CLASS-XI

Study Notes

Problem Solving

Period-01

Introduction:

Python programming language was developed by Guido Van Rossum in February 1991. Python is based on or influenced with two programming languages:

- ABC language, a teaching language created as a replacement of BASIC, and
- Modula-3

Python is an easy-to-learn yet powerful object oriented programming language. It is a very high level programming Language yet as powerful as many other middle-level not so high-level languages like C, C++, Java etc.

COMPUTATIONAL THINKING

Because of its power and capabilities, modern age computers can be used for solving a variety of problems from diverse areas, whether simple or complex ones. However, before a problem can be tackled, the problem itself and the ways in which it could be solved need to be understood.

Computational Thinking refers to the collective thought processes involved in formulating problem and solutions so that they are represented in a form that can be effectively carried out by a computer.

When thinking is focused on solving or computing a solution for a specific problem, it is the computational thinking.

Computational thinking is an approach to solving problems using concepts and ideas from computer science, and expressing solutions to those problems so that they can be run on a computer.

COMPUTATIONAL THINKING

Computational Thinking is a problem-solving process that includes the following characteristics.

- ❖ Decomposition. It is the breaking down a complex problem, data or process into smaller, more manageable parts.
- ❖ Pattern recognition/Data Representation. It refers to looking for similarities, patterns and trends in data.
- ❖ Abstraction. It is the filtering out unnecessary details/ information to focus only on important Information.
- ❖ Generalisation. It is identifying the common or shared characteristics. Sometimes generalisation is considered as a part of abstraction.
- ❖ Algorithms. It is the developing step-by-step instructions to solve this problem, or others like it

Based on the above characteristics, when a solution is devised, it must be evaluated for its effectiveness. Let us now talk about these characteristics one by one.

I. Decomposition:

Decomposition is the process of breaking down a big or complex problem into a set of smaller sub-processes to allow us to describe, understand, or execute the problem better. Decomposition involves:

- ➢ Dividing a task into a sequence of subtasks
- ➢ Identifying elements or parts of a complex system

Consider some examples of decomposition:

- ❖ Everyday example. Making cookies is a complex task that can be broken down into smaller simpler tasks such as mixing up the dough, forming into shapes via cookie cutters, and baking.
- ❖ Academic example. Writing an essay is a complex task that can be broken down into smaller tasks such as developing a thesis, gathering evidence, and creating a bibliography page.
- ❖ Engineering example. Designing a solution to construct a bridge by considering site conditions, technology available, technical capability of the contractor, foundation, etc.
- ❖ Computer Science example. Writing a computer program/ software by determining a well defined series of smaller steps (mostly in the form of modules and functions) to solve the problem or achieve a desired outcome.

2. **Pattern Recognition**

The goal of pattern recognition is to find common similarities and differences among objects. With proper patterns identified, we can solve seemingly diverse problems by a single algorithm.

Pattern recognition involves:

- ❖ Identifying similarities or common differences that lead us to shortcuts.
- ❖ using identified shortcuts, mapping problem characteristics to possible solution

Once the patterns are identified, they can be represented through appropriate data. Consider some examples of pattern recognition.

- Everyday example. While driving on roads, switching lanes promptly may cause accidents.

So the drivers look for patterns in traffic to decide whether and when to switch lanes.

- Academic example. Pattern recognition is required when categorizing rocks as igneous, metamorphic, or sedimentary.
- Science/Research example. Scientists and engineers look for patterns in data to derive theories and models.

3. **Abstraction:**

Abstraction or Data abstraction refers to focusing on information relevant to a context/problem and suppressing other details.

Abstraction involves:

- Recognising the context
- Identifying the information relevant to the context

Consider some examples of decomposition:

- **Everyday example**. When we tell a story or describe a movie to our friend, why don't we describe every single detail of the story or movie?
- **Academic example**. When we write a book report, we summarize and discuss only the theme or key aspects of the book, it is abstraction.
- **Engineering example**. When we overview an engineering solution in a proposal or a plan we highlight the underlying approach and hide the details.

