CHAPTER 2

Microsoft Excel - Using VBA and VLOOKUP Programmatically

Introduction

Someone once said, "Give a small boy a hammer and before long the small boy discovers that everything needs to be hammered". Microsoft Excel is that hammer for many engineers. In short, the computer spreadsheet has been used in countless ways since its inception in the 1970's. (Some of you may remember such spreadsheet applications as VisiCalc and Lotus 1-2-3, all of which predate the advent of the Windows version of Microsoft Excel in 1986.)

While most users of Excel are reasonably proficient at setting up a simple spreadsheet with rows and columns to track whatever needs to be tracked, fewer users are familiar with the use of functions and even fewer still, use Excel's programming language, Visual Basic for Applications (VBA); to automate the use of functions in spreadsheets.

This chapter was written for anyone who wishes to improve their Microsoft Excel skills. The chapter will provide enough background information to allow someone who has little to no understanding of Microsoft Excel to use the VLOOKUP function both in a standalone formula and as part of a simple VBA program. The ability to use VBA, immediately takes the use of Microsoft Excel to the next level. To make VBA accessible to everyone, we will review formulas and functions first and then finally, work with VBA. Within VBA we will introduce object oriented programming (OOP), something that every engineer (and everyone else) should know.

The chapter makes use of Microsoft Windows Excel 2013 Desktop version. Users who use Excel 2007 and later versions, should be able to make easy use of the information and code that is provided. Expert users of versions of Excel predating 2007 should find it relatively easy to modify what is provided.

Formulas

Before we start talking about formulas, let' s review some of the fundamental details associated with a typical Microsoft Excel worksheet, making sure that we understand some of the terms that we will be using.

Figure 2.1 shows a screen shot of a Microsoft workbook the first time we open a new spreadsheet in Microsoft Excel 2013 (your spreadsheet may look different - not to worry, we are just going to focus on the parts of the worksheet that we need). Note that we have saved the new workbook using the name "My First Workbook.xlsx".

If you have only used Microsoft Excel 2007 or later then the user interface that you see in **Figure 2.1** is the only Microsoft Excel user interface that you will have seen - it is called the Ribbon. For Excel versions prior to 2007 the user would use menus and tool bars. Some of the more important items in **Figure 2.1** include the following.

- The workbook will contain all our work and can contain one or more worksheets. In our example we can see that the default name for our worksheet ("Sheet1") is displayed on the worksheet tab.

- Our worksheet is made up of cells and we can reference a single cell (e.g. C2) or a range of cells (e.g. C10:E15). We can make use of the Name Box to rename cells or ranges.

- If you don't see the **DEVELOPER** Tab on your worksheet, then complete the following steps:
    1. Click the **FILE** Tab;
    2. Click **Options**;
    3. Click **Customize Ribbon**;
    4. In the **Main Tabs** list box, Click the check box beside **Developer;** and
    5. Click **OK.**

- Formulas can be entered either directly in a cell or in the Formula Bar and formulas always start with an equal sign (**=**).

