Entry

This example illustrates the End property of the Range object in Excel VBA. We will use this property to select the range from the Active Cell to the last entry in a column.

Situation:

Some sales figures in column A. Assume that you will be adding more sales figures over time.



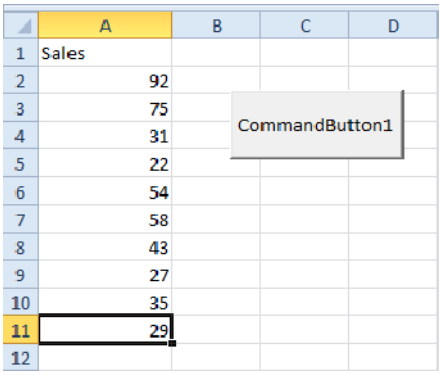
Place a command button on your worksheet and add the following code lines:

1. To select the last entry in a column, simply add the following code line:

```
Range("A5").End(xlDown).Select
```

Note: instead of Range("A5"), you can also use Range("A1"), Range("A2"), etc. This code line is equivalent to pressing the END+DOWN ARROW.

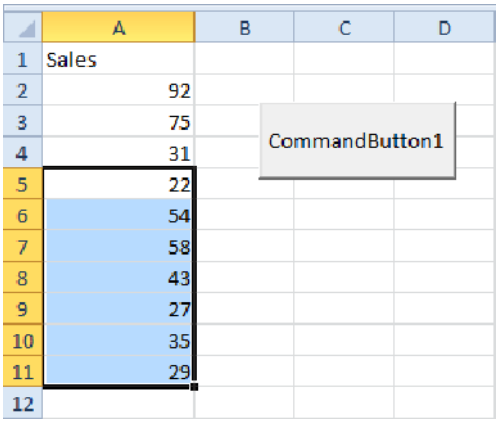
Result when you click the command button on the sheet:



2. To select the range from cell A5 to the last entry in the column, add the following code line:

```
Range(Range("A5"), Range("A5").End(xlDown)).Select
```

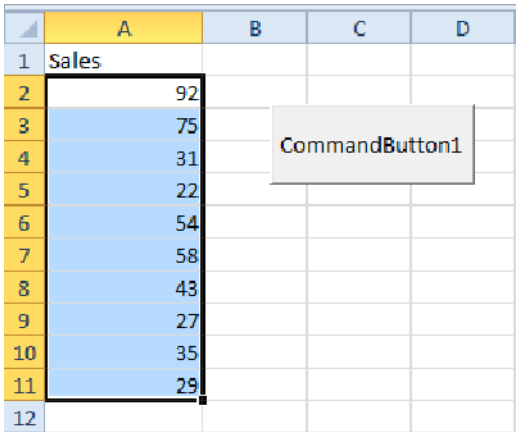
Result when you click the command button on the sheet:



3. To select the range from the Active Cell to the last entry in the column, simply replace Range("A5") with ActiveCell.

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Result when you select cell A2 and click the command button on the sheet:



Use of Font Property

The Font property of the Range object in Excel VBA gives access to a lot of other properties. That is because the Font property returns an object itself; the Font object. The Font object has many properties like the Color property and the Bold property.

Color property

To change the color of an Excel range, use the Font property of the Range object, and then the Color property of the Font object.

1. Add the following code line:

```
Range("A1").Font.Color = -16776961
```

Explanation: Where do we get this strange number from? Well, we started the Macro Recorder and changed the color of a cell to red. You can to this for every color!

2. The following code line gives the exact same result.

```
Range("A1").Font.Color = vbRed
```

Explanation: vbRed is a sort of built-in constant in Excel VBA. Place your cursor on vbRed in the Visual Basic Editor and click F1 to see which other constants you can use.

3. The following code line gives the exact same result.

```
Range("A1").Font.Color = RGB(255, 0, 0)
```

Explanation: RGB stands for Red, Green and Blue. These are the three primary colors. Each component can take on a value from 0 to 255. With this function you can make every color. RGB(255,0,0) gives the pure Red color.

Bold property

The following code line bolds a range:

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

To unbold a range, you can use the False keyword. The Font object has many more properties. If you want to program these sort of things, just use the Macro Recorder to see how to do it! Usually code created by the Macro Recorder is too long. For example, the Macro Recorder creates the following code when we bold Range("A1").

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

'
    Range("A1").Select
    Selection.Font.Bold = True
End Sub
```

Background Colors

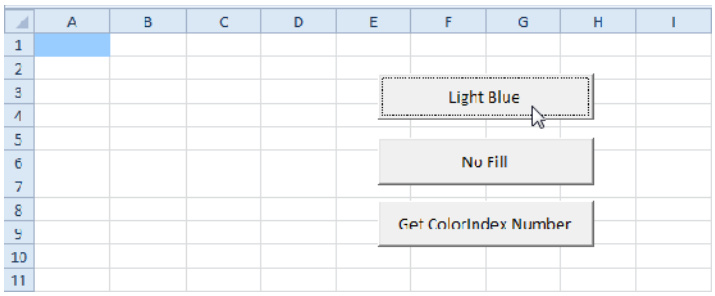
Changing background colors in Excel VBA is easy. Use the Interior property to return an Interior object. Then use the ColorIndex property of the Interior object to set the background color of a cell.

Place three command buttons on your worksheet and add the following code lines:

1. The code line below sets the background color of cell A1 to light blue.

```
Range("A1").Interior.ColorIndex = 37
```

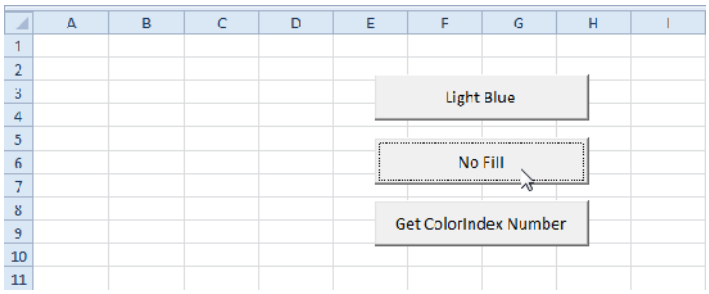
Result:



2. The following code line sets the background color of cell A1 to 'No Fill'.

```
Range("A1").Interior.ColorIndex = 0
```

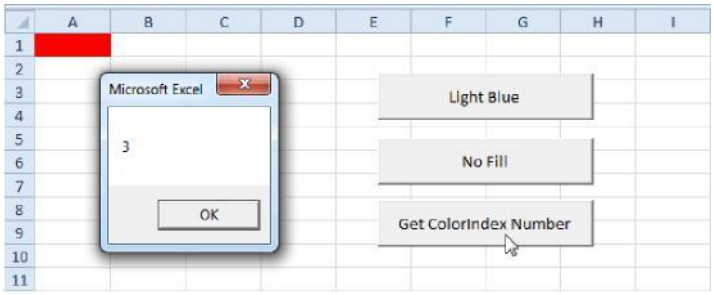
Result:



3. If you want to know the ColorIndex number of a color, simply ask Excel VBA.

```
MsgBox Selection.Interior.ColorIndex
```

Select cell A1 and click the command button on the sheet:



4. The ColorIndex property gives access to 56 "preset" colors. If you can't find the specific color you are looking for, use the Color property and the RGB function.