- **Computer Science example**. A calculator program only shows the numbers and operates to function buttons to user without giving details of algorithm /program underneath.

3A Generalization:

Generalisation refers to identifying common or shared characteristics between two domains or problems such that models or solutions of one could be adapted or applied to the other.

Consider following examples.

- Mammals are warm blooded, give live birth, have hair, and so on. An elephant is a mammal. Therefore, it is warm blooded, give live birth, have hair.

Solids dissolve faster if they are smaller and the solution is warmer. A specific solid is not dissolving fast so either it is big in size or the solution is not warmer.

4. **Algorithm Design:**

It refers to what steps are needed to solve the problem and how can the steps best be organized? In other words, algorithms are a step-by-step strategy for solving a problem and Algorithm design is the development of algorithms.

An algorithm is a sequence of steps that solves a problem by working on some input data and producing a desired outcome (effective solution). Algorithmic thinking involves both creation and execution of an algorithm.

Four Principles of Computational Thinking

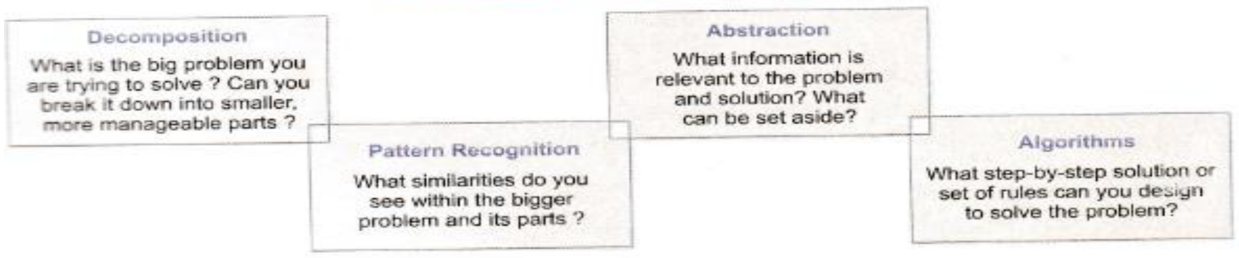

Decomposition
What is the big problem you are trying to solve ? Can you break it down into smaller, more manageable parts ?
Pattern Recognition
What similarities do you see within the bigger problem and its parts ?
Abstraction
What information is relevant to the problem and solution? What can be set aside?
Algorithms
What step-by-step solution or set of rules can you design to solve the problem?

Figure 5.1 Four principles of Computational Thinking

Evaluation:

This is the essential part of every type of constructive thinking including Computational Thinking. Evaluation aims to check to see whether a solution reached via decomposition, pattern recognition, abstraction/generalisation, algorithm design is good and effective.

Evaluation involves the consideration of:

- Algorithm correctness
- Requirements (meeting constraints, design principles, etc.)
- Performance (usability, efficiency, speed, complexity, reliability, etc.)

Examples:

When we carry out a physics experiment, say, to find the relationship between temperature and pressure, we check our data, investigate why it does not match the theory, redo our experimental setup, and recollect data points.

Period-02

Introduction :

Introduction to Problem solving

- A computer is a tool that can be used to implement a plan for solving a problem.
- A computer program is a set of instructions for a computer. These instructions describe the steps that the computer must follow to implement a plan.
- An algorithm is a plan for solving a problem.
- A person must design an algorithm.
- A person must translate an algorithm into a computer program.

An Algorithm Development Process:

Every problem solution starts with a plan. That plan is called an algorithm.

There are many ways to write an algorithm. Some are very informal, some are quite formal and mathematical in nature, and some are quite graphical. The instructions for connecting a DVD player to a television are an algorithm. A mathematical formula such as πR2 is a special case of an algorithm. The form is not particularly important as long as it provides a good way to describe and check the logic of the plan.

The development of an algorithm (a plan) is a key step in solving a problem. Once we have an algorithm, we can translate it into a computer program in some programming language. Our algorithm development process consists of five major steps.

An Algorithm Development Process:

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Step 1: Obtain a description of the problem:

This step is much more difficult than it appears. In the following discussion, the word client refers to someone who wants to find a solution to a problem, and the word developer refers to someone who finds a way to solve the problem. The developer must create an algorithm that will solve the client's problem.

