VBA
Visual Basic for Applications
Learner Guide
# Table of Contents

## SECTION 1 ◊ WORKING WITH MACROS ................................................................. 5

WORKING WITH MACROS .................................................................................. 6
  - About Excel macros ......................................................................................... 6
  - Opening Excel (using Windows 7 or 10) ....................................................... 6
  - Recognizing the macro tools and what they are .......................................... 8
  - Practice macros first .................................................................................... 8
  - Create your macro ........................................................................................ 9
  - Recording a macro ...................................................................................... 10
  - Recording your second macro .................................................................... 12
  - Testing your recorded macro shortcut keys .............................................. 13
  - Testing your recorded macro from the menu ............................................. 14
  - View my macro code in the VBE .............................................................. 15

## SECTION 2 ◊ ASSIGNING MACROS ................................................................. 21

LOCATION OF RECORDED MACRO .................................................................. 22
  - Absolute or Relative macro recording ....................................................... 22
  - Using Absolute ......................................................................................... 22
  - Viewing Absolute code ............................................................................ 24
  - Running Absolute ................................................................................... 24
  - Using Relative ......................................................................................... 25
  - Testing Absolute and Relative macros ..................................................... 26
  - Viewing code of both Absolute and Relative macros .............................. 27
  - Code for Absolute .................................................................................. 27
  - Code for Relative .................................................................................. 28

## SECTION 3 ◊ MAKE DECISIONS ..................................................................... 33

- Assigning your macros to menus, buttons and toolbars ........................... 28
  - Adding a button to your toolbar ............................................................... 28
  - Testing your button .................................................................................. 29
  - Adding a new button to the Quick Access Toolbar ............................... 29
  - Add an Icon to the QAT for your macro ................................................. 30
Making Decisions with code .................................................................................................................. 34
Multiple-line Else .................................................................................................................................. 35
Testing your If statement ...................................................................................................................... 37
Turn on the Relative Reference .......................................................................................................... 38
Copying Code ......................................................................................................................................... 38
Adding Do… Loop statement ................................................................................................................. 38
Testing the Do Loop .............................................................................................................................. 39
Copying Code ......................................................................................................................................... 40
Testing your new If statement .............................................................................................................. 41
Adding Do… Loop statement ................................................................................................................. 41
Testing the Do Loop .............................................................................................................................. 42
How to count .......................................................................................................................................... 46
For… Next statement .............................................................................................................................. 46
Variable Naming Conventions ............................................................................................................... 48
Constants .................................................................................................................................................. 48
Built-in Constants .................................................................................................................................... 49
Object Variables ....................................................................................................................................... 50
Working with objects using a For … Next statement ............................................................................ 51
How to select a range based on starting point ....................................................................................... 52
How to add one macro in another’s routine .......................................................................................... 54
SECTION 4 ◊ PERSONAL.XLSB ............................................................................................................. 56
Personal Workbook Macros .................................................................................................................... 57
How do I create a Personal.xlsm file? ................................................................................................. 57
Auto Run Macro(s) ............................................................................................................................... 58
Auto_Open ............................................................................................................................................... 59
Auto_Close ............................................................................................................................................. 60
Final Exercise .......................................................................................................................................... 60
SECTION 1 ◊ WORKING WITH MACROS

What will I learn in this section?

Create a macro
Run a macro
View a macro
WORKING WITH MACROS

About Excel macros

This will be an introduction to Visual Basic for Applications (VBA) in Excel 2013/2016. For Excel power user who is not yet a programmer or anyone who just wants to know more. VBA is easy to learn, by creating a macro and then viewing the code anyone can quickly learn how to read the language.

The more you learn the more you can modify VBA to have the macro do the extra task that a user needs.

The Visual Basic Editor (VBE) is the user-friendly program that you will use to talk with Excel. In it you can create your VBA procedures (macros). You will then be able to modify and test these components easily with short cut keys that will help you step through your code.

The first macro you will create is a macro you record using Excel’s macro recorder. You will record yourself doing a few steps with the AutoFilter feature.

Opening Excel (using Windows 7 or 10)

1. Click the Start button in Windows 7 on the taskbar.

2. In Windows 7 point to All Programs.

In Windows 10 type: Excel in the Search box.
3. Click **Microsoft Excel** (on the menu *in Windows 7*) (on the menu in *Windows 10*), click **Excel 2016**.

Your screen should look like this.

![Microsoft Excel window]

1. Click the **File** tab and **Open** menu or press **Ctrl + F12**.

![File tab and Open menu]

Notice: If you do not see the file extension `.xlsm` your computer is fine. Normally this feature is turned off. It is shown here to identify this file as different. This file was saved as a Macro-enabled file format.

2. Open the **Sort&Filter.xlsm** file.

![Sorting and filtering]

Your screen should look like this.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td><strong>Employee’s Pay Roll List</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>EmpID</td>
<td>FirstName</td>
<td>LastName</td>
<td>DateHired</td>
</tr>
<tr>
<td>5</td>
<td>EMP123</td>
<td>Hazel</td>
<td>Abdul</td>
<td>22-Jul-02</td>
<td>TR</td>
</tr>
<tr>
<td>6</td>
<td>EMP124</td>
<td>Liza</td>
<td>Able</td>
<td>4-Jul-02</td>
<td>BF</td>
</tr>
<tr>
<td>7</td>
<td>EMP125</td>
<td>Robert</td>
<td>Albert</td>
<td>20-Oct-02</td>
<td>ND</td>
</tr>
<tr>
<td>8</td>
<td>EMP126</td>
<td>Howard</td>
<td>Alexi</td>
<td>23-Apr-02</td>
<td>PP</td>
</tr>
<tr>
<td>9</td>
<td>EMP127</td>
<td>Maxine</td>
<td>Al-Sabah</td>
<td>19-Mar-02</td>
<td>DR</td>
</tr>
</tbody>
</table>
Recognizing the macro tools and what they are

Under the View tab in Excel you will find the Macro group. In the Group there are three submenu items.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Macros...</td>
<td>List the Macros available in Excel</td>
</tr>
<tr>
<td>Record Macro...</td>
<td>Start recording a macro by typing a name for your macro and a location to store the macro</td>
</tr>
<tr>
<td>Use Relative References</td>
<td>Tool used to view, edit and create macros</td>
</tr>
</tbody>
</table>

On the View tab in the Macro group the Macro menu when clicked should look like this.

Practice macros first

Before you record a macro, it is a good idea to practice the macro steps and if there are more than 6-8 steps it is a good idea to write them down to make sure you do not miss a step when recording.

1. Click in cell F5 and click the Data tab.
2. On the Data ribbon in the Sort & Filter group, click to Filter button.

Filter is a toggle button for turning the filter on or off.

Your screen should look like this.

<table>
<thead>
<tr>
<th>EmplID</th>
<th>FirstName</th>
<th>LastName</th>
<th>DateHired</th>
<th>Dept Code</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP123</td>
<td>Hazel</td>
<td>Abdul</td>
<td>22-Jul-02</td>
<td>TR</td>
<td>Training Region</td>
</tr>
<tr>
<td>EMP124</td>
<td>Liza</td>
<td>Able</td>
<td>4-Jul-02</td>
<td>BF</td>
<td>Backup Floor</td>
</tr>
<tr>
<td>EMP125</td>
<td>Robert</td>
<td>Albert</td>
<td>20-Oct-02</td>
<td>NB</td>
<td>North Bank</td>
</tr>
</tbody>
</table>
3. Click the down-arrow for Division and click **Select All** to remove checkmarks.

