

Agenda for Class 1

Introduction about

- a) Problem Solving
- b) Algorithm
- c) Flow Chart

a) Problem Solving

Example Problems:

- Calculating the distance between the Sun and the Moon (**Space research**)
- Calculating the relative velocity between a missile and its target (**Defence**)
- Measuring heartbeat, temperature of a patient (**Health management**)
- Calculating the speed of a vehicle ahead of your car in road (**Road accident prevention – Road safety**)
- Finding the average for students in a class (**Result processing**)
- Checking whether a water tank in a terrace is filled or not (**Water conservation**)
- Checking the moisture content of soil (**Smart Agriculture**)
- Calculating the amount of data consumed by a customer in a particular time period (Eg: data consumption in 1 hour / one day) (**Network facilitator's profit**)
- Searching for a particular book (**Library Management**)
- Comparing photo of a person with records in crime database (**Crime control**)
- Providing list of cinema theatres to a person (**Entertainment**)
- Predicting a cyclone/heavy rain fall (**Weather report / public safety**)

Activity: Identify at least 5 such problems similar to the above list, which are useful in day to day life.

About Problem Solving:

- Problem solving consists of moving from a given initial situation to a desired goal situation.
- That is, problem solving is the process of designing and carrying out a set of steps to reach a goal.
- Solving problems is the core of computer science.
- Programmers must first understand how a human solves a problem, then understand how to translate this "algorithm" into something a computer can do, and finally how to "write" the specific syntax (required by a computer) to get the job done.
- It is sometimes the case that a machine will solve a problem in a completely different way than a human.
- The problem solving is an art at this point and there are no universal approaches one can take to solving problems.
- Basically, one must explore possible avenues to a solution one by one until one comes across a right path to a solution.
- Thus, generally speaking, there is guessing and hence an element of luck involved in problem solving. (This brings innovation).
- However, in general, as one gains experience in solving problems, one develops one's own techniques and strategies, though they are often intangible.
- Computer Programmers are problem solvers.
- In order to solve a problem on a computer you must
 - Know how to **represent** the information (data) describing the problem.
 - Determine the steps to **transform** the information from one representation into another.

Here is a formal definition of the term problem.

1. You have a clearly defined given initial situation.
2. You have a clearly defined goal (a desired end situation).
3. You have a clearly defined set of resources that may be applicable in helping you move from the given initial situation to the desired goal situation. There may be specified limitations on resources, such as rules, regulations, and guidelines for what you are allowed to do in attempting to solve a particular problem.
4. You have some ownership--you are committed to using some of your own resources, such as your knowledge, skills, and energies, to achieve the desired final goal.

A Framework for Problem Solving (used by experts)

The following four phases can be identified in the process of solving problems:

- (1) Understanding the problem
- (2) Making a plan of solution
- (3) Carrying out the plan
- (4) Looking back i.e. verifying

One way to describe the process of problem solving might be the following:

1. understand the problem
2. understand what is necessary to achieve the solution, what are necessary inputs, outputs, tools, techniques, etc.
3. subdivide problem into more easily solvable pieces (if necessary)
4. solve each piece (prioritize subdivided pieces)
5. reassemble the solved pieces
6. verify results match expectations

