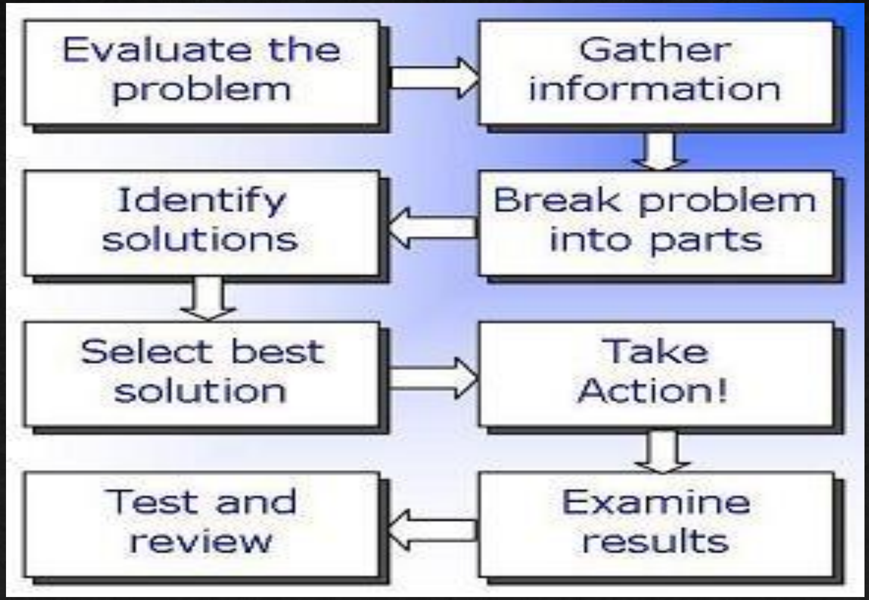
Lesson 1 : Problem Solving & Logic Building

Lesson 1 : topics

- Problem Solving
- Problem Solving Examples
- Logic Building
- Logic Building Examples
- Problem Solving vs Logic Building

What is Problem Solving ?

- Problem solving is a broader concept that includes the entire process of addressing & overcoming challenges.
- It is applicable to various aspects of life.
- Problem solving is a life skill you can grow over time.
- It involves
  - Assessing a situation
  - Identifying possible solutions
  - Choose the most appropriate course of actions
- Let's look at some real life examples….



Problem Solving : Real life example-1

- **Problem:**

You come home after a long day and realize you've lost your house keys.

- **Problem-Solving:**

You might first identify potential places where you might have left the keys.

You could retrace your steps, check your pockets, or think about any recent activities.

If you can't find them immediately, you might consider alternative solutions like contacting a roommate with a spare key, calling a locksmith, or seeking help from a neighbor.

Problem Solving : Real life example-2

- **Problem:**

While driving, you discover that your car has a flat tire.

- **Problem-Solving:**

Your initial step might involve finding a safe place to pull over.

Then, you could assess the damage, considering whether you have a spare tire and the necessary tools.

If you do, you might change the tire.

If not, you might need to call for roadside assistance or ask for help from a passerby.

Problem Solving : Real life example-3

- **Problem:**

You have a tight deadline at work and a significant amount of tasks to complete.

- **Problem-Solving:**

You could start by prioritizing tasks,

breaking down the larger project into smaller, more manageable parts.

Consider whether there are any tasks that can be delegated or postponed.

Create a schedule or to-do list to organize your work and ensure that you focus on the most critical aspects first.

Problem Solving in Programming 1/4

In the context of Programming , Problem Solving involves

1. Understanding the Problem:
   - Analysis: Start by thoroughly understanding the problem. What is the input? What is the expected output? What are the constraints or requirements?
   - Clarification: If the problem is not clear, seek additional information or clarification. Understand the specific conditions and constraints that the solution must meet.

2. Breaking Down the Problem:
   - Decomposition: Divide the problem into smaller, more manageable sub-problems. This makes it easier to tackle each part individually and solve the problem incrementally.
   - Identifying Patterns: Look for patterns or similarities to problems you've solved before. Recognizing patterns can help in applying known solutions or algorithms.