![Image](https://via.placeholder.com/150)

4. Check the box for **Bremerton** and click the **OK** button.

5. On the **Data** tab, **Sort & Filter** group, click **Filter** to turn off filter.

   **Filter** is a toggle button for turning the filter on or off.

   ![Image](https://via.placeholder.com/150)

   **Notice** that your data is now without any filter and the AutoFilter has been turned off.

---

**Create your macro**

The first character of the macro name must be a letter. Other characters can be letters, numbers, or underscore characters. Spaces are not allowed in a macro name; an underscore character works well as a word separator.

- Do not use a macro name that is also a cell reference, or you can get an error message that the macro name is not valid.
- You can use CTRL+ *letter* (for lowercase letters) or CTRL+SHIFT+ *letter* (for uppercase letters), where *letter* is any letter key on the keyboard. The shortcut key letter you use cannot be a number or special character such as @ or #.
- In the **Store macro in** box, click the location where you want to store the macro. If you want a macro to be available whenever you use Excel, select **Personal Macro Workbook**.
- If you want to include a description of the macro, type it in the **Description** box. Anything you type here will show up as green commented text in the macro module.
Look at the bottom right corner of the status bar in Excel. You should see a macro button.

The macro button is just to the right of the word “Ready”. You can click on this button to open the Record Macro dialog box. Once you are recording you can click this button again to stop the macro from recording.

The macro button changes when recording, depending on the Excel version it will be white or black.

**Recording a macro**

The Macro Recorder works like a tape recorder. Whereas a tape recorder records all the sounds, Excel recorder records all the actions that you perform when you work in Excel.

In the next few steps you will record two macros. These macros will be stored in the Sort&Filter.xlsm workbook.

1. Click the View tab, point to Macro, click the down arrow to open the menu and click Record Macro...on the menu.

**Notice**: The macro button has two parts; the top half opens the Macro dialog box used to see macros you have already recorded. You will use the top half later in the book.

The bottom half opens the menu that you will be using in this lesson.
Your screen should look like this.

**Notice** you have four items to complete as explained in the bullet points above.

2. In the Macro name: text box type: **TurnOnOffAutoFilter**
   
   *AutoFilter is a toggle menu for on or off.*

3. Click in the Shortcut key: *text box*, and press: **Shift + A**
   
   Your screen should look like this.

4. In the Macro name: text box type: **TurnOnOffAutoFilter**
   
   *AutoFilter is a toggle menu for on or off.*

5. Click in the Shortcut key: *text box*, and press: **Shift + A**
   
   Your screen should look like this.
**Notice** your shortcut key is now, **Ctrl + shift + A**. You have assigned a shortcut key. Remember any shortcut key you assign will over write Excel’s shortcut keys. Your shortcut key is senior and become the primary.

If you need to assign a macro after you have created your macro, you will need to click the **Options** button in the Macro dialog box.

6. In the **Store macro in**: drop down leave “This workbook” as the selection.
7. In the Description: text box, select the text and type: **My toggle for Filter**
8. Click the **OK** button to turn on the recorder.

At the bottom, your screen should look like this.

Notice the black or white square box (depending on your Excel version) on the Status bar. This is now the **Stop Recording** button. You can also click the down arrow for the Macros menu and select **Stop Recording**.

9. Click on the cell **F5**.
10. Click the **Data** tab, point to **Filter** and click the **Filter** button.
11. Click the **Stop Recording** button.

You have completed all the step needed and you have created your first macro.

**Recording your second macro**

Your second macro will be to select Bremerton as the filtered item.

1. Click the **View** tab and click the **Macros** down-arrow.
2. Click the **Record Macro**... menu.
3. In the Macro name: text box type: **MyBremerton**.
4. Click in the Shortcut key: text box, click in the box and press: **Shift + B**
Your screen should look like this.

Notice your shortcut key is now, $\text{Ctrl} + \text{shift} + B$.

5. In the Description: text box, select the text and type: **My Bremerton filtered list**.

6. Click the **OK** button to turn on the recorder.

7. Click the **down arrow** for Division, uncheck **Select All** and check **Bremerton**.

8. Click the **Stop Recording** button.

You have now created the second macro.

**Testing your recorded macro shortcut keys**

1. Press the shortcut key: $\text{Ctrl} + \text{Shift} + A$

   **Remember the Filter is a toggle and this turns it off.**

2. Press the shortcut key: $\text{Ctrl} + \text{Shift} + A$

   **Now the shortcut key turn the Filter back on.**

3. Press the shortcut key: $\text{Ctrl} + \text{Shift} + B$

   **Now only Bremerton is showing.**

Your screen should look like this.
Testing your recorded macro from the menu

1. Click the View tab (if needed) and point to Macro and click the down-arrow.

2. Click the View Macros menu.

   Your screen should look like this.

3. Select the TurnOnOffAutoFilter from the macro list window.

4. Click the Run button in the Macro dialog box.

   Your filter is turned off.

5. Click the down-arrow and click View Macros.

6. Select TurnOnOffAutoFilter from the macro list window.

7. Click the Run button in the Macro dialog box.

   Your filter is turned on.
8. Click the *down-arrow* and click **View Macros**.

9. Select **MyBremeton** from the macro list window.

10. Click the **Run** button in the Macro dialog box.

Your screen should look like this.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>FirstName</th>
<th>LastName</th>
<th>DateHired</th>
<th>DeptCode</th>
<th>DeptName</th>
<th>Division</th>
<th>Hours</th>
<th>PayRate</th>
<th>GrossPay</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP124</td>
<td>Liza</td>
<td>Able</td>
<td>4-Jul-02</td>
<td>BF</td>
<td>Backup Floor</td>
<td>Bremerton</td>
<td>32</td>
<td>22.25</td>
<td>712</td>
</tr>
<tr>
<td>EMP131</td>
<td>Sheryl</td>
<td>Bankler</td>
<td>20-Oct-02</td>
<td>PP</td>
<td>Pepper Park</td>
<td>Bremerton</td>
<td>40</td>
<td>14.00</td>
<td>560</td>
</tr>
<tr>
<td>EMP139</td>
<td>Ari</td>
<td>Bellwood</td>
<td>26-Nov-02</td>
<td>TR</td>
<td>Training Region</td>
<td>Bremerton</td>
<td>40</td>
<td>18.00</td>
<td>720</td>
</tr>
<tr>
<td>EMP145</td>
<td>Burt</td>
<td>Chu</td>
<td>14-Sep-02</td>
<td>NB</td>
<td>North Bank</td>
<td>Bremerton</td>
<td>40</td>
<td>23.00</td>
<td>920</td>
</tr>
<tr>
<td>EMP199</td>
<td>Linda</td>
<td>Chu</td>
<td>14-Sep-02</td>
<td>NB</td>
<td>North Bank</td>
<td>Bremerton</td>
<td>40</td>
<td>23.00</td>
<td>920</td>
</tr>
<tr>
<td>EMP152</td>
<td>Randy</td>
<td>Cummins</td>
<td>5-Aug-02</td>
<td>DR</td>
<td>Docking Range</td>
<td>Bremerton</td>
<td>40</td>
<td>21.00</td>
<td>840</td>
</tr>
<tr>
<td>EMP206</td>
<td>Randy</td>
<td>Cumminston</td>
<td>9-Aug-02</td>
<td>DR</td>
<td>Docking Range</td>
<td>Bremerton</td>
<td>40</td>
<td>21.00</td>
<td>840</td>
</tr>
<tr>
<td>EMP213</td>
<td>MaryAnne</td>
<td>Fontaine</td>
<td>22-Jul-02</td>
<td>BF</td>
<td>Backup Floor</td>
<td>Bremerton</td>
<td>36</td>
<td>15.45</td>
<td>566.2</td>
</tr>
<tr>
<td>EMP156</td>
<td>Alice</td>
<td>Haplsbuch</td>
<td>7-Nov-02</td>
<td>PP</td>
<td>Pepper Park</td>
<td>Bremerton</td>
<td>40</td>
<td>23.00</td>
<td>920</td>
</tr>
<tr>
<td>EMP159</td>
<td>Mary</td>
<td>Hodge</td>
<td>22-Jul-02</td>
<td>BF</td>
<td>Backup Floor</td>
<td>Bremerton</td>
<td>36</td>
<td>15.45</td>
<td>566.2</td>
</tr>
<tr>
<td>EMP173</td>
<td>Jules</td>
<td>Kaneko</td>
<td>2-May-02</td>
<td>TR</td>
<td>Training Region</td>
<td>Bremerton</td>
<td>40</td>
<td>17.00</td>
<td>680</td>
</tr>
<tr>
<td>EMP173</td>
<td>Jules</td>
<td>Kaneko</td>
<td>2-May-02</td>
<td>TR</td>
<td>Training Region</td>
<td>Bremerton</td>
<td>40</td>
<td>17.00</td>
<td>680</td>
</tr>
<tr>
<td>EMP180</td>
<td>Iain</td>
<td>Lemporst</td>
<td>2-Oct-02</td>
<td>NB</td>
<td>North Bank</td>
<td>Bremerton</td>
<td>32</td>
<td>24.00</td>
<td>768</td>
</tr>
<tr>
<td>EMP187</td>
<td>Sam</td>
<td>McCormick</td>
<td>27-Aug-02</td>
<td>DR</td>
<td>Docking Range</td>
<td>Bremerton</td>
<td>40</td>
<td>22.00</td>
<td>880</td>
</tr>
</tbody>
</table>