The client is responsible for creating a description of the problem, but this is often the weakest part of the process. It's quite common for a problem description to suffer from one or more of the following types of defects:

(1) the description relies on unstated assumptions,

(2) the description is ambiguous,

(3) the description is incomplete, or

(4) The description has internal contradictions. These defects are seldom due to carelessness by the client. Instead, they are due to the fact that natural languages (English, French, Korean, etc.) are rather imprecise. Part of the developer's responsibility is to identify defects in the description of a problem, and to work with the client to remedy those defects.

Step 2: Analyze the problem:

The purpose of this step is to determine both the starting and ending points for solving the problem. This process is analogous to a mathematician determining what is given and what must be proven. A good problem description makes it easier to perform this step.

- ❖ When determining the starting point, we should start by seeking answers to the following questions:
- ❖ What data are available?
- ❖ Where is that data?
- ❖ What formulas pertain to the problem?

- What rules exist for working with the data?

What relationships exist among the data values?

When determining the ending point, we need to describe the characteristics of a solution. In other words, how will we know when we're done? Asking the following questions often helps to determine the ending point.

- What new facts will we have?
- What items will have changed?
- What changes will have been made to those items?
- What things will no longer exist?

Step 3: Develop a high-level algorithm:

An algorithm is a plan for solving a problem, but plans come in several levels of detail. It's usually better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later. We can use an everyday example to demonstrate a high-level algorithm.

Problem: I need a send a birthday card to my brother, Mark.

Analysis: I don't have a card. I prefer to buy a card rather than make one myself.

High-level algorithm:

- Go to a store that sells greeting cards
- Select a card
- Purchase a card
- Mail the card

This algorithm is satisfactory for daily use, but it lacks details that would have to be added were a computer to carry out the solution. These details include answers to questions such as the following.

- "Which store will I visit?"
- "How will I get there: walk, drive, ride my bicycle, take the bus?"

❖ "What kind of card does Mark like: humorous, sentimental, risqué?"

These kinds of details are considered in the next step of our process.

Step 4: Refine the algorithm by adding more detail:

A high-level algorithm shows the major steps that need to be followed to solve a problem. Now we need to add details to these steps, but how much detail should we add? Unfortunately, the answer to this question depends on the situation. We have to consider who (or what) is going to implement the algorithm and how much that person (or thing) already knows how to do. If someone is going to purchase Mark's birthday card on my behalf, my instructions have to be adapted to whether or not that person is familiar with the stores in the community and how well the purchaser known my brother's taste in greeting cards.

When our goal is to develop algorithms that will lead to computer programs, we need to consider the capabilities of the computer and provide enough detail so that someone else could use our algorithm to write a computer program that follows the steps in our algorithm. As with the birthday card problem, we need to adjust the level of detail to match the ability of the programmer. When in doubt, or when you are learning, it is better to have too much detail than to have too little.

Most of our examples will move from a high-level to a detailed algorithm in a single step, but this is not always reasonable. For larger, more complex problems, it is common to go through this process several times, developing intermediate level algorithms as we go. Each time, we add more detail to the previous algorithm, stopping when we see no benefit to further refinement. This technique of gradually working from a high-level to a detailed algorithm is often called **stepwise refinement**.

Stepwise refinement is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.

Step 5: Review the algorithm:

The final step is to review the algorithm. What are we looking for? First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem. Once we are satisfied that the algorithm does provide a solution to the problem, we start to look for other things.

The following questions are typical of ones that should be asked whenever we review an algorithm. Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

Does this algorithm solve a very specific problem or does it solve a more general problem? If it solves a very specific problem, should it be generalized?

For example, an algorithm that computes the area of a circle having radius 5.2 meters (formula π*5.22) solves a very specific problem, but an algorithm that computes the area of any circle (formula π*R2) solves a more general problem.

Review the algorithm:

Can this algorithm be simplified?

