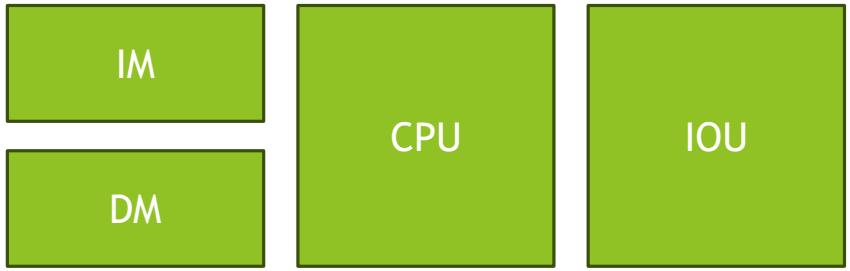
Programming Constructs

Problem Statement

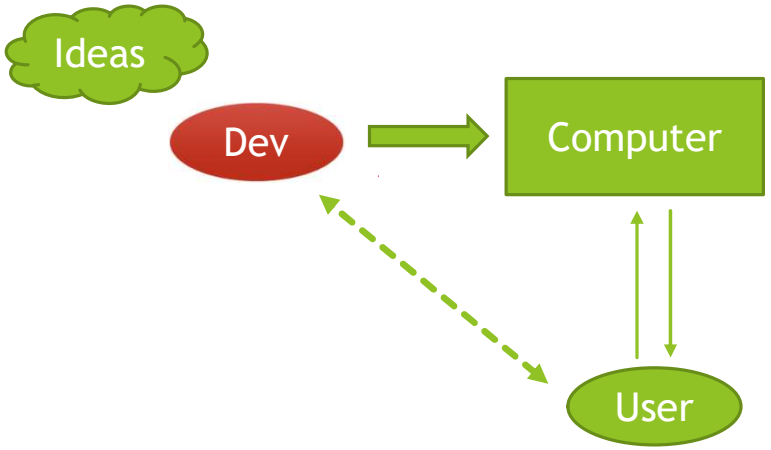
- Given three points in space, determine whether three points make a right angled triangle and/or an isosceles triangle

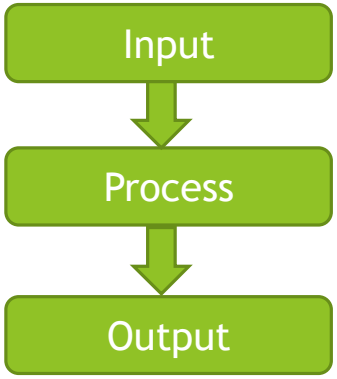
Computer Architecture

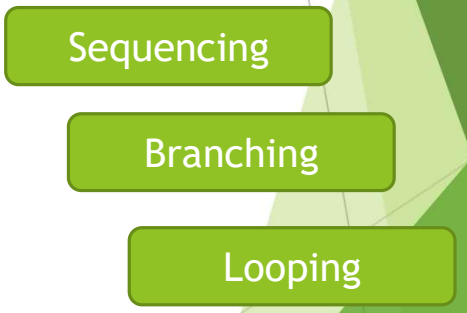


Computer Programming Concepts Review

- Computer is a hardware capable of performing certain task
- It needs to be instructed on what task to perform, hence the use of computer languages

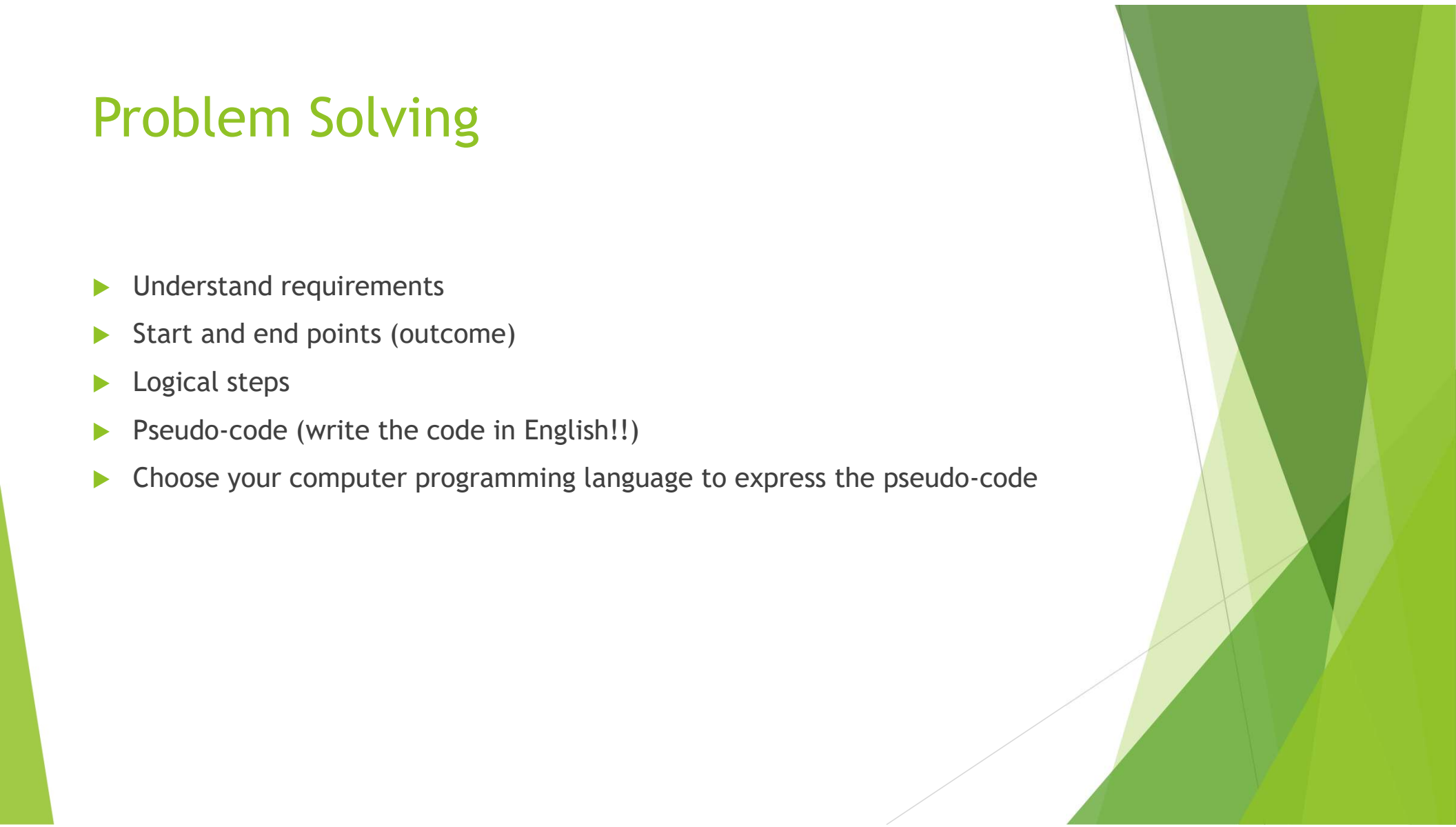






Problem Solving

- Understand requirements
- Start and end points (outcome)
- Logical steps
- Pseudo-code (write the code in English!!)
- Choose your computer programming language to express the pseudo-code

