**Excel Assignment – 20**

1. Write a VBA code to select the cells from A5 to C10. Give it a name “Data Analytics” and fill the cells with the following cells “This is Excel VBA”

**Ans:**

Sub MACR21()

Range("A5:C10").Select

ActiveWorkbook.Names.Add "Data\_Analytics", "=Sheet1!$A$5:$c$10"

Range("A5:C10").Value = "This is Excel VBA"

End Sub

2. Use the above data and write a VBA code using the following statements to display in the next column if the number is odd or even

a. IF ELSE statement : = **IF(ISEVEN(A2), "Even", "Odd")**

|  |  |
| --- | --- |
| Number | Odd or Even |
| 56 | Even |
| 89 | Odd |
| 26 | Even |
| 36 | Even |
| 75 | Odd |
| 48 | Even |
| 92 | Even |
| 58 | Even |
| 13 | Odd |
| 25 | Odd |

b. Select Case statement :

Sub OddEven()

Dim num As Integer

num = CInt(InputBox("Please enter any Integer"))

Select Case num Mod 2

Case 0

MsgBox Str(num) + " is an Even Integer"

Case 1

MsgBox Str(num) + " is an Odd Integer"

End Select

End Sub

c. For Next Statement

Sub Solution()

Dim LastRow As Long

Dim CellVal As Long

Dim i As Integer

LastRow = Cells(Rows.Count, 1).End(xlUp).Row

For i = 2 To LastRow

CellVal = Cells(i, 1).Value

If CellVal Mod 2 = 1 Then

Cells(i, 2).Value = "Odd"

Else

Cells(i, 2).Value = "Even"

End If

Next i

End Sub

1. What are the types of errors that you usually see in VBA?

### Ans: Types of VBA Errors

i)  **Syntax errors**– A specific line of code is not written correctly

ii) **Compile errors**– Issues that happen when putting together lines of code, though the individual lines of code seem to make sense

iii) **Runtime errors**– When the code is usually correct in principle, but an action taken by the user or the data being used leads to unexpected errors.

1. How do you handle Runtime errors in VBA?

Ans: Error Trapping: The first step in dealing with runtime errors is to set a "trap" to catch the error. This is to be done by including an On Error statement in your macro. When a runtime error occurs, the On Error statement transfers control to an error-handling routine.

To trap errors, we must set error trap above the point in the procedure where errors are likely to occur. A good practice is to place the error trap near the top of the procedure. To avoid having the error-handling routine execute even when no error occurs, should include an Exit Sub or Exit Function statement just before the error-handling routine's label.

1. Write some good practices to be followed by VBA users for handling errors?

**Ans:** Following are some best practices we can use when it comes to error handling in Excel VBA.

1. Use ‘On Error Go [Label]’ at the beginning of the code. This will make sure any error that can happen from there is handled.
2. Use ‘On Error Resume Next’ ONLY when you’re sure about the errors that can occur. Use it with expected error only. In case you use it with unexpected errors, it will simply ignore it and move forward. You can use ‘On Error Resume Next’ with ‘Err.Raise’ if you want to ignore a certain type of error and catch the rest.
3. When using error handlers, make sure you’re using Exit Sub before the handlers. This will ensure that the error handler code is executed only when there is an error (else it will always be executed).
4. Use multiple error handlers to trap different kinds of errors. Having multiple error handler ensures that an error is properly addressed. For example, you would want to handle a ‘type mismatch’ error differently than a ‘Division by 0’ run-time error.

1. What is UDF? Why are UDF’s used? Create a UDF to multiply 2 numbers in VBA?

**Ans:** UDF: A User Defined Function is a procedure (a group of commands) written in VBA that (usually) accepts inputs and returns a result. A UDF cannot modify the formatting of a cell or workbook or move values around on a worksheet. With VBA, we can create a custom Function (also called a User Defined Function) that can be used in the worksheets just like regular functions.These are helpful when the existing Excel functions are not enough. In such cases, we can create our own custom User Defined Function (UDF) to cater to our specific needs.

There are a few different reasons why we might want to consider creating a custom User Defined Function in our worksheet.

The first advantage of functions is that they can clean up our spreadsheets. Instead of cell after cell of sequential calculations, we can combine many successive calculations into a single function. This can significantly clean up our spreadsheets.

User Defined Functions can increase our productivity by allowing us to store and re-use calculations that we use over and over again. Rather than having to recall an equation from memory or looking it up in a reference, we can build it into a UDF and call the UDF instead of retyping the calculation. This has the added benefit of minimizing typing errors.

Finally, User Defined Functions provide us with all of the flexibility of the Visual Basic for Applications language. With UDF’s we can take advantage of loops, expanded logic, and other functionality. This is where we can really increase the capability of you engineering spreadsheets.

**UDF to multiply 2 numbers in VBA**

Sub Multiply\_Range\_with\_Number()

'Declaring variables

Dim numbers As Range, multiplier As Range

Dim a As Range

Dim c As Integer

'Setting variables

Set numbers = Range("A2:A" & Range("A65000").End(xlUp).Row)

Set multiplier = Range("b2")

c = 2

'For each loop

For Each a In numbers

Cells(c, 3) = a \* multiplier

c = c + 1

Next a

End Sub