b) Algorithm

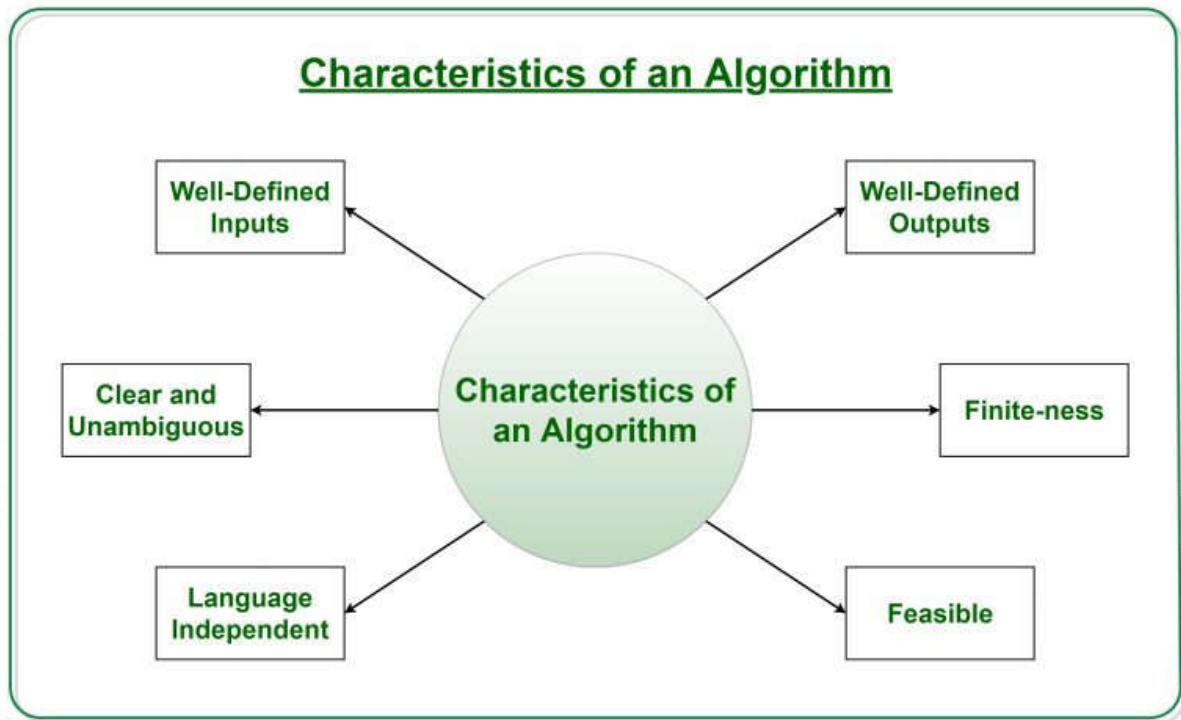
- Algorithm refers to a set of rules/instructions that define step-by-step how a work is to be executed upon in order to get the expected results
- **Before we can start finding programming solutions for problems,** we need a way to describe the solution steps.
- The formal name given to these solution steps is an **algorithm**.
- An algorithm is a **finite series of well-defined steps for converting the inputs of a specific problem into meaningful**, desired outputs.
- Take note that prior to developing an algorithm, we must know and understand what the **required inputs and desired outputs** are.
- The core of what good programmers do is being able to define the steps necessary to accomplish a goal.
- Unfortunately, a computer, only knows a very restricted and limited set of possible steps.
 - For example, a computer can add two numbers.
 -
 - But if you want to find the average of two numbers, this is beyond the basic capabilities of a computer.
 -
 - To find the average, you must:
 - First: Add the two numbers and save this result in a variable
 - Then: Divide this new number the number two, and save this result in a variable.
 - Finally: provide this number to the rest of the program (or print it for the user).

- The Algorithms designed are language-independent, i.e. they are just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.
- Eg: Pseudocode can be an English language-like description of the problem solution.
- Pseudocode is not representative of any specific programming language, i.e. algorithm is not computer code, and therefore cannot be used for actual programming.
- It is merely a solution description.
- it has a start, a middle, and an end. In fact, you will probably label the first step 'start' and the last step 'end.'

Thus, sample pseudocode for the problem above might look something like:

```
get persons age
if persons age >= 18 then
    output "come on in"
otherwise
    output "come back next year"
endif
```

- Note that not every individual's pseudocode will look exactly the same.
- For example, one may use "get person's age" as above, or one may use "input person's age", but the intent should be clear in any case.



It must have the following characteristics:

- **Clear and Unambiguous:**
 - Algorithm should be clear and unambiguous.
 - Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs:**
 - If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs:**
 - The algorithm must clearly define what output will be yielded and it should be well-defined as well.