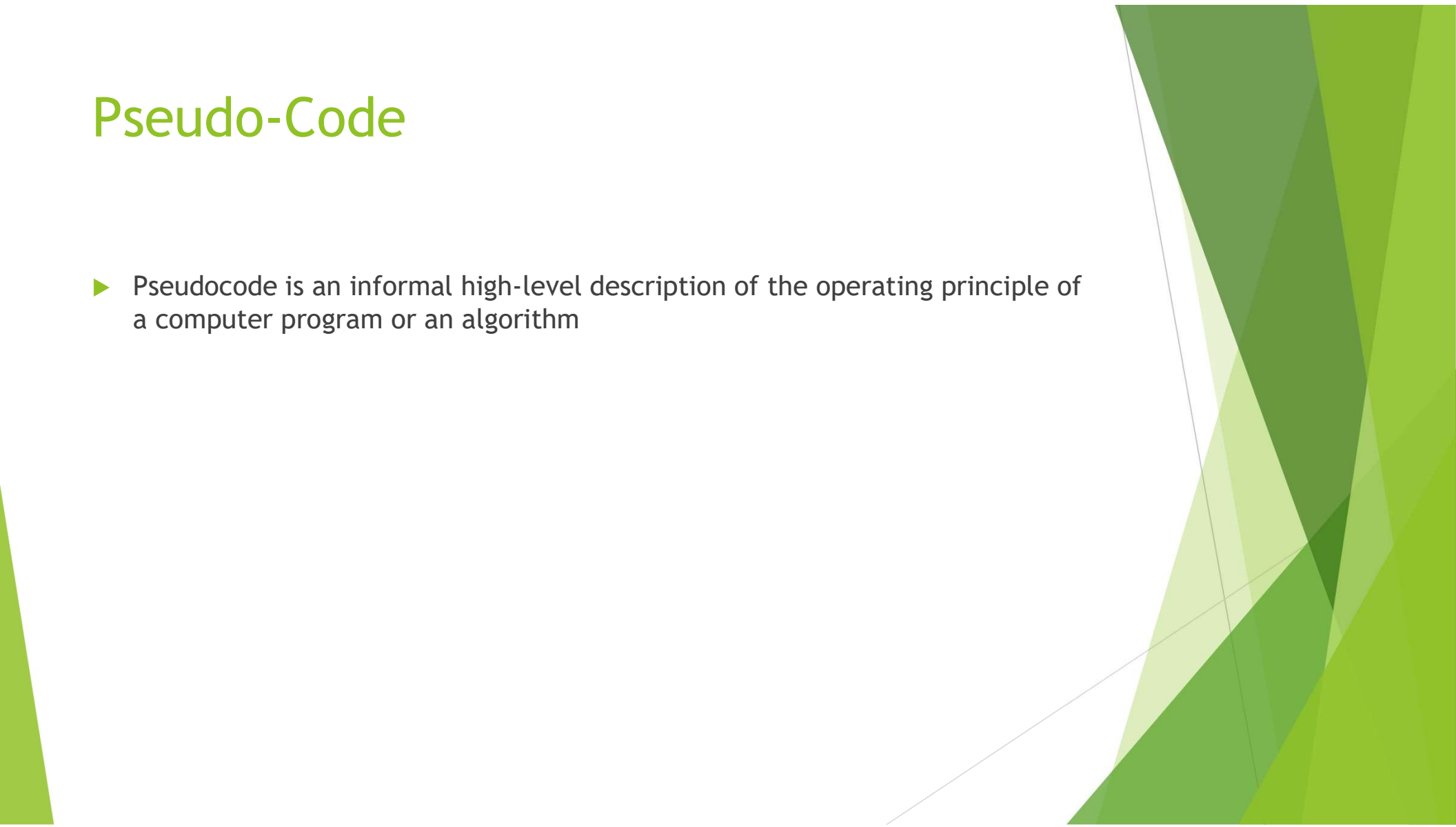


Programming Constructs

- Variables
- Expressions
- Statements
- Conditionals
- Loops
- Functions
- Modules
- Objects

Pseudo-Code

- Pseudocode is an informal high-level description of the operating principle of a computer program or an algorithm



Pseudo-Code

- **Pseudocode lets programmers write their application plans in plain English before transferring it to a programming language, like Java or Python**. This makes it easier to determine user flows and eliminate top-level flow errors.
- Developing software is a collaborative process. Along with programmers, you also have product designers, marketing managers, and other teams who have a stake in the application's development. Resolving problems among team members with varying levels of programming knowledge can eat up a lot of resources, which is one reason why pseudocode helps dev teams save both time and effort.
- Pseudocode helps developers find pitfalls and problems before development begins. Once the pseudocode outline is complete, it's usually inspected and verified by other programmers or system designers. In short, pseudocode makes it easier to write algorithms that work with minimal issues and deliver the desired result before you get to the nitty gritty coding work.

Advantages

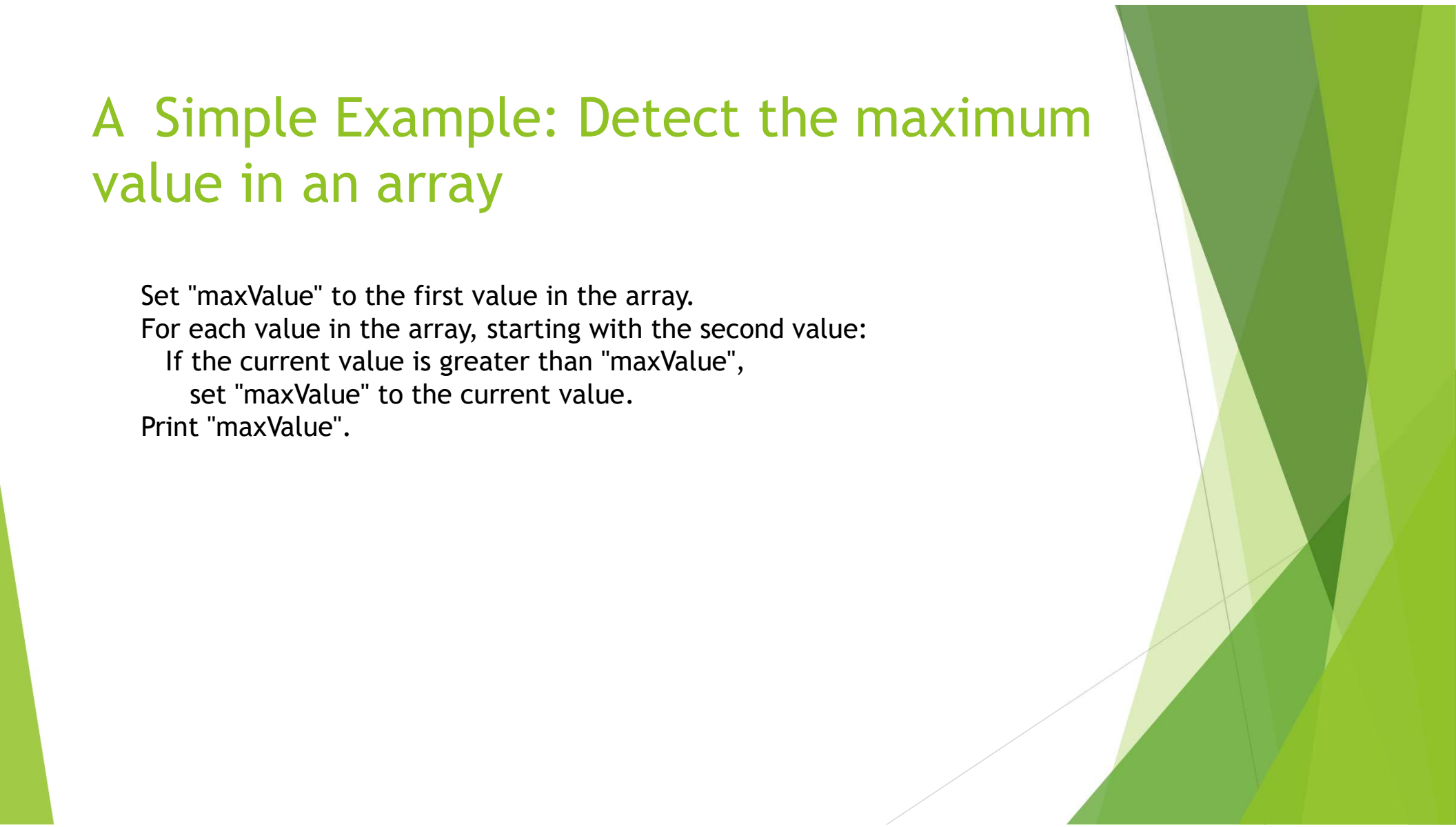
- Better readability. Pseudocode makes communication clearer for people from an array of disciplines, such as mathematicians, business partners, and managers.
- Better code construction. Pseudocode makes it easier to convert those principles into code written in any programming language faster.
- Faster development: Outlining a project with pseudocode helps solidify an idea and expedite production.
- Easier bug detection and repair. The higher-level format in pseudocode makes finding and repairing bugs faster and smoother before the first line of code is even written.

