1. What are the data types used in VBA?

Excel VBA Data Types

Computer cannot differentiate between the numbers (1,2,3..) and strings (a,b,c,..). To make this differentiation, we use Data Types.

VBA data types can be segregated into two types

- **Numeric Data Types**

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

Type	Storage	Range of Values
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places)

- **Non-numeric Data Types**

Data Type	Bytes Used	Range of Values
String (fixed Length)	Length of string	1 to 65,400 characters
String (Variable Length)	Length + 10 bytes	0 to 2 billion characters
Boolean	2 bytes	True or False
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

In VBA, if the data type is not specified, it will automatically declare the variable as a Variant.

Let see an example, on how to declare variables in VBA. In this example, we will declare three types of variables string, joining date and currency

2. What are variables and how do you declare them in VBA? What happens if you don't declare a variable?

When declaring variables, you usually use a **Dim** statement. A declaration statement can be placed within a procedure to create a procedure-level variable. Or it may be placed at the top of a module, in the Declarations section, to create a module-level variable.

The following example creates the variable and specifies the String data type.

VBCopy
Dim strName As String

If this statement appears within a procedure, the variable strName can be used only in that procedure. If the statement appears in the Declarations section of the module, the variable strName is available to all procedures within the module, but not to procedures in other modules in the project.

To make this variable available to all procedures in the project, precede it with the **Public** statement, as in the following example:

VBCopy
Public strName As String

For information about naming your variables, see Visual Basic naming rules.

Variables can be declared as one of the following data types: **Boolean**, **Byte**, **Integer**, **Long**, **Currency**, **Single**, **Double**,

Date, **String** (for variable-length strings), **String** ***** ***length*** (for fixed-length strings), **Object**, or **Variant**. If you don't specify a data type, the **Variant** data type is assigned by default. You can also create a user-defined type by using the **Type** statement.

You can declare several variables in one statement. To specify a data type, you must include the data type for each variable. In the following statement, the variables intX, intY, and intZ are declared as type **Integer**.

VBCopy Dim intX As Integer, intY As Integer, intZ As Integer

In the following statement, intX and intY are declared as type **Variant**, and only intZ is declared as type **Integer**.

VBCopy Dim intX, intY, intZ As Integer

You don't have to supply the variable's data type in the declaration statement. If you omit the data type, the variable will be of type **Variant**.

The shorthand to declare x and y as Integer in the statement above is:

VBCopy
Dim intX%, intY%, intZ as Integer

The shorthand for the types is: % -integer; & -long; @ -currency; # -double; ! -single; $ -string

3. What is a range object in VBA? What is a worksheet object?

Range is a property in VBA that helps specify a particular cell, a range of cells, a row, a column, or a three-dimensional range. In the context of the Excel worksheet, the VBA range object includes a single cell or multiple cells spread across various rows and columns.

For example, the range property in VBA is used to refer to specific rows or columns while writing the code. The code "Range("A1:A5").Value=2" returns the number 2 in the range A1:A5.

In VBA, **macros are recorded** and executed to automate the Excel tasks. This helps perform the repetitive processes in a faster and more accurate way. For running the macros, VBA identifies the cells on which the called tasks are to be performed. It is here that the range object in VBA comes in use.
The VBA range property is similar to the worksheet property and has several applications.

4. What is the difference between worksheet and sheet in excel?

1. Spreadsheet is a program that allows users to create a work report.

It is like an electronic table divided into rows and columns which can be used for the following:

- To prepare final accounts of a business - We can use spreadsheet software for recording business data.

- To present data in an orderly manner - A spreadsheet helps to present user data in an attractive and orderly manner.
- To prepare charts and graphs - Spreadsheet programs help us to represent our data using different tools like tales, pie charts, diagrams, etc.
- To make quick calculations. - Spreadsheets can be of great help to do quick calculations of complex mathematical data using formulas

2. Work Book:

- A workbook is a set of worksheets, which the users can use to record and store their data.
- Each individually saved file of a spreadsheet program is known as a workbook.
- There can be many worksheets in any workbook.
- We can store the information in an organized way in a single workbook.
- A workbook opens with three worksheets and the maximum worksheet can be 255

5. What is the difference between A1 reference style and R1C1 Reference style? What are the advantages and disadvantages of using R1C1 reference style?

In A1 reference style, you have column name as an alphabet and row name as a number and when you select the A1 cell that means you are in column A and row 1.

But in R1C1 both column and row are in numbers.

So, when you select cell A1 it shows you R1C1, which means row 1 and column 1, and if you go to A2 then it will be R2C1.

6. When is offset statement used for in VBA? Let's suppose your current highlight cell is A1 in the below table. Using OFFSET statement, write a VBA code to highlight the cell with "Hello" written in it.

	A	B	C
1	25	354	362
2	36	6897	962
3	85	85	Hello
4	96	365	56
5	75	62	2662

select a cell in Excel, you have two basic methods: RANGE and CELLS:

```
Range ("A1").Select

Range("RangeName").Select

Cells(3, 4).Select   'Selects Row 3, Column 4, i.e. cell D3
```

Range works well for hard-coded cells. Cells works best with calculated cells, especially when you couple it with a loop:

```
For i = 1 to 10

     Cells(i, 1).value = i    ' fill A1 through A10 with the value of i
```

```
Next i
```

Note that your focus does not change. Whatever cell you were in when you entered the loop is where you are when you leave the loop. This is way faster than selecting the cell, changing the value, selecting the next cell, etc. If you are watching the sheet, the values simply appear.

There are times when you are processing a list when you might want to look at the values in the same row, but a couple of columns over. You can accomplish this best with the .Offset clause. You might see the .Offset clause when you record a macro using relative references:

```
ActiveCell.Offset(1, 0).Range("A1").Select
```

This is both confusing and overkill. Translated into English, it takes the current cell (ActiveCell) and selects the row that is one row down from the current row and in the same column. The "Range("A1")" clause is not necessary. So if you want to stay in the current cell and read a value two columns to the right, you could use syntax like the following:

```
strMyValue = ActiveCell.Offset(0,2).Value
```

If you are in cell D254, the code above will reference the cell F254 and read its value into the variable strMyValue. This is far more efficient than selecting the cell two columns to the right, processing your data, then remembering to select two columns to the left and continue.

If you want to offset to a column to the left of you or a row above you, use a negative number. If you are in cell G254, the code

ActiveCell.Offset(-3,-2).Select will select E251 (3 rows up and 2 columns left).

You can loop through a list much more efficiently with Offset. It is easier to program and way faster to execute.

```
While ActiveCell.Value <> ""

    strFirstName = ActiveCell.Value

    strLastName = ActiveCell.Offset(0,1).Value

    dblSalary = ActiveCell.Offset(0,2).Value

    ActiveCell.Offset(0,2).Value = dblSalary * 1.05 'give a 5% raise

    MsgBox(strFirstName & " " & strLastName & ": Your new salary is " & dblSalary)

    ActiveCell.Offset(1,0).Select

Wend
```