```
Range("A1").Interior.Color = RGB(255, 0, 0)
```

Explanation: RGB stands for Red, Green and Blue. These are the three primary colors. Each component can take on a value from 0 to 255. With this function you can make every color. RGB(255,0,0) gives the pure Red color (exact same result as above).

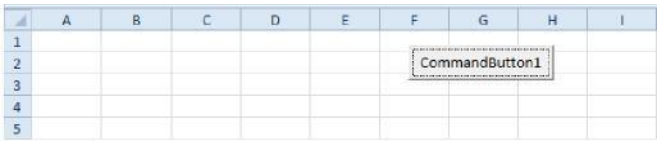
Option Explicit

We strongly recommend to use Option Explicit at the start of your Excel VBA code. Using Option Explicit forces you to declare all your variables.

For example, place a command button on your worksheet and add the following code lines:

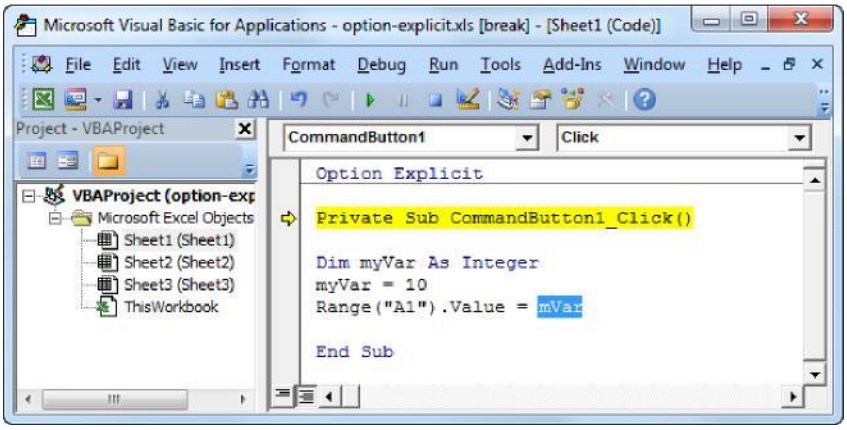
```
Dim myVar As Integer
myVar = 10
Range("A1").Value = mVar
```

Result when you click the command button on the sheet:

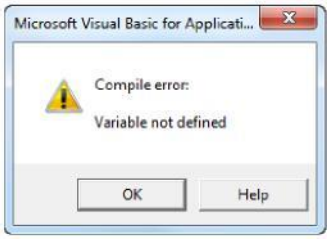


Clearly, cell A1 does not contain the value 10. That is because we accidentally misspelled myVar. As a result, Excel VBA places the value of the undeclared, empty variable mVar into cell A1.

When using Option Explicit, the code lines above generate an error because we did not declare the variable mVar.



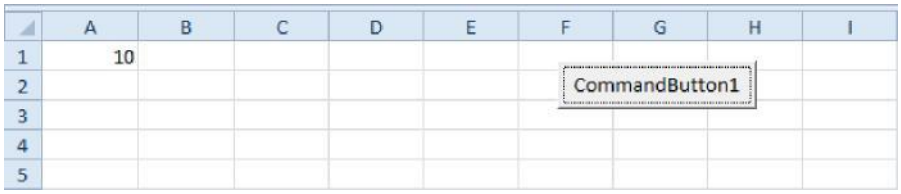
Result:



1. Click OK. Then Click on Reset (Stop) to stop the debugger.

2. Correct mVar so that it reads myVar.

Result when you click the command button on the sheet:

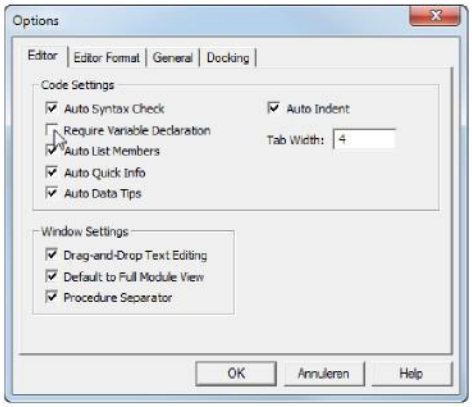


Now you know why you should always use Option Explicit at the start of your Excel VBA code. It avoids incorrectly typing the name of an existing variable.

Fortunately, you can instruct Excel VBA to automatically add Option Explicit.

1. In the Visual Basic Editor, click on Tools and then click on Options.

2. Check Require Variable Declaration.

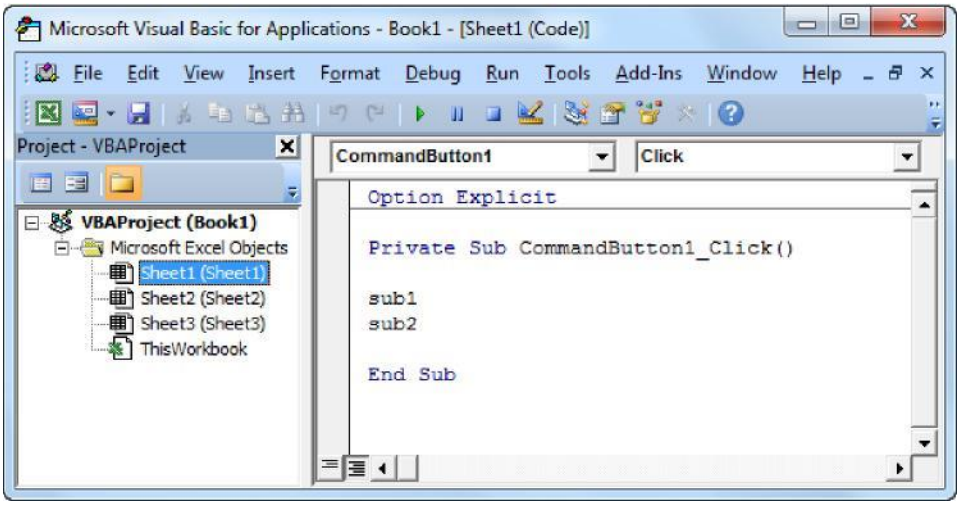


Note: Option Explicit will not be added automatically to existing Excel files. Simply type in Option Explicit yourself if you want to use it.

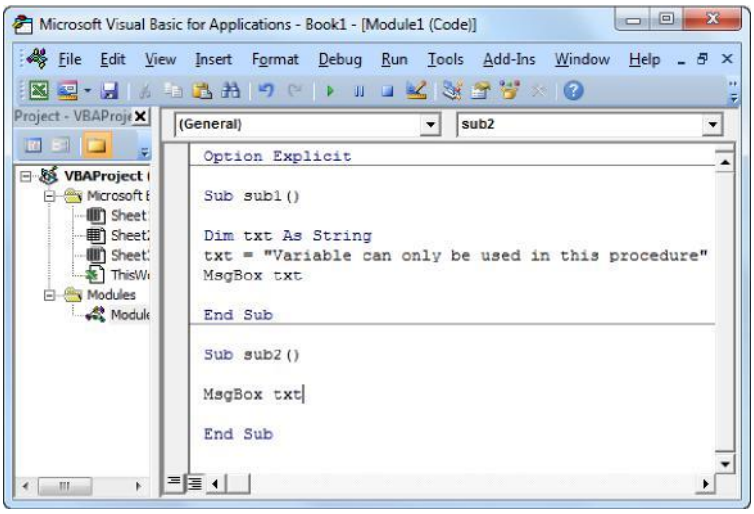
Scope of a variable

The scope of a variable in Excel VBA determines where that variable may be used. You determine the scope of a variable when you declare it. There are three scoping levels: procedure level, module level, and public module level.

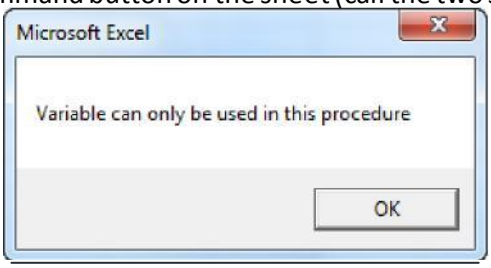
Place a command button on your worksheet and add the following code lines:

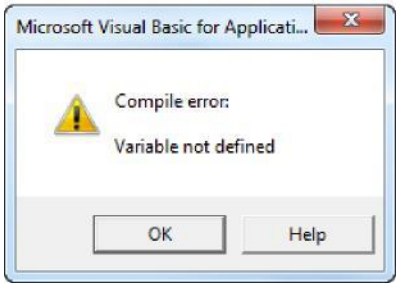


1. Place the two procedures (a procedure is either a sub or a function) into a module. In the Visual Basic Editor, click Insert, Module. Add the following code lines:



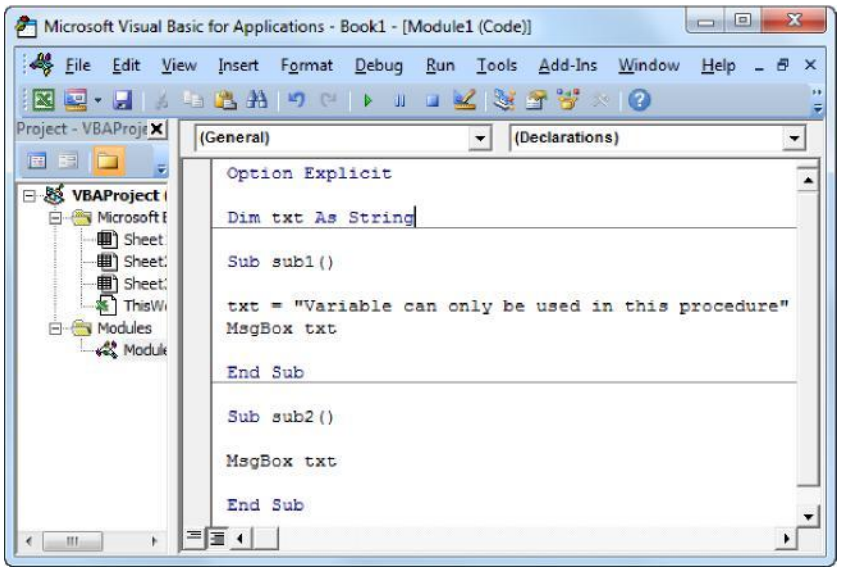
2. Result when you click the command button on the sheet (call the two subs):



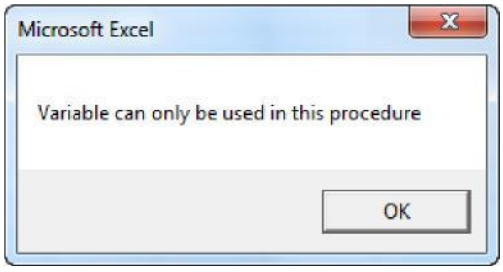


Explanation: the variable txt has scope procedure level because it is declared in the procedure (between Sub and End Sub). As a result, you can only use this variable in sub1. The variable txt cannot be used in sub2.

3. When you want a variable to be available to all procedures in a module, you are saying you want the variable to have module level scope. You need to declare the variable in the General Declarations section (at the top of the module). Slightly adjust the code as follows:

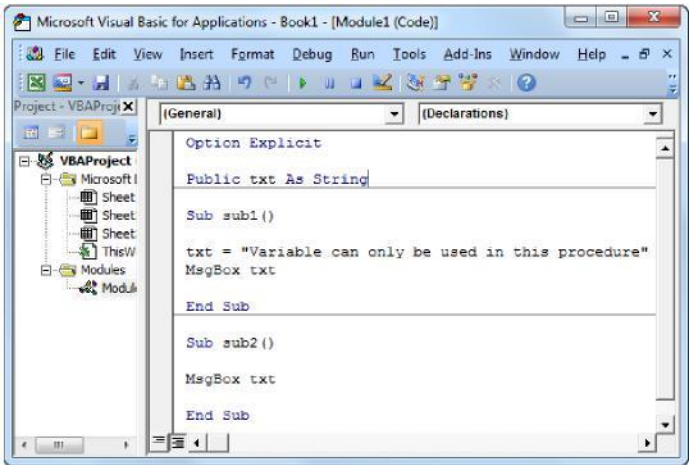


4. Result when you click the command button on the sheet:



Explanation: the variable txt can now be used in sub2. Module level is used interchangeably with private module level. That is because by default variables declared with the Dim statement in the General Declarations section are scoped as private. You can also scope a variable as public. Read on.

5. By using the Public keyword, your variable will be available to all procedures in all modules in a workbook. This is called public module level scope. Slightly adjust the code as follows:

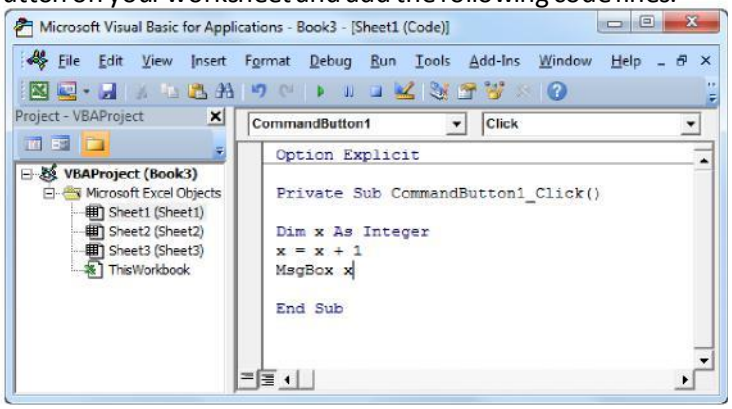


Explanation: now you can create a new module and place a sub called sub3 into this module. Use the same code as sub2. Add sub3 to your command button code. When you click the command button on the worksheet, you will get three message boxes saying "Variable can only be used in this procedure" (see downloadable Excel file).

Life of Variables

Sometimes you want to retain the value of a variable in Excel VBA when a procedure ends. You can achieve this by using the Static keyword.

1. Place a command button on your worksheet and add the following code lines:

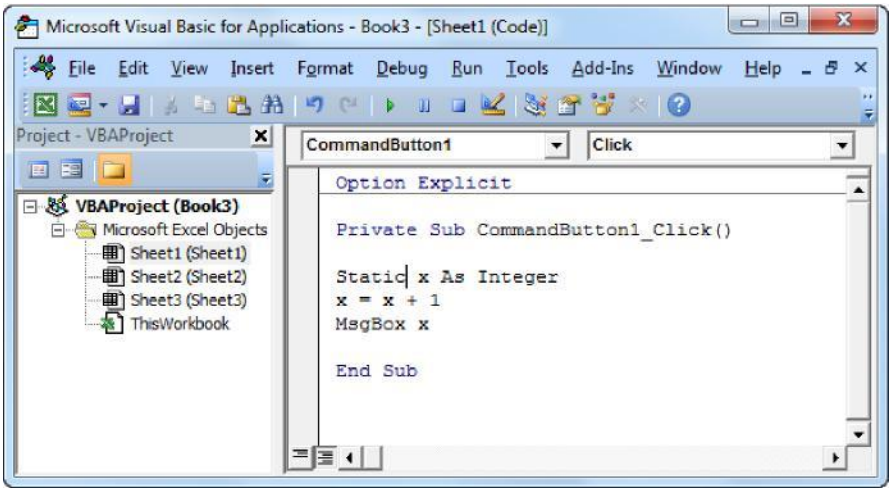


1. Result when you click the command button on the sheet:

3. Result when you click another time:

Explanation: Excel VBA destroys the variable when the procedure ends. Each time you click the command button on the sheet, Excel VBA creates the variable x again, adds the value 1 to it, and displays the result.

4. Now replace the keyword Dim with the keyword Static.



5. Result when you click the command button on the sheet:

6. Result when you click another time:

Conclusion: static variables retain their values, even when a procedure ends.

Note: static variables will be destroyed when you click the Reset (Stop) button or when you close your workbook.

Logical Operators

The three most used logical operators in Excel VBA are: And, Or and Not. As always, we will use easy examples to make things more clear.

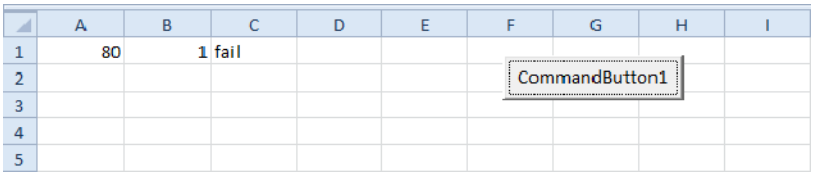
Logical Operator And

Place a command button on your worksheet and add the following code lines:

```
Dim score1 As Integer, score2 As Integer, result As String

score1 = Range("A1").Value
score2 = Range("B1").Value

If score1 >= 60 And score2 > 1 Then
  result = "pass"
Else
  result = "fail"
End If

Range("C1").Value = result
```

Explanation: if score1 is greater than or equal to 60 and score2 is greater than 1, Excel VBA returns pass, else Excel VBA returns fail.

Result when you click the command button on the sheet:



Conclusion: Excel VBA returns fail because score2 is not greater than 1.

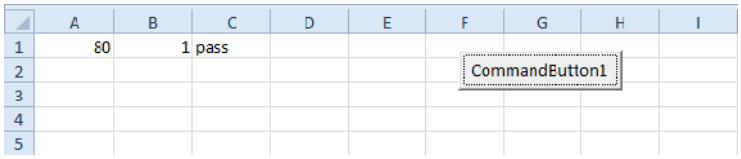
Logical Operator Or

Place a command button on your worksheet and add the following code lines:

```
Dim score1 As Integer, score2 As Integer, result As String

score1 = Range("A1").Value
score2 = Range("B1").Value

If score1 >= 60 Or score2 > 1 Then
  result = "pass"
Else
  result = "fail"
End If

Range("C1").Value = result
```

Explanation: if score1 is greater than or equal to 60 or score2 is greater than 1, Excel VBA returns pass, else Excel VBA returns fail.

Result when you click the command button on the sheet:



Conclusion: Excel VBA returns pass because score1 is greater than or equal to 60.

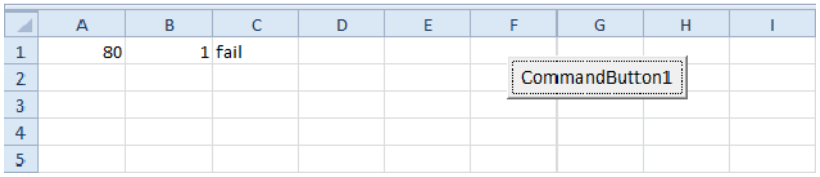
Logical Operator Not

Place a command button on your worksheet and add the following code lines:

```
Dim score1 As Integer, score2 As Integer, result As String

score1 = Range("A1").Value
score2 = Range("B1").Value

If score1 >= 60 And Not score2 = 1 Then
  result = "pass"
Else
  result = "fail"
End If

Range("C1").Value = result
```

Explanation: if score1 is greater than or equal to 60 and score2 is not equal to 1, Excel VBA returns pass, else Excel VBA returns fail.

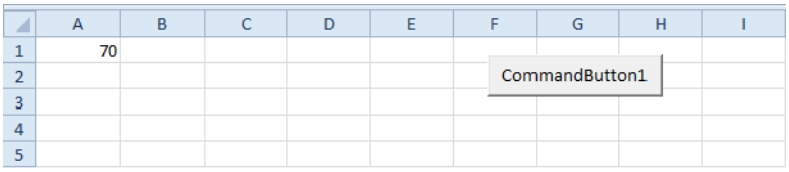
Result when you click the command button on the sheet:



Conclusion: Excel VBA returns fail because score2 is equal to 1.

Select Case

Instead of multiple If Then statements in Excel VBA, you can use the Select Case structure. Situation:



Place a command button on your worksheet and add the following code lines:

1. First, declare two variables. One variable of type Integer named score and one variable of type String named result.