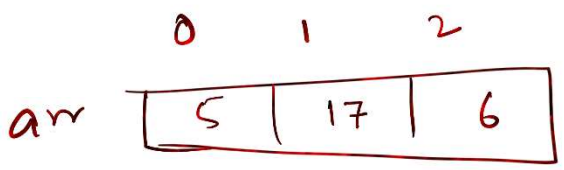
Key Elements

- SEQUENCE: It means a sequence performed right after another.
- WHILE: With a condition at the beginning, it is a loop.
- REPEAT-UNTIL: A loop with a condition at the bottom.
- FOR: Additional way to enact looping.
- IF-THEN-ELSE: Directs an algorithms flow.
- CASE: A generalization of IF-THEN-ELSE.

A Simple Example: Detect the maximum value in an array

Set "maxValue" to the first value in the array.
For each value in the array, starting with the second value:
  If the current value is greater than "maxValue",
    set "maxValue" to the current value.
Print "maxValue".





len(arr) = 3 - 1 = 2

Step = 1

max = arr[0]

ind = 1

{ if (max < arr[ind])

max = arr[ind]

Step : 2

ind = 2 (ind + 1)

if (max < arr[ind])

max = arr[ind]

Step 3 :

ind ← ind + 1

if (max < arr[ind])

max = arr[ind]

⋮

do this until you reach last index

ind == len(arr) - 1

=

max

```
arr = [10, 18, 35, 23, 12, 19]
max = arr[0]
ind = 1

repeat until ind reaches end of arr
begin
    if max < arr[ind]
        max = arr[ind]
    ind = ind + 1
end

print max
```

Ideas

↓

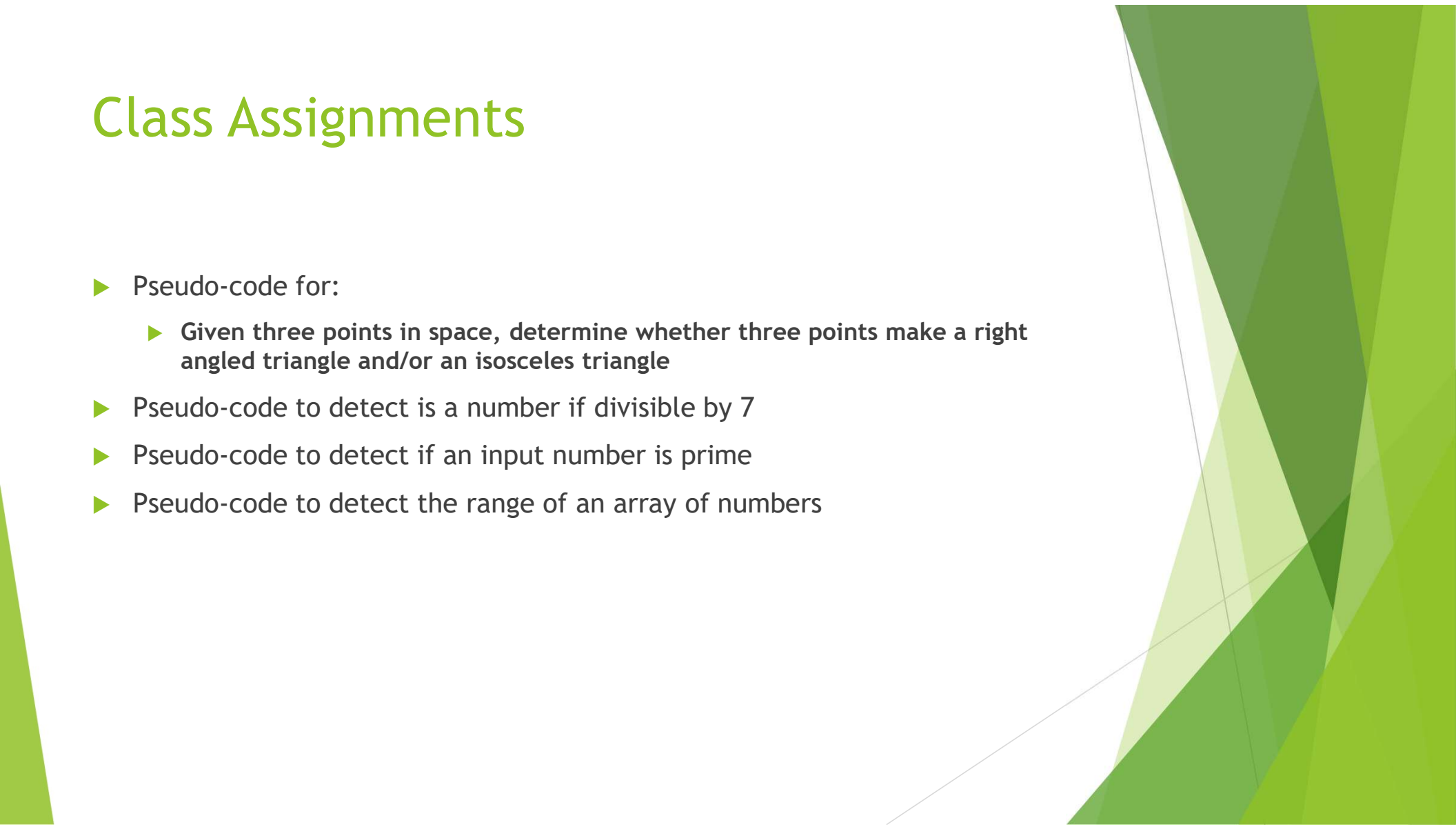
Systematic Approach

↓

Maintain sequence

Class Assignments

- Pseudo-code for:
  - **Given three points in space, determine whether three points make a right angled triangle and/or an isosceles triangle**
- Pseudo-code to detect is a number if divisible by 7
- Pseudo-code to detect if an input number is prime
- Pseudo-code to detect the range of an array of numbers



Assignment

- Pseudocode Development - Task: Write a detailed pseudocode for a simple program that takes a number as input, calculates the square if it's even or the cube if it's odd, and then outputs the result. Incorporate conditional and looping constructs

Flow Chart

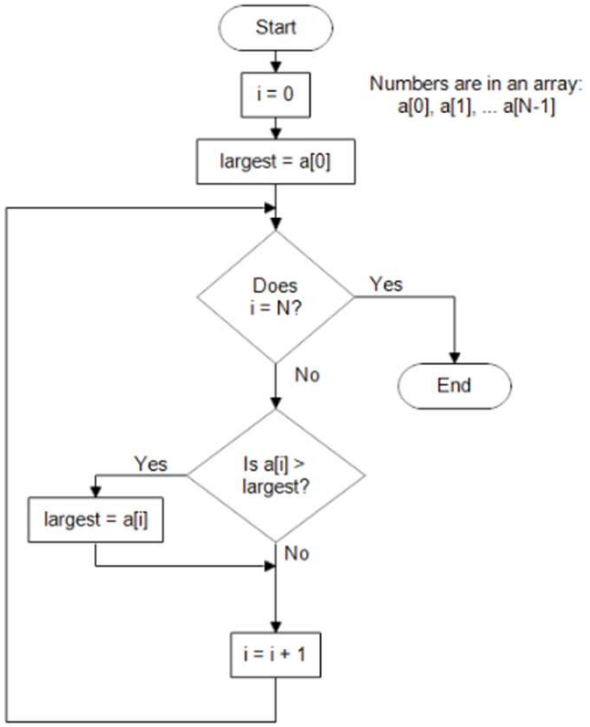
- A flowchart is a diagram that depicts a process, system or computer algorithm.

