Algorithms, Flowcharts and Pseudocodes

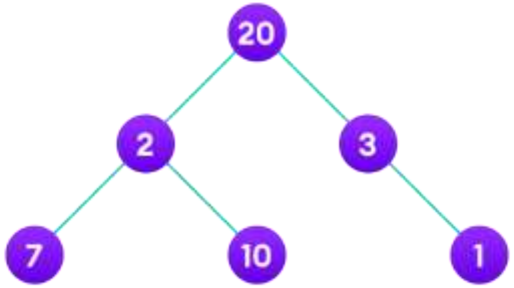
Algorithms

- ***Definition***
  - An algorithm is a step-by-step procedure or set of rules designed to perform a specific task or solve a problem.
- **Characteristics of a Good Algorithm:**

1. **Finiteness:** The algorithm should have a finite number of steps.
2. **Definiteness:** Every step must be clearly and unambiguously defined.
3. **Input:** The algorithm takes zero or more inputs.
4. **Output:** It produces at least one output.
5. **Effectiveness:** Each step must be basic enough to be carried out manually or mechanically.

Algorithms

- **Examples of Algorithms:**
  - **Real-life algorithms:** Directions to a location, finding a book in a library etc.
  - **Simple coding algorithms:** How to add two numbers, sorting numbers in a list, etc.
- **Types of Algorithms:**
  - **Greedy Algorithm:** A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result. (e.g., longest path in a tree).
  - **Divide and Conquer Algorithm:** Split a problem into smaller parts (e.g., merge sort).
  - **Dynamic Programming:** Breaking problems into sub-problems and storing the results of sub-problems (e.g., Fibonacci series i.e. *0, 1, 1, 2, 3, 5, 8, 13, 21, 34,*).
  - **Brute Force:** Simple, exhaustive search approach.



Algorithms

- **Steps in Writing an Algorithm:**
  1. Understand the problem.
  2. Break the problem into smaller steps.
  3. Define inputs and outputs.
  4. Write the procedure clearly and sequentially.
  5. Test the algorithm with various inputs.

Algorithms

- **Algorithm to add two numbers:**
- Steps:
  1. Start
  2. Declare two variables, `a` and `b`
  3. Input the values of `a` and `b`
  4. Calculate the sum of `a` and `b` and store it in a variable `sum`
  5. Display `sum`
  6. Stop
- Example:
  - Input: 50, 64
  - Output: 114

Algorithms

- **Algorithm to Check if a Number is Even or Odd**
- Steps:

1. Start
2. Declare a variable `num`
3. Input the value of `num`
4. If `num % 2 == 0`, print "Even"
5. Else, print "Odd"
6. Stop

- Example:

Input: num = 8

Output: Even

Algorithms

- **Algorithm to find the largest number in an array:**

```
Step 1: Start
Step 2: Initialize a variable max to the first element of the array
Step 3: For each element in the array
    If the element is greater than max, update max
Step 4: Return max
Step 5: Stop
```

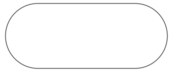
- Example: 30, 45, 67, 23, 56, **80**, 76

Flowcharts

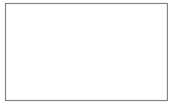
- **Definition:**
  - A flowchart is a diagram that graphically represents the flow of steps in a process or algorithm.
- **Advantages of Flowcharts:**
  - Easy visualization of the process.
  - Clear communication of ideas.
  - Helps in debugging and identifying errors in logic.

Flowcharts

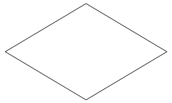
- **Symbols:**
  - **Oval (Start/End):** Represents the start or end of a process.



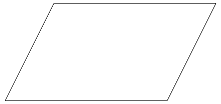
  - **Rectangle (Process):** Represents a process, action, or operation.



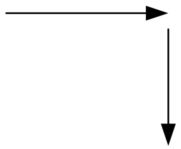
  - **Diamond (Decision):** Represents a decision point (e.g., Yes/No or True/False).



  - **Parallelogram (Input/Output):** Represents input/output operations.



  - **Arrow:** Shows the flow of control between different steps.