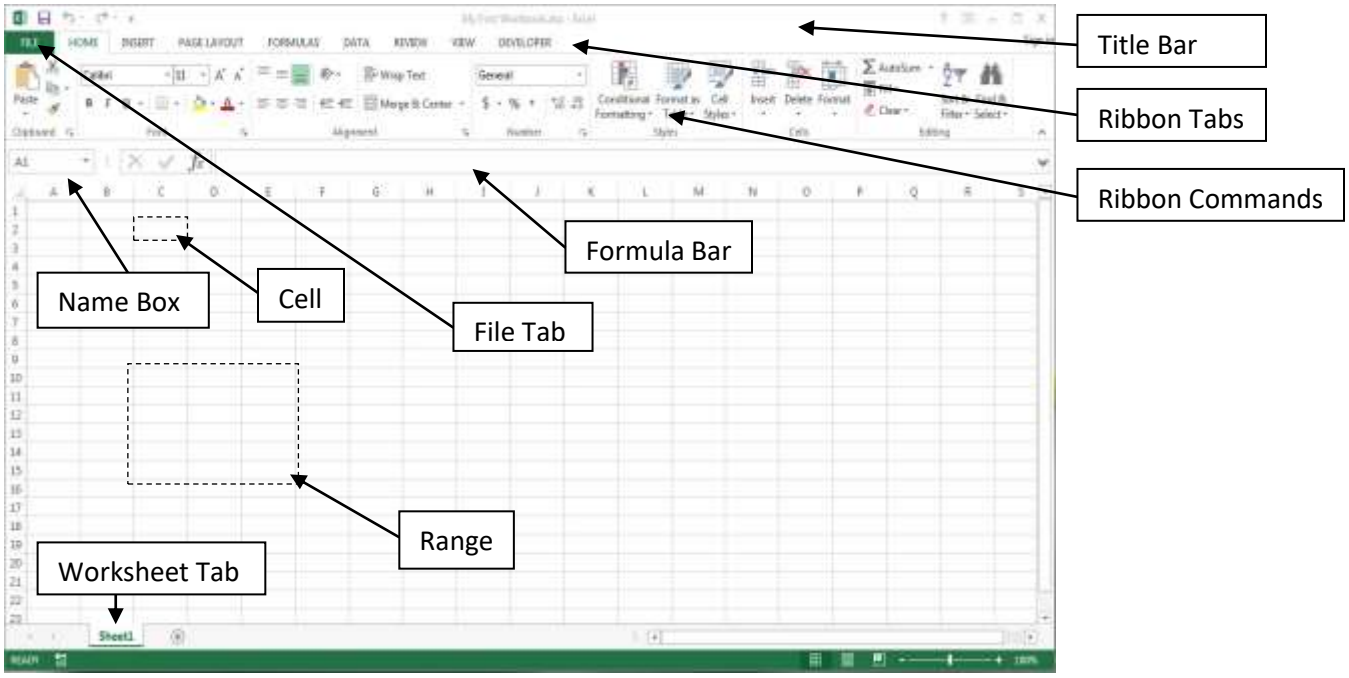


Figure 2.1 - Excel 2013 Blank Worksheet

Formulas are at the heart of what Excel is all about.

A formula entered in a cell can include:

- **Operators**

  An operator is simply a symbol that represents an operation. A formula can contain multiple operators. For a partial list of operators see **Table 2.1**

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
=	Equal to
<	Less than value on right side
>	Greater than value on right side
<=	Less than or equal to value on right side
>=	Greater than or equal to value on right side
<>	Not equal to

Table 2.1 Partial List of Operators

- **Parentheses**

  Parentheses are used to control the order in which the expressions of the formula are calculated. Terms in parentheses are calculated first.

- **Numerical Values or Character Strings**

- **Microsoft Excel Built-in Functions**

  The over 300 Microsoft Excel functions allow the user to manipulate data quickly and easily. We will be working with the **VLOOKUP** function, but more common functions include **SUM** or **COUNT**.

- **Cell References**

  We can refer to cells or ranges of cells and, if we want, we can give these cells or ranges unique names. We can refer to cells on the worksheet that we are working on, on a different worksheet, or even on a worksheet in a different workbook.

Now for a simple example of a formula. Click on cell A4 (i.e. with your Excel cell pointer, click cell A4) and type: **=A1+A2**. To check to see if the formula works enter a numerical value in cell A1 and a numerical value in cell A2. The sum of A1 and A2 should be displayed in cell A4. It is just that easy to use formulas in Excel.

Using Built-in Functions in Formulas

As stated in the section above, Microsoft Excel has over 300 functions. Out of these 300+ functions, Microsoft Excel has three basic lookup functions (HLOOKUP, VLOOKUP and LOOKUP) that allow a user to search a column or row for a specific value and return another value as a result.

VLOOKUP is a vertical lookup, that is, the search for a match moves from row-to-row in the first column of the lookup table. If a match is found in the first column, a value in the same row is returned from a column in the lookup table that you specified. (This should become clearer once we take a closer look at the VLOOKUP function.)

The syntax for the vertical lookup function, VLOOKUP, is as follows.

VLOOKUP (lookup_value, table_array, col_index_num, range_lookup)

Where the four arguments are defined as:

lookup_value - This is a required argument. The value is used to search the first column of the lookup table.

table_array - This is a required argument. The range of cells that contains the lookup table. Uppercase and lowercase text are equivalent.

col_index_num - This is a required argument. It is the column number of the lookup table that contains the return value when a match is found.

range_lookup - This is an optional argument. A logical value that specifies whether you want **VLOOKUP** to find an exact match or an approximate match.

- If an exact value is required (i.e. the **range_lookup** argument is set to **FALSE**) and not found, **VLOOKUP** returns a value of **#N/A**.