Flow Chart

- 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 an algorithm 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.

Flow Chart


Start
i = 0
Numbers are in an array:
a[0], a[1], ... a[N-1]
largest = a[0]
Does
i = N?
Yes
End
No
Yes
Is a[i] >
largest?
largest = a[i]
No
i = i + 1

Flow Chart Symbols

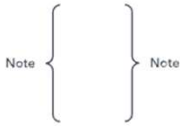
Terminal/Terminator

Process

Decision

Document

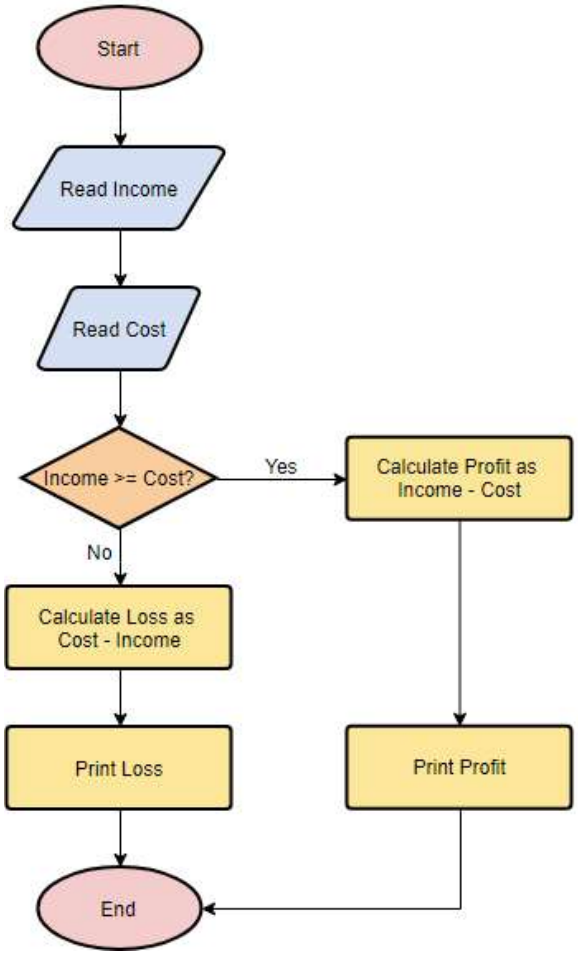
Flow Chart Symbols

Data, or Input/Output	
Stored Data	
Flow Arrow	
Comment or Annotation	
Predefined process	

Flow Chart Symbols

On-page connector/reference

Off-page connector/reference



Find the profit/loss when income = 1,000, cost = 800

Start

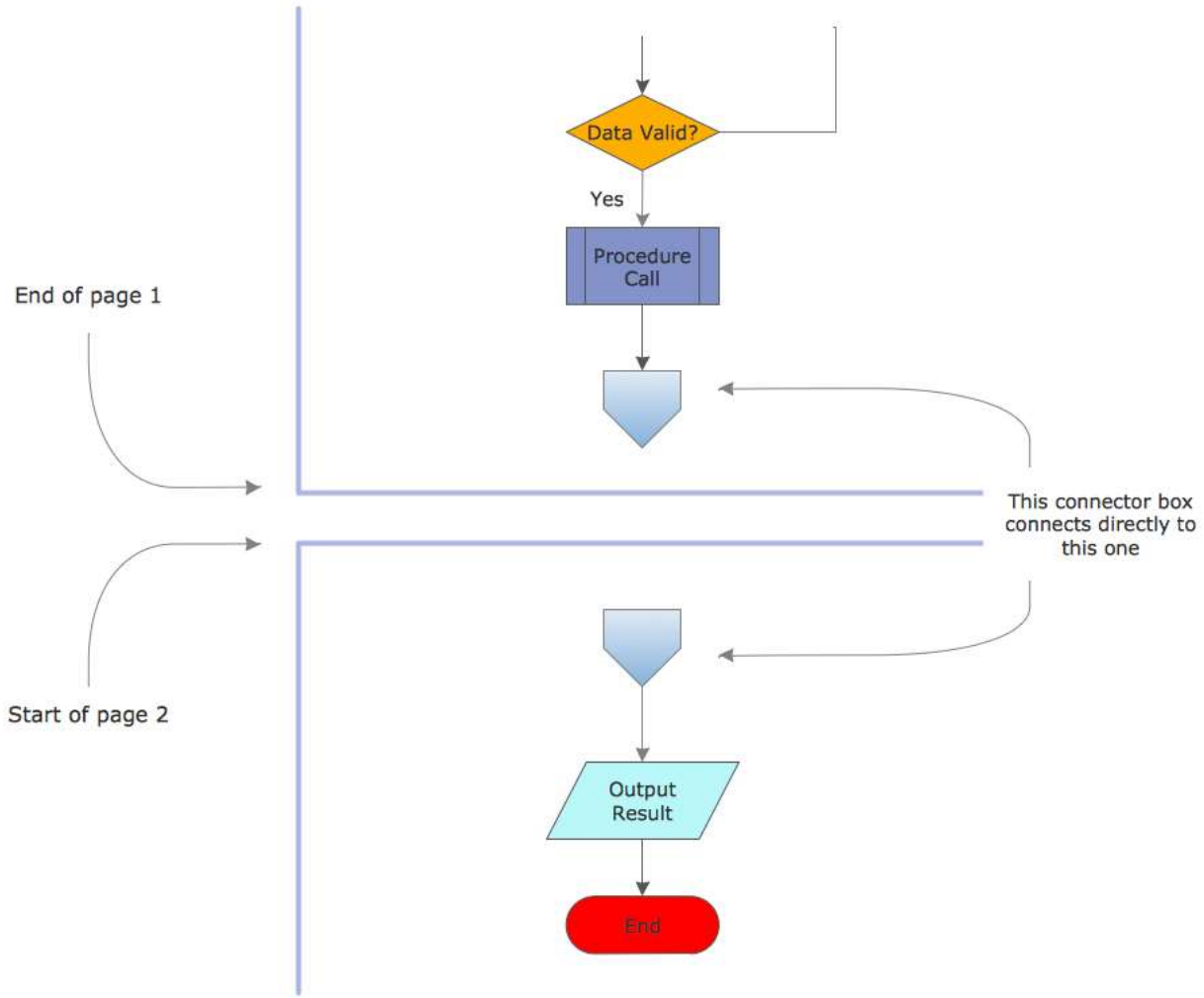
Income = 1,000

Cost = 800

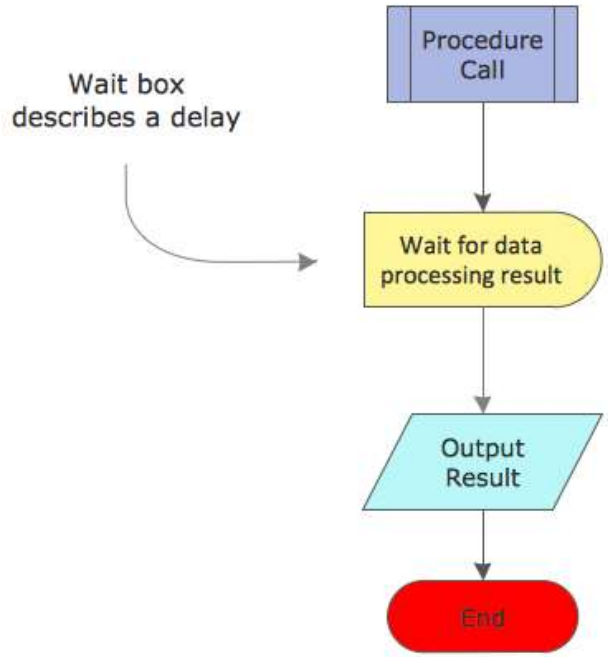
1,000 > 800? (Yes) **Profit = 1,000 - 800**