```
Dim score As Integer, result As String
```

2. We initialize the variable score with the value of cell A1.

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

3. Add the Select Case structure.

```
Select Case score
Case Is >= 80
    result = "very good"
Case Is >= 70
    result = "good"
Case Is >= 60
    result = "sufficient"
Case Else
    result = "insufficient"
End Select
```

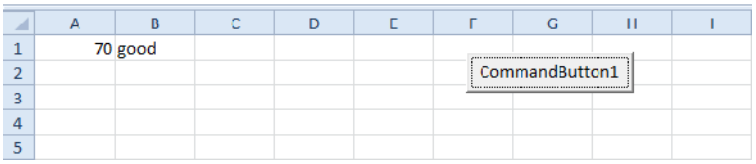
Explanation: Excel VBA uses the value of the variable score to test each subsequent Case statement to see if the code under the Case statement should be executed.

4. Write the value of the variable result to cell B1.

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

5. Test the program.

Result when you click the command button on the sheet:



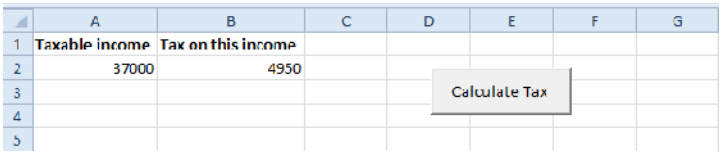
Note: Excel VBA executes the code under the second Case statement for all values greater than or equal to 70 and less than 80.

Calculate Tax Rates

Below we will look at a program in Excel VBA that calculates the tax on an income. The following tax rates apply to individuals who are residents of Australia.

Taxable income	Tax on this income
0 - $6,000	Nil
$6,001 - $35,000	15c for each $1 over $6,000
$35,001 - $80,000	$4,350 plus 30c for each $1 over $35,000
$80,001 - $180,000	$17,850 plus 38c for each $1 over $80,000
$180,001 and over	$55,850 plus 45c for each $1 over $180,000

Situation:



1. First, we declare two double variables. One double variable we call income, and one double variable we call tax.

```
Dim income As Double
Dim tax As Double
```

2. We initialize the variable income with the value of cell A2 and round it.

```
income = Round(Range("A2").Value)
```

3. We place the rounded value into cell A2 again.

```
Range("A2").Value = income
```

4. We use the Select Case statement to calculate the tax on an income. Excel VBA uses income to test each subsequent Case statement to see if the code under the Case statement should be executed.

```
Select Case income
 Case Is >= 180001
    tax = 55850 + 0.45 * (income - 180000)
 Case Is >= 80001
    tax = 17850 + 0.38 * (income - 80000)
 Case Is >= 35001
    tax = 4350 + 0.3 * (income - 35000)
 Case Is >= 6001
    tax = 0.15 * (income - 6000)
 Case Else
```

```
    tax = 0
End Select
```

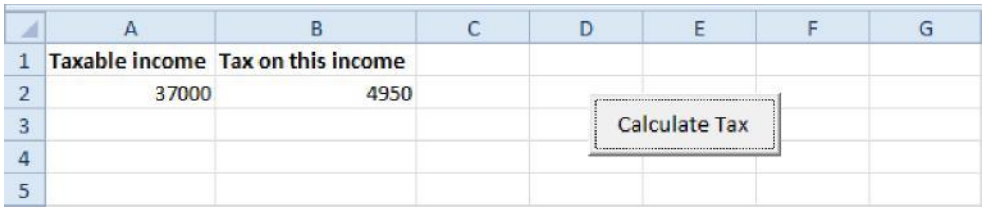
Example: if income is 37000, tax equals 4350 + 0.3 * (37000-35000) = 4350 + 600 = $4950

5. We write the value of the variable tax to cell B2.

```
Range("B2").Value = tax
```

6. Place this code in a command button and test it.

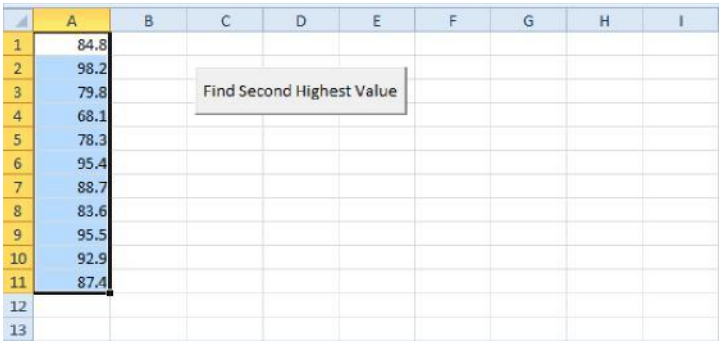
Result:



Find Second Highest Value from a list

Below we will look at a program in Excel VBA that finds the second highest

value. Situation:



1. First, we declare two Range objects and two variables of type Double. We call the Range objects rng and cell. One double variable we call highestValue, and one double variable we call secondHighestValue.

```
Dim rng As Range, cell As Range
Dim highestValue As Double, secondHighestValue As Double
```

2. We initialize the Range object rng with the selected range and the two Double variables with value 0.

```
Set rng = Selection
highestValue = 0
secondHighestValue = 0
```

3. First, we want to find the highest value. We want to check each cell in a randomly selected range (this range can be of any size). In Excel VBA, you can use the For Each Next loop for this. Add the following code lines:

```
'Find Highest Value
For Each cell In rng

Next cell
```

Note: rng and cell are randomly chosen here, you can use any names. Remember to refer to these names in the rest of your code. The green line is a comment and is only added here to provide information about this piece of code.

4. We check each cell in this range. If it's higher than highestValue, we write the value to the variable highestValue. Add the following code line to the loop.

```
If cell.Value > highestValue Then highestValue = cell.Value
```

Note: the first value is always higher than highestValue because the starting value of highestValue is 0.

5. Second, we want to find the second highest Value. We add another For Each Next loop.

```
'Find Second Highest Value
For Each cell In rng

Next cell
```

6. We check each cell in the selected range again. If it's higher than secondHighestValue and lower than highestValue, we write the value to the variable secondHighestValue. Add the following code line to the loop.

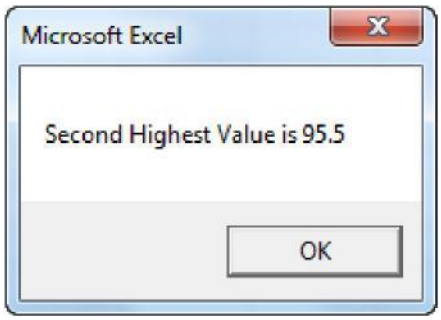
```
If cell.Value > secondHighestValue And cell.Value < highestValue ThensecondHighestValue = cell.Value
```

7. Finally, we display the second highest value using a MsgBox.

```
MsgBox "Second Highest Value is " & secondHighestValue
```

8. Place your macro in a command button, select the numbers, and click on the command

button. Result:



Sum of numbers by Color

Below we will look at a program in Excel VBA that sums numbers by color.
Situation:

You have landed money to two twelve people. Some people have given it back (in black) and some still owe you money (red). You want to know how much money you still receive.

	A	B	C	D	E	F	G	H	I
1	10								
2	11								
3	5								
4	6								
5	7								
6	5								
7	8								
8	3								
9	8								
10	4								
11	9								
12	10								
13									
14									

Sum By Color

1. First, we declare two variables of type Integer. One named toReceive and one named i. We initialize the variable toReceive with value 0.

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

2. Second, we start a For Next loop.