One formula for computing the perimeter of a rectangle is:

Length + width + length + width

A simpler formula would be:

2.0 * (length + width)

Is this solution similar to the solution to another problem? How are they alike? How are they different?

For example, consider the following two formulae:

Rectangle area = length * width
Triangle area = 0.5 * base * height

Similarities: Each computes an area. Each multiplies two measurements.

Differences: Different measurements are used. The triangle formula contains 0.5.

Hypothesis: Perhaps every area formula involves multiplying two measurements.

Period-03

Introduction :

Introduction to Algorithm & Pseudo code

Algorithm:

An algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem, based on conducting a sequence of specified actions. A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

Algorithms are widely used throughout all areas of IT (information technology). A search engine algorithm, for example, takes search strings of keywords and operators as input, searches its associated database for relevant web pages, and returns results.

Algorithm:

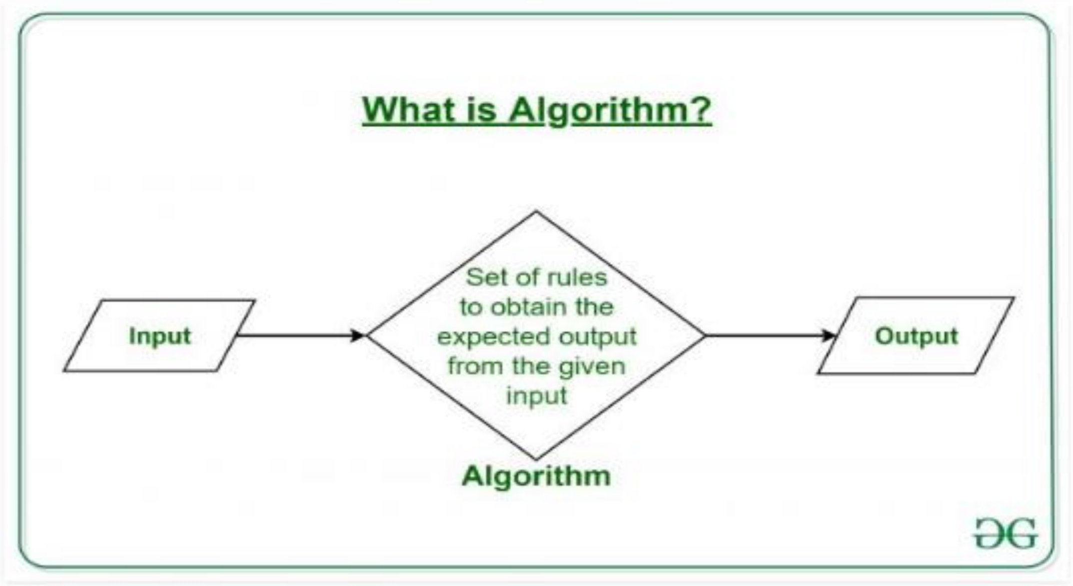
An encryption algorithm transforms data according to specified actions to protect it. A secret key algorithm such as the U.S. Department of Defense's Data Encryption Standard

(DES), for example, uses the same key to encrypt and decrypt data. As long as the algorithm is sufficiently sophisticated, no one lacking the key can decrypt the data.

The word algorithm derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word algebra as well.

Algorithm Basics:

The word Algorithm means “a process or set of rules to be followed in calculations or other problem-solving operations”. Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

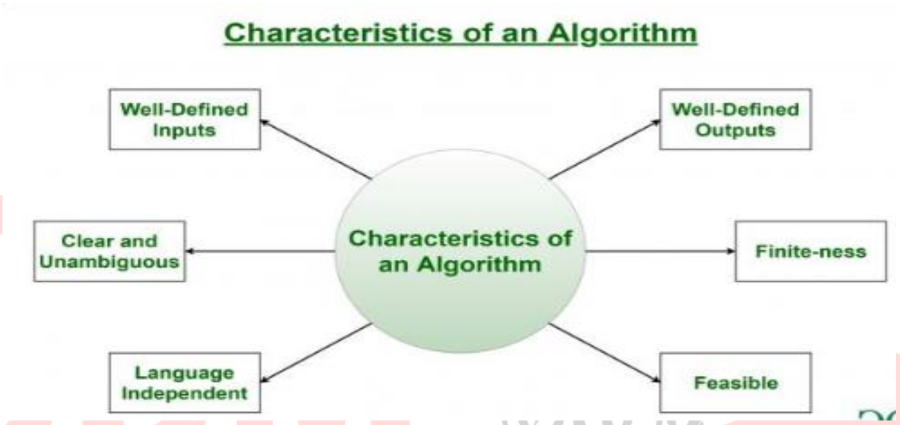
Algorithm Basics:

It can be understood by taking an example of cooking a new recipe. To cook a new recipe, one reads the instructions and steps and execute them one by one, in the given sequence.

The result thus obtained is the new dish cooked perfectly. Similarly, algorithms help to do a task in programming to get the expected output.

The Algorithm 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.

What are the Characteristics of an Algorithm?



Algorithm Basics:

As one would not follow any written instructions to cook the recipe, but only the standard one. Similarly, not all written instructions for programming is an algorithm. In order for some instructions to be an algorithm, 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 or similar.
- Feasible: The algorithm must be simple, generic and practical; such that it can be executed upon will the available resources. It must not contain some future technology, or anything.

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:

- The problem that is to be solved by this algorithm.
- The constraints of the problem that must be considered while solving the problem.
- The input to be taken to solve the problem.
- The output to be expected when the problem the is solved.
- 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.

Example: Consider the example to add three numbers and print the sum.

- Step 1: Fulfilling the pre-requisites

As discussed above, in order to write an algorithm, its pre-requisites must be fulfilled.

1. The problem that is to be solved by this algorithm: Add 3 numbers and prints their sum.
2. The constraints of the problem that must be considered while solving the problem: The numbers must contain only digits and no other characters.

3. The input to be taken to solve the problem: The three numbers to be added.
4. The output to be expected when the problem is solved: The sum of the three numbers taken as the input.
5. The solution to this problem, in the given constraints: The solution consists of adding the 3 numbers. It can be done with the help of '+' operator, or bit-wise, or any other method.

Step 2: Designing the algorithm

Now let's design the algorithm with the help of above pre-requisites:

Algorithm to add 3 numbers and print their sum:

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

Step 3: Testing the algorithm by implementing it.

In order to test the algorithm, let's implement it in C language.

ALGORITHMS AND FLOWCHARTS

A typical programming task can be divided into TWO phases:

- Problem solving phase
- Produce an ordered sequence of steps that describe solution of problem.

  This sequence of steps is called an **algorithm**

Implementation phase:

- implement the program in some programming language

STEPS IN PROBLEM SOLVING

- First produce a general algorithm (one can use pseudocode)
- Refine the algorithm successively to get step by step detailed algorithm that is very close to a computer language.
- Pseudocode is an artificial and informal language that helps programmers develops algorithms. Pseudocode is very similar to everyday English

PSEUDOCODE & ALGORITHM:

Example 1: Write an algorithm to determine a student's final grade and indicate whether it is Passing or failing. The final grade is calculated as the average of four marks.

Pseudocode:

```
• Input a set of 4 marks
• calculate their average by summing and dividing by 4
• if average is below 50
          Print "FAIL"
      Else
```

Print “PASS”

Detailed Algorithm

```
Step 1: Input M1, M2, M3, M4
Step 2: GRADE = (M1+M2+M3+M4)/4
Step 3: if (GRADE < 50) then
            Print “FAIL”
        Else
            Print “PASS”
        End if
```