Profit = 200

End


Data Valid?
Yes
Procedure Call
End of page 1
This connector box connects directly to this one
Start of page 2
Output Result
End

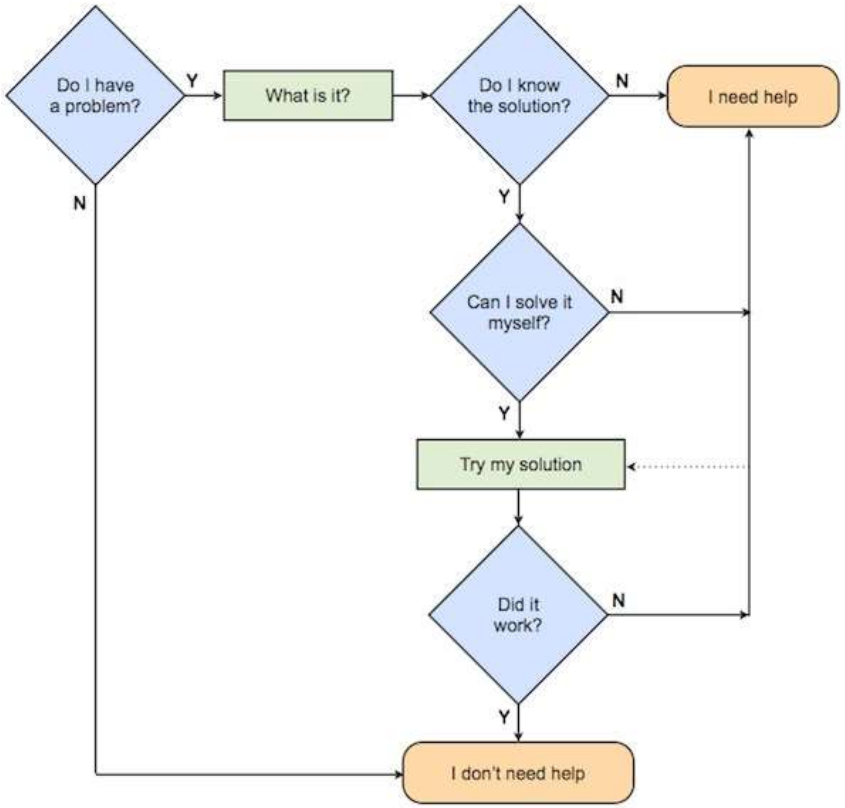
Delays

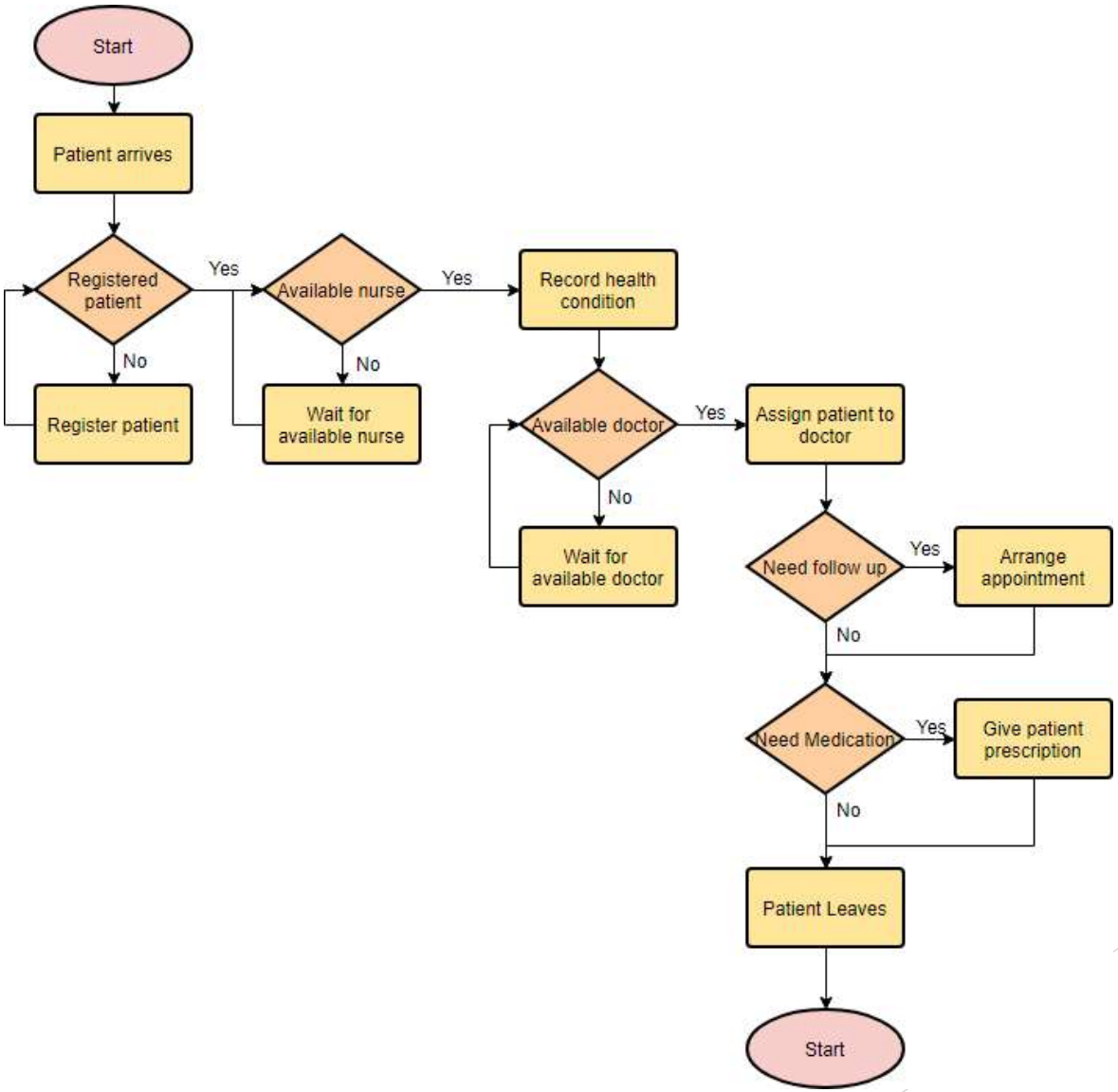

Wait box
describes a delay
Procedure
Call
Wait for data
processing result
Output
Result
End