- If an exact value is not required (i.e. the **range_lookup** argument is set to **TRUE** or left blank)

To ensure that the function works properly, the first column of the lookup table must be in ascending order. If the **lookup_value** is smaller than the smallest value in the first column of the **table_array, VLOOKUP** will return a value of **#N/A**.

Now, making use of the VLOOKUP function, let's create a worksheet that rates vehicle gasoline usage (i.e. miles-per-US-gallon). Depending on the mileage we are getting the formula will rate our miles-per-US-gallon somewhere between **Very Poor** and **Excellent** (see **Figure 2.2**).

Note that our formula, **=VLOOKUP (B5,D2:F6,3)**, has been entered in cell B6. Using what we know about the **VLOOKUP** function, the value to be rated is entered in cell **B5** (in our example the value to be rated is 35 miles/USG). The three-column lookup table is defined by the range **D2:F6**. If a match is found, the rating will be returned from column **3** (in this case the rating returned is **Very Good**). Since we did not enter a fourth argument (**TRUE** or **FALSE**) into our VLOOKUP, the default is TRUE and an approximate match is returned; this works well since in this case we are working with gas mileage ranges.

So, that is how easy it is to use Microsoft Excel functions in your formulas.

Next, before we use the same VLOOKUP function in a VBA program, let's look at some of the theory behind an objected-oriented programming language like VBA.

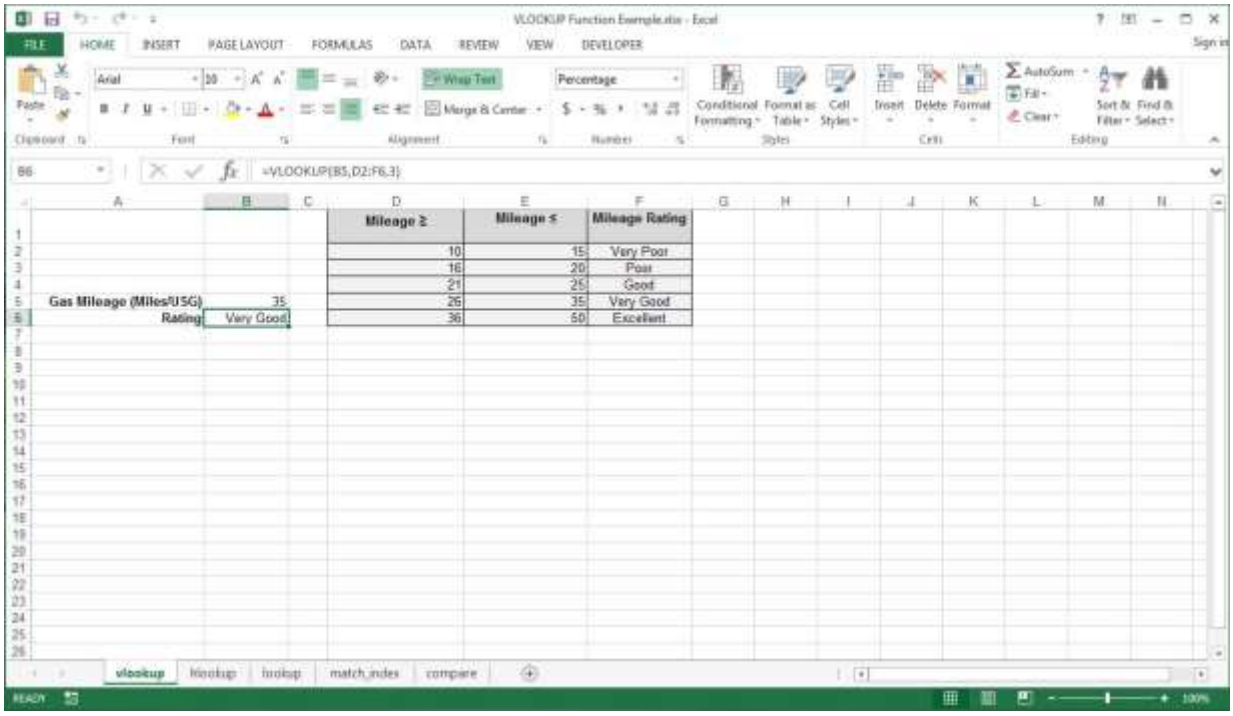


Figure 2.2 - Using VLOOKUP in a Formula

Objected-Oriented Programming

What is object-oriented programming? Object oriented programming has been around since the 1970's, but really did not become well known until the early 1980's. Simply stated, objected-oriented programming in some way tries to emulate how we think about things ("objects") in the real world. In order to describe the real world around us object-oriented programming language has defined the following terms.