```
For i = 1 To 12
```

3. We now check each number and only if the color of the number is red we add the number to toReceive.

```
If Cells(i, 1).Font.Color = vbRed Then
toReceive = toReceive + Cells(i, 1).Value
End If
```

4. Don't forget to close the loop.

```
Next i
```

5. Finally, we display the money still to receive. We use the & operator to concatenate (join) two strings. Although toReceive is not a string it works here.

```
MsgBox "Still to receive "& toReceive & " dollars"
```

6. Place your macro in a command button and test it.

Result:

65534								
65535								
65536	2							

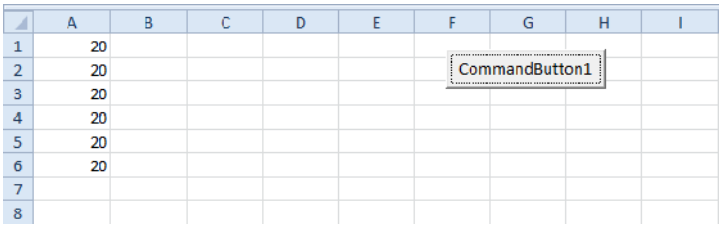
Use of Do Until Loop

Although not used very often on this site, you might find yourself in a situation where you want to use the Do Until Loop in Excel VBA. Code placed between Do Until and Loop will be repeated until the part after Do Until is true.

Place a command button on your worksheet and add the following code lines:

```
Dim i As Integer
i = 1

Do Until i > 6
 Cells(i, 1).Value = 20
 i = i + 1
Loop
```

Result when you click the command button on the sheet:

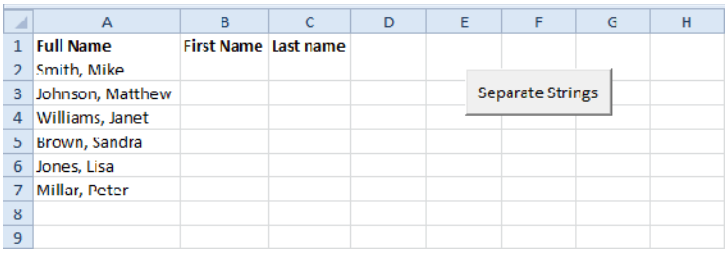


Explanation: until i is higher than 6, Excel VBA places the value 20 into the cell at the intersection of row i and column 1 and increments i by 1. As a result, the value 20 will be placed into column A six times (not seven because Excel VBA stops when i equals 7).

Separate Strings

Below we will look at a program in Excel VBA that separates strings.

Situation:



Place a command button on your worksheet and add the following code lines:

1. First, we declare a variable called fullname of type String, a variable called commaposition of type Integer, and a variable called i of type Integer.

```
Dim fullname As String, commaposition As Integer, i As Integer
```

The problem we are dealing with is that we need to tell Excel VBA where we want to separate the string. In case of Smith, Mike the comma is at position 6 while in case of Williams, Janet the comma is at position 9.

2. We use a loop to execute the operations on each name entered in Excel. First, we initialize the variable fullname. Next, we use the Instr function to find the position of the comma.

```
For i = 2 To 7
 fullname = Cells(i, 1).Value
 commaposition = InStr(fullname, ",")
```

3. Finally, we want to write the part after the comma to column B and the part in front of the comma to column C. You can achieve this by adding the lines:

```
Cells(i, 2).Value = Mid(fullname, commaposition + 2)
Cells(i, 3).Value = Left(fullname, commaposition - 1)
```

Mid(fullname, commaposition + 2) means we want the part of fullname starting at character 'commaposition + 2' (this is exactly the first name).

Left(fullname, commaposition - 1) means we want the part of fullname starting at the beginning until character 'commaposition- 1' (this is exactly the last name).

4. Don't forget to close the loop.

```
Next i
```

5. Add six names separated by a comma and space to Range("A2:A7").
6. Test the program.

Result:

	A	B	C	D	E	F	G	H
1	**Full Name**	**First Name**	**Last name**					
2	Smith, Mike	Mike	Smith					
3	Johnson, Matthew	Matthew	Johnson					
4	Williams, Janet	Janet	Williams					
5	Brown, Sandra	Sandra	Brown					
6	Jones, Lisa	Lisa	Jones					
7	Millar, Peter	Peter	Millar					
8								
9								

Separate Strings

Convert to Proper Case

Below we will look at a program in Excel VBA that converts text to proper case. That is, the first letter in each word in uppercase, and all other letters in lowercase.

Situation:

	A	B	C	D	E	F	G	H
1								
2	name:	JACK						
3	surname:	WATSON						
4	address:	KINGSTON HILL 95						
5								

Convert To Proper Case

1. First, we declare two Range objects. We call the Range objects rng and cell.

```
Dim rng As Range, cell As Range
```

2. We initialize the Range object rng with the selected range.

```
Set rng = Selection
```

3. We want to check each cell in a randomly selected range (this range can be of any size). In Excel VBA, you can use the For Each Next loop for this. Add the following code lines:

```
For Each cell In rng


Next cell
```

Note: rng and cell are randomly chosen here, you can use any names. Remember to refer to these names in the rest of your code.

4. To ignore a cell that contains a formula, add the following code line between For Each and Next (only if cell.HasFormula is false we continue).

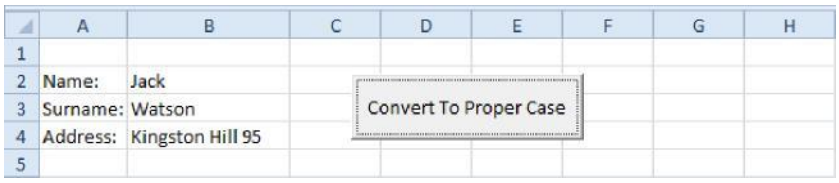
```
If Not cell.HasFormula Then

End If
```

5. Next, we want to convert each word in this range to 'proper case'. You can use the worksheet function Proper for this task. Add the following code line in your if statement.

```
cell.Value = WorksheetFunction.Proper(cell.Value)
```

6. Test the program.

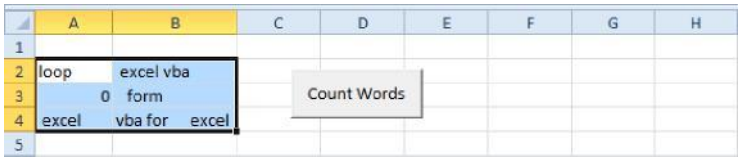
Result:



Count Words

Below we will look at a program in Excel VBA that counts the number of words in a selected range. One or more spaces are assumed to separate words.

Situation:



1. First, we declare two Range objects and three variables. We call the Range objects rng and cell. One Integer variable we call cellWords, one Integer variable we call totalWords, and one String variable we call content.

```
Dim rng As Range, cell As Range
Dim cellWords, totalWords As Integer, content As String
```

2. We initialize the Range object rng with the selected range and the two variables of type Integer with value 0.

```
Set rng = Selection
cellWords = 0
totalWords = 0
```

3. We want to check each cell in a randomly selected range (this range can be of any size). In Excel VBA, you can use the For Each Next loop for this. Add the following code lines:

```
For Each cell In rng

Next cell
```

Note: rng and cell are randomly chosen here, you can use any names. Remember to refer to these names in the rest of your code.

4. Next, we determine for each cell in this range how many words it contains. To ignore a cell that contains a formula, add the following code line between For Each and Next (only if cell.HasFormula is false we continue).

```
If Not cell.HasFormula Then

End If
```

5. First, we write the content of the cell to the variable content. Next, we remove the spaces at the beginning and the end (if there are any). In Excel VBA, you can use the Trim function for this. For example, " excel vba" will be converted to "excel vba". Add the following code lines in your If statement.

```
content = cell.Value
content = Trim(content)
```

Note: the trim function in Excel VBA does not remove extra spaces between words, but that's OK in this example.

6. At this point, a cell can still be empty. If the cell is empty, we assign the value 0 to the variable cellWords. If not, it contains at least one word and we assign the value 1 to the variable cellWords. Add the following code lines in your If statement.

```
If content = "" Then
cellWords = 0
Else
cellWords = 1
End If
```

A cell can contain more than one word of course. That's exactly what we want to find out now. As an example we take: "excel vba". If a cell contains at least one space at this stage, it contains at least one more word. You can use the Instr function in Excel VBA to look for a space. Instr(content, " ") finds the position of the first space in content.

7. We will make use of the Do While Loop structure. Code placed between these words (at step 8, 9 and 10) will be repeated as long as the part after Do While is true. We want to repeat these steps as long as Instr(content, " ") > 0 is true (as long as content contains a space and thus more words). Add the Do While Loop in your If statement.