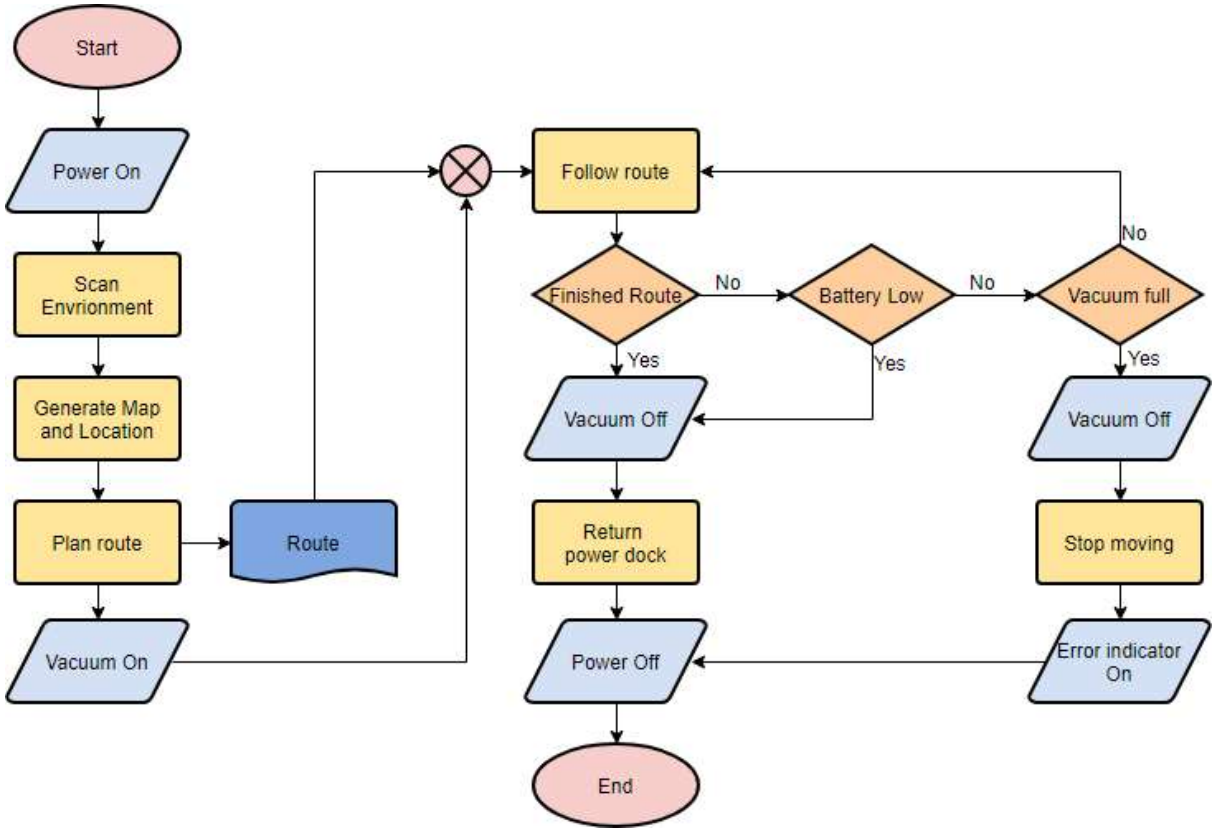
Class Assignments

- **Repeat practice: Find maximum of an array -> All participants**
- Given three points in space, determine whether three points make a right angled triangle and/or an isosceles triangle: Mounika, Vijay, Yashwanth, Suyog
- Flow chart to detect is a number if divisible by 7: Tanvi, Sudheer, Sushil, Durga Rani
- Flow chart to detect if an input number is prime: Vaibhav, Faizan, Jahnavi, Tarun
- Flow chart to detect the range of an array of numbers: Sushma, Vishnu, Utkarsh, Laya

Decision Making Process: Do you need help?

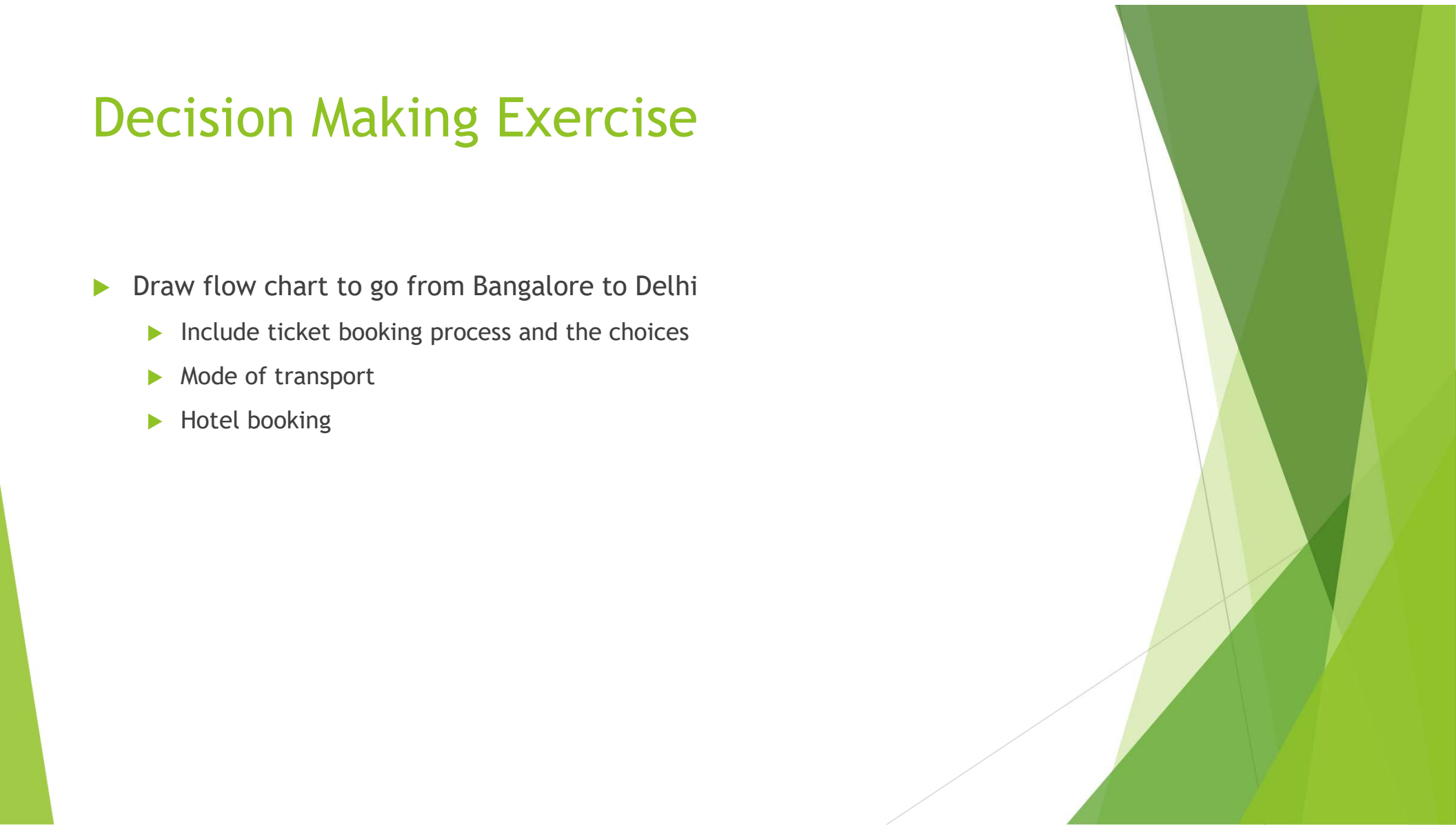



Start
Patient arrives
Registered patient
Yes
Available nurse
Yes
Record health condition
No
No
Register patient
Wait for available nurse
Available doctor
Yes
Assign patient to doctor
No
Wait for available doctor
Need follow up
Yes
Arrange appointment
No
Need Medication
Yes
Give patient prescription
No
Patient Leaves
Start


Start
Power On
Scan Envrionment
Generate Map and Location
Plan route
Route
Vacuum On
Follow route
Finished Route
No
Battery Low
No
Vacuum full
No
Yes
Yes
Yes
Vacuum Off
Vacuum Off
Return power dock
Stop moving
Power Off
Error indicator On
End

Decision Making Exercise

- Draw flow chart to go from Bangalore to Delhi
  - Include ticket booking process and the choices
  - Mode of transport
  - Hotel booking



Modular Design

- Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Modular Design: Components

- Modules: These are self-contained units within a program. Each module focuses on a specific task or functionality. By dividing the program into modules, we achieve better organization and maintainability.
- Module Interface: A module interface defines the elements (functions, variables, etc.) that the module provides and requires. Other modules can interact with a module through its interface.
- Implementation: The implementation of a module contains the actual working code corresponding to the elements declared in the interface.