- **Finite-ness:**
 - The algorithm must be finite, i.e. it should not end up in an infinite loops (never ending) or similar.
- **Feasible:**
 - The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources.
- **Language Independent:**
 - The Algorithm designed must be language-independent,
 - i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

How to Design an Algorithm?




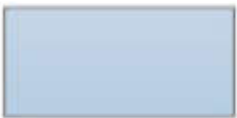
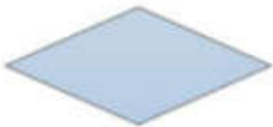
In order to write an algorithm, following things are needed as a pre-requisite:

1. The **problem** that is to be solved by this algorithm.
2. The **constraints** of the problem that must be considered while solving the problem.
3. The **input** to be taken to solve the problem.
4. The **output** to be expected when the problem the is solved.
5. The **solution** to this problem, in the given constraints.

Then the algorithm is written with the help of above parameters such that it solves the problem.

C) Flow Charts

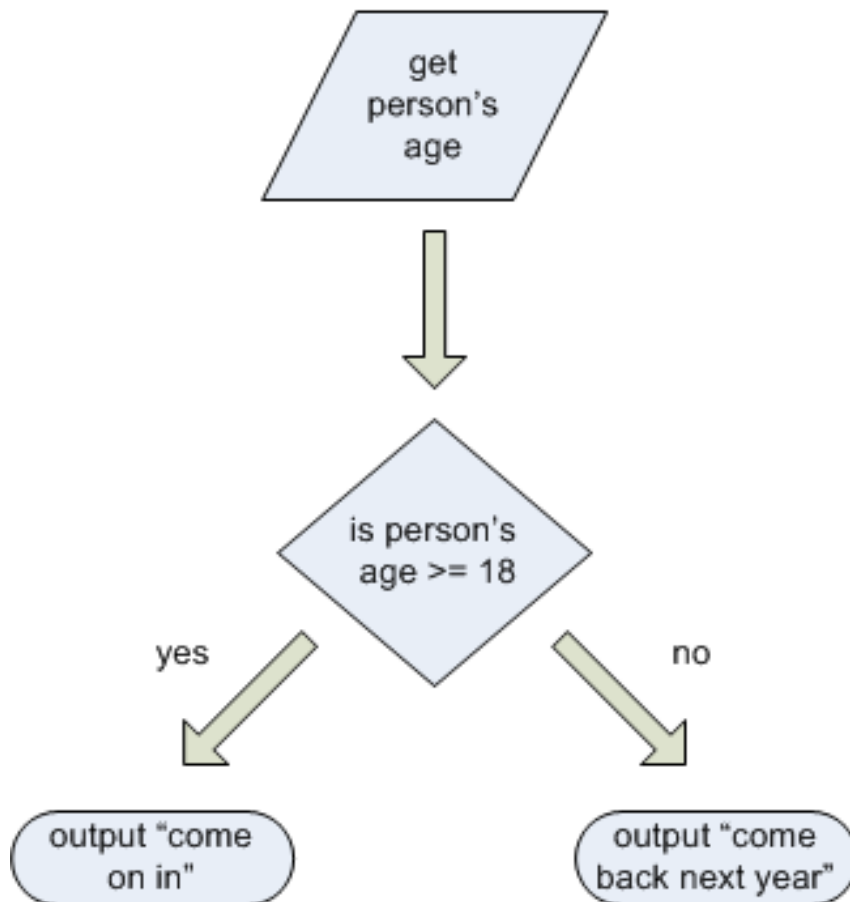
- Flowcharts are graphical representations of how a problem or process flows.
- There are special symbols that have predefined meaning in the flowchart diagram, as given in the following table.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Sample flowchart,

Representing the problem "Is a person old enough to vote?" (ignoring other factors), we can observe the following flowchart:

Is a person old enough to vote?