```
Do While InStr(content, " ") > 0

Loop
```

8. Next, we take the part of content starting at the position of the first space. We use the Mid function for this.

```
content = Mid(content, InStr(content, " "))
```

For example: Mid("excel vba", InStr("excel vba", " ")) will give " vba".

9. We trim the string again.

```
content = Trim(content)
```

Result: "vba"

10. We increment cellWords by 1.

```
cellWords = cellWords + 1
```

This Do While Loop will be repeated as long as content contains a space and thus more words. In our example, we exit the Do While Loop since "vba" does not contain a space anymore! Result: this cell contains 2 words.

11. After having checked one cell, we add cellWords to the variable totalWords. This code line should be placed outside the Do While Loop but in the If statement.

```
totalWords = totalWords + cellWords
```

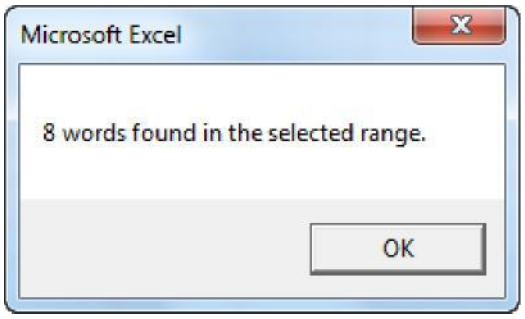
The whole process starts again for the next cell until all cells have been checked.

12. Finally, we display the value of totalWords using a msgbox. This code line should be placed outside the For Each Next loop.

```
MsgBox totalWords &" words found in the selected range."
```

13. Test the program.

Result:



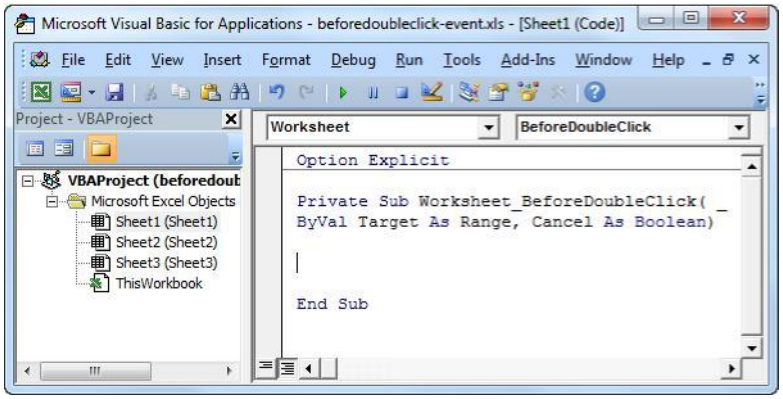
BeforeDoubleClick Event

Code added to the Worksheet BeforeDoubleClick Event will be executed by Excel VBA when you double click a cell on a worksheet.

1. Open the Visual Basic Editor.

2. Double click on a sheet (for example Sheet1) in the Project Explorer.

3. Choose Worksheet from the left drop-down list. Choose BeforeDoubleClick from the right drop-down list.



Note: the '_' symbol is used to continue the statement on a new line (in order to show you the complete code).

Add the following code lines to the Worksheet BeforeDoubleClick Event:

4. The code line below colors the active cell red.

```
Target.Font.Color = vbRed
```

5. Set the Cancel argument to true if you don't want the default double click action (cell edit mode) to occur.

```
Cancel = True
```

6. Test the program by double clicking a cell on Sheet1.
Result:

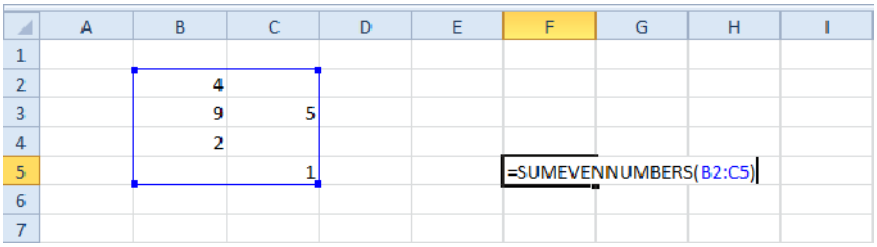
	A	B	C	D	E	F	G	H	I
1	3	78	27	0	78				
2	15	99	69	5	22				
3	88	64	84	31	77				
4	75	63	10	33	74				
5	24	2	3	51	23				
6	62	86	100	81	100				
7	14	20	82	88	65				
8	94	74	69	17	80				
9	29	2	46	67	33				
10	17	25	33	74	48				
11									
12									

User Defined Function

Below we will look at a program in Excel VBA that creates a User Defined Function. Excel has a large collection of functions. In most situations those functions are sufficient to get the job done. If not, you can create your own function called User Defined Function or custom Excel function. You can access a User Defined Function just like any other Excel function.

We want to create a function called SUMEVENNUMBERS that finds the sum of the even numbers of a randomly selected range.

Situation:



User defined functions need to be placed into a module.

1. Open the Visual Basic Editor and click Insert, Module.

2. Add the following code line:

```
Function SUMEVENNUMBERS(rng As Range)
```

The name of our Function is SUMEVENNUMBERS. The part between the brackets means we give Excel VBA a range as input. We name our range rng, but you can use any name.

3. Next, we declare a Range object and call it cell.

```
Dim cell As Range
```

4. We want to check each cell in a randomly selected range (this range can be of any size). In Excel VBA, you can use the For Each Next loop for this. Add the following code lines:

```
For Each cell In rng

Next cell
```

Note: cell is randomly chosen here, you can use any name.

5. Next, we check for each value in this range whether it is even or not. We use the Mod operator for this. The Mod operator gives the remainder of a division. So 7 mod 2 would give 1. 7 is divided by 2 (3 times) to give a remainder of 1. Having said this, it is easy to check whether a number is even or not. Only if the remainder of a number divided by 2 is 0, the number is even. 8 mod 2 gives 0, 8 is divided by 2 exactly 4 times, and therefore 8 is even. Add the following If statement to the For Each Next loop.

```
If cell.Value Mod 2 = 0 Then

End If
```

6. Only if this statement is true, we add the value to SUMEVENNUMBERS. Add the following code line in the If statement.

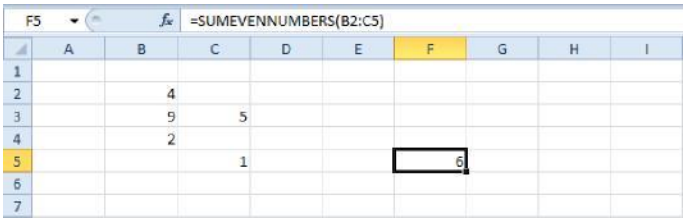
```
SUMEVENNUMBERS = SUMEVENNUMBERS + cell.Value
```

7. Don't forget to end the function (outside the loop).