**View my macro code in the VBE**

There are a few things to know before you jump into the Visual Basic Editor.

Let’s start with code that you will be viewing and a few other items.

- **Code**: black text for instructions, blue for key words in code and green text for comments for humans to read.
  - Instructions, operators and variables will be black text
  - Keywords: terms that have special meaning will be blue text
  - Comments: text for only humans to read will be green text

- **Module**: VBA modules are stored in an Excel workbook in folders that can be viewed using the VBE. Your VBA code is stored in the module sheet. You can have many module sheets and you can name them.

- **Procedures**: A procedure is the basic unit of computer code that performs an action. There are two procedures; Sub procedure and Function procedure.
• **Sub** and **End Sub**: A sub routine consist of a single or many statements that can be executed in many ways. **Example:**

```vba
Sub Test()
    Sum = 1 + 1
    MsgBox "The answer is “ & Sum
End Sub
```

• **Function**: Just as its name implies you can create your own functions using the power of VBA. **Example:**

```vba
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

• **Objects**: VBA manipulates objects; active objects would be workbooks, sheets, and ranges. **Example**: `worksheets("Sheet1").range("A1")`

• **VBA variables**: You can assign values to VBA variables. (**Interest is the variable below**) **Example**: `Interest = worksheets("Sheet1").Range("A1").Value`

• **Object methods**: A method is an action that is performed with the object. **Example**: `Range("A1").ClearContents`

• **Standard programming constructs**: VBA can and does all of the standard programming language items; for – next, If – then, Case and loops etc.

• **Events**: Excel VBA recognizes specific events, for example; Workbook open or close, object is clicked (called focus), keyboard is pressed, worksheet is activated or deactivated, and cell is entered or edited. Many more…

1. To enter the VBE, you can click on the Developer’s tab, Code group and click **Visual Basic** button or press the **Alt + F11** keys.

2. Double click the **Module1** sheet in the Modules folder.
Your screen will look like this.

Depending on how much you get into VBA you may or may not need to know everything you have access to in the VBE. In this booklet, only the main items that will be used will be described.

Under the View menu you will find the menu items for Project Explorer and Properties Window, with shortcut keys.
You can run some macros from within VBE but not all. You can use the Break and Reset for your macros, you will learn how to use those latter in this booklet.

You can also view your macro dialog box from inside the VBE.

When you have a number of macros in a module sheet you can select one from the drop-down list (top right side) to reach the code on the module sheet.

This will select the macro and bring the macro code to the top of your screen.

A **Sub** procedure (the basic structure of a macro)
The parts you see here in the screen shot above are broken down into:

**Sub** and **End Sub** - keywords

**Comments** - easy to read information that describes what the macro and commands are all about.

**Statement block** – the executable commands, what the macro code will tell Excel to do
Section 1 – Review
You must be able to answer the following questions on your own. As you review the questions below, write in your answer below each one.

1. What is the first character a macro should start with?

2. What color are comments in a sub routine?

3. What is the shortcut key to open the VB Editor?

4. Can you have more than one module in a workbook?

Additional Study
Use the Microsoft Excel help features to look up the key words from this section for additional information. Example: AutoSum, AutoFilter.
SECTION 2 ◊ ASSIGNING MACROS

What will I learn in this section?

Relative References in macros
Assign Macro to a button or QAT
LOCATION OF RECORDED MACRO

When you record a macro, its code is recorded in a module in the active workbook. You can select another workbook or the Personal.xls workbook. If you would like the macro you record to be global, meaning you can run the macro regardless of the workbook you open then record the macro in the Personal.xls workbook.

If a macro is recorded in the wrong workbook the macro can be copied from one module to another module.

Absolute or Relative macro recording

Just like formulas that are relative reference or absolute, when you record a macro it can be one or the other. When using Absolute; if you click on cell D2, then when you run the macro D2 is used. But the Relative works by moving the same number of cells based on the action during the recording. Example would be that with relative if you pressed the Enter key the action would be that you moved down one cell.

You will do two macros in this section; one absolute and one relative. Then take a look at how the code is different.

Using Absolute

1. Right click one of the tabs of your ribbon.
2. Select the Customize the Ribbon… menu.
3. Click the check box next to Developer.
4. Open Absolute&Relative.xlsx class file.
5. Click the **Record Macro** button.

6. Type: **TitleMyAbsolute** for Macro name.

7. In the Description box type: **Use Absolute title for heading.**

8. Click the **OK** button to start recording.

9. Click in C2 and type: **Western Region**

10. Press [Enter] and type: **First Quarter Sales**

11. Press the [Enter] key.

12. Select cell C2 and drag to cell F2.

13. Click the **Merge and Center** button.

14. Select cell C3 and drag to cell F3.

15. Click the **Merge and Center** button.

Your screen should look like this.

16. Click the **Stop Recording** macro button.
Viewing Absolute code

1. On the Developer's ribbon, click the Visual Basic button.

![Visual Basic button](image)

2. Verify that Module 1 is selected in the open workbook.

Your screen should look like this.

![Visual Basic Editor](image)

Examine the code with your instructor. Notice the actual cell addresses used.

3. Close the Visual Basic Editor.

Running Absolute

1. Select Sheet2 and click on cell A1.

2. Run the TitleMyAbsolute macro.

Your screen should look like this.

![Excel table](image)
Using Relative

1. Select **Sheet3** and click on cell **C2** to select.

2. Click the **Relative Reference** button on the Macros menu. (*Relative References is on*)

3. Click the **Record Macro** button.

4. Type: **TitleMyRelative** for Macro name.

5. In the Description box type: *Use Relative title for heading.*

6. Click the **OK** button.

7. Type: **Western Region (in cell C2)** and press [Enter].

8. Type: **First Quarter Sales** and press [Enter].

9. Select **C2** and drag to **F2**.

10. Click the **Merge and Center** button.

11. Select **C3** and drag to **F3**.

12. Click the **Merge and Center** button.
Your screen should look like this.

<table>
<thead>
<tr>
<th>Western Region</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First Quarter Sales</td>
</tr>
<tr>
<td>Sales</td>
<td>Jan</td>
</tr>
</tbody>
</table>

13. Click the **Stop Recording** macro button.

**Testing Absolute and Relative macros**

The Absolute macro will always place your information in the same location. No matter where the active cell maybe the titles in the macro will always end up on the second and third roll between the C and F column.

You will see that the Relative macro will place the title where you place the active cell.

1. Click on **Sheet4** and click on cell **J24**.
2. Run the **TitleMyAbsolute** macro.  
   *(Hint: Tools, Macro, Macros)*

   **Notice** the heading is placed in between column C and F, on row 2 and 3.

3. Click on cell **K8**.
4. Run the **TitleMyRelative** macro.

   **Notice** the heading is placed from K8 to K9, merged and centered.
Viewing code of both Absolute and Relative macros

1. On the Developer’s ribbon, click on the Visual Basic button.

Your screen should look like this.

2. If the Project – VBAProject pane is not open, press the Ctrl + R hotkey.

3. If the Module1 window is not open, double click on the Module1 icon in the project pane.

Code for Absolute