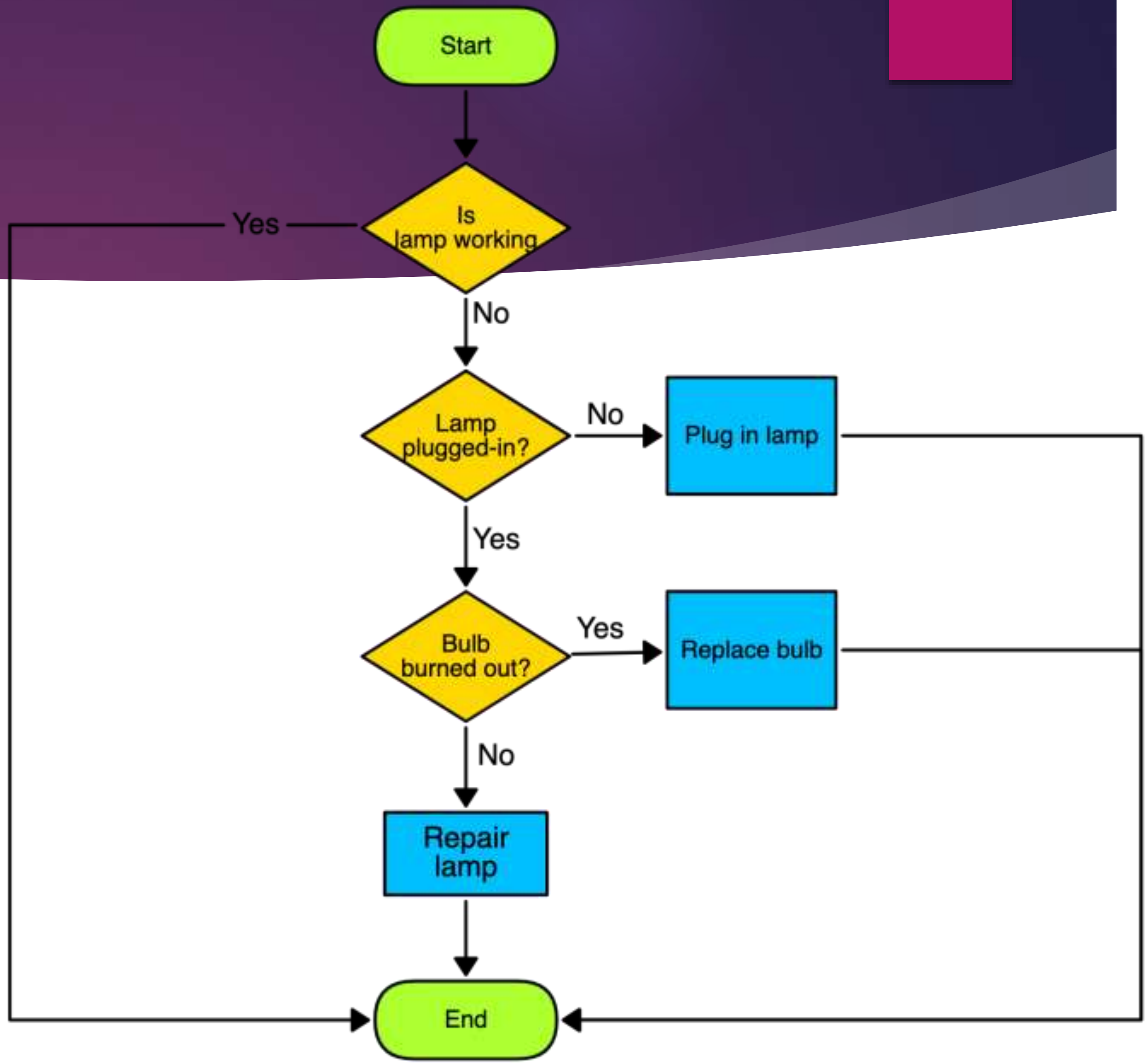


Flowcharts

- **Flowchart Rules:**
  - Flowchart begins with Start and ends with End.
  - Use the appropriate symbols for each step.
  - Maintain a left-to-right or top-to-bottom flow.
  - Ensure clarity in decision-making processes with clear labels for Yes/No or True/False.