Classes

A class describes or is a template for one or more objects. It describes an object or objects. In the programming world, to create an instance of an object we **instantiate** the class giving us an instance of the object.

Object

An object is the actual thing itself. Just as in the real world, an object is a stand-alone item, but just as in the real-world, objects can contain other objects. An object has everything it needs to

be what it is and do what it needs to do. The **object model** or **object library** defines all the objects available for application development.

The object model for Microsoft Excel contains over 100 objects. There are many different object models, including Microsoft Word, Microsoft Access and Microsoft Outlook. Each of these object libraries can be accessed or a **reference** set from within Microsoft Excel.

Methods

Objects can perform actions called Methods.

An **event** is something that happens to an object and the resulting action is defined by a **method**. Excel has a number of built-in routines that can watch for an event to occur. As we will see later in the chapter, clicking on a command button is an example of an event.

VBA has two types of methods, **sub** procedures and **function** procedures. One of the key differences between the two types methods is that functions return a result, subs do not return a result.

Properties

Properties or attributes define the physical characteristics of the object. Each property may have one or more possible values, it depends on the property.

Using a real-world example from the automotive industry:

Class: Truck blue prints

Object: Truck

Method: Start Engine

Properties: Color; Cab Size; Box Size

A truck is a complex machine and we would expect that the Truck object would have a large number of objects contained within the Truck object. For example, we would expect an Engine object to be contained in the Truck object and the Start Engine method to be associated with (an action of) the Engine object. We would also not be surprise to find a Body object and the properties of Color, Cab Size and Box Size to all be properties of the Body object.

So what are the advantages of object-oriented programming? Let's list some of the more significant advantages.

Abstraction

Simply put, abstraction is a process the result of which allows the user to use an object without knowing how it does what it does. Can you imagine how difficult life would be if you had to know the details of how a computer works before you could use it to run your spreadsheets? We will use a number of objects when we use VBA to develop our small VLOOKUP application.

We will not have any idea how the objects do what they do or even how the objects were originally developed. This is a very important advantage.

Encapsulation

Encapsulation is closely related to abstraction and refers to the requirement that objects are able to be standalone. That is, they contain all their own methods, properties and data in order to both exist autonomously and carry out their own actions. The advantage is that classes and, as a result, objects can be reused thus shortening development time.

Inheritance

Classes can inherit another class's methods and properties. This ability, inheritance, can significantly speedup the development of new classes.

Polymorphism

Objects can share methods and properties and yet the implementation of these methods and properties can differ depending on the object that is referenced. Here the advantage is that a familiar or consistent interface be provided between the user and objects - while what is named (method or property) is the same, the differences in implementation can be hidden. The advantage is that little additional programming may be required to add specialized methods.

An example of polymorphism might be that we create a class called "insect". Additionally, we want the insect to move so we create a **move** method. Then we decide to start creating specific insects (these would be known as subclasses) for example, an ant. (The ant subclass would inherit the methods and properties of the insect class.) The ant has its own way of moving so when we move the ant using the ant **move** method, the subclass (with some additional programming) provides the information necessary to move the ant in an ant-like way. We could do the same for any number of different insects. The result would be that the move method has many forms (i.e. polymorphic).

Finally, we are ready to work with Visual Basic for Applications and the Microsoft Excel VLOOKUP function.

VLOOKUP Using VBA

If you have been following this chapter from the beginning, so far, we have learned how to use the Excel function VLOOKUP in a formula which we placed directly onto our worksheet. We will now extend our mileage rating example (see **Using Built-in Functions in Formulas** above) by using VBA to move through a list of mileages (perhaps we are comparing vehicles in order to make a new purchase based on published gasoline mileages - miles/USG) rating each mileage as we move through the list.

Once we are finished our worksheet should look very similar to **Figure 2.3**. To construct the worksheet, complete the following steps.

1. Fill each cell exactly as shown in **Figure 2.3** to create the Results Table.

2. Fill each cell exactly as shown in **Figure 2.3** to create the VLOOKUP Table.
3. Create the ActiveX Command Button by completing the following steps:
    a. Click the **DEVELOPER** tab
    b. Click **Insert**
    c. Click the **Command Button (Active X Control)**
    d. Move the cursor onto the worksheet; Click and hold to draw the button – then release
    e. Click on the button to select it and then Right-Click on the button to display a menu
    f. Scroll down the menu; Click the **CommandButton Object;** Click **Edit**
    g. Change the button caption from "CommandButton1" to "Start Rating"
    h. Click anywhere off the button (on the worksheet) to complete this task

You should now have a worksheet that looks like **Figure 2.3**.

