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

**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”   
   **Answer –**

Sub SetNamedRange()

' Select the range from A5 to C10

Range("A5:C10").Select

' Name the selected range as "DataAnalytics"

ActiveWorkbook.Names.Add Name:="DataAnalytics", RefersTo:=Selection

' Fill the selected cells with the text

Selection.Value = "This is Excel VBA"

End Sub

1. 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
   1. IF ELSE statement
   2. Select Case statement
   3. For Next Statement

**Answer –** “”””””””””””””””””””””””””””””””””””””””””””””””””””””’

Sub DetermineOddEven()

Dim dataRange As Range

Dim cell As Range

' Set the range containing the numbers (assuming numbers are in column A starting from A2)

Set dataRange = Range("A2:A11")

' Using IF-ELSE statement

For Each cell In dataRange

If cell.Value Mod 2 = 0 Then

cell.Offset(0, 1).Value = "Even"

Else

cell.Offset(0, 1).Value = "Odd"

End If

Next cell

' Clear the next column for the Select Case and For-Next examples

dataRange.Offset(0, 1).ClearContents

' Using Select Case statement

For Each cell In dataRange

Select Case cell.Value Mod 2

Case 0

cell.Offset(0, 1).Value = "Even"

Case 1

cell.Offset(0, 1).Value = "Odd"

End Select

Next cell

' Clear the next column for the For-Next example

dataRange.Offset(0, 1).ClearContents

' Using For-Next statement

For i = 1 To dataRange.Rows.Count

If dataRange.Cells(i, 1).Value Mod 2 = 0 Then

dataRange.Cells(i, 1).Offset(0, 1).Value = "Even"

Else

dataRange.Cells(i, 1).Offset(0, 1).Value = "Odd"

End If

Next i

End Sub

“””””””””””””””””””””””””””””””””””””””””””””””””””””””

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

**Answer -** Common types of errors in VBA:

1. Syntax Errors - Mistakes in code syntax.

2. Runtime Errors - Occur during code execution.

3. Logic Errors - Code runs but produces unexpected output.

4. Object Variable Not Set - Using uninitialized object variables.

5.Type Mismatch - Assigning incompatible data types.

6. Overflow- Result exceeds allowable data type range.

7. File Not Found - Attempting to access a nonexistent file.

8. Out of Memory - Insufficient system memory.

9. Automation Errors - Issues with external applications or COM objects.

Handling errors is crucial for effective debugging and application robustness.

1. How do you handle Runtime errors in VBA?

**Answer -** To handle runtime errors in VBA:

1. On Error Resume Next

- Allows code to continue to the next line if an error occurs.

- Useful for selectively ignoring errors for specific lines.

2. On Error GoTo [label]

- Directs VBA to jump to a specified label when an error occurs.

- Useful for custom error handling and cleanup.

3. On Error GoTo 0

- Disables error handling, resetting it to the default behavior.

4. Err Object

- Provides information about the most recent runtime error.

- Properties like `Err.Number` and `Err.Description` give error details.

5. Resume Statement

- Controls how the code proceeds after an error.

- Options include `Resume`, `Resume Next`, and `Resume [label]`.

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

**Answer -** Implementing effective error handling in VBA involves strategic use of statements such as `On Error Resume Next` for selective error skipping and `On Error GoTo [Label]` for custom handling. The `Err` object and its properties aid in retrieving error details, and `Err.Clear` resets errors for subsequent code. User-friendly messages and error logging enhance the user experience and facilitate debugging. Selective use of `Resume` statements allows controlled code execution post-error. Validate inputs, handle expected errors, and document error-handling logic. Include proper exit strategies, reset error handling with `On Error GoTo 0`, and regularly test error-handling procedures for a robust VBA application.

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

**Answer -** A UDF (User-Defined Function) in VBA is a custom function created by users for specific tasks, providing custom functionality, reusability, and improved readability in Excel. UDFs automate tasks and offer flexibility not found in built-in functions.

Function MultiplyNumbers(ByVal num1 As Double, ByVal num2 As Double) As Double

MultiplyNumbers = num1 \* num2

End Function