Modular Design: Benefits

- Code Reusability: Modules can be reused across different projects, saving development time.
- Ease of Maintenance: Isolating functionality in modules simplifies debugging and updates.
- Scalability: New features can be added by creating new modules without affecting existing ones.

Modular Design: Top Down Approach

- If we look at problem as a whole, it may seem impossible to solve because it may be too complex:
  - Writing a tax computation program
  - Writing a word processor program
- Complex problems can be solved using top-down design also known as step wise refinement
  - We break the problem in to multiple functional block
  - Then we break the blocks in to further blocks
  - As we continue, soon each of the parts will be easy to design

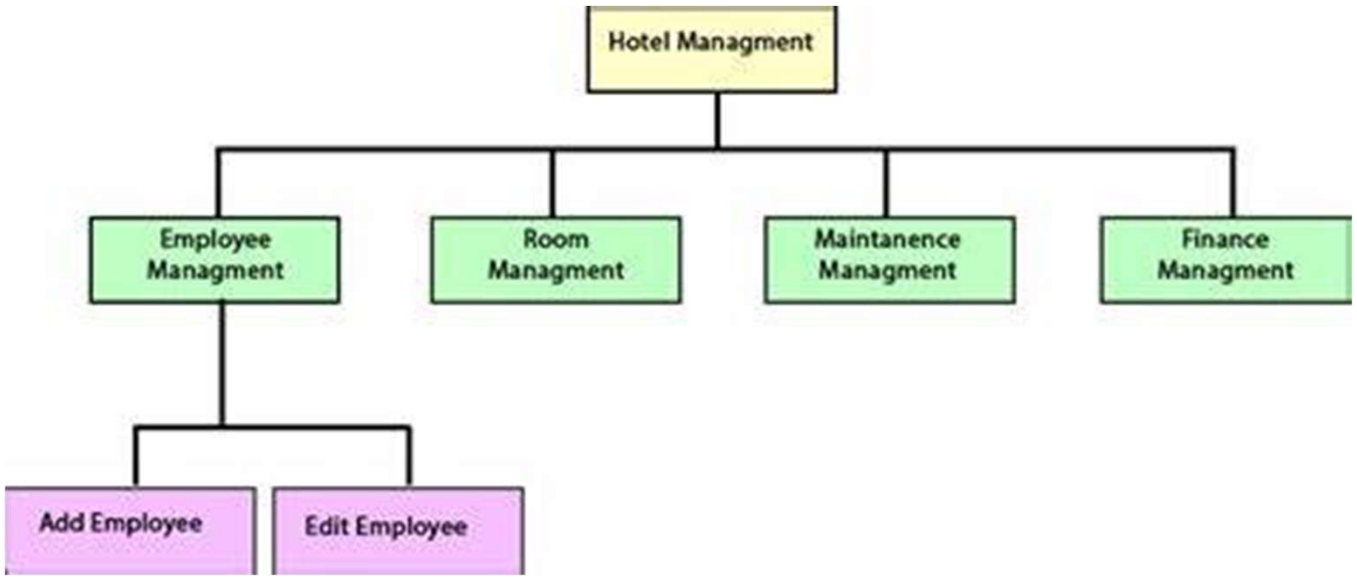
Top-Down Approach: Advantages

- Breaking problems into parts help us to identify what needs to be done.
- At each step of refinement, new parts will become less complex and therefore easier to solve.
- Parts of the solution may turn out to be reusable.
- Breaking problems into parts allows more than one person to solve the problem.

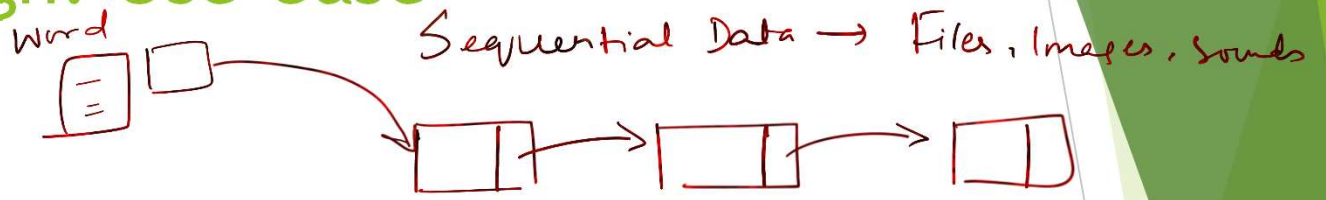
Top Down: Applications

- Tax Computation Program: Imagine you're tasked with creating a program that calculates income tax for individuals.
- Top-Down Design Steps:
  - High-Level Overview: Begin by outlining the main features of the tax computation program, such as inputting income, applying tax rules, and generating the final tax amount.
  - Refinement: Break down each feature into smaller components. For instance:
    - Input Module: Design how users will input income data.
    - Tax Calculation Module: Define the rules for calculating tax based on income brackets.
    - Output Module: Specify how the final tax amount will be displayed.
  - Further Refinement: Continue refining each module until you have detailed specifications for all components.
  - Integration: Combine the modules to create the complete tax computation program