```
End Function
```

8. Now you can use this function, just like any other Excel function, to find the sum of the even numbers of a randomly selected range.

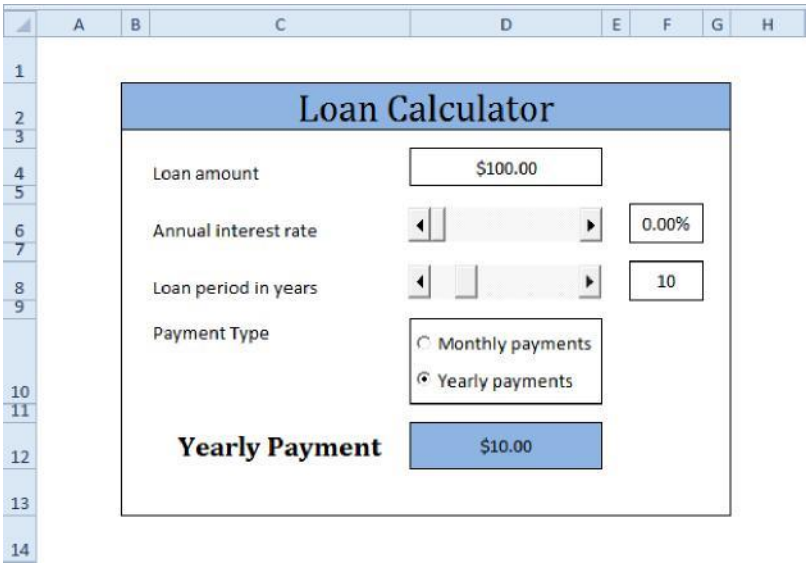
Result:



Well done! That's how easy User Defined Functions in Excel VBA are. Note: this function is only available in this workbook.

Create a Loan Calculator

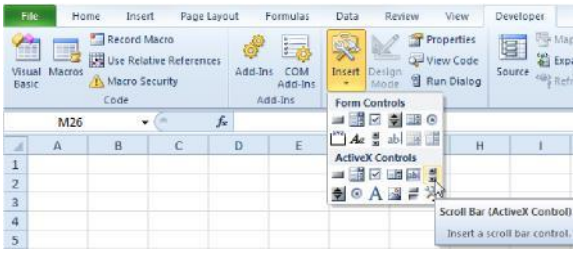
This page teaches you how to create a simple loan calculator in Excel VBA. The worksheet contains the following ActiveX controls: two scrollbars and two option buttons.



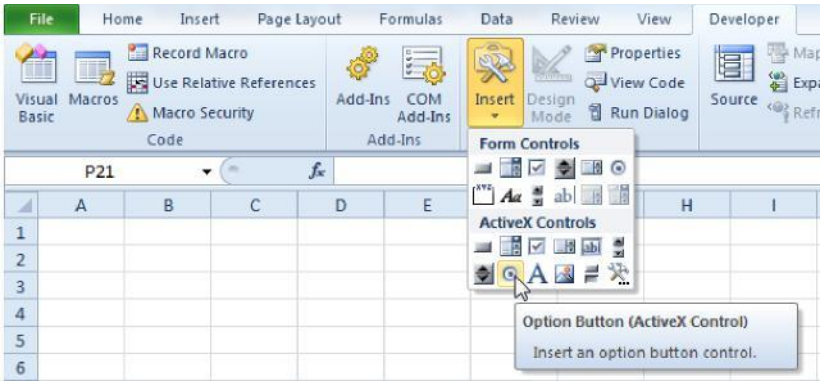
Note: the instructions below do not teach you how to format the worksheet. We assume that you know how to change font types, insert rows and columns, add borders, change background colors, etc.

Execute the following steps to create the loan calculator:

1. Add the two scrollbar controls. Click on Insert from the Developer tab and then click on Scroll Bar in the ActiveX Controls section.



2. Add the two option buttons. Click on Insert from the Developer tab and then click on Option Button in the ActiveX Controls section.



Change the following properties of the scrollbar controls (make sure Design Mode is selected).

3. Right mouse click on the first scrollbar control, and then click on Properties. Set Min to 0, Max to 20, SmallChange to 0 and LargeChange to 2.

4. Right mouse click on the second scrollbar control, and then click on Properties. Set Min to 5, Max to 30, SmallChange to 1, LargeChange to 5, and LinkedCell to F8.

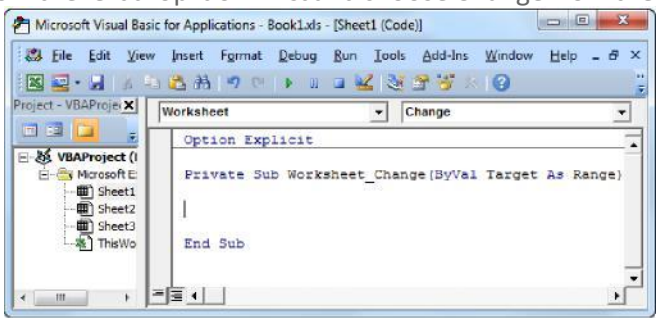
Explanation: when you click on the arrow, the scrollbar value goes up or down by SmallChange. When you click between the slider and the arrow, the scrollbar value goes up or down by LargeChange.

Create a Worksheet Change Event. Code added to the Worksheet Change Event will be executed by Excel VBA when you change a cell on a worksheet.

5. Open the Visual Basic Editor.

6. Double click on Sheet1 (Sheet1) in the Project Explorer.

7. Choose Worksheet from the left drop-down list and choose Change from the right drop-down list.



8. The Worksheet Change Event listens to all changes on Sheet1. We only want Excel VBA to run the Calculate sub if something changes in cell D4. To achieve this, add the following code line to the Worksheet Change Event (more about the Calculate sub later on).

```
If Target.Address = "$D$4" Then Application.Run "Calculate"
```

9. Get the right percentage in cell F6 (change the format of cell F6 to percentage). Right mouse click on the first scrollbar control, and then click on View Code. Add the following code lines:

```
Private Sub ScrollBar1_Change()

Range("F6").Value = ScrollBar1.Value / 100
Application.Run "Calculate"

End Sub
```

10. Right mouse click on the second scrollbar control, and then click on View Code. Add the following code line:

```
Private Sub ScrollBar2_Change()

Application.Run "Calculate"

End Sub
```

11. Right mouse click on the first option button control, and then click on View Code. Add the following code line:

```
Private Sub OptionButton1_Click()

If OptionButton1.Value = True Then Range("C12").Value = "Monthly Payment"
Application.Run "Calculate"

End Sub
```

12. Right mouse click on the second option button control, and then click on View Code. Add the following code line:

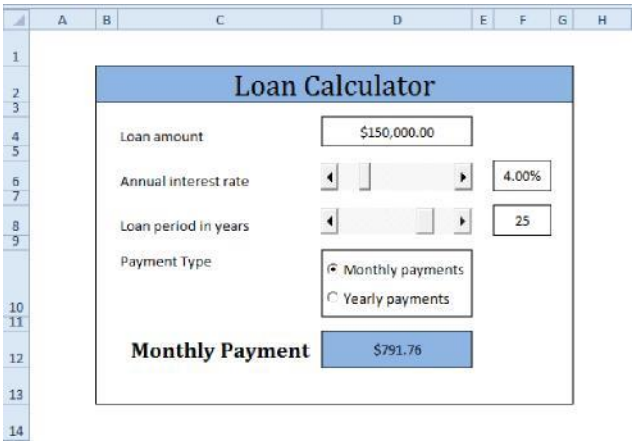
```
Private Sub OptionButton2_Click()

If OptionButton2.Value = True Then Range("C12").Value = "Yearly Payment"
Application.Run "Calculate"

End Sub
```

13. Time to create the sub. You can go through our Function and Sub chapter to learn more about subs. If you are in a hurry, simply place the sub named Calculate into a module (In the Visual Basic Editor, click Insert, Module).

```
Sub Calculate()

Dim loan As Long, rate As Double, nper As Integer

loan = Range("D4").Value
```

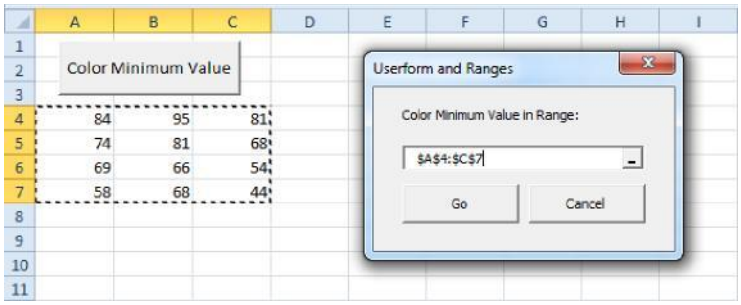
```
rate = Range("F6").Value
nper = Range("F8").Value

If Sheet1.OptionButton1.Value = True Then
  rate = rate / 12
  nper = nper * 12
End If

Range("D12").Value = -1 * WorksheetFunction.Pmt(rate, nper, loan)

End Sub
```

Explanation: the sub gets the right parameters for the worksheet function Pmt. The Pmt function in Excel calculates the payments for a loan based on constant payments and a constant interest rate. If you make monthly payments (Sheet1.OptionButton1.Value = True), Excel VBA uses rate / 12 for rate and nper *12 for nper (total number of payments). The result is a negative number, because payments are considered a debit. Multiplying the result by -1 gives a positive result.



Userform and Ranges

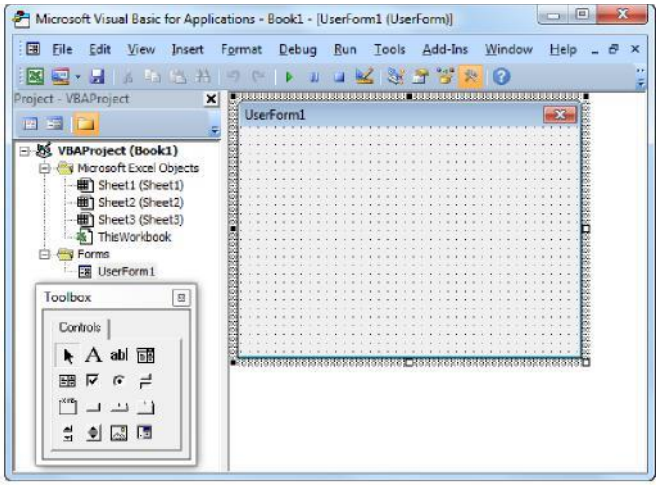
You can use a RefEdit control in Excel VBA to get a range from a user. The Userform we are going to create colors the minimum value of the range stored in the RefEdit control.



To create this Userform, execute the following steps.

1. Open the Visual Basic Editor. If the Project Explorer is not visible, click View, Project Explorer.

2. Click Insert, Userform. If the Toolbox does not appear automatically, click View, Toolbox. Your screen should be set up as below.



3. Add the label, RefEdit control and command buttons. Once this has been completed, the result should be consistent with the picture of the Userform shown earlier. For example, create a RefEdit control by clicking on RefEdit from the Toolbox. Next, you can drag a RefEdit control on the Userform.

Note: If your toolbox does not have a RefEdit control, set a reference to RefEdit control. Click Tools, References, and check Ref Edit Control.

4. You can change the names and the captions of the controls. Names are used in the Excel VBA code. Captions are those that appear on your screen. It is good practice to change the names of the controls, but it is not necessary here because we only have a few controls in this example. To change the caption of the Userform, label and command buttons, click View, Properties Window and click on each control.

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