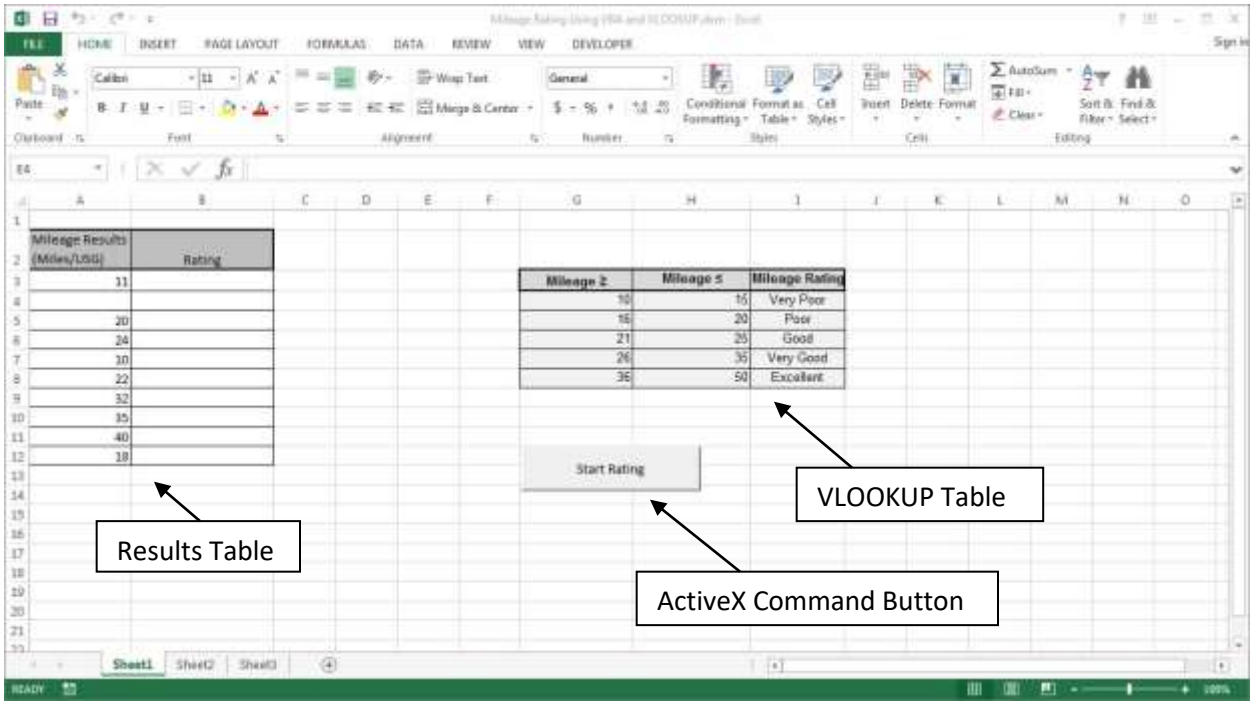


Figure 2.3 - Using VLOOKUP in VBA

We are going to use our command button (the **Start Rating** button) to store and run our code; we will do this using the following steps.

1. Click the **Start Rating** button; Right-Click the **Start Rating** button
2. Scroll down the menu; Click **View Code** which immediately opens the Visual Basic Editor Window (VB Editor)
3. In the Code window type the following code starting just below **Private Sub CommandButton1_Click()**

```
' Declare what each variable will hold using Variable Types
```

```
    Dim intMileage As Integer
    Dim strRating As String

  ' Declare a variable as a "Range" object
    Dim rngLookupTable As Range

 ' Set the "Range" object(rngLookupTable)to point to a defined range
    Set rngLookupTable = Range("g4:i8")

 ' Use VLOOKUP in a loop to determine each Rating of the Mileage Results

        For j = 3 To 12

          intMileage = Cells(j, 1).Value

          If intMileage = 0 Then

            MsgBox ("A Mileage Result was Missed!")

          Else

            strRating = Application.WorksheetFunction.VLookup(intMileage, rngLookupTable, 3, True)

            Cells(j, 2).Value = strRating

          End If

        Next j
```

After inputting the code, your VB Editor should look similar to **Figure 2.4**.

Before we start looking at the various lines of code let's take a look at some of the detail displayed in the VB Editor.

