**Lab Workshop #7**

Purpose: review the setup for programming in VBA

 learn to use the Locals Window and single-stepping of VBA code

  adding user input and output display to VBA code

 use program flow control to implement input validation

 implement a VBA procedure based on a flowchart

1. Launch Excel. When you are going to be working with VBA and Excel in the Boulder Campus computer labs, there are several start-up steps that are necessary. Carry out the following:

1.1 Display the Developer tab.

From the Excel spreadsheet window, click on the **Office Button**,  , and on the **Excel Options**

button,  . With the **Popular** tab selected, 

 (it is the default),

click the checkbox to display the **Developer** tab,  ,

and then click the **OK** button,  .

You should now see the Developer tab in the Excel ribbon,



1.2 Setting Macro Security

 The easiest way to do this is to click the **Macro Security** button shown above on the Developer ribbon. Then, click on the **Enable all macros** option button as shown below.

 

 Finish this step by clicking the **OK** button.

1.3 Setting for Option Explicit

 Switch over to the Visual Basic Editor (VBE) by pressing the ***Alt-F11*** key combination. On the

 menu, select **Tools → Options**,  , and then check the box for

 **Require Variable Declaration**,  . Finish again by clicking the **OK** button.

If you complete these steps on your own personal computer, they will be remembered from one work session to the next; however, in the computer labs on the Boulder Campus, it is likely that you will have to complete these steps each time you begin a login session. Get in the habit.

2. Using VBA's Debugging Tools

 In developing computer programs, bugs[[1]](#footnote-1) inevitably occur. Modern programming tools include features to aid in debugging computer programs, and VBA is no exception.

 Make sure your current project is selected in the Project Explorer. See below.

  Then, from the VBE menu select **Insert → Module**

  (make sure you don't click Class Module). You should see a blank module in

 the Code Window with Option Explicit at the top. 

 Now, type the following subroutine procedure ("Sub", for short) into the Code Window:

 

 You noted that, after you pressed the ***Enter*** key on the first line, Sub Test(), the End Sub line was inserted automatically by the VBE.

 Since no types (Double, Integer, etc.) are listed with the variables in the **Dim** statement, these are all automatically of the Variant type. To examine what happens to variables as the program is executed step-by-step, open the Locals Window. Do this via the View menu.

  The Locals Window should appear at the bottom of the Code Window. You can resize it by dragging the mouse on the border between the two windows. 

 With the editor cursor on the first line of the Sub (it can actually be on any code line), press the ***F8*** key. This puts the VBE into "debug mode" and the first line of the Sub should be highlighted in yellow.

  The yellow arrow on the left shows the VBA statement that will be executed next. In the Locals Window below, you will see a list of the Sub's local variables. They should all be shown at Variant type. Their type personality will change as the Sub is executed line by line.

 

 Press ***F8*** four times to highlight the **s=...** statement. How has VBA "typed" the **a**, **b** and **c** variables? -- note this in the Locals Window.

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Press ***F8*** again. Let your mouse cursor "hover" over the **s** variable, and a tooltip window should open showing you the current value of **s**.

  This is a useful way to follow the values of variables, especially when you don't have the Locals Window open.

 Notice how VBA has "typed" **s** and its value in the Locals Window. Press ***F8*** again to complete the calculations. What are the result and type for **r** shown in the Locals Window?

 **r =** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **r's type:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Press ***F8*** again to terminate execution. You will notice that the Locals Window goes blank.

 If you ever need to stop the Debug Mode partway through the execution, you can click the Reset button up on the VBE toolbar, .

 If you want to execute a Sub straight through, without single-stepping, there are several alternatives:

 from the VBE toolbar,

• click the **Run** button, 

 • from the menu, select **Run → Run Sub/UserForm**, 

 • press the ***F5*** key

 from the Excel spreadsheet window,

 • press the ***Alt-F8*** key combination to get the Macro dialog window

 select the Sub and click the **Run** button

 • define and use a shortcut key combination that runs the Sub

 • place a button on the spreadsheet

 • place a command button on a toolbar

 We will review all of these as we go along learning VBA.

3. Adding User Input and Display Output

 A simple way to add user input to VBA code is via the InputBox function. Using the VBE, replace the three **a =**, **b =**, and **c =** statements with those shown below.

 

 If you single-step execute the Sub, you will see a sequence of three InputBoxes in the Excel spreadsheet window.

 

 Enter values of **7**, **10** and **8** and observe the values and variable types in the Locals Window. What do you observe?

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Now, go back and change the Dim statement to force the type of the **a**, **b** and **c** variables to Double.

 

 Make sure to click the cursor onto another line after making the changes; otherwise, they won't take.

 Step-execute the code again. What do you observe now?

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Add the following Message Box display statement right after the **r =** statement.

 

 The MsgBox function actually displays text strings, not numerical values. So, the ampersand (**&**) is used to join two text strings and the numerical value of r is automatically converted to a text string before it is displayed by MsgBox.

 Run the Sub, but this time run it continuously by pressing the ***F5*** key or clicking the **Run** button.

 You should see a Message Box display like

 

 The only thing about this display that is a bit troublesome is all the figures shown for the value of the radius. You can format that by adding the FormatNumber function to the MsgBox statement, as shown below.

 

 Then, the result rounded for display.

 

 That looks better.

4. Program Flow Control

 In designing computer algorithms, the need to select among alternatives arises frequently. The "If" statements in a computer language generally take care of decisions or selection in program flow. Also, there is a need to repeat sequences of code and "Do-Loop" and "For-Next" statement pairs in VBA provide for this.

 One way to illustrate the use of program flow control is to add validation and protection to the simple Sub you have written to calculate the radius of the largest circle that will fit inside a triangle.

 By observation of the diagram to the right, for a triangle to form, it is necessary that

  and 

 Also, although it's obvious that a requirement is

 

 The reason for stating this is that it would be possible for a negative value to be entered when the Sub is executed.

 A flowchart that illustrates how an input value for **a** can be validated is

 

 You can see a loop in the flowchart. That structure is called "repetition." And, you can see a selection structure, denoted by the diamond with the True/False alternate paths. The VBA code to implement the flowchart is

 

 Notice that the interior of the Do-Loop structure is indented. This is a useful style that shows clearly what code is inside the loop.

 Modify your code as shown above and then add similar validation loops for the entries of the **b** and **c** values.

 Test your Sub to make certain it works. Try out a negative or zero value of one of the entries to check that the validation loop works correctly.

 The remaining issue we have is to check whether the value of **c** provides for a valid triangle. There are many ways to do this. The flowchart below shows one.

 

 This is obviously a more complicated structure, but the flowchart helps immensely in writing the VBA code to implement it. Study the flowchart carefully. Here is the code:

 

 We wouldn't expect you to be able to write this from scratch at this point, but enter it into your Sub and then test it with different cases to check whether it catches invalid inputs. With more practice and a little time, you will be able to develop and code structures like this.

 Add a comment at the top of your Code Window with your name, section number and the date. Print out your Code Window and attach it to your lab worksheet. Save your Excel workbook at **lab7a.xlsm** and close it out.

5. On the following page you will find a flowchart. Open a new Excel workbook and create a VBA procedure (Sub) that implements this algorithm. For the input of the **C** value, use an InputBox function. For display of the result, use a MsgBox function.

 You should first write out your VBA code on a separate sheet of paper. Make certain to start with the Option Explicit statement.

 This algorithm has two loop structures. Make sure you use the correct VBA loop statements for these structures. Make reference to the **Class12** and **Class13** notes, if you need to.

 When you have completed the VBA code, test it with an input of **2**. Demonstrate it to your TA and have her/him initial in the box below.

 In general terms, what does this code compute? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Note: the use of the back-arrow, 🡨 , in a flowchart sequence box means assignment.

 Save your workbook at **Lab7b.xlsm** and close it out.



6. Leave Excel and return to Windows. Log off your computer.

**End of Lab Workshop #7**

1. The origin of the term "bug" in computing was a moth that, in 1947, caused one of the early electronic computers to fail by shorting out one of its relay contacts. The computer's operators coined the term "debugging" when they removed the moth and recorded the incident in their logbook, taping the insect onto the logbook page. [↑](#footnote-ref-1)