Assignment-20

iNeuron

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"

Number Odd or even

56 89 26 36 75 48 92 58 13 25

Ans:- Here is a VBA code that can select the cells from A5 to C10, name it "Data Analytics" and fill the cells with the data you provided:

```
Sub DataAnalytics()

    Range("A5:C10").Select

    Selection.Name = "Data Analytics"

     Range("A5:C10").Value = Array("This is Excel VBA", "Number", "Odd or even",
56, 89, 26, 36, 75, 48, 92, 58, 13, 25)

End Sub
```

You can run this code by pressing the "Alt + F11" keys to open the VBA editor and then pasting the code in a new module. To run the code, you can either press the "F5" key or go to the "Run" menu and select "Run Sub/UserForm".

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

b. Select Case statement

c. For Next Statement

Ans:- Here is a VBA code that uses the data in the "Data Analytics" range and adds a column to display if the number is odd or even, using the following statements:

a. IF ELSE statement:

```
Sub OddEven_IFELSE()

    Dim rng As Range

    Set rng = Range("Data Analytics")

    For i = 5 To 10

        If rng(i, 3) Mod 2 = 0 Then

            rng(i, 4) = "Even"

        Else

            rng(i, 4) = "Odd"

        End If

    Next i
```

```
End Sub
```

b. Select Case statement:

```
Sub OddEven_SelectCase()
    Dim rng As Range
    Set rng = Range("Data Analytics")
    For i = 5 To 10
        Select Case rng(i, 3) Mod 2
            Case 0
                rng(i, 4) = "Even"
            Case Else
                rng(i, 4) = "Odd"
        End Select
    Next i
End Sub
```

c. For Next statement:

```
Sub OddEven_ForNext()
    Dim rng As Range
    Set rng = Range("Data Analytics")
    For i = 5 To 10
```

```
        If IsOdd(rng(i, 3)) Then

            rng(i, 4) = "Odd"

        Else

            rng(i, 4) = "Even"

        End If

    Next i

End Sub

Function IsOdd(num As Long) As Boolean

    IsOdd = (num Mod 2 <> 0)

End Function
```

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

Ans:- There are several types of errors that you can encounter while working with VBA:

1. Compile errors: These occur when the code is being compiled, before it is executed. They occur when the code contains syntax errors, such as incorrect keywords, incorrect operator usage, or missing punctuation.
2. Runtime errors: These occur when the code is being executed. Examples of runtime errors include division by zero, type mismatch, and object required.
3. Logical errors: These are errors in the code logic and are not necessarily detectable by the compiler. These errors can cause the code to produce incorrect results or not produce the intended result.
4. Object not set errors: These occur when a reference to an object is not set correctly, either because the object is not created, or because it is not properly assigned to a variable.
5. Automation errors: These occur when you try to automate a task in an Office application using VBA, but the task fails because of a problem in the application or the operating system.

6. Type mismatch errors: These occur when you try to assign a value to a variable of a different data type.
7. Out of memory errors: These occur when there is not enough memory available to execute the code.

It is important to be aware of these errors and to test your code thoroughly to avoid them. It is also a good idea to include error handling code in your VBA projects, so that you can handle any errors that occur in a graceful manner.

4. How do you handle Runtime errors in VBA?

Ans:- There are several ways to handle runtime errors in VBA:

1. The On Error statement: This statement allows you to specify what should happen if a runtime error occurs. You can use this statement to direct the code to jump to a specific label or line number if an error occurs, and you can also use it to provide custom error messages.

For example:

```
On Error GoTo ErrorHandler

'Your code here

Exit Sub

ErrorHandler:

    'Error handling code here

    MsgBox "An error has occurred."

End Sub
```

2. The Resume statement: This statement is used to continue execution after an error has been handled. It can be used to resume execution at the same line where the error occurred, or at a specific line or label.

For example:

```
On Error GoTo ErrorHandler

'Your code here

Exit Sub

ErrorHandler:

    'Error handling code here

    If Err.Number = 1001 Then

        Resume Next

    End If

End Sub
```

3. The Err object: This is a built-in object in VBA that contains information about the error that has occurred. You can use the properties of the Err object to determine the error number and description, and to handle specific errors in a custom manner.

For example:

```
n Error GoTo ErrorHandler

'Your code here

Exit Sub

ErrorHandler:
    'Error handling code here

    If Err.Number = 1001 Then

        MsgBox "Error 1001: " & Err.Description

    End If

End Sub
```

It is a good practice to handle runtime errors in your VBA code, as it makes your code more robust and less likely to produce unintended results.

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

<u>Ans</u>:- Here are some good practices for handling errors in VBA:

1. Use the On Error statement: This statement allows you to specify what should happen if a runtime error occurs. You can use this statement to direct the code to jump to a specific label or line number if an error occurs, and you can also use it to provide custom error messages.
2. Test your code thoroughly: It is important to thoroughly test your code, especially the parts of the code that are prone to errors. This will help you identify and fix any errors before the code is used in a production environment.
3. Use descriptive error messages: When handling errors, it is a good practice to provide descriptive error messages that explain what went wrong and how the user can correct the error. This makes it easier for users to understand what has happened and how to fix it.

4. Use the Err object: The Err object is a built-in object in VBA that contains information about the error that has occurred. You can use the properties of the Err object to determine the error number and description, and to handle specific errors in a custom manner.
5. Be specific in error handling: When handling errors, it is a good practice to be specific in your error handling code. This means that you should handle specific errors, rather than handling all errors in the same way.
6. Avoid hiding errors: It is a bad practice to hide errors by simply ignoring them or by using the On Error Resume Next statement. Hiding errors can make it difficult to identify the source of the problem and to correct it.
7. Document error handling: It is a good practice to document the error handling code in your VBA projects, including the type of error that is being handled and what actions are taken when the error occurs. This makes it easier for others to understand how errors are handled in your code.

By following these best practices, you can make your VBA code more robust and less prone to errors, and you can handle errors in a graceful and informative manner.

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

Ans:- A User-Defined Function (UDF) is a function that you can create in VBA to perform custom calculations. UDFs are used to extend the built-in functions in Excel, and to make it easier to perform complex calculations that cannot be done with the standard functions.

Here's an example of how to create a UDF in VBA to multiply two numbers:

1. Open the Visual Basic Editor (VBE) by pressing ALT + F11.
2. In the VBE, go to the "Insert" menu and select "Module".
3. In the new module, enter the following code:

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

    MultiplyNumbers = num1 * num2
```

```
End Function
```

4. Close the VBE and go back to your Excel sheet.
5. In a cell, enter the following formula to use the UDF:

```
=MultiplyNumbers(A1, B1)
```

Where A1 and B1 are the cells that contain the numbers to be multiplied.

UDFs are a powerful and flexible tool in VBA, as they allow you to create custom calculations that can be reused throughout your workbook. They can also be shared with other users, making it easier to collaborate and streamline your work processes.