INTRODUCTION TO FORMS IN VISUAL BASIC IN EXCEL

The forms are used to create a user interface for the visual basic programs. This is a useful tool to learn as it will help create intuitive user-friendly interfaces instead of presenting a spreadsheet.

The learning objectives of this tutorial are:

1. To create a userform
2. Learn the controls and their properties
3. Develop a scientific calculator
4. Develop an input form for a spreadsheet based database with a search function.

1. HOW TO GET STARTED?

To create a new userform, click on the “Developer” tab in the Excel ribbon and click on the “View Code” button. This will open the code viewer for the Microsoft Excel. This is usually the place where we enter code for the command buttons in the excel programs. Now click on the “Insert Userform” button as shown in the Figure 1. This will open a blank form for us to work on. There is a Toolbox with the controls that can be inserted into this form.

![Figure 1. Showing the button that inserts the UserForm](image)

2. THE CONTROLS

A brief explanation of the most commonly used controls is given below.

1. Select objects: This is used to select the objects present in the form.
2. Label: This is used to insert labels which can display text on the form.
3. Textbox: This is used to insert a textbox in the form. This can be used either as a means of getting input from the user or displaying output to the user. This is similar to the textboxes used in web pages or cells we use in the spreadsheets.
4. **ComboBox:** This is a control used to insert a pull down menu in the form. The combo box allows the user to select from a given set of values. For example: If you want the user to select a month, the choices are limited to the 12 months in the calendar.

5. **Listbox:** This is similar to a combo box except the list of options is displayed in a box instead of a pull down menu.

6. **Checkbox:** This is a toggle box where you can check or uncheck an option. This can be used where you want to give the user an option to select a function when running the program. Several checkboxes can be inserted into the program and each checkbox can be toggled irrespective of the other checkbox.

7. **Optionbutton:** The option button is a radiobutton which is similar to a checkbox. However, only one optionbutton can be selected.

8. **Command Button:** This is similar to the command button used in the spreadsheet. A set of instructions can be programmed to run every time the button is pressed.

### 3. EXAMPLE-1: SCIENTIFIC CALCULATOR

1. Create a userform as explained above. The program will create a form called “Userform1” as shown in Figure 1. If the name is different, change the name to “Userform1” or use that name everywhere in the code.

2. The properties of the form can be changed just like the properties of a command button in an excel sheet. To display these properties, right click on the form and select properties.

3. To change the title of the form from “Userform1”, you can change the caption in the properties. Type “Calculator” in the caption and notice the title changes to “Calculator” as shown in Figure 3.

4. Add three text boxes and three labels as shown in Figure 3 and give appropriate captions for all the labels. The labels can be changed by changing the caption field in their properties tab. This is similar to the above step. Furthermore, the “Font” property can also be changed if necessary.

5. Add 9 more buttons as shown in Figure 3 and make sure that the caption field for each of button is as shown in the figure for consistency.

6. To open this spreadsheet, we can either press F5 or press the play button while we’re in the editor menu. However, a better way would be to place a button in the spreadsheet and enter the code “Userform1.show”. This tells the program to show the form when we press the button.

7. Double click the “Add” button. This will open a code editor to write the instructions to run when the button is pressed. We write a code that will take the numbers 1 & 2 and displays them as a solution.

8. To read a value in a textbox, we use the statement, variableName=<textboxname>.Value where textboxname is obtained from the “Name” field in the textbox properties tab. Each textbox has a unique name. In the form that was generated, the names for number 1, number 2 and solution are Textbox2, Textbox3 and Textbox1 respectively.
Figure 2. Showing the caption and font properties for a UserForm

Figure 3. Form layout for Calculator
9. Write the code segment as below to perform the addition for number 1 and number 2.

```vbnet
Dim num1 As Single, num2 As Single, solution As Single
num1 = TextBox2.Value
num2 = TextBox3.Value
solution = num1 + num2
TextBox1.Value = solution
```

We are declaring 3 variables to store the two numbers to be added and their solution. The number 1 and number 2 in Textbox2 and Textbox3 are assigned to `num1` and `num2`. These two numbers are added and the result is stored in the variable named `solution`. The result is displayed in Textbox1. Here, the assignment to variables is same as using the cells except we’re using textboxes to read and output the numbers.

10. Double click the rest of the buttons and add code for the rest of the buttons. The buttons in the second row use only number 1. The function for sinh(x) is given by \((\exp(x)-\exp(-x))/2\). Also make sure you have a check for \(\log(x)\) and \(\sqrt{x}\) functions where an error message is displayed when a negative number is entered as shown below.

```vbnet
If (num1 < 0) Then
    MsgBox("Please enter a positive real number")
    TextBox1.Value = NA
Else
    solution = Sqr(num1)
    TextBox1.Value = solution
End If
```

11. Finally the “Close” button at the bottom is used to close the form. Double click the button and enter the line “Unload Me”. This closes the current form.