```vba
Sub TitleMyAbsolute()
    ' TitleMyAbsolute Macro
    ' Use Absolute title for heading.

    Range("C2").Select
    ActiveCell.FormulaR1C1 = "Western Region"
    Range("C3").Select
    ActiveCell.FormulaR1C1 = "First Quarter Sales"
    Range("C2:F2").Select
```

Notice the cell address, this command selects C2
This command directs the text to go in the Active Cell
Assigning your macros to menus, buttons and toolbars

After a macro has been created you might like to have a button on your toolbar to quickly run your macro(s).

As you have seen earlier shortcut keys can be used also. But assigning buttons and menus is easy to do.

**Adding a button to your toolbar**

1. On the Developer's ribbon, click **Insert** button and click the **Button** in the **Form Controls** row.

2. Drag the mouse plus from H2 to J3 and release.

3. In the **Assign Macro** dialog box, click on **TitleMyRelative** and click **OK**.

4. Drag across the text: **Button 1**

   Notice an I beam will appear when the pointer is next to the “B”.

5. Type: **My Relative Title**
6. Click a blank cell in the spreadsheet to activate the button.

**Testing your button**

1. Select the cell H24 on Sheet4.
2. Click your new button.
   
   *Your heading is placed, starting with H24 to K24.*

**Adding a new button to the Quick Access Toolbar**

1. Click the More button on the *Quick Access Toolbar*.

![Image of Customize Quick Access Toolbar]

2. Click the More Commands… menu.

![Image of More Commands menu]

3. Click the down-arrow for “Choose commands from:”.

![Image of Choose commands from menu]

4. Select Macros from the menu.
5. In the list of macros, select TitleMyRelative.

![Image of available macros]

6. Click the Add >> button in the middle of the dialog box.

*The macro has been added to the Quick Access Toolbar.*
Add an Icon to the QAT for your macro

1. Click the Modify… button.

2. In the Modify Button dialog box, click the symbol you would like for your macro button.

3. In the Display name: text box, type: Title My Relative.

   Example:

4. Click the OK button to close the dialog box.
Section 2 – Review
You must be able to answer the following questions on your own. As you review the questions below, write in your answer below each one.

1. Where can you find the Relative button to record your macro relative?

2. If you record a relative macro how is this different?

3. Where can you find a macro button to add to the Quick Access Toolbar?

4. How do you assign a macro to a button?

Additional Study
Use the Microsoft Excel help features to look up the key words from this section for additional information. Example: Macros, VBA.
SECTION 3 ◊ MAKE DECISIONS

What will I learn in this section?

IF ... ELSE ... END IF
DO... LOOP
FOR... NEXT
SELECT CASE ... CASE IS...
WORKING WITH RANGES
Making Decisions with code

There are a number of ways to use code to help make decisions. In this section you will first take a look at If… Then statement and the Do… Loop statement. You will end up with For… Next statement.

With the If statement you have a conditional statement. Such as X=2 or another example is “book” = “book”.

<table>
<thead>
<tr>
<th>Operator Symbol</th>
<th>Operator meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

When creating the If statement it can be one line execute or multiple lines.

**Single-line**

If A1content = B1content  Then  ActiveCell.Font.Bold = True

**Multiple-line**

If A1content = B1content  then

ActiveCell.Font.Bold = True
Selection.NumberFormat = "m/d/yy"
End If

**Multiple-line Else**

If A1content = B1content then
Activecell.Font.Bold = True
Selection.NumberFormat = "m/d/yy"

Else

Activecell.Font.Italic = True

End If

Now a little about how a Do – Loop statement works. Do loops are great tools to have when you are working with your spreadsheet. They will until a condition is met or until a condition is changes to something.

In our example ( after the IF ) you will have the loop continue until it encounters a blank cell.

Understanding the **DO...Loop** syntax:

```
Do
    Statement
    Loop while condition
Examples:
    Do
        ActiveCell > 5000 then ActiveCell.Font.Bold = True
    Loop while activecell <> ""
```

```
Do
    Statement
    Loop until condition
Examples:
    Do
        ActiveCell > 5000 then ActiveCell.Font.Bold = True
    Loop until activecell = ""
```
In the next few steps you will build an If statement to check to see if a number is over 5000 and if it is make it bold.

1. Open the file **IfCase.xlsm**.
2. Click on cell **C2**.
3. Press the hotkeys **ALT + F11**
   
   *This will open the VBA editor.*
4. Right click on the Microsoft Excel Objects folder under **VBAProject (IfCase.xls)**.

   ![VBA Project with Insert and Module options](image)

5. With the mouse point to Insert menu, then click the **Module** menu.

   Your screen should look like this.

   ![Module options in VBA Project](image)

   *You now have a Module1.*
6. Type: **Sub MyFirstIf()**
7. Press the **[ENTER]** key.
Your screen should look like this.

<table>
<thead>
<tr>
<th>Sub MyFirstIf()</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Sub</td>
</tr>
</tbody>
</table>

8. Type: **If ActiveCell > 5000 then ActiveCell.Font.Bold = True**

9. Click the **Save** button in the Visual Basic Editor.

**Testing your If statement**

1. Click the **Macros, View Macros** menu item

2. Run the macro **MyFirstIf**.

3. The cell **C2** should now be bold.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Apr</td>
<td>May</td>
<td>Jun</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Widget 201</td>
<td>2310</td>
<td>2430</td>
<td>2520</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Widget 202</td>
<td>5230</td>
<td>4870</td>
<td>5440</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Widget 203</td>
<td>3450</td>
<td>3670</td>
<td>3980</td>
<td></td>
</tr>
</tbody>
</table>

Now let’s take the IF and have it check all the numbers. To do this you will need the macro to move the active cell during the process of checking. First you will record a macro to move down one cell. This is a quick way to generate code without the need to look it up.

Once you have recorded the code, you will take that recorded code and add it to your **MyFirstIf** macro.
**Turn on the Relative Reference**

You want to use this code in different columns, so an absolute is not needed in this case. You will need to use relative for the macro to work correctly.

1. On the Developer’s ribbon, click the Use Relative References if needed.
2. Click Record Macro button and name the macro **MoveDownOne**.
3. Confirm the dialog box is set to Store macro in: This Workbook.
4. Click OK to start the macro.
5. Press the [ENTER] key and click the Stop Recording button.

**Copying Code**

1. Press **ALT + F11** to open the Visual Basic Editor.
2. Double click the Module2 to see the new macro and code.
3. **Copy**: `ActiveCell.Offset(1, 0).Range("A1").Select`
4. Double click the Module1 folder to see the macro **MyFirstIf**.
5. Place the cursor at the end of the word **True** and press [ENTER].
6. With the cursor under your IF statement, click the Paste button.

Your screen should look like this.