Problem Solving in Programming .. 2/4

3. **Designing an Algorithm:**
   - Algorithmic Thinking: Devise a step-by-step plan (algorithm) to solve each sub-problem. Consider the logical flow of the solution and the necessary control structures (if statements, loops).
   - Pseudocode: Express your algorithm in a high-level, human-readable pseudocode before translating it into actual code. This helps in refining the logic without getting bogged down by syntax.

4. **Choosing the Right Data Structures:**
   - Data Representation: Determine how to represent and organize the data involved in the problem. Choose appropriate data structures (arrays, lists, dictionaries) to efficiently store and manipulate information.
   - Efficiency Considerations: Consider the efficiency of data structures to ensure optimal performance of your program.

Problem Solving in Programming … 3/4

5. **Coding the Solution:**
   - Translation: Implement your algorithm in a programming language. Pay attention to the syntax and semantics of the chosen language.
   - Modularity: Break your code into functions or modules. This promotes reusability, readability, and easier debugging.

6. **Testing and Debugging:**
   - Testing: Check your program with various test cases to ensure it works correctly under different scenarios.
   - Debugging: If errors or unexpected behavior occurs, use debugging tools to identify and fix issues in your code.

Problem Solving in Programming …. 4/4

7. **Optimization:**
   - **Efficiency Improvements:** Analyze your code for opportunities to make it more efficient. This could involve optimizing algorithms, reducing unnecessary computations, or improving memory usage.

8. **Documentation:**
   - **Comments and Documentation: Clearly document your code, including comments that explain the purpose of each section. This helps others (and yourself) understand the code and its rationale**

What is Logic Building ?

- Logic building involves step by step details of implementation like breaking down a problem, making decisions based on conditions and creating a systematic plan.

- In the context of programming, logic building refers to the ability to create a logical and systematic sequence of steps or instructions (logic) to solve a specific problem or perform a task using a programming language.

Logic Building in programming 1/3

- In the context of programming, logic building includes :

1. **Algorithm Design:**
   - Creating step-by-step procedures or algorithms to solve a problem or perform a specific task within the software.
2. **Control Flow:**
   - Determining the order in which different parts of the program are executed. This involves using control structures like conditionals (if-else statements) and loops to manage the flow of the program.
3. **Data Flow:**
   - Defining how data is processed, transformed, and manipulated throughout the software. This includes decisions on data storage, retrieval, and manipulation.

Logic Building in programming 2/3

4. **Conditional Statements:**
   - Implementing logic that allows the program to make decisions based on specific conditions. Conditional statements, such as if-else, are essential for controlling the execution flow.
5. **Loop Structures:**
   - Using loop structures to repeat a set of instructions iteratively. This is often necessary for processing collections of data or performing repetitive tasks.
6. **Logical Operators:**
   - Employing logical operators (AND, OR, NOT) to combine and manipulate conditions in order to make more complex decisions in the software.
7. **Error Handling:**
   - Incorporating mechanisms to handle errors and unexpected situations gracefully, ensuring the software remains robust and user-friendly.

Logic Building in programming 3/3

8. **Abstraction:**
   - Simplifying complex functionality by focusing on essential details and hiding unnecessary complexities. Abstraction is crucial for creating modular and maintainable software.
9. **Efficiency Considerations:**
   - Optimizing algorithms and data structures to ensure that the software runs efficiently, especially for large datasets or resource-intensive tasks.
10. **Debugging Skills:**
    - Identifying and resolving errors or bugs in the code to ensure the correctness and reliability of the software.

Problem Solving/Logic Building : Building a house

❖ **Problem-Solving:**

- **Problem:** You want to build a house.
- **Problem-Solving Steps:**
  1. *Understanding the Problem:* Identify the requirements and constraints, such as the size of the house, budget, location, and legal regulations.
  2. *Breaking Down the Problem:* Divide the construction process into phases, such as planning, designing, obtaining permits, and actual construction.
  3. *Algorithm Design:* Develop a comprehensive plan for each phase, including architectural designs, budget allocations, and a construction timeline.
  4. *Efficiency Considerations:* Optimize the budget by finding cost-effective materials and construction methods.
  5. *Testing and Debugging:* Regularly inspect the construction site to ensure that the work aligns with the architectural plans and that any issues are addressed promptly.