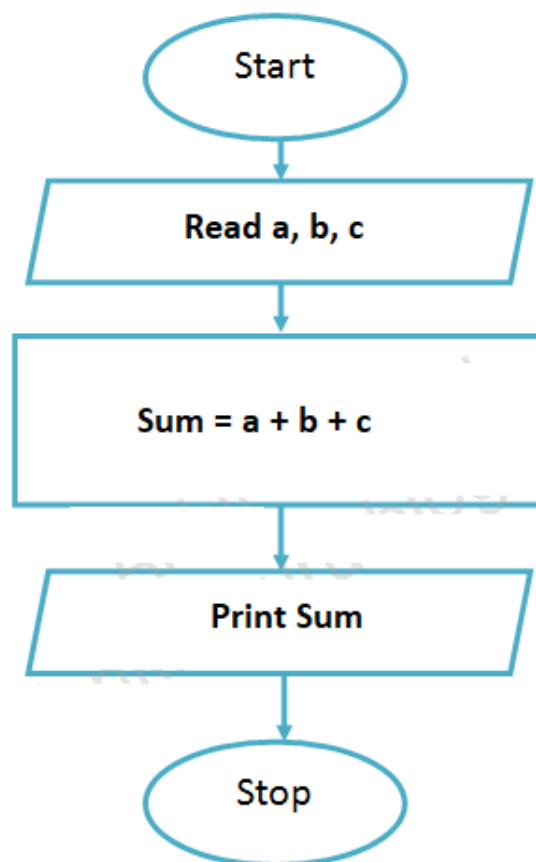
Examples:

(a) Problem: To add 3 numbers and print their sum:

Algorithm:

1. START
2. Read the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
3. Declare an integer variable sum to store the resultant sum of the 3 numbers.
4. Add the 3 numbers and store the result in the variable sum.
5. Print the value of variable sum
6. END

Flow Chart:



(b) Problem: To find whether a given number is 'odd' or 'even'

Algorithm:

Step 1: Start

Step 2: [Take Input] Read: Number

Step 3: Check: If Number % 2 == 0 Then

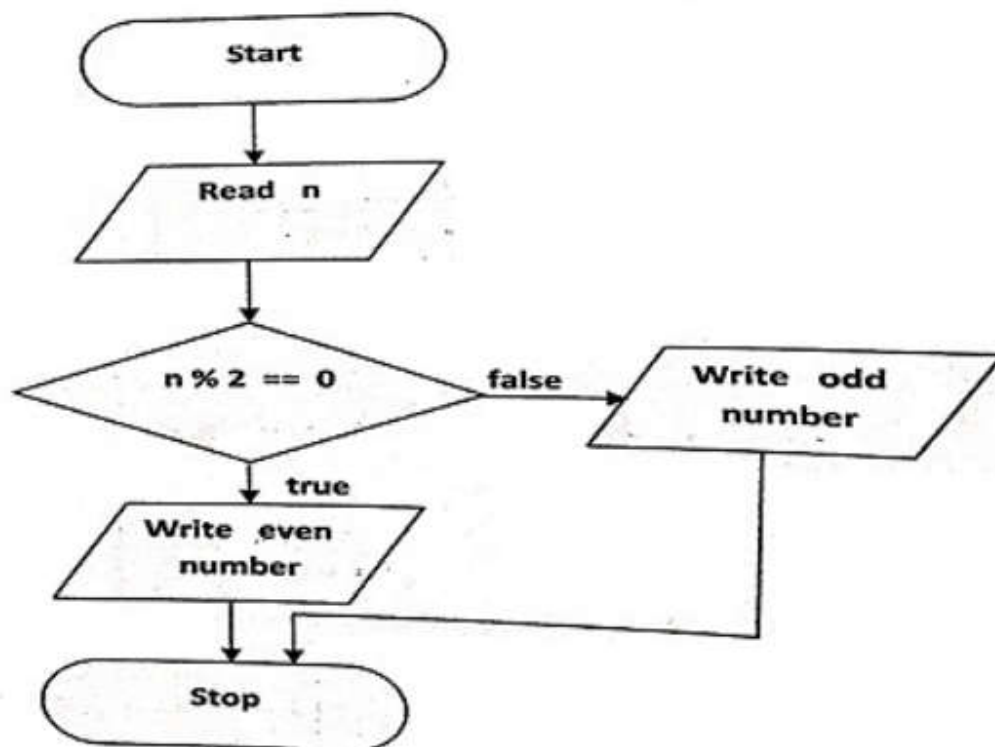
Print : N is an Even Number.