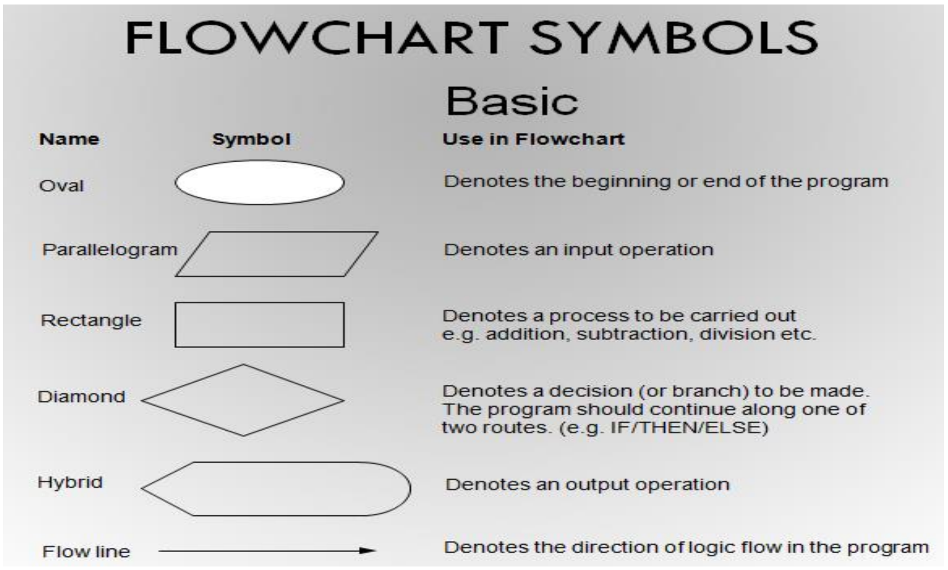
THE FLOWCHART:

(Dictionary) A schematic representation of a sequence of operations, as in a manufacturing process or computer program.

(Technical) A graphical representation of the sequence of operations in an information system or program. Information system flowcharts show how data flows from source documents through the computer to final distribution to users. Program flowcharts show the sequence of instructions in a single program or subroutine. Different symbols are used to draw each type of flowchart.

A Flowchart:

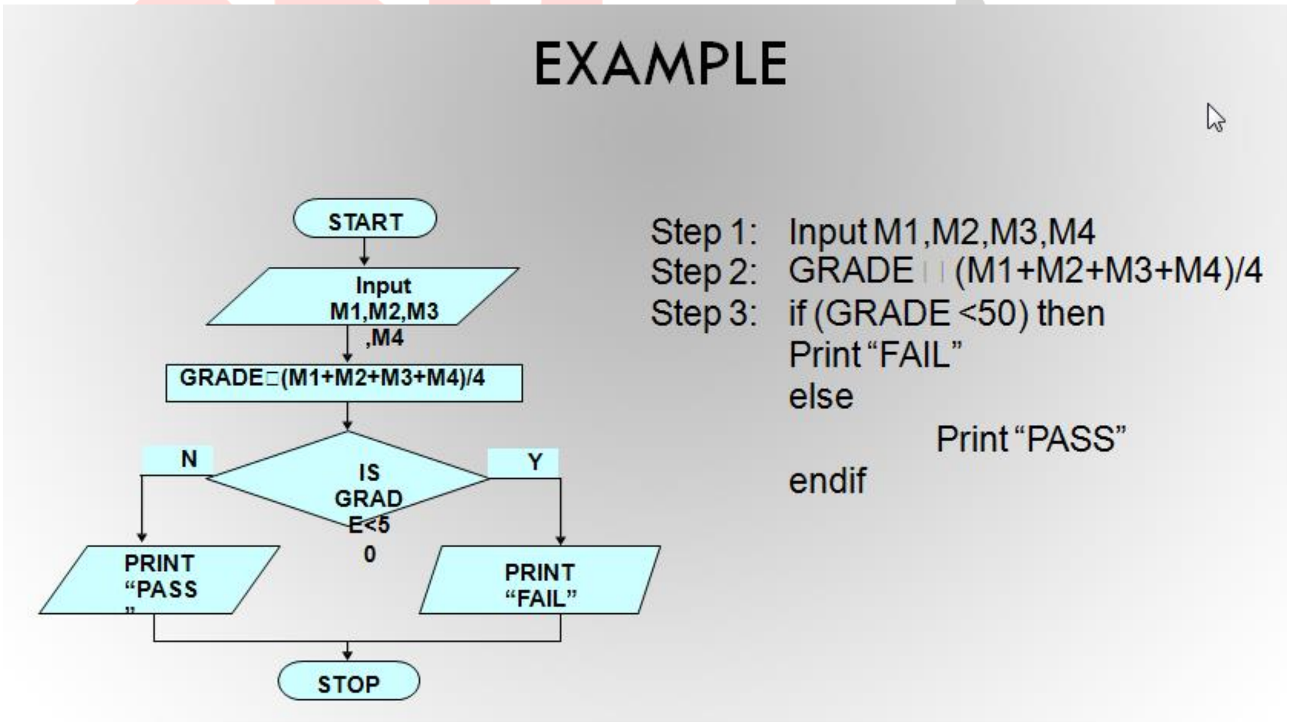
- Shows logic of an algorithm
- emphasizes individual steps and their interconnections e.g. control flow from one action to the next