### 4.1 EXAMPLE-2: A SIMPLE DATABASE WITH SEARCH FUNCTION

The goal of this example is to develop a form to enter the basic details of people. Let us say, an employer wants to maintain a database of all his employees with some basic information about them. We will create a form with labels and textboxes to input these details as shown in Figure 4. After the details have been entered, pressing the next button will transfer these entries onto a spreadsheet as shown in Figure 5. A search form will be subsequently generated to search for a specific name in the database. We will use two forms for this problem. One form is used to enter the details and the second form is used for a search function.

Create a form with textboxes as shown in Figure 4 and label the first row of “Sheet 1” in the spreadsheet as shown in Figure 5. There are four buttons in the form. The function of the “Reset” button is to clear the form quickly. “Next” button enters the values into the database and clears the form for the next entry. “Search” button will open the search form and “Close” button will exit the form.

**Note:** We will use a Combobox for the state and two option buttons for the gender.
If we open the form now, we will notice that the combo box is empty. We need to add values to this combo box so the user could select values from it. There are several ways to populate a combo box. We will use the Userform_initialize() subroutine to populate the combo box. There are several “events” that can be programmed. The form can perform certain tasks when it’s clicked on or when it’s open. The “UserForm_initialize()” subroutine will perform these tasks as soon as the form loads. To enter the contents of this subroutine, right click on the form and select “view code”. The code below tells the form to add the following states listed to be added to the combo box as soon as it is initialized.

```vbnet
Private Sub UserForm_Initialize()
'Populate combo box.
    Me.ComboBox1.AddItem "Alabama"
    Me.ComboBox1.AddItem "California"
    Me.ComboBox1.AddItem "Florida"
    Me.ComboBox1.AddItem "North Carolina"
    Me.ComboBox1.AddItem "Texas"
    Me.ComboBox1.AddItem "Others"
End Sub
```

If we open the form now, we will notice that the combo box is empty. We need to add values to this combo box so the user could select values from it. There are several ways to populate a combo box. We will use the Userform_initialize() subroutine to populate the combo box. There are several “events” that can be programmed. The form can perform certain tasks when it’s clicked on or when it’s open. The “UserForm_initialize()” subroutine will perform these tasks as soon as the form loads. To enter the contents of this subroutine, right click on the form and select “view code”. The code below tells the form to add the following states listed to be added to the combo box as soon as it is initialized.

```vbnet
Private Sub UserForm_Initialize()
'Populate combo box.
    Me.ComboBox1.AddItem "Alabama"
    Me.ComboBox1.AddItem "California"
    Me.ComboBox1.AddItem "Florida"
    Me.ComboBox1.AddItem "North Carolina"
    Me.ComboBox1.AddItem "Texas"
    Me.ComboBox1.AddItem "Others"
End Sub
```
Go back to the UserForm2 by double clicking it in the “Project” window on the left side. If the project explorer is not visible, click on “View” in the file menu and select “Project explorer”. We will start entering code for all the buttons now. Double click on the “Reset” button and enter the code below:

```vbnet
TextBox1.Value ="
TextBox2.Value ="
TextBox3.Value ="
TextBox4.Value ="
TextBox5.Value ="
ComboBox1.Value ="
OptionButton1.Value = False
OptionButton2.Value = False
```

Make sure that the textbox names are same or use the names of the textboxes that are in your form. We are setting blank values to all the textboxes and the option buttons are unchecked.

Double click the “Next” button and enter the code below:

```vbnet
Dim i As Integer, gender As String
i = TextBox5.Value
Cells(i + 1, 1) = i
Cells(i + 1, 2) = TextBox1.Value
Cells(i + 1, 3) = TextBox2.Value
Cells(i + 1, 4) = TextBox3.Value
Cells(i + 1, 5) = TextBox4.Value
If OptionButton1.Value = True Then
    gender = "Male"
ElseIf OptionButton2.Value = True Then
    gender = "Female"
Else
    gender = "NA"
End If
Cells(i + 1, 6) = gender
Cells(i + 1, 7) = ComboBox1.Value
TextBox1.Value ="
TextBox2.Value ="
TextBox3.Value ="
TextBox4.Value ="
TextBox5.Value ="
ComboBox1.Value ="
OptionButton1.Value = False
OptionButton2.Value = False
```
We need two variables in this part of the program. An integer value is used to place the database entry into the right place in the spreadsheet. This will be done by using the “S. No”. A string variable is necessary to get the gender based on which optionbutton is selected. The optionbutton is assigned a Boolean value based on whether it is checked or not. If the optionbutton is checked, it will return a “True” value and a “False” value if it is unchecked. An “if condition statement” is used to check if the optionbutton is checked or not. When the first optionbutton is selected, we conclude that the employee is male and so on. After the entries are placed on the spreadsheet, we reset the form for a new entry.

Double click the “Search” button and paste the code below.

```
Unload Me
UserForm3.Show
```

This essentially closes the current form and opens the form with the search function capabilities. And finally double click the “Close” button and write “Unload me”. This will close the current form and discards all the entries entered into the form.