```
Sub MyFirstIf()
    If ActiveCell > 5000 Then ActiveCell.Font.Bold = True
    ActiveCell.Offset(1, 0).Range("A1").Select
End Sub
```

7. Click the Save button to save the macro code.

**Adding Do... Loop statement**

In the macro above one of the problems is the macro is not checking each number in the list. So, you need some code that will check the first number, then move
down one row and check the next number. This needs to continue until all the numbers have been checked.

The Do loop statement is idea for repeating a process over and over until a condition is met or changes. You will now add the Do Loop to your current macro to have it repeat the two steps until the macro reaches an empty cell.

1. Press **ALT + F11** to open the Visual Basic Editor.
2. Double click the **Module1** to see the macro *MyFirstIf* and code.
3. Click at the end of the "Sub MyFirstIf()" line and press [ENTER].
4. Type: **Do**
5. Click at the end of the "ActiveCell.Offset(1, 0).Range("A1").Select" line and press [ENTER].
6. Type: **Loop Until Active Cell = " "**

Your screen should look like this.

![Code snippet showing the Do Loop](image)

**Notice** the loop statement: Loop Until ActiveCell = ""

With this statement you are asking the Do Loop to process the code in between the do and the Loop statement but before it is repeated again check the active cell to see if it is empty or not. If the active cell is empty the loop will not continue. If the cell is NOT empty VB will jump back up to the DO and process the code again.

**Testing the Do Loop**

1. Select **C5** on **Sheet2** in the **IfCase.xls** workbook.
2. Run the **MyFirstIf** macro.
3. The cell C2 should now be **bold**.
Now let’s take the IF and have it check all the numbers. To do this you will need the macro to move the active cell during the process of checking. First you will record a macro to move down one cell. This is a quick way to generate code without the need to look it up.

Once you have recorded the code, you will take that recorded code and add it to your MyFirstIf macro.

You want to use this code in different columns, so an absolute is not needed in this case. You will need to use relative for the macro to work correctly.

1. On the Developer’s ribbon, click the Use Relative References if needed.

2. Click Record Macro button and name the macro MoveDownOne.

3. Confirm the dialog box is set to Store macro in: This Workbook.

4. Click OK to start the macro.

5. Press the [ENTER] key and click the Stop Recording button.

Copying Code

1. Press ALT + F11 to open the Visual Basic Editor.

2. Double click the Module2 to see the new macro and code.

3. Copy: ActiveCell.Offset(1, 0).Range(" A1").Select

4. Double click the Module1 folder to see the macro MyFirstIf.

5. Place the cursor at the end of the word True and press [ENTER].

6. With the cursor under your IF statement, click the Paste button.
Your screen should look like this.

![Code Snippet]

7. Click the **Save** button to save the macro code.

**Testing your new If statement**

1. Select the cell **C6** and click the **Bold** button to remove bold.
2. Select the cell **C5**.
3. Run the **MyFirstIf** macro.

What happen to the numbers in the list?

What is the macro doing at this point?

How can we make it work?

**Adding Do... Loop statement**

In the macro above one of the problems is the macro is not checking each number in the list. So you need some code that will check the first number, then move down one row and check the next number. This needs to continue until all the numbers have been checked.

The Do loop statement is idea for repeating a process over and over until a condition is met or changes. You will now add the Do Loop to your current macro to have it repeat the two steps until the macro reaches an empty cell.

1. Press **ALT + F11** to open the Visual Basic Editor.
2. Double click the **Module1** to see the macro **MyFirstIf** and code.
3. Click at the end of the "Sub MyFirstIf()" line and press [ENTER].
4. Type: **Do**
5. Click at the end of the "ActiveCell.Offset(1, 0).Range("A1").Select" line and press [ENTER].
6. Type: **Loop Until ActiveCell = ""**
Your screen should look like this.

```
Sub MyFirstIf()
    Do
        If ActiveCell > 5000 Then ActiveCell.Font.Bold = True
        ActiveCell.Offset(1, 0).Range("A1").Select
    Loop Until ActiveCell = ""
End Sub
```

Notice the loop statement: Loop Until ActiveCell = ""

With this statement you are asking the Do... Loop to process the code in between the Do and the Loop statement but before it is repeated check the active cell to see if it is empty or not. If the active cell is empty the loop will not continue. If the cell is NOT empty the code will be repeated again.

**Testing the Do Loop**

1. Select **C5** on Sheet2 in the **IfCase.xlsm** workbook.
2. Run the **MyFirstIf** macro.

Your screen should look like this.

<table>
<thead>
<tr>
<th></th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2310</td>
<td>2430</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2300</td>
<td>4870</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2300</td>
<td>4870</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4390</td>
<td>6321</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>2310</td>
<td>2430</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>5230</td>
<td>4870</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>3450</td>
<td>3670</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>4390</td>
<td>4010</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>2310</td>
<td>6842</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>5230</td>
<td>4870</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>3450</td>
<td>3670</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>4390</td>
<td>4010</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>2310</td>
<td>2430</td>
</tr>
<tr>
<td>18</td>
<td>14</td>
<td>5230</td>
<td>7890</td>
</tr>
<tr>
<td>19</td>
<td>15</td>
<td>3450</td>
<td>3670</td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>4390</td>
<td>5213</td>
</tr>
<tr>
<td>21</td>
<td>17</td>
<td>2310</td>
<td>2430</td>
</tr>
<tr>
<td>22</td>
<td>18</td>
<td>5230</td>
<td>4870</td>
</tr>
<tr>
<td>23</td>
<td>19</td>
<td>3450</td>
<td>3670</td>
</tr>
<tr>
<td>24</td>
<td>20</td>
<td>4390</td>
<td>4010</td>
</tr>
<tr>
<td>25</td>
<td>21</td>
<td>2310</td>
<td>2430</td>
</tr>
<tr>
<td>26</td>
<td>22</td>
<td>5230</td>
<td>4870</td>
</tr>
<tr>
<td>27</td>
<td>23</td>
<td>3450</td>
<td>3670</td>
</tr>
<tr>
<td>28</td>
<td>24</td>
<td>4390</td>
<td>4010</td>
</tr>
<tr>
<td>29</td>
<td>25</td>
<td>2310</td>
<td>2430</td>
</tr>
</tbody>
</table>

Now that you have learned how to repeat a process you can expand your selections by using the Select Case statement.

Understanding the **SELECT CASE** syntax:
Select Case {expression}

Case Is {value 1}
  Statement
Case Is {value 2}
  Statement
Case Is {value 3}
  Statement
Case else (optional)
  Statement
End Select

**Example of a Case statement:**

Select Case ActiveCell
  Case Is < 4000
    Selection.Font.ColorIndex = 7
  Case Is < 5000
    Selection.Font.ColorIndex = 5
  Case Is < 6000
    Selection.Font.ColorIndex = 3
End Select

In your next macro you will use a *Select Case* to check the size of the numbers and based on the size of the number give them a color.

In the Microsoft Visual Basic Help you can look up the Pattern ColorIndex Property to find the below table of colors.
Color is a great way to have items on your spreadsheet show up and be easily identified.

1. In Module1 type the following code above the MyFirstIf macro:

```
Sub colorformatting()
    Do
        Select Case ActiveCell
            Case Is < 4000
                Selection.Font.ColorIndex = 7
            Case Is < 5000
                Selection.Font.ColorIndex = 5
            Case Is < 6000
                Selection.Font.ColorIndex = 3
        End Select
        ActiveCell.Offset(1, 0).Range("A1").Select
    Loop Until ActiveCell = ""
End Sub
```

Notice that you are using the Do Loop and the line of code that moves the active cell down one. Here you are beginning to add code that you have already learned to make your code more versatile.

2. After the macro is complete, select D5 on Sheet2 in the IfCase.xlsm workbook.
3. Run the colorformatting macro.
Your screen should look like this.

<table>
<thead>
<tr>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>2310</td>
<td>2430</td>
<td>2520</td>
</tr>
<tr>
<td>5230</td>
<td>4870</td>
<td>5440</td>
</tr>
<tr>
<td>3450</td>
<td>3670</td>
<td>3980</td>
</tr>
<tr>
<td>4390</td>
<td>5321</td>
<td>4580</td>
</tr>
<tr>
<td>2310</td>
<td>2430</td>
<td>2520</td>
</tr>
<tr>
<td>5230</td>
<td>4870</td>
<td>5440</td>
</tr>
<tr>
<td>3450</td>
<td>3670</td>
<td>3980</td>
</tr>
<tr>
<td>4390</td>
<td>4010</td>
<td>4580</td>
</tr>
<tr>
<td>2310</td>
<td>6842</td>
<td>2520</td>
</tr>
<tr>
<td>5230</td>
<td>4870</td>
<td>5440</td>
</tr>
<tr>
<td>3450</td>
<td>3670</td>
<td>3980</td>
</tr>
<tr>
<td>4390</td>
<td>4010</td>
<td>4580</td>
</tr>
<tr>
<td>2310</td>
<td>2430</td>
<td>2520</td>
</tr>
<tr>
<td>5230</td>
<td>7890</td>
<td>5440</td>
</tr>
</tbody>
</table>

**Notice** here that the numbers take on a different color based on their size. Your case statement has given a different color to the numbers based on **Case Is** test for less than a certain number.
How to count

One of our issues for working with spreadsheets is to know how many rows or columns we must work with. To find out you can use a simple counter.

Example: Bucket = Bucket + 1

Do

ActiveCell.Offset(1, 0).Range("A1").Select

Bucket = Bucket + 1

Loop Until ActiveCell = ""

This is just one of many ways to count the rows with content.

For… Next statement

The For Next loop uses a counter variable that increases or decreases in value during each repetition or rotation of the loop.

Understanding the For…Next syntax:

<table>
<thead>
<tr>
<th>Declare variable (counter) to hold number</th>
<th>Dim bucket as integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>For counter = start to end Statement</td>
<td>Bucket = 1</td>
</tr>
<tr>
<td>Next counter</td>
<td>For bucket = 1 to 4</td>
</tr>
<tr>
<td></td>
<td>If Activecell &gt; 5000 then</td>
</tr>
<tr>
<td></td>
<td>Activecell.font.bold = true</td>
</tr>
<tr>
<td></td>
<td>Activecell.offset(1, 0).Range(&quot;A1&quot;).Select</td>
</tr>
<tr>
<td></td>
<td>Next bucket</td>
</tr>
</tbody>
</table>

In the example on the right the For… Next loop will count to 4. There are three (3) numbers in this structure; Bucket will keep track of loops are done, the one (1) in the 1 to 4 is the starting point of the loops (the loop starts counting at 1), the four (4) is the number of times the loop will be done and the ending point of the loops.

Another way to think of this is it will repeat the command lines 4 times (1 to 4). Bucket is used here to match the starting number and keep track of each repeat that the loop does. That is why we call it the counter, it retains the current number the loop has done.

When the counter (Bucket) reaches the number to the right of “to” then the loop will stop.
(Dim is an abbreviation of dimension, a holdover from the old BASIC language. It would make more sense to use the word Declare, but we are stuck with Dim.)

Declaring variables has two advantages. First, it helps catch spelling mistakes. Suppose you use the variable bucket several times in a sub, but in one case you misspell it as buckte. If you have already declared bucket in a Dim statement, VBA will catch your spelling error, reasoning that buckte is not on the list of declared variables.

The second reason for declaring variables is that you can then specify the types of variables you have. Each type requires a certain amount of computer memory, and each is handled in a certain way by VBA. It is much better for you, the programmer, to tell VBA what types of variables you have than to let it try to determine them from context. The variable types used most often are the following.

- String (for text like “Bob” or “The program ran without errors.”)
- Integer (for integer values in the range −32,768 to 32,767)
- Long (for really large integers beyond the Integer range)
- Boolean (for variables that can be True or False)
- Single (for numbers with decimals)
- Double (for numbers with decimals where you require more accuracy than with Single)
- Currency (for monetary values)
- Variant (a catch-all, where you let VBA decide how to deal with the variable)

### Symbols for Data Types

It is also possible to declare (some) data types by the symbols in Table 5.1. For example, you could use Dim bucket@ or Dim bucket%, where the symbol follows the variable name. This practice is essentially a holdover from older versions of the BASIC language, and you might see it in legacy code. However, I don’t recommend using this rather obscure shorthand way of declaring variables. After all, would you remember them better?

<table>
<thead>
<tr>
<th>Symbols for Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer %</td>
</tr>
<tr>
<td>Long &amp;</td>
</tr>
<tr>
<td>Single !</td>
</tr>
<tr>
<td>Double #</td>
</tr>
<tr>
<td>Currency @</td>
</tr>
<tr>
<td>String $</td>
</tr>
</tbody>
</table>

When you use the Dim statement in a procedure, you put the Dim statement at the beginning of the procedure. Use a Dim statement to declare the object type of a variable. In the above the word bucket is used as a variable. You can use any word you like for the variable name.

A variable is the named storage location that can contain data that can be modified during program execution. Each variable has a name that uniquely identifies it within its scope. A data type can be specified or not.
Variable names must begin with an alphabetic character, must be unique within the same scope, can’t be longer than 255 characters, and can’t contain an embedded period or type-declaration character.

If you don’t specify a data type, the variable is a Variant by default.

**Variable Naming Conventions**

Programmers have surprisingly strong feelings about variable naming conventions. The one thing they all agree on is that variable names should indicate what the variables represent. So, it is much better to use a name such as taxRate than to use a generic name like x. Your code becomes much easier to read, both for others and for yourself, if you use descriptive names.

Beyond this basic suggestion, however, there are at least three naming conventions used in the programming world, and each has its proponents. The Pascal convention uses names like TaxRate, where the first letter in each “word” in the name is upper case. The camel convention is similar, but it does not capitalize the first word. Therefore, it would use the name taxRate. (The term camel indicates that the hump is in the middle, just like a camel.) Finally, the Hungarian convention, named after a Hungarian programmer, prefixes variables with up to three characters to indicate their variables types. For example, it might use the name sngTaxRate to indicate that this variable is of type Single. Other commonly used prefixes are int (for Integer), bln (for Boolean), str (for String), and so on. The proponents of the Hungarian convention like it because it is self-documenting. If you see the variable sngTaxRate in the middle of a program, you immediately know that it is of type Single, without having to go back to the Dim statement that declares the variable.

Which convention should you use? This seems to depend on which convention is currently in style, and this changes over time. For a while, it seemed that the Hungarian convention was the “in thing,” but it results in some rather long and ugly variable names. At present, the camel convention appears to be the most popular, so I have adopted it throughout this book. But if you end up programming for your company, there will probably be a corporate style that you will be required to follow.

**Constants**

The term *variable* means that it can change. Specifically, the variables discussed earlier can change values as a program runs—and they often do. There are times, however, when you want to define a *constant* that never changes during the program. The reason is usually the following. Suppose you have a parameter such as a tax rate that plays a role in your program. You know that its value is 28% and that it will never change (at least, not within your program). You could type the value 0.28 every place in your program where you need to use the tax rate.
However, suppose the tax rate changes to 29% next year. To use your old program, you would need to search through all of the lines of code and change 0.28 to 0.29 whenever it appears. This is not only time-consuming, but it is prone to errors. (Maybe one of the 0.28 values you find is not a tax rate but is something else. You don’t want to change it!) A better approach is to define a constant with a line such as the following.

\[
\text{Const taxRae} = 0.28
\]

This line is typically placed toward the beginning of your sub, right below the variable declarations (the Dim statements). Then every place in your sub where you need a tax rate, you type taxRate rather than 0.28. If the tax rate does happen to change to 29% next year, all you have to change is the value in the Const line.\(^2\) Another advantage to using constants is that your programs don’t have “magic numbers.” A magic number is a number found in the body of a program that seems to appear out of nowhere. A person reading your program probably has no idea what a number such as 0.28 represents (unless you explain it with a comment or two). In contrast, if the person sees taxRate, there is no question what it means. So, try your best to use constants and avoid magic numbers.\(^3\)

**Built-in Constants**

There are many built-in constants that you will see in VBA. They are either built into the VBA language, in which case they have the prefix vb, they are built into the Excel library, in which case they have the prefix xl, or they are built into the Microsoft Office library, in which case they have the prefix mso.

Actually, these constants all have integer values, and they are all members of enumerations. A simple example illustrates the concept of an enumeration. Consider the Color property of a Font object. It can be one of eight possible integer values, and no one on earth would possibly memorize these eight values. (They are not 1 through 8.) Instead, you remember them by their constant names: vbBlack, vbBlue, vbCyan, vbGreen, vbMagenta, vbRed, vbWhite, and vbYellow. Using these constants, you can change the color of a font in a line such as:

\[
\text{Range(“A1”).Font.Color = vbBlue}
\]

Similarly, Excel has a number of enumerations. One that is useful when dealing with ranges is the set of possible directions, corresponding to the four arrows keys: xlDown, xlUp, xlToRight, and xlToLeft. Again, these constants are really integer values that no one in the world remembers. You remember them instead by their more suggestive names.

To view the many enumerations for VBA, Excel, and Office, open the Object Browser, select the VBA, Excel, or Office library, and search the class list for items.
starting with Vb, Xl, or Mso. Each of them is an enumeration that holds a number of built-in constants. For example, the XlDirection enumeration holds the constants xlDown, xlUp, xlToLeft, and xlToLeft, and the VbMsgBoxStyle enumeration holds all the constants that correspond to message box icons and buttons.

**Object Variables**

There is one other type of variable. This is an Object variable, which “points” to an object. For example, suppose you have a Range object, specified by the range name Scores on a worksheet named Data, that you intend to reference several times in your program. To save yourself a lot of typing, you can Set a range object variable named scoreRange to this range with the lines:

```
Dim scoreRange as Range
Set scoreRange = ActiveWorkbook.Worksheets("Data").Range("Scores")
```

From then on, you can simply refer to scoreRange. For example, you could change its font size with the line:

```
scoreRange.Font.Size = 12
```

In the next few steps you will check a list of 25 numbers. There are 25 products and no matter which spreadsheet you open with a product list, there is all ways 25 items. Because you know the total number ahead of time you want to create a For Next loop to check your numbers.

1. Click on Sheet2 tab and click E5 in the IfCase.xls workbook.
2. Press ALT + F11 to open the Visual Basic Editor.
3. At the bottom of Module1 type the following:

   ```vba
   Sub CheckMy25() 'begin
   Dim number1 As Integer
   number1 = 1
   For number1 = 1 To 25
   If ActiveCell > 5000 Then ActiveCell.Font.Bold = True
   ActiveCell.Offset(1, 0).Range("A1").Select
   Next number1
   End Sub
   ```

4. Click on Sheet2 tab and click E5 in the IfCase.xlsm workbook.
5. Run the **CheckMy25** macro.

**Notice** you have a number of items Bold and the active cell ended on the blank cell below the last column. Why because you told the macro to do the **IF** 25 times.

50
**Working with objects using a For ... Next statement**

An Object in VBA is something you name, control, and manipulate in procedures. In Excel an object can be; worksheet, selected range, or charts.

You can assign variables to refer to an object, it is easier to refer to the objects themselves. The SET statement is used to assign a variable to an object. A common object is the active cell or the activecell in a selected range.

Set Selection1 = ActiveCell

Here you are using "Selection1" to be the name of the object.

With this you have given the active cell a name that you can use in your macro.

In the next few steps you will use the SET variable to loop through the For...Next statement. Plus you will use an IF... Else... End If statement to make a decision on Bold or Italic formatting based on the size of the number.

You will change your current CheckMy25 macro to check all the numbers in a selected area; the range you have selected.

1. Change your **CheckMySelection** macro to look like this:

```vba
Sub CheckMySelection()
    Set Selection1 = ActiveCell
    For Each Selection1 In Selection
        If Selection1 > 5000 Then
            Selection1.Font.Bold = True
        Else
            Selection1.Font.Italic = True
        End If
    Next Selection1
End Sub
```

At the top you have SET your variable - then in the FOR EACH - NEXT statement you have setup your loop to step through the highlighted area.
2. Select the range **E5:E29** on Sheet2.

3. Run the macro **CheckMySelection**.

You could select the whole range of numbers and run the macro again and it would do all the numbers.

**How to select a range based on starting point**

In Excel you can extend the highlight from the active cell to the last cell in the range. In the next few steps you will type out the code that will do just that.

- **anchor_cell** will be used to pickup the current cell address and hold it.
- **anchor_cell** is a bucket to hold the cell address.

**ActiveCell.End** can be used to go in all four directions.

*Example:* `xlDown, xlToLeft, xlToRight, xlUp`

In your code you want it to go down and to the right to find the last cell in the bottom row.

The last bit of your code is to select the complete range:

```
Range(anchor_cell, ActiveCell).Select
```

*Think of it as:* `Range(“E5”, “G29”).Select`

This last step takes the point of the anchor (the cell you first select) and the last cell at the bottom right and extends the highlight from top left to the bottom right.

After you have created and used the code, in the next section you will copy this macro over to the personal.xls file to use globally. This will be explained in the next section.

1. Type the following in your **IfCase.xlsm** Module1 sheet below the other macros.

   ```
   Sub SelectWholeRange()
   Dim anchor_cell As String
   anchor_cell = ActiveCell.Address
   ActiveCell.End(xlDown).Select
   ActiveCell.End(xlToLeft).Select
   Range(anchor_cell, ActiveCell).Select
   End Sub
   ```

2. Select Sheet2, click on cell **C5**.
3. Run the macro `SelectWholeRange`.
Your screen should look like this.

<table>
<thead>
<tr>
<th>Widget</th>
<th>2104</th>
<th>2105</th>
<th>2106</th>
</tr>
</thead>
<tbody>
<tr>
<td>2101</td>
<td>2102</td>
<td>2103</td>
<td>2104</td>
</tr>
<tr>
<td>2105</td>
<td>2106</td>
<td>2107</td>
<td>2108</td>
</tr>
<tr>
<td>2109</td>
<td>2110</td>
<td>2111</td>
<td>2112</td>
</tr>
<tr>
<td>2113</td>
<td>2114</td>
<td>2115</td>
<td>2116</td>
</tr>
<tr>
<td>2117</td>
<td>2118</td>
<td>2119</td>
<td>2120</td>
</tr>
<tr>
<td>2121</td>
<td>2122</td>
<td>2123</td>
<td>2124</td>
</tr>
<tr>
<td>2125</td>
<td>2126</td>
<td>2127</td>
<td>2128</td>
</tr>
<tr>
<td>2129</td>
<td>2130</td>
<td>2131</td>
<td>2132</td>
</tr>
</tbody>
</table>

This can be a handy bit of code to use in other macros.

Now that you have two macros you can place one in another and have the two macros work together.

**How to add one macro in another's routine**

In Excel you can use macros stored in your workbook to run from another macro.

Here you just add the name of the macro. In step 1 of the prior page you created a macro to select a range of cells with data. Here you can see that we added the name of the macro “CheckMySelection” to the next to the last line.

```vba
Sub SelectWholeRange()
    Dim anchor_cell As String
    anchor_cell = ActiveCell.Address
    ActiveCell.End(xlDown).Select
    ActiveCell.End(xlToRight).Select
    Range(anchor_cell, ActiveCell).Select
    CheckMySelection
End Sub
```

On page 40 you create the macro to check your selection of items and if greater than 5000 make them bold if not make them italic.

This first part of the macro will highlight the range for you and the name you added, **CheckMySelection** will mark them.
Section 3 – Review

You must be able to answer the following questions on your own. As you review the questions below, write in your answer below each one.

1. Can an IF statement be just one line?

2. Will a DO loop check before it runs or after it runs?

3. Why would you use a FOR – NEXT loop?

4. Why would you declare a variable at the beginning of your macro?

Additional Study
Use the Microsoft Excel help features to look up the key words from this section for additional information. Example: AutoSum, AutoFilter.
SECTION 4 ◊ PERSONAL.XLSB

What will I learn in this section?

Personal.xlsb store for global use
Setup for Auto run
Final Exercise
Personal Workbook Macros

There will be many of your macros that you would like to use over and over; no matter what workbook you open. A macro of this type is a global macro and should always be stored in the Personal.xls workbook.

This workbook is stored in the XLSTART subfolder under your login. Example:  C:\Documents and Settings\{userlogin}\Application Data\Microsoft\Excel\XLSTART

How do I create a Personal.xlsx file?

1. In Excel, click the Record Macro button.

2. Type: test

3. Click the drop down button for Store macro in: list and select Personal Macro Workbook.
4. Click the **OK** button to start the recording.

5. Press **[ENTER]** and click the **Stop Recording** macro button.

6. Press the **ALT + F11** shortcut key to open the VBE.

   Your screen should look like this.

   ![VBE Screen](image)

   7. Click the **Save** button in the VBE to save your new **PERSONAL.XLSB** workbook.

   You can now delete the test macro and copy any macro that you have already created to the personal.xlsb workbook to use globally.

   In the future you can record your macro in the personal.xlsb when you would like them to be global.

   **Auto Run Macro(s)**

   In Excel there is a way to make sure a macro is ran based on the event, that event is the opening of an excel workbook.

   You might like to have a macro run to check something in the spreadsheet before the user starts working, or you might like to prompt the user to do or check something before they start to work.

   To have your macro run as soon as the workbook is open you just name the macro **Auto_Open**. If you would like something to be done as the workbook is closing, you name the macro **Auto_Close**. These two names are **unique** in
Excel and Excel understands what it should do with the code placed in either macro.

Auto_Open

Try the below example to see how the Auto_Open macro can work for you.

1. Open a **new** workbook.
2. Save the workbook as **test.xlsm**
3. Press **ALT + F11** to open the VBE.
4. Add a module to the workbook.
   
   (*Hint: right click the Microsoft Excel Objects folder.*)

5. Type the **below** macro.

   ```vba
   Sub auto_open()
   MsgBox "This is just a message!"
   End Sub
   ```

6. **Save** the workbook and then **reopen**.

   Your screen should look like this.

7. Click the **OK** button to close the message.

   You can use the Auto_Open to do a number of things; pre-fill cells, formatting, check for… anything you like.
Auto_Close

You also use the Auto_Close, this can be a life saver for any workbook that you need to check items before it closes. This macro will run every time you close, or should I say try to close your workbook. If the items, you are check on are not completed or left blank etc. you can have the file not close and prompt the user to do something more to the spreadsheet.

Example of an Auto_Close macro:

```vba
Sub Auto_Close()
    Range("A1").Select
    If ActiveCell = "" Then
        ActiveCell = Format(Date, "mm/dd/yy")
        ActiveCell = "=today()"
        ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveWorkbook.Save
    ThisWorkbook.Close
    Else
    ThisWorkbook.Saved = True ' this will stop the prompt
    ThisWorkbook.Close
End Sub
```

Final Exercise

You will open the file FinalExercise. Write a macro using a Do Loop to count how many sales where made this day (using the Cost column) of your file. Click on cell F5 and create a macro to 1) count the number of sales made for the day. (see page 46)

2) have a message box appear giving the total number of sales for the day. (see page 59) Remember to add an item on to the text of your message you need to insert the & symbol.

3) After you have completed step 1 & 2, take the number and use it in a macro to make all numbers $6,000 or over bold. (see page 50)

Now all three steps should run together.
## INDEX

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute or Relative</td>
<td>20</td>
</tr>
<tr>
<td>Auto_Close</td>
<td>55</td>
</tr>
<tr>
<td>Auto_Open</td>
<td>54</td>
</tr>
<tr>
<td>Case statement</td>
<td>41</td>
</tr>
<tr>
<td>Code</td>
<td>13</td>
</tr>
<tr>
<td>Color</td>
<td>42</td>
</tr>
<tr>
<td>Customize the Ribbon</td>
<td>20</td>
</tr>
<tr>
<td>Dim statement</td>
<td>44</td>
</tr>
<tr>
<td>Do… Loop statement</td>
<td>36</td>
</tr>
<tr>
<td>DO…Loop</td>
<td>33</td>
</tr>
<tr>
<td>Filter</td>
<td>7</td>
</tr>
<tr>
<td>If statement</td>
<td>32</td>
</tr>
<tr>
<td>Modify</td>
<td>28</td>
</tr>
<tr>
<td>Open</td>
<td>5</td>
</tr>
<tr>
<td>Record Macro</td>
<td>8</td>
</tr>
<tr>
<td>Relative Reference</td>
<td>23</td>
</tr>
<tr>
<td>Stop Recording</td>
<td>10</td>
</tr>
<tr>
<td>Use Relative References</td>
<td>36</td>
</tr>
<tr>
<td>View Macros</td>
<td>6</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>25</td>
</tr>
</tbody>
</table>