```
Private Sub CommandButton1_Click()

UserForm1.Show

End Sub
```

We are now going to create the Sub UserForm_Initialize. When you use the Show method for the Userform, this sub will automatically be executed.

6. Open the Visual Basic Editor.

7. In the Project Explorer, right click on UserForm1 and then click View Code.

8. Choose Userform from the left drop-down list. Choose Initialize from the right drop-down list.

9. Add the following code lines:

```
Private Sub UserForm_Initialize()

Sheet1.Cells.Font.Color = vbBlack
```

```
UserForm1.RefEdit1.Text = Selection.Address

End Sub
```

Explanation: the first code line changes the font color of all the cells on sheet1 to black. The second code line obtains the address of the current selection and displays it in the RefEdit control.

We have now created the first part of the Userform. Although it looks neat already, nothing will happen yet when we click the command buttons on the Userform.

10. In the Project Explorer, double click on UserForm1.
11. Double click on the Go button.
12. Add the following code lines:

```
Private Sub CommandButton1_Click()

Dim addr As String, rng, cell As Range, minimum As Double

addr = RefEdit1.Value
Set rng = Range(addr)
minimum = WorksheetFunction.Min(rng)

For Each cell In rng
    If cell.Value = minimum Then cell.Font.Color = vbRed
Next cell

End Sub
```

Explanation: first, we get the address from the RefEdit control and store it into the String variable addr. Next, we set rng to the range specified in the RefEdit control. Next, we use the worksheet function Min to find the minimum value in the range. Finally, we color the minimum value(s) using a loop.

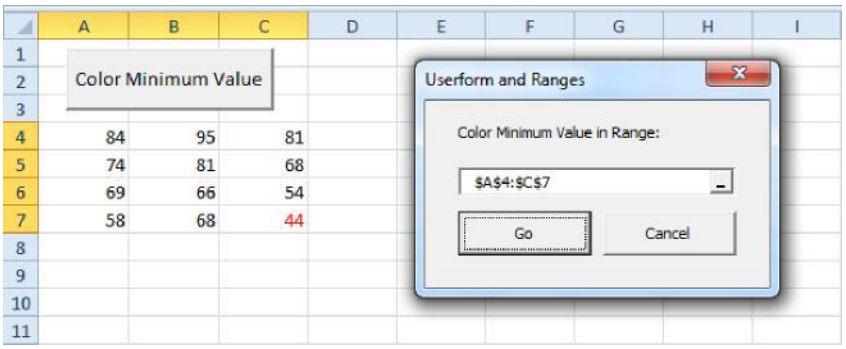
13. Double click on the Cancel button.
14. Add the following code line:

```
Private Sub CommandButton2_Click()

Unload Me

End Sub
```

Explanation: this code line closes the Userform when you click on the Cancel button.

15. Test the Userform.

Result:

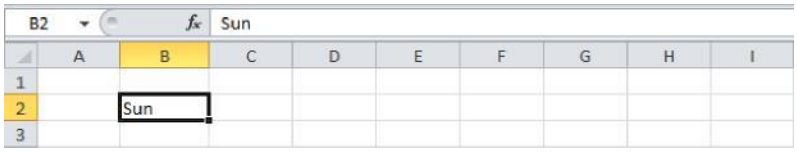


Custom Lists

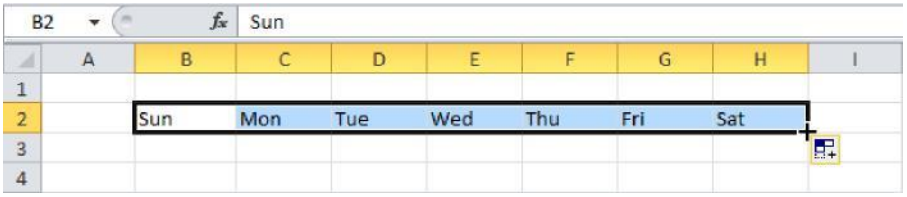
If you create a custom list in Excel, you can easily fill a range with your own list of departments, clients, cities, credit card numbers, etc. This can save time and reduce errors.

First, we will look at an example of a built-in list.

1. Type Sun into cell B2.

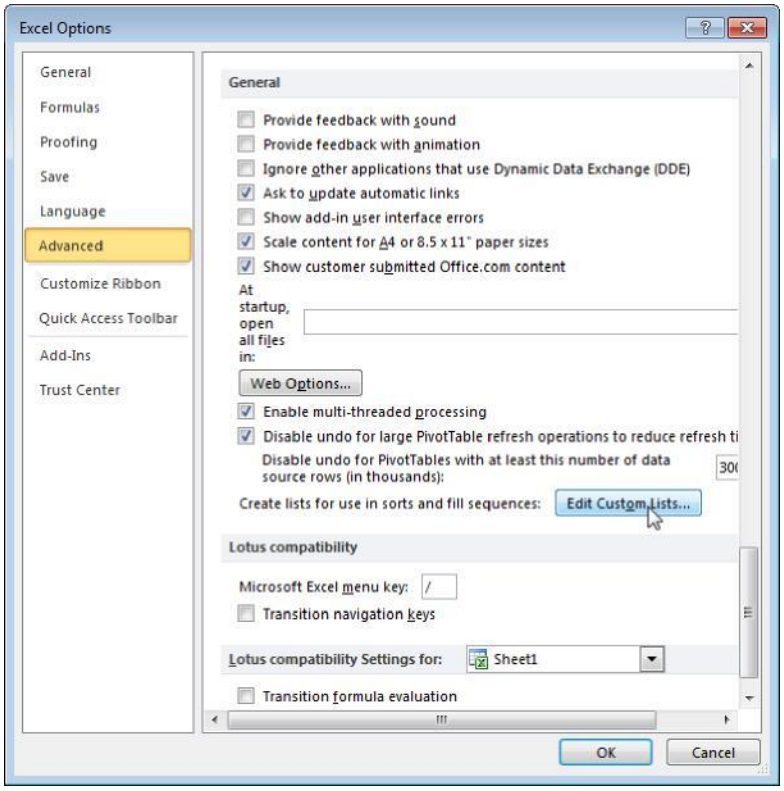


2. Select cell B2, click on the lower right corner of cell B2 and drag it across to cell H2.

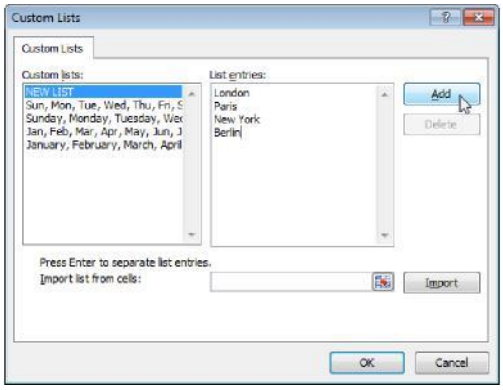


How does Excel know this?

3. On the green File tab, click Options.
4. Under Advanced, go to General and click Edit Custom Lists.



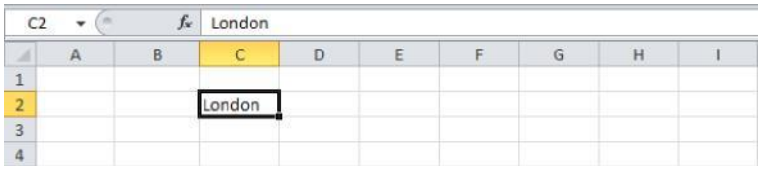
Here you can find the built-in 'days of the week' lists. Also notice the 'months of the year' lists. 5. To create your own custom list, type some list entries, and click Add.



Note: you can also import a list from a worksheet.

6. Click OK.

7. Type London into cell C2.



8. Select cell C2, click on the lower right corner of cell C2 and drag it down to cell C5.