FLOWCHART SYMBOLS

Basic

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program



EXAMPLE

```
Step 1:  Input M1,M2,M3,M4
Step 2:  GRADE □ (M1+M2+M3+M4)/4
Step 3:  if (GRADE <50) then
         Print "FAIL"
         else
                 Print "PASS"
         endif
```

Period-04, 05 & 06

Introduction:

Activity to draw flow chart

What is a flow Chart?

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence. They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes. If we consider all the various forms of flowcharts, they are one of the most common diagrams on the planet, used by both technical and non-technical people in numerous fields. Flowcharts are sometimes called by more specialized names such as Process Flowchart, Process Map, Functional Flowchart, Business Process Mapping, Business Process Modeling and Notation (BPMN), or Process Flow Diagram (PFD).

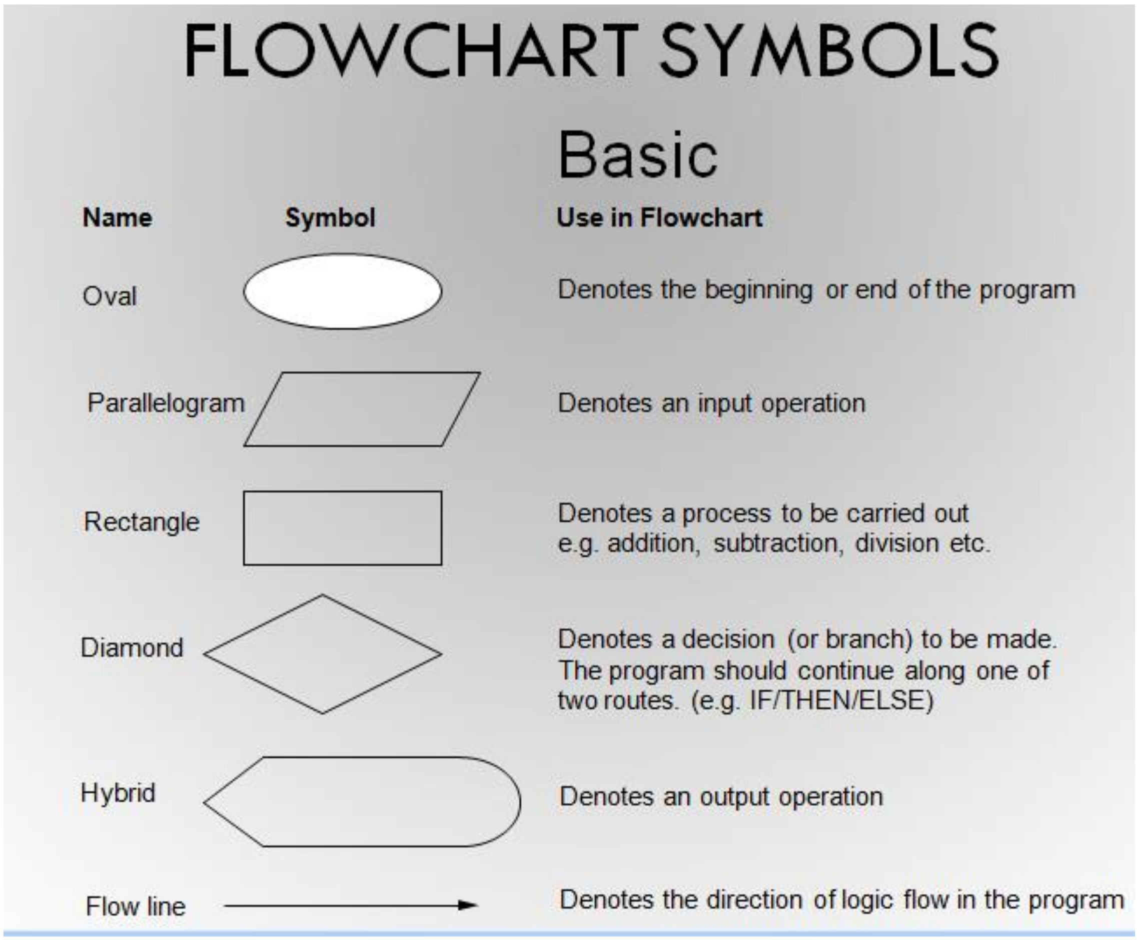
More flowchart tips

- Keep your audience in mind and gear the detail in your chart to them. Clear communication is a key goal of flowcharts.
- If the process you are charting involves different teams or departments, consider using a Swimlane Diagram to clearly delineate responsibilities and handoffs.
- Use on-page or off-page connectors to "edit" your chart and make it flow logically. This can allow you to break up a chart into separate pages and still flow well.

History

Flowcharts to document business processes came into use in the 1920s and '30s. In 1921, industrial engineers Frank and Lillian Gilbreth introduced the "Flow Process Chart" to the

American Society of Mechanical Engineers (ASME). In the early 1930s, industrial engineer Allan H. Morgensen used Gilbreth's tools to present conferences on making work more efficient to business people at his company. In the 1940s, two Morgensen students, Art Spinanger and Ben S. Graham, spread the methods more widely. Spinanger introduced the work simplification methods to Procter and Gamble. Graham, a director at Standard Register Industrial, adapted flow process charts to information processing. In 1947, ASME adopted a symbol system for Flow Process Charts, derived from the Gilbreths' original work.


FLOWCHART SYMBOLS

Basic

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

Flowcharts for computer programming/algorithm:

