VBA3-Creating User-Defined Functions

Excel has over 150 built-in functions that you can use to create quite complex formulas. If you need to reuse a formula more than several times, you can create your own user-defined function and use it like any of the built-in functions. You'll learn the ins and outs of creating a user function: what you can and cannot do in a function, how to use variable types to control the output of a function, how to use IF-THEN-ELSE and SELECT CASE statements to control processing in your function and how to distribute a user function to others in your organization.

Overview

1. You can create a function that works like Excel's built-in functions.
2. User functions let you add specific operations that Excel does not provide.
3. Somewhere in your code you need to assign the Function name to the results of a calculation. You can do this more than once in the function, based on conditions in your code.
4. There are certain limits to what a User-Defined Function can do.
5. You distribute them by:
    a. Including the function in a workbook.
    b. Giving the recipient a copy of your Personal Macro Workbook.
    c. Exporting the VBA module and having the recipient import it.
6. If you do not include the code, the workbook will show a #NAME? error.
7. User function arguments in Excel can be values or formulas or cell references just as with standard Excel functions.

Function Rules

1. Your function can only affect the value of the current cell.
2. You can refer to other cells, but cannot change them.
3. Your function cannot move to another cell. The cell cursor must remain in the original cell.
4. Functions can have arguments of any type. You can have as many arguments as you need.
5. Functions return a value of a specific type. This value can be inserted into the current cell or used in a formula in the current cell.

Creating a User-Defined Function

1. Define a Function using the statement:
   `Function(ArgumentName as DataType) as DataType`
2. VBA adds the `End Function` statement.
3. You can have as many arguments as you need and each can be a different datatype.

The Celsius to Fahrenheit Function

1. Excel does not provide a Celsius to Fahrenheit or Fahrenheit to Celsius conversion function.
2. The formulas for a these conversions are:

```
Celsius = (Fahrenheit  - 32) * 5/9
Fahrenheit = (Celsius * 9/5) + 32
```

3. The Celsius to Fahrenheit function is:

```
Function CtoF(Fahrenheit As Double) As Double
'Enter a Fahrenheit temperature
'It can be a fraction: 98.6 or 33.5

      CtoF = (Fahrenheit - 32) * 5 / 9
      'VBA does the calculation and assigns the value to CtoF
End Function
```

4. Note that the ' defines a Comment: all text after the ' will be ignored while the code is running.
5. This function will now appear in your list of functions under User-Defined.
6. When you add it to a cell, you specify a temperature or a cell reference to a cell containing a temperature.
7. The converted temperature will appear in the cell containing the function.

EXERCISE: Create CtoF

1. Create the CtoF function.
2. Test using 212°, 100°, 50°, 32°, 0°, -40°
3. Create the FtoC function.
4. Test using the same values

IF-THEN-ELSE Processing

1. You have probably used the IF function in Excel.
2. The definition of IF-THEN-ELSE is:

a.	IF [condition]	'A formula that results in TRUE or FALSE
b.	THEN [code]	'One formula to process or value to insert if the test is true
c.	ELSE [code]	'One formula to process or value to insert if the test is false

3. Excel encloses the IF statement arguments in parentheses:
   `=IF([Test],[True],[False])`.
4. In VBA, the syntax needs to include the keywords `Then, Else, End If`.
5. The THEN appears on the same line as the IF: `If Sum <> 0 Then`
6. You can write a simple IF statement on one line if it contains only one statement for the True and zero or one statement for the false. (Note that the _ is a continuation character. Use it at the end of a line to continue the statement to the next line without causing an error)

```
If ActiveCell.Offset(0,-1).Value = 0 THEN ActiveCell.Value = "No Data" _
      ELSE ActiveCell.Value = "DataFound"
```

7. Useful to selectively process your data.
8. Useful to test for errors and work around them.

```
If ActiveCell.Value = "" Then ActiveCell.Value = 0
```

9. Compound tests require restating the comparison:

```
If ActiveCell.Offset(0,-1).Value = "" or _
     ActiveCell.Offset(0,-1).Value = 0 Then
          ActiveCell.Value = 0
Else
     ActiveCell.Value = ActiveCell.Offset(0,-1).Value / 2
End If
```

10. Notice the indentation. VBA ignores indentation and blank lines so add as many as you need to make your code more readable.

Fiscal Year Function

1. Another common operation is converting a date to the proper Fiscal Year.
2. For example, if your Fiscal Year starts on July 1 and the current Fiscal Year is 2017, then:
   2/16/17 = Fiscal Year: 2017
   7/16/17 = Fiscal Year: 2018
3. To convert a date in the calendar year to the Fiscal Year:
   a. Find the current year.
   b. Find the current month.
   c. If the current month is 1 to 6, then the `Fiscal Year=Current Year`
   d. If the current month is 7-12, then the `Fiscal Year = Current Year + 1`
   e. Use the `MONTH(date)` and `YEAR(date)` functions to extract the necessary values.
   f. These are available in VBA as well as in Excel:

```
CurrentYear = Year(TestDate)
CurrentMonth = Month(TestDate)
```

EXERCISE: Create a Fiscal Year Function

1. Create the Function FiscalYear(TestDate as Date) to return the Fiscal Year as an Integer.
2. Test it with different months.

