Macros 3

There are a lot of helpful functions with macros. We'll explore using message box prompts through macros in this chapter. In the previous chapters, we recorded macros to create macro code, and we did some light editing to the code. However, we haven't directly written macro code.

Practice Spreadsheet

Use the workbook you created in the Macros 1 and Macros 2 chapter.

Message Box

In order to use a message box, we have to write the code in Visual Basic. For this example, we'll create a basic message box to learn how it works.

1. Open **Visual Basic** from the **Developer** tab in the Excel ribbon toolbar.
2. Type **Sub firstMessage()** as the first line to begin a new subroutine.
3. Type **Dim box1 As String** in the second line.
    a. The keyword Dim stands for *dimension*. Basically, the code is telling the computer to set aside memory to save the value of this variable.
4. Type **box1 = MsgBox("You created a message box!", vbOKOnly, "Box 1")** in the third line.
    a. The first argument in the MsgBox function is the text that will appear in the message box to the user.
    b. The second argument defines the type of message box. The options include boxes with OK and Cancel buttons, Yes and No buttons, and others. For this example, a simple OK button will be sufficient.
    c. The third argument is the title text for the message box. This text appears along the top of the box.
5. Type **End Sub** in the fourth line. (See **Figure 30.1**)
6. Press the **Play** button in Visual Basic's toolbar to run the code.

```
Sub firstMessage()

    Dim box1 As String

    box1 = MsgBox("You created a message box!", vbOKOnly, "Box 1")

End Sub
```

Figure 30.1

Congratulations! You have created your first message box in Excel using Visual Basic. There are many useful situations for a message box. For example, at the end of a macro, you may want a message box to notify that the macro has been

completed successfully. (See **Figure 30.2**)

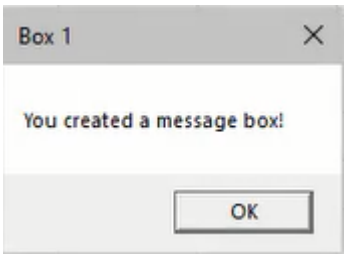

Figure 30.2

Input Box

In other situations, we may want an input box to allow the user to input a response.

1. Open **Visual Basic** from the **Developer** tab in the Excel ribbon toolbar.
2. Type **Sub firstInput()** as the first line to begin a new subroutine.
3. Type **Dim box1 As String** in the second line.
4. Type **box1 = InputBox("Please enter your name", "Name", "Example Name")** in the third line.
    a. Similar to the previous MsgBox function, the InputBox function's first and second arguments define the text displayed in the box's title and main message.
    b. The third argument displays example text in the input field. You should be aware that the user will have to delete this text to type their own.
5. Type **MsgBox box1, vbOKOnly, "Your name"** in the fourth line.
    a. This code will display another box after the user types their name and presses OK in the input box.
    b. The first argument will call the saved string (the user's input) instead of displaying a predetermined message as performed in the previous example.
6. Type **End Sub** in the fifth line. (See **Figure 30.3**)
7. Press the **Play** button in Visual Basic's toolbar to run the code.

```
Sub firstInput()

    Dim box1 As String

    box1 = InputBox("Please enter your name", "Name")

    MsgBox box1, vbOKOnly, "Your name"

End Sub
```

Figure 30.3

The new input box will prompt the user to enter their name and press **OK**. Another message box will be displayed with the user's response. Now we have successfully completed an input box combined with a message box. (See **Figure 30.4**)

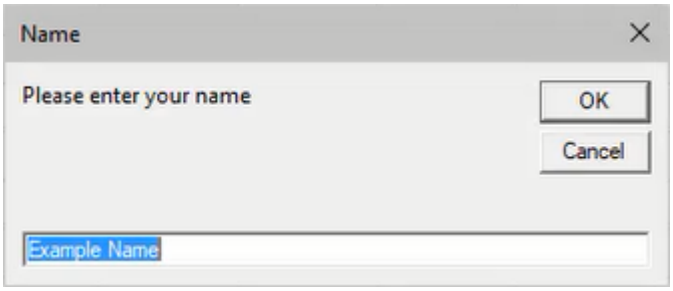

Figure 30.4

Apply If Logic to a Message Box

What if we want to prompt the user with a yes or no question and allow them to follow up by inputting a response? We may also want to record their response in the spreadsheet for review. Code can be written with logical checks to perform the desired function.