### 4.2 SEARCH FUNCTION FORM

Create a form as shown in Figure 6. Note that the big white box is a list box and not a text box. We will use the list box to populate it with the search results.

![Figure 6. Form layout for the search function](image)

Double click the “Back” button and enter the code below. The purpose of the back button is to close this button and return to the original form.
Double click the “Search” button and enter the code below.

*Dim searchname As String, i As Integer, sno(10) As String, counter As String*

*searchname = TextBox1.Text*

*counter = 0*

*For i = 1 To 10*

*If (searchname = Cells(i + 1, 2)) Then*

*counter = counter + 1*

*sno(counter) = CStr(Cells(i + 1, 1)) + "     " + Cells(i + 1, 2) *

*End If*

*Next i*

*If (counter = 0) Then*

*MsgBox ("no results found")*

*ElseIf (counter = 1) Then*

*MsgBox ("1 result found")*

*Else*

*MsgBox (counter + " results found")*

*End If*

*ListBox1.Clear*

*For i = 1 To counter*

*ListBox1.AddItem sno(i)*

*Next i*

The code above assumes that there are only 10 entries in the database. Modify it as necessary for the length of your database. It reads the search string from the textbox1 and stores in a variable named searchname. Subsequently this search string is used to check against the names of all the database entries. If there is a match, a counter keeps track of all the number of results. We store the “S. No” and the name of the matched result in a string. A Cstr() function is used to convert the integer value of “S.No” into a string. We use a message box to display the number of results kept track by the counter. To give meaningful message box prompts, when there are 0 results found, we show a message a saying “no results found” instead of a generic “0 results found”. After the user closes the msgbox prompt, we will populate the listbox below with the stored search results. Each time the listbox is cleared before the new results are displayed by using the ListBox1.Clear statement.

**Further resources:**

http://www.mrexcel.com/articles.shtml#VBA
http://www.excelgames.org/
5. WORKING WITH MULTIPLE SHEETS IN SAME SPREADSHEET

There could be several scenarios where we want to segregate the data into multiple spreadsheets for better organization. In this example, we will read two matrices A and B from “Sheet 1” and “Sheet 2” respectively and output the result in “Sheet 3”. The first two rows of the sheets are allocated for the dimensions of matrix as shown in Figure 7.

To read the information from a different sheet in the spreadsheet, we will use the statement Worksheets("Sheetname").Cells(row #, col #) where sheetname is the name of the sheet where the information is present. Cells(row #, col #) gives the information where the numbers are present. The matrix multiplication code is rewritten for different sheets below. Create a button and set up the spreadsheet as explained above and copy paste the code. It should print the output in “Sheet3”.

![Figure 7. Matrix layout in the spreadsheet](image)

Option Explicit
Option Base 1

Private Sub cmdMultiply_Click()
Dim A() As Single, B() As Single, C() As Single
Dim A_nrows As Integer, A_ncols As Integer, B_nrows As Integer, B_ncols As Integer, i As Integer, j As Integer

A_nrows = Worksheets("Sheet1").Cells(1, 2)
A_ncols = Worksheets("Sheet1").Cells(2, 2)

B_nrows = Worksheets("Sheet2").Cells(1, 2)
B_ncols = Worksheets("Sheet2").Cells(2, 2)
ReDim A(A_nrows, A_ncols) As Single, B(B_nrows, B_ncols) As Single, C(A_nrows, B_ncols) As Single

For i = 1 To A_nrows
    For j = 1 To A_ncols
        A(i, j) = Worksheets("Sheet1").Cells(3 + i, j)
    Next j
Next i

For i = 1 To B_nrows

For j = 1 To B_ncols
    B(i, j) = Worksheets("Sheet2").Cells(3 + i, j)
Next j
Next i

If (A_ncols <> B_nrows) Then
    MsgBox ("This is not possible. Please check the dimensions.")
    Exit Sub
End If

Call matmultiply(A_nrows, A_ncols, B_ncols, A(), B(), C())

Worksheets("Sheet3").Cells(1, 2) = A_nrows
Worksheets("Sheet3").Cells(2, 2) = B_ncols

For i = 1 To A_nrows
    For j = 1 To B_ncols
        Worksheets("Sheet3").Cells(3 + i, j) = C(i, j)
    Next j
Next i

End Sub

Sub matmultiply(nn1 As Integer, nn2 As Integer, nn3 As Integer, aa() As Single, bb() As Single, cc() As Single)
' This routine multiplies two matrices a and b and computes c
' aa (nn1 x nn2) and bb (nn2 x nn3) are the two input matrices and cc (nn1 x nn3) is the result
Dim i As Integer, j As Integer, k As Integer, sum As Single
For i = 1 To nn1
    For j = 1 To nn3
        sum = 0#
        For k = 1 To nn2
            sum = sum + aa(i, k) * bb(k, j)
        Next k
        cc(i, j) = sum
    Next j
Next i
End Sub