As a visual representation of data flow, flowcharts are useful in writing a program or algorithm and explaining it to others or collaborating with them on it. You can use a flowchart to spell out the logic behind a program before ever starting to code the automated process. It can help to organize big-picture thinking and provide a guide when it comes time to code. More specifically, flowcharts can:

- Demonstrate the way code is organized.
- Visualize the execution of code within a program.
- Show the structure of a website or application.
- Understand how users navigate a website or program.

How to plan and draw a basic flowchart:

1. Define your purpose and scope. What do you hope to accomplish? Are you studying the right things with appropriate start and end points to accomplish that purpose? Be detailed enough in your research but simple enough in your charting to communicate with your intended audience.

2. Identify the tasks in chronological order. This might involve talking to participants, observing a process and/or reviewing any existing documentation. You might write out the steps in note form, or begin a rough chart.

3. Organize them by type and corresponding shape, such as process, decision, data, inputs or outputs.

4. Draw your chart, either sketching by hand or using a program such as Lucidchart.

5. Confirm your flowchart, walking through the steps with people who participate in the process. Observe the process to make sure you haven't missed anything important to your purpose.

Flowchart Examples

Q.1. Addition of two inputted number

Q.2. largest among two inputted number

Q.3. Area of circle of inputted radius

Q.4. Area of Rectangle whose length and breadth are inputted by the user

Q.5. Area of Square whose length is inputted by the user

Flowchart Examples

Q.1. largest among three inputted number

Q.2. Fahrenheit into centigrade and vice versa

Q.3. Area of triangle whose sides are inputted by the user

Q.4. Swapping of two inputted number

Q.5. 1 + 2 + 3 + ... + n, where n is inputted by the user

Flowchart Examples

Q.1. Factorial of an inputted number

Q.2. Summation of each individual digit of an inputted number

Q.3. Input a number and check for Armstrong

Q.4. Reverse of an inputted number

Q.5. Input a number and check for palindrome

****