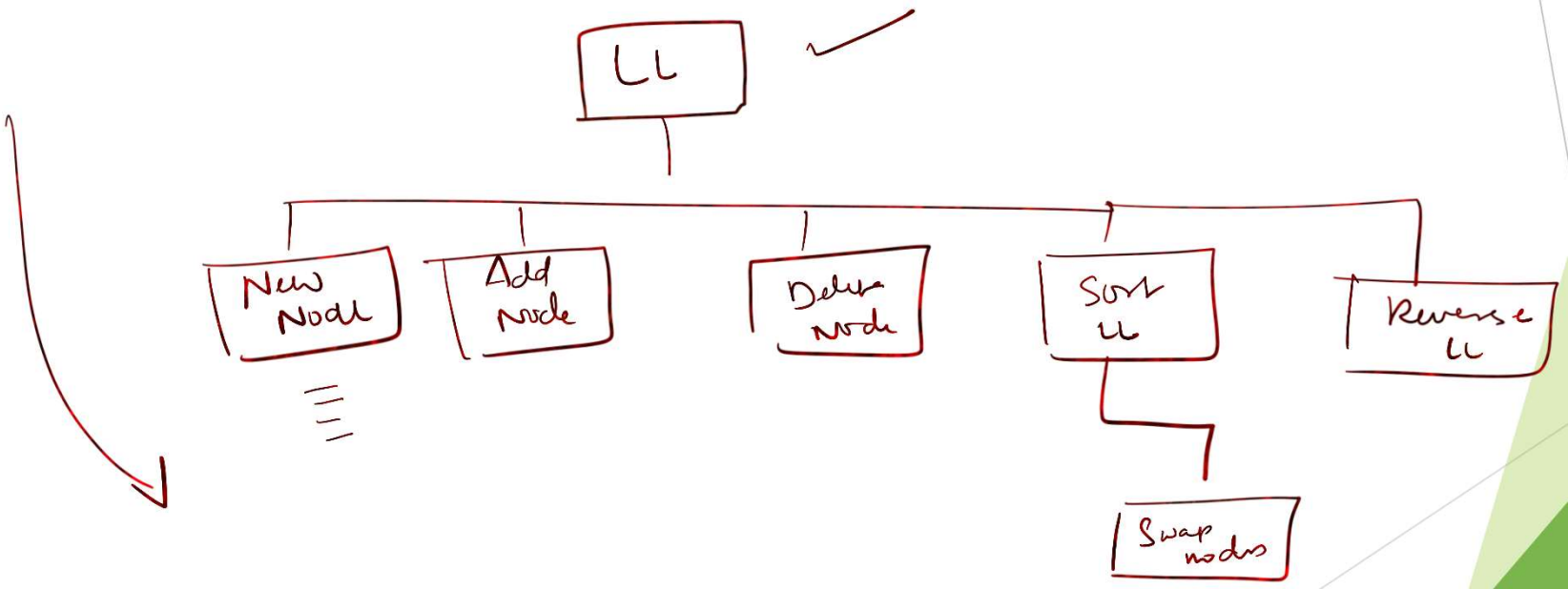
Modular Design: Top Down Approach



Modular Design: Use Case



- Designing and Managing a Linked List



Modular Design: Bottom Up Approach

- In bottom-up design, we begin with the basic building blocks—the smallest, fundamental components.
- These elements are then linked together to form subsystems.
- Subsystems are further connected, sometimes across multiple levels, until the complete top-level system emerges.
- Advantages: Make decisions about reusable low-level utilities then decide how there will be put together to create high-level construct. ,

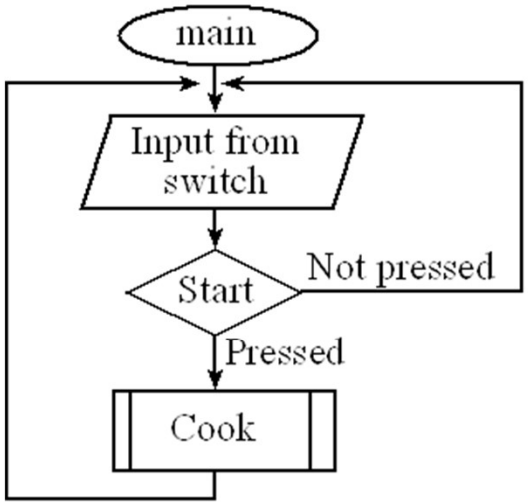
Modular Design: Bottom Up Approach

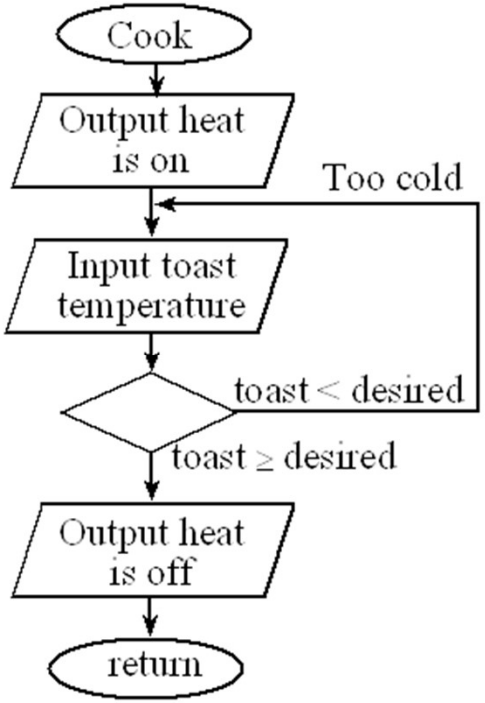
- Incremental Construction: Like growing from a seed, bottom-up design starts small and progressively becomes more complex.
- Local Optimization: However, this organic growth may lead to a tangle of elements, optimized locally without a global purpose.
- Contrast:
  - Top-down design begins with the big picture and breaks it down into smaller segments.
  - In contrast, bottom-up design starts with the details and assembles them into a cohesive whole.

Bottom Up: Applications

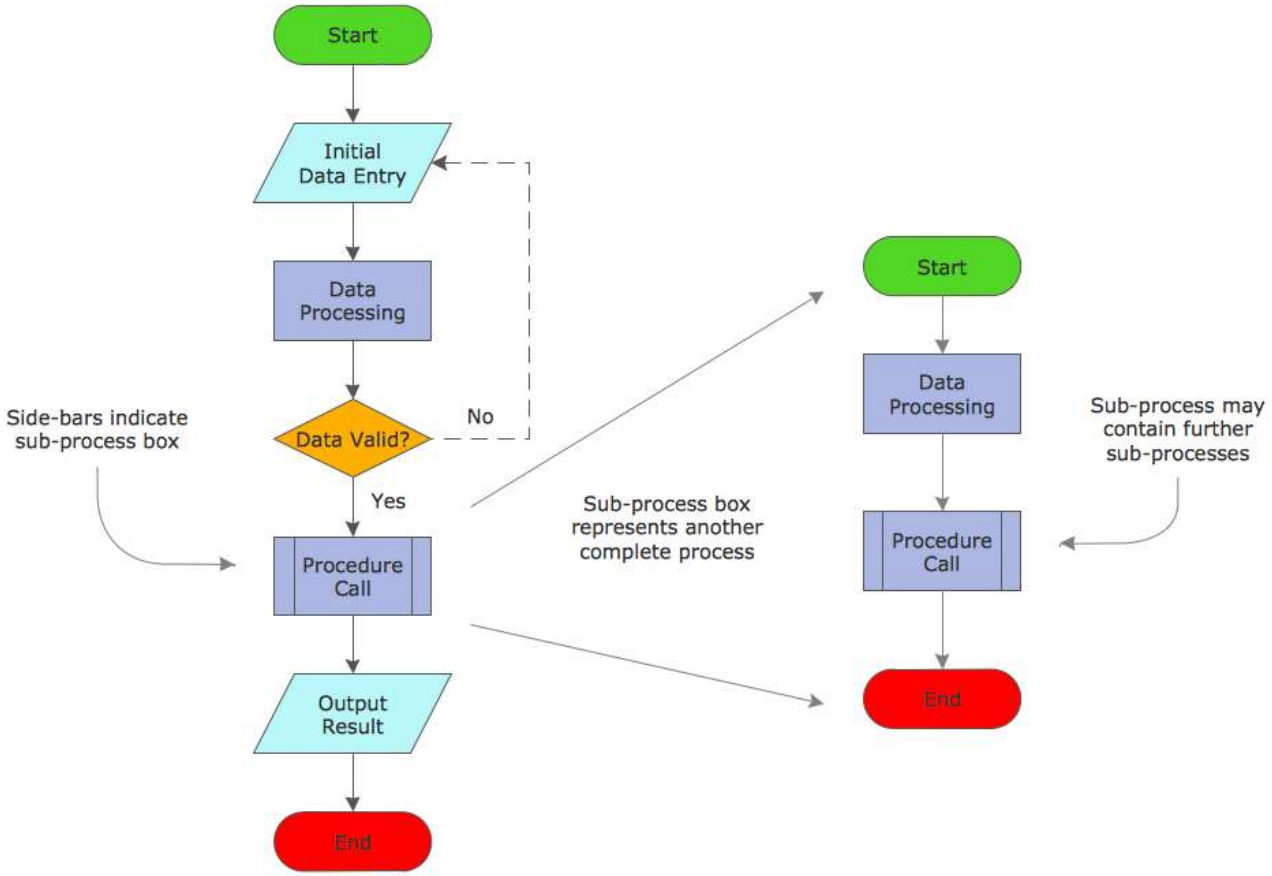
- Software Engineering: Bottom-up design is commonly used in software development, especially when integrating existing components or libraries.
- Product Design: Engineers and designers also apply both bottom-up and top-down approaches during product development

Flow Chart for Modular Designs