The Project Explorer window, titled **Project - VBAProject** in **Figure 2.4** shows the current workbook as **VBA Project Mileage Rating Using VBA and VLOOKUP.xlsm.** If you recall, our last workbook just had the extension **.xlsx**. The **.xlsm** extension started to be used in Excel 2007 to indicate that the workbook contained programming code (otherwise called **macros** in Excel terminology). Since programming code can be a security risk, this became an important flag for users of Excel workbooks.

Underneath **VBAProject** we find four objects listed, three worksheet modules and the whole workbook (**ThisWorkbook**). Our programming code is stored in **Sheet1** module.

Directly below the Project Explorer window, we find the Properties window (if needed, Click **View** and Select **Properties Window**). This window displays all of the properties associated with the selected object. In our case we can see the properties for the command button that we

created. Note that the name of the command button is the default name **CommandButton1** and the caption for the button is "Start Rating".

Now let's look at the various lines of code in detail.

The first line of code tells Excel the name of the sub procedure that will run when we Click the command button that we placed on the worksheet. Code is stored in what are called modules. The word Private indicates that the sub can only be accessed in this module (**Sheet1**).

```
Private Sub CommandButton1_Click()
```

Methods (sub procedures and functions) almost always make use of variables that allow for the temporary storage of data. The Dim or dimension statement is used to declare variables. The values stored in variables can be of different types and a type is declared for each variable.

```
Dim intMileage As Integer
Dim strRating As String
```