Else

Print : N is an Odd Number.

Step 4: End

Flow Chart:



(c) Problem: To find the largest among two different numbers entered by the user.

Algorithm:

Step 1: Start

Step 2: Declare variables a and b

Step 3: Read variables a and b (Eg: a=1, b=3)

Step 4: If a > b (i.e. 1 > 3)

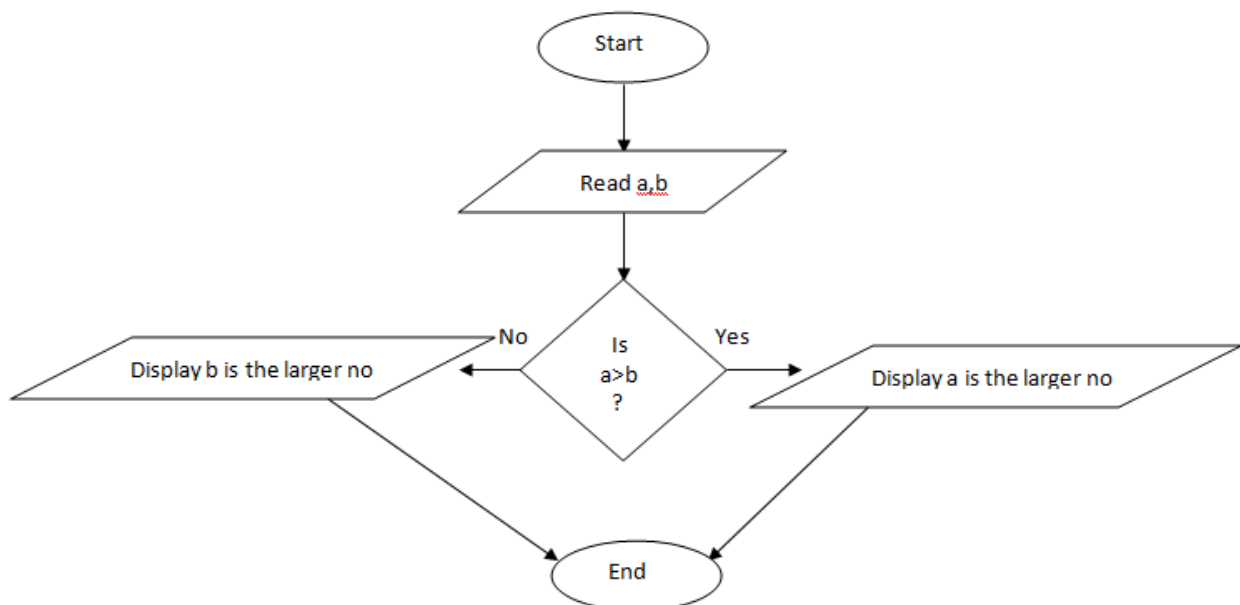
Display a is the largest number.

Else (i.e. 1 < 3)