1. Type **Sub logicExample()** as the first line to begin a new subroutine.
2. Type **Dim answerOne As String, answerTwo As String** in the second line.
3. Type **answerOne = MsgBox("Did you enjoy this course?", vbYesNo, "Course Eval")** in the third line.
4. Type **If answer One = vbYes Then** in the fourth line.
    a. The **If** and **Then** keywords perform a logic function to run the next line of code only if the user's response meets the specified requirement; in this case, the user must answer *yes*. Otherwise, this If logic check and the associated code will be skipped.
5. Type **MsgBox "Thank you!", vbOKOnly** in the fifth line.
6. Type **End If** in the sixth line.
7. Type **If answerOne = vbNo Then** in the seventh line.
8. Type **answerTwo = InputBox("Please tell us what can be improved", "Improvements")** in the eighth line.
    a. If the user answers *no*, this logic check will run code to prompt the user to type a response.
9. Type **Sheets ("MessageBox").Range("C10").Value = answerTwo** in the ninth line.
    a. This line of code will record the user's response to the designated spreadsheet and cell.
10. Type **End If** in the tenth line. (See **Figure 30.5**)
11. Type **End Sub** in the eleventh line.

```
Sub logicExample()

    Dim answerOne As String, answerTwo As String

    answerOne = MsgBox("Did you enjoy this course?", vbYesNo, "Course Eval")

    If answerOne = vbYes Then

        MsgBox "Thank you!", vbOKOnly

    End If

    If answerOne = vbNo Then

        answerTwo = InputBox("Please tell us what can be improved", "Improvements")

        Sheets("MessageBox").Range("C10").Value = answerTwo

    End If

End Sub
```

Figure 30.5

Apply If and Else Logic to Toggle a Chart

This final example demonstrates another use for macro code to hide or display a chart in a spreadsheet. We will create a checkbox button and source its true or false value (whether or not the box is checked) to a separate cell. Then, the macro code will reference the indicated cell in an If and Else logic check to hide or display the designated chart.

1. Create a simple chart referencing random values. After creating the chart, identify its title for later reference.
    a. Go to the **Home** tab in the Excel ribbon toolbar.
    b. Press the **Find & Select** button.
    c. Select **Selection Pane** to show a list of objects in the spreadsheet.
    d. Identify the title for the appropriate chart.
        i. The eye icons on the right can be used to hide and show the object.
        Note: It may be useful to record a macro of this action to copy the code for the macro in step 3.
2. Create a checkbox button to toggle between checked and unchecked, and source its value to a separate cell.
    a. Go to the **Developer** tab in the Excel ribbon toolbar.
    b. Press the **Insert** button and select the checkbox form control.
    c. Draw the checkbox button across the desired cells and rename it.
    d. Right-click the checkbox button and select **Format Control…**.
    e. Select or type an empty cell in the **Cell link** field.
    f. Press **OK**.
    g. Select the designated cell and rename it for quick reference later.
3. Write a macro code in Visual Basic to hide or display the chart based on the button's value.
    a. Type **Sub showHideChart()** in the first line.
        i. You should be aware the range name in the parenthesis argument will vary on the name you use.
    b. Type **If Range("show_chart").Value = True Then** in the second line.
    c. Type **ActiveSheet.Shapes.Range(Array("Chart 2")).Visible = msoTrue** in the third line.
        i. You should be aware the chart array name will vary on the name designated in Excel. See step 1 to identify the chart's title.
    d. Type **Else:** in the fourth line.
    e. Type **ActiveSheet.Shapes.Range(Array("Chart 2")).Visible = msoFalse** in the fifth line.
    f. Type **End If** in the sixth line. (See **Figure 30.6**)
    g. Type **End Sub** in the seventh line.

```
Sub showHideChart()

    If Range("show_chart").Value = True Then

        ActiveSheet.Shapes.Range(Array("Chart 2")).Visible = msoTrue

    Else:

        ActiveSheet.Shapes.Range(Array("Chart 2")).Visible = msoFalse

    End If


End Sub
```

Figure 30.6

1. Add the macro to the checkbox button.
    a. Right-click the checkbox button.
    b. Select **Assign Macro…**.
    c. Select the recently-made macro.
    d. Press **OK**.

The completed checkbox button will now hide and display the chart. The examples in this chapter are small demonstrations of macro coding to enhance Excel spreadsheets and user experience. There are more extensive uses for Visual Basic code in Excel that you can find on the web to help improve your spreadsheets. The best way to learn macro coding is to practice.

Supplemental Resource