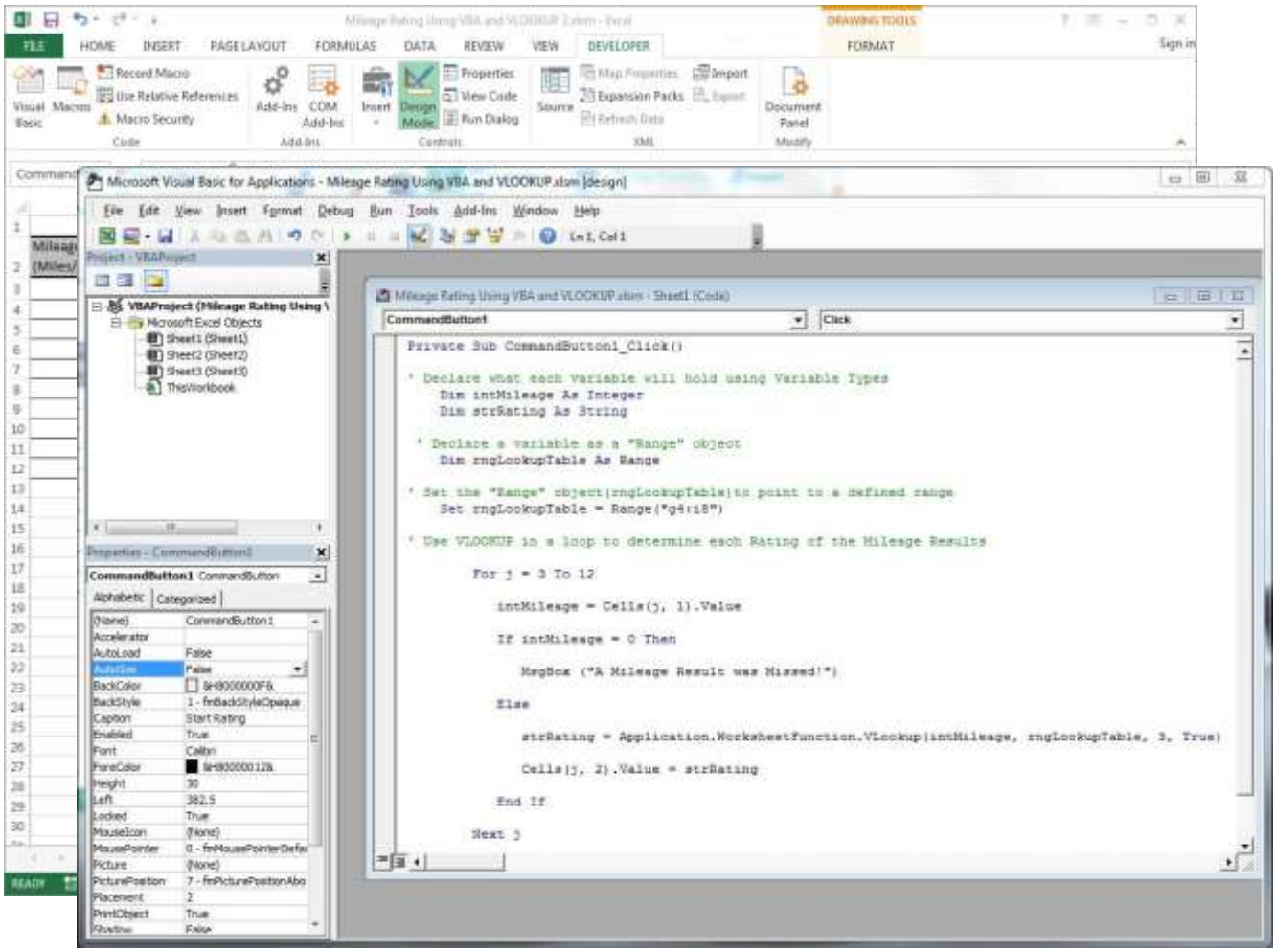


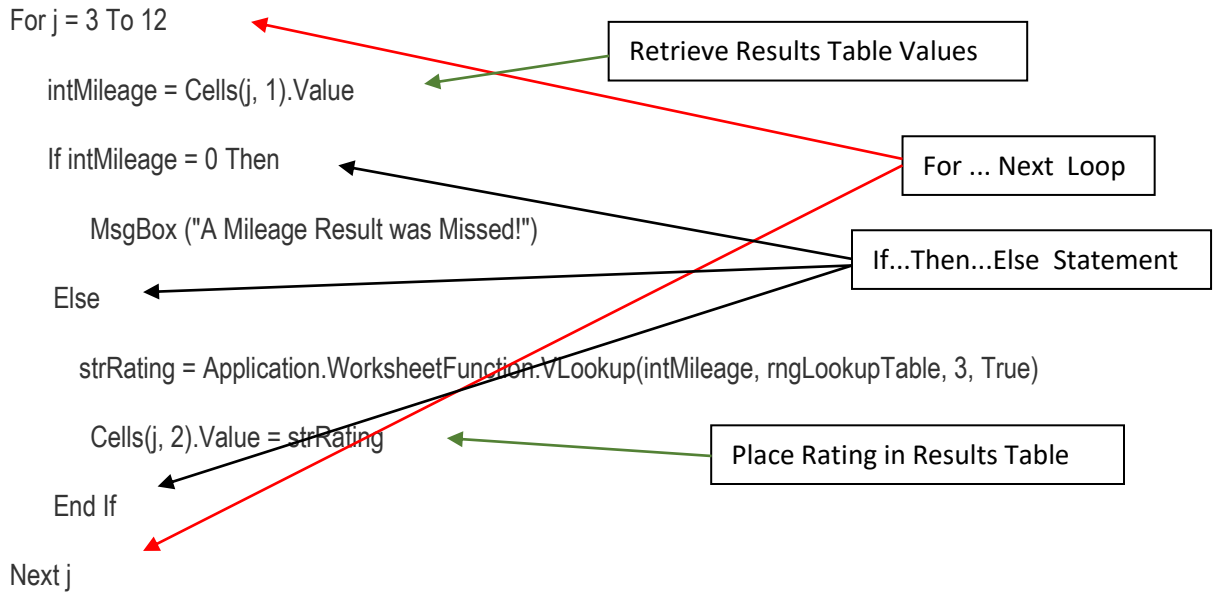
Figure 2.4 - VBA Editor Window - Complete VBA Program

Integers can have a value from -32,768 to 32,767. Strings can hold up to approximately 2 billion characters (depending on your computer's memory).

Declaring and setting an object is a two-step process. First, the object variable name is declared as a specific type of object (in this case Range). Second, the Set command is used to point to the object that the object variable will hold.

```
Dim rngLookupTable As Range
Set rngLookupTable = Range("g4:i8")
```

The next block of code is where the real work is accomplished.



Hopefully it is easy to see that the loop retrieves Results Table values one at a time starting with Cells(3,1).Value where the syntax for Cells is Cells(IndexRow,IndexColumn). The loop will stop after retrieving the value stored in Cell(12,1). Each time the loop is completed, a new value is stored in intMileage.

The value in intMileage is then compared to the value 0 to see if we missed placing a value in our Results Table. If the value stored in intMileage is equal to 0 a message window opens, displaying the message A Mileage Result was Missed!. If the value stored in intMileage is not equal to 0 then the VLOOKUP function is used to determine the mileage rating, the result of which is stored in the variable strRating.