Problem Solving/Logic Building : Building a house

- **Logic Building:**
  - **Problem:** You have the architectural plans, and now you need to implement the construction phase.
  - **Logic Building Steps:**
    1. *Algorithmic Thinking:* Develop a step-by-step plan for the construction process. This involves deciding the order of tasks, such as laying the foundation, framing, roofing, and interior finishing.
    2. *Control Flow:* Determine the sequence in which construction tasks are executed. For example, ensure that the foundation is laid before building the walls.
    3. *Data Flow:* Manage the flow of materials and resources, ensuring that they are delivered to the construction site when needed.
    4. *Conditional Statements:* Use conditional statements to handle variations, like adjusting construction plans if unexpected challenges arise.
    5. *Loop Structures:* Implement loops for repetitive tasks, such as laying bricks or installing tiles.
    6. *Error Handling:* Incorporate mechanisms to address unforeseen issues during construction, like unexpected weather conditions or material shortages.

Problem Solving vs Logic Building

- In the programming world, logic building and problem-solving are closely related concepts.
- While logic building is more about the detailed coding and algorithmic aspects, problem-solving in programming involves a broader set of skills, including requirements analysis, system design, and overall strategy formulation.
- Effective programmers need both logic building skills for detailed implementation and problem-solving skills for addressing the broader challenges associated with software development.

Lesson 1: Summary

Here is what we learned

- Problem Solving
- Logic Building

Lesson 2 : topics

- Algorithms
- Examples of Algorithm
- Flowcharts
- Examples of Flowchart

Lesson 2 : Algorithms & Flowcharts

What is Algorithm?

- An algorithm is a step-by-step procedure or set of rules for solving a specific problem or accomplishing a particular task.

- **Qualities of a Good Algorithm**
  - Input and output should be defined precisely.
  - Each step in the algorithm should be clear and unambiguous.
  - Algorithms should be most effective among many different ways to solve a problem.
  - An algorithm shouldn't include computer code. Instead, the algorithm should be written in such a way that it can be used in different programming languages.

Algorithm Example

- Problem : Calculate the average of two given numbers?

- Algorithm : Average of two numbers

```
Step 1 : Start
Step 2 : Declare variables num1, num2, sum and average.
Step 3 : Read values num1, num2.
Step 4 : Add num1 and num2 and assign the result to sum.
            sum ← num1 + num2
Step 5 : Divide sum by 2 and assign the result to average.
            average ← sum / 2
Step 6 : Display average
Step 7 : Stop
```

Problem Solving steps ?

Logic Building steps ?

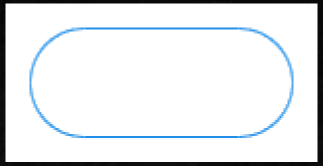
Anything missing?

Flowchart

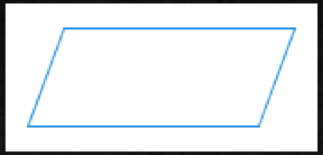
- Flowchart is a graphical representation of an algorithm.
- A well-designed flowchart should be easy to read and follow a logical sequence of steps.
- Advantages:
  - It is easy to understand.
  - It is a better way of communicating the logic of the system.
  - It provides better documentation.
- Disadvantages:
  - It is difficult to draw flowcharts for large and complex system.
  - If changes are done in software, then the flowchart must be redrawn.

Flowchart Symbols 1/2

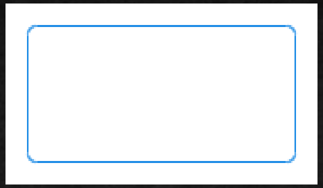
- **Terminal :** The oval symbol indicates Start, Stop and Halt in a program's logic flow.



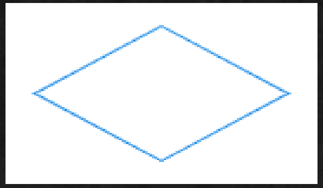
- **Input/Output :** A parallelogram denotes any function of input/output type.



- **Processing :** A box represents processing. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by process symbol.

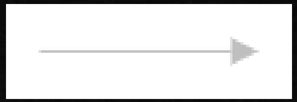


- **Decision :** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.

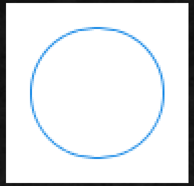


Flowchart Symbols 2/2

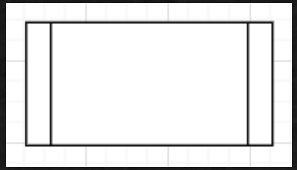
- **Flow lines :** Arrows represent the direction of flow of control and relationship among different symbols of flowchart.



- **Connectors :** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.



- Predefined Functions/Process : is represented as a box with double lines on its sides.

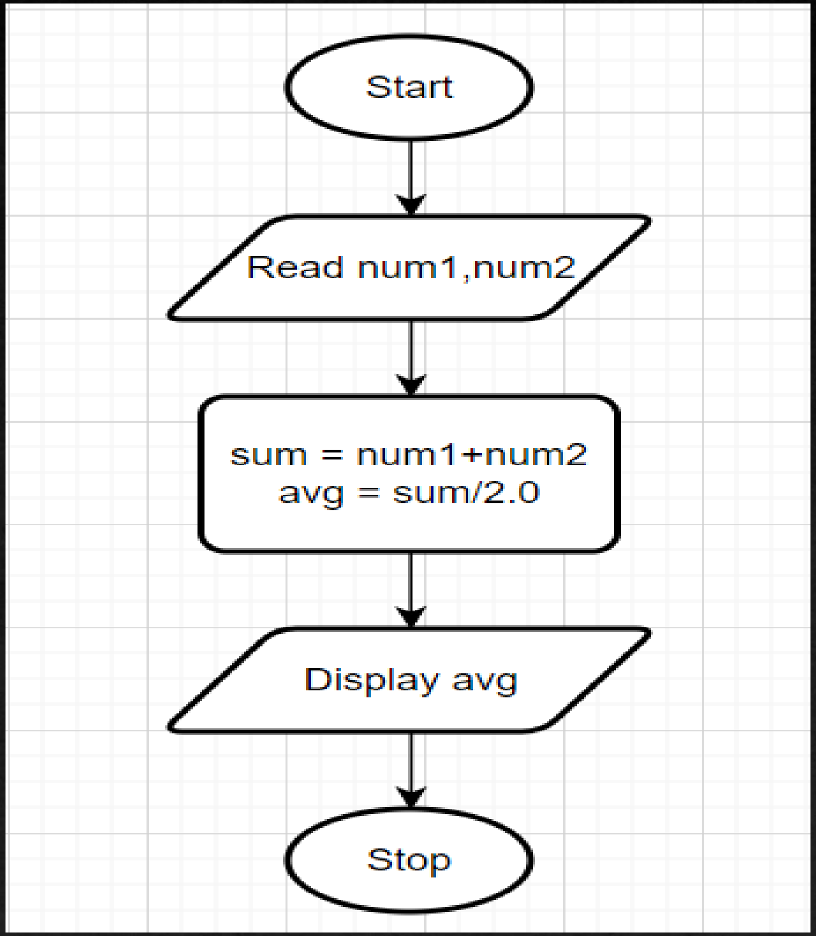


Rules for creating Flowchart

- Flowchart opening statement must be ‘start’ keyword.
- Flowchart ending statement must be ‘end’ keyword..
- All symbols in the flowchart must be connected with an arrow line.
- The decision symbol in the flowchart is associated with the arrow line.

Flowchart Example

Problem : Calculate the average of two given numbers?



Lesson 2: Summary

Here is what we learned

- Algorithm
- Flowchart