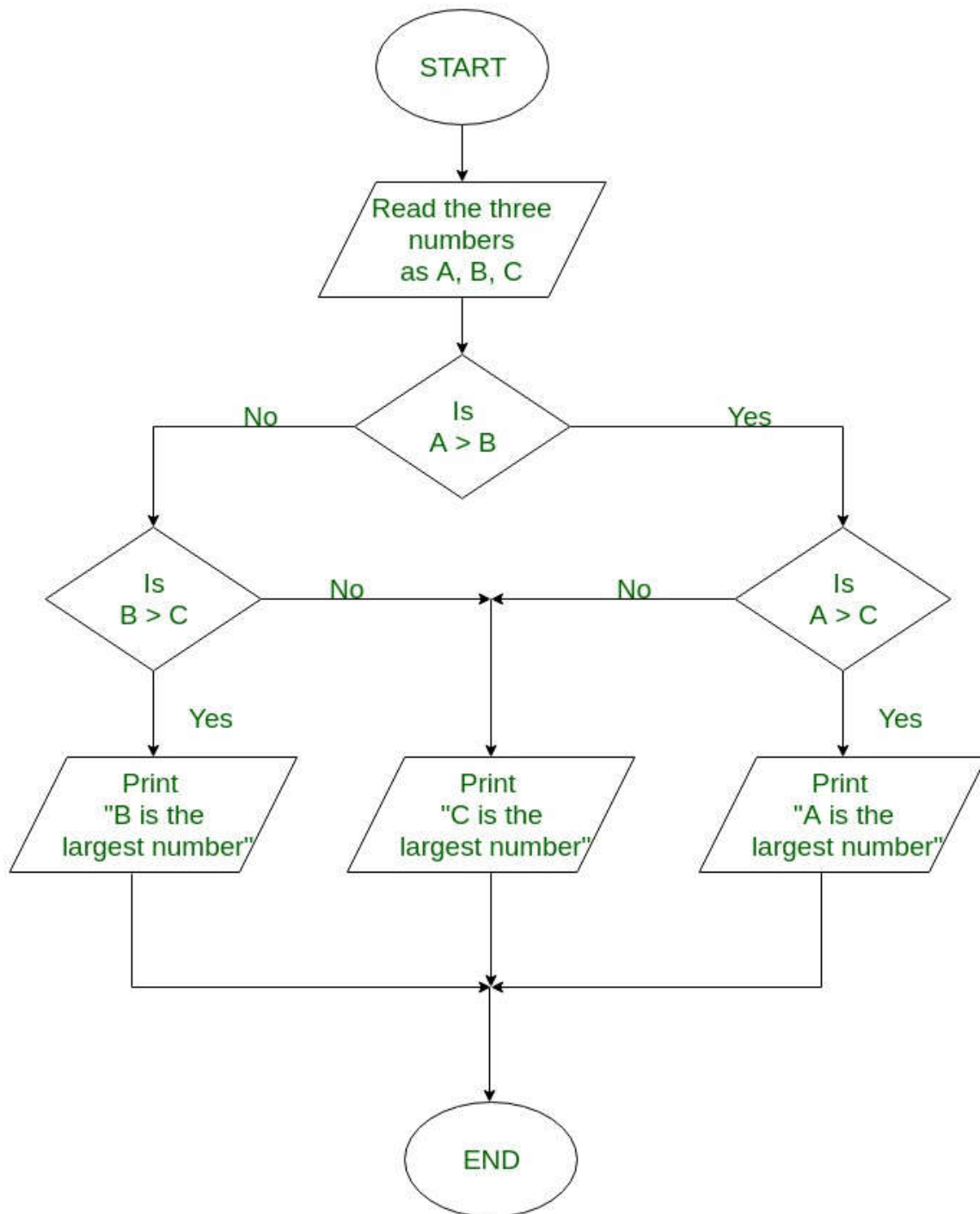
Display b is the largest number.

Step 5: Stop

Flowchart:



(d) Flowchart: To find largest among three different numbers



Activity:

Write algorithms and draw flow charts for the following problems.

Problems:

- 1) To find the area of a circle and print the result ($\text{Area} = \pi r^2$)
- 2) To print the sum of first '5' whole numbers (0,1,2,3,4,5,6,7,...)
- 3) To find the total marks and average of a student, who has 5 subjects.

Ref: <https://www.smartdraw.com/flowchart/flowchart-symbols.htm>

Notebook:

- Write your name and roll number in each page
 - Always mention date for each activity
 - Write/draw legibly
 - Randomly, the students will be called to show the notebook for evaluation (online)
-