Nesting IF THEN ELSE

1. IF THEN ELSE statements can be nested if you have more than two tests to perform:

```
Function ShippingCost(Weight as Double) as Currency
     If Weight <= 5 Then
          ShippingCost = 2.95
     Else If Weight <=10 Then
          ShippingCost = 4.95
     Else If Weight <=50 Then
          ShippingCost = 8.95
     Else
          Shipping Cost = 0
     End If
End Function
```

2. VBA will process this code and stop at the first True statement. Then it will skip to the statement after the End If.

3. Notice that we are using a Currency data type as the return value. This will make sure the returned value is a legitimate currency amount and contains no other characters except numbers, a period, a dollar sign, and a minus sign (if necessary).
4. Notice also that we have hard-coded the shipping costs into the function. To make it more universal, we can store the costs in spreadsheet cells and read them as necessary. If you name the cells, it makes finding the values much easier:

   ```
   curShipping5=Range("Shipping5").Value
   curShipping10=Range("Shipping10").Value


   If Weight <= 5 Then
         ShippingCost = curShipping5
   ```
   If you need to change a shipping weight, you can change the cell containing the value without changing your VBA code.

SELECT CASE Statement

1. Nested Ifs are useful, but can be unwieldy when there are multiple conditions, especially if you are expecting to add conditions over time.
2. The SELECT CASE statement lets you add multiple conditions easily. The format of the statement is:

   ```
   Select Case [VariableName]
         Case [value1]:   [VBA Code]
         Case [value2]:   [VBA Code]
         Case [value3]:   [VBA Code]
         Case [value…n]:  [VBA Code]
         Else:  [VBA Code]
   End Select
   ```
3. You can have as many Case statements as necessary.
4. Each Case value can be a string ("Tennessee") or a number (15 or 3.14159).
5. You can string together multiple cases using commas, operators or the keyword To

   ```
   Case 1,2,3: [VBA Code]      'If VariableName = 1 or 2 or 3,
                               ' execute code
   Case 1 To 9: [VBA Code]     'If VariableName is between 1 and 9,
                               '(including 1 and 9), execute code
   Case Is < 10: [VBA Code]    'If VariableName is less than 10,
                               'execute code
   ```
6. You can add as many lines of code as you need between Case statements:

   ```
   Case 1,2,3:
         ActiveCell.Value = "3Q"
         ActiveCell.Offset(0,1).Value = Year(TestDate)+1
   Case 4,5,6:
   ```
7. Use indentation to format your code for readability.
8. The `Else` keyword covers any value outside the range you want. It also handles any odd user input.

EXERCISE: Adding the Quarter to the Fiscal Year Function

1. You can modify your FiscalYear function to add the Fiscal Quarter to the Fiscal Year.
2. Assuming the Fiscal Year begins in July and the current year is 2017, the following conditions apply:

Month	Month Number	Current Year	Quarter	Return Value
1/1/2017	1	2017	3Q	3Q 2017
2/1/2017	2	2017	3Q	3Q 2017
3/1/2017	3	2017	3Q	3Q 2017
4/1/2017	4	2017	4Q	4Q 2017
5/1/2017	5	2017	4Q	4Q 2017
6/1/2017	6	2017	4Q	4Q 2017
7/1/2017	7	2018	1Q	1Q 2018
8/1/2017	8	2018	1Q	1Q 2018
9/1/2017	9	2018	1Q	1Q 2018
10/1/2017	10	2018	2Q	2Q 2018
11/1/2017	11	2018	2Q	2Q 2018
12/1/2017	12	2018	2Q	2Q 2018

3. Modify the Fiscal Year function to return the quarter and the year: 3Q 2017, etc.

Distributing a User Function

1. If you add a User Defined function to a module in the current Workbook, it will go with the workbook when you send the workbook to someone else.
2. Make sure to save the workbook as an .XLSM file.
3. If you want to use the function whenever you have Excel open, add the function to a module in the Personal Macro Workbook.
4. You can give someone your Personal Macro Workbook and if they put it in the correct folder, it will be available to them when they open Excel.
   a. The correct location may not be editable due to your company's policies.
   b. This will replace their Personal Macro Workbook. Any macros they have created will be lost. All your macros will be available to them.
5. You can export a module into a .BAS file. The recipient can then import the .BAS file into their Personal Macro Workbook or to a specific workbook.
   a. You cannot send a .BAS file using Outlook. You must rename the extension (to say .BXX) then your recipient must rename it back to .BAS after they save it and before they import it.
   b. You can export your modules into .BAS files and store them in order to have a backup of your code. Recommended.
6. To export a module:
   a. Open the Visual Basic editor.
   b. In the Project Explorer, right-click on the module you want to export.
   c. Select Export File.
   d. Find the folder and/or drive into which you want to place the file.

    e. Click Save.
    f. VBA will save the file using the name of the module plus .BAS.

7. To import a module:
    a. Open the Visual Basic editor.
    b. In the Project Explorer, right-click on the VBAProject (workbook) into which you want to import the module. (For example, right-click on VBAPersonal to place it into your Personal Macro Workbook.)
    c. Select Import File.
    d. Find the folder and/or drive where the file is saved.
    e. Click Open.
    f. VBA will add the module to the VBA project, using the name of the module file.
    g. Make sure to click Save in the VBA Editor or save the changes to Personal when you exit Excel.