The rating stored in strRating is then placed on the worksheet next to (one cell to the right of) the associated mileage result.

```
Cells(j, 2).Value = strRating
```

Some final notes on the code.

- Remarks are lines of code started with an apostrophe ('). These comment lines are used to document the program.

- The dot operator (.), seen in lines of code like `intMileage = Cells(j, 1).Value`, is used to separate objects and properties.
- While our application performs as designed there are a number of enhancements that we could add. For example, error handling would be a must if we wanted to share our application with other users.

So let's run the code, but before we do, let's save our work.

1. Select the main window containing the worksheet.
2. Click **File** tab
3. Click **Save As**
4. Click **Browse**
5. Click the down arrow associated with **Save as type**
6. Click **Excel Macro Enabled Workbook (*.xlsm)**
7. If desired, change the file name - do not change the extension (.xlsm)
8. If desired, change the save location
9. Click **Save**

To run the code.

1. Make sure that **Design Mode** Ribbon Command is off (if highlighted, Click **Design Mode)**
2. Click **Start Rating** command button.

If you have completed all the tasks as listed above, you should now have a worksheet that looks similar to **Figure 2.5**

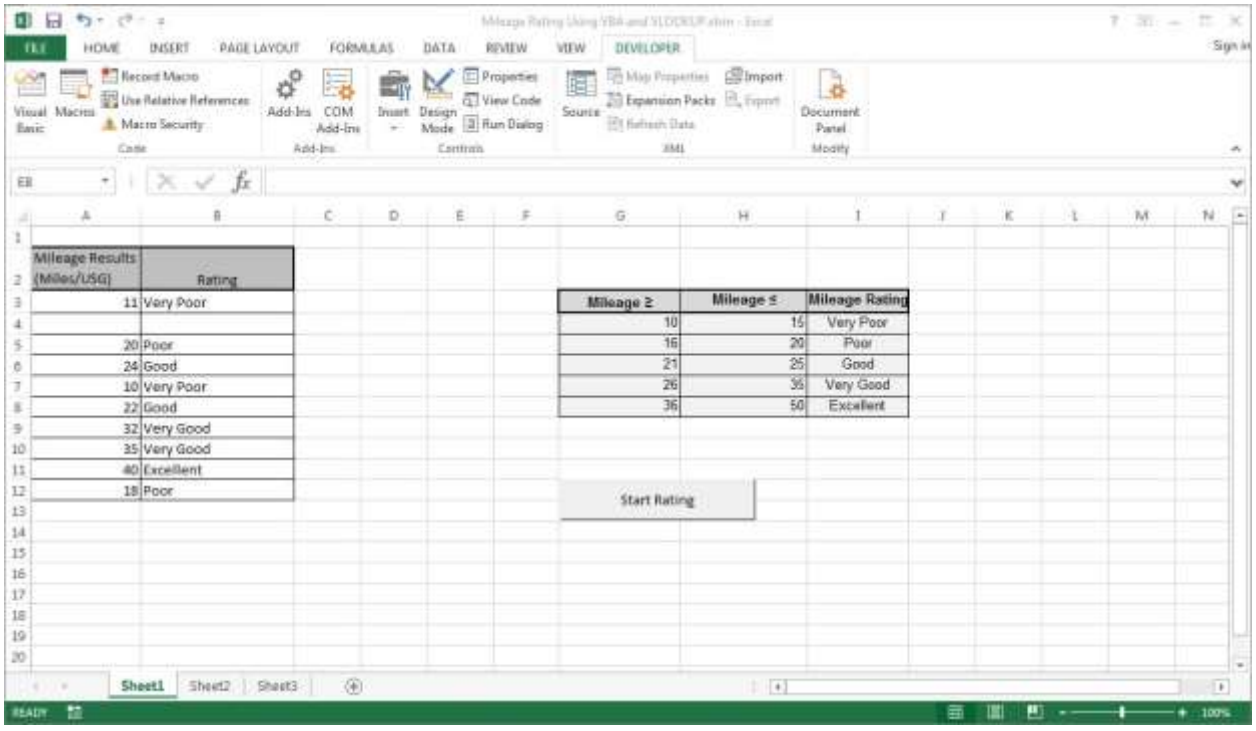


Figure 2.5 - Rating Completed

Conclusion

In this chapter we developed a formula using VLOOKUP function, reviewed the key principles and advantages of object oriented programmming (OOP), and coded and ran a VBA based application making use of the VLOOKUP function.