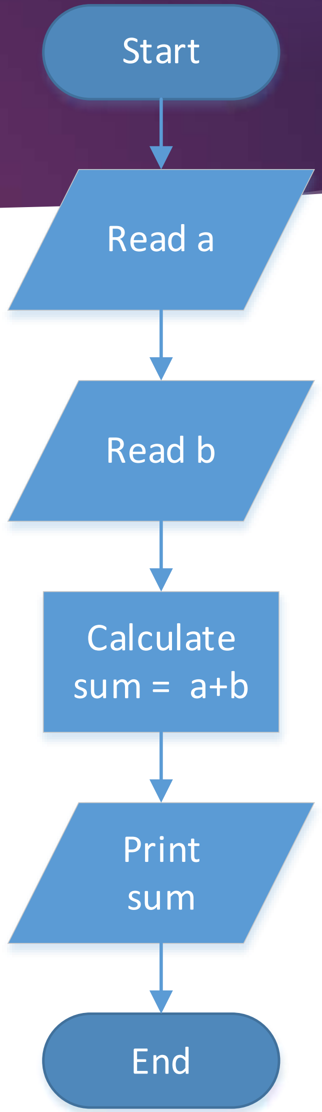
Flowcharts

- Lamp example



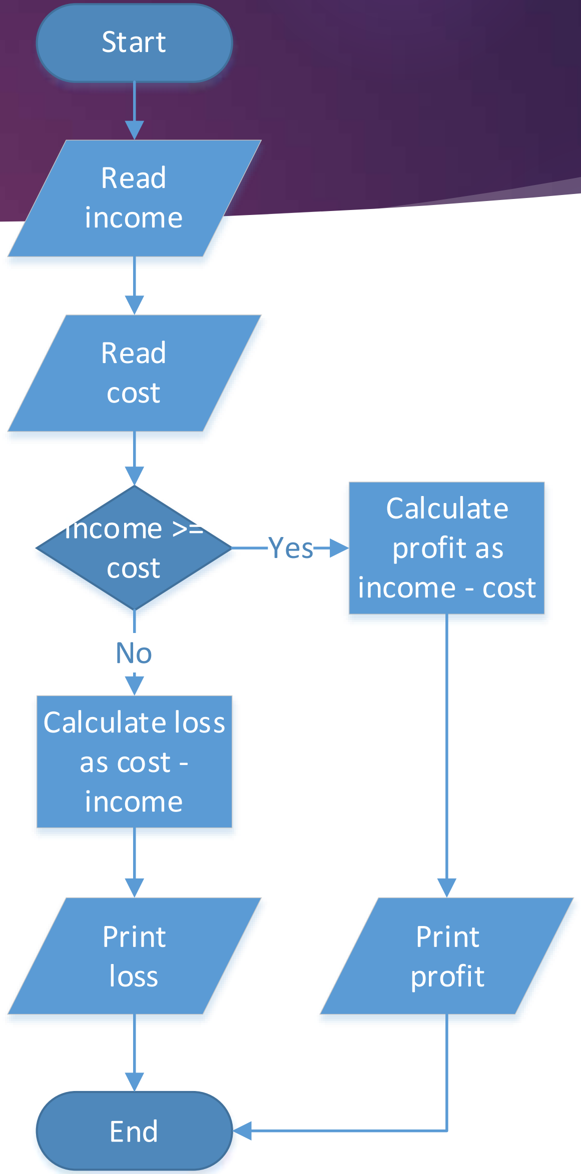
Flowcharts

- Flowchart for calculating sum of two numbers



Flowcharts

- Calculating profit/loss



Flowcharts

- **Comparing Algorithms and Flowcharts**
  - Algorithms are more abstract, focusing on logical steps.
  - Flowcharts provide a visual representation of these steps, making them easier to understand.
- **Common Mistakes and Best Practices**
  - Ensure every algorithm has a clear beginning and end.
  - Make flowcharts as simple as possible to avoid confusion.
  - Emphasize using pseudocode as a bridge between algorithms and actual programming.

Pseudocodes

- ***Introduction to Pseudocode***
  - Before we jump into actual programming code, it’s often useful to write something called pseudocode
  - This is a way to describe algorithms using a mix of natural language and programming-like syntax
  - It’s easier to understand
  - Doesn't rely on a specific programming language

Pseudocodes

- ***Convert an Algorithm to Pseudocode***
- **Example:**
- "Let's take our even-or-odd algorithm and write it in pseudocode:"

```
Start
Input number
If number % 2 == 0
    Print "Even"
Else
    Print "Odd"
Stop
```

- Pseudocode is more structured than regular language but simpler than actual code
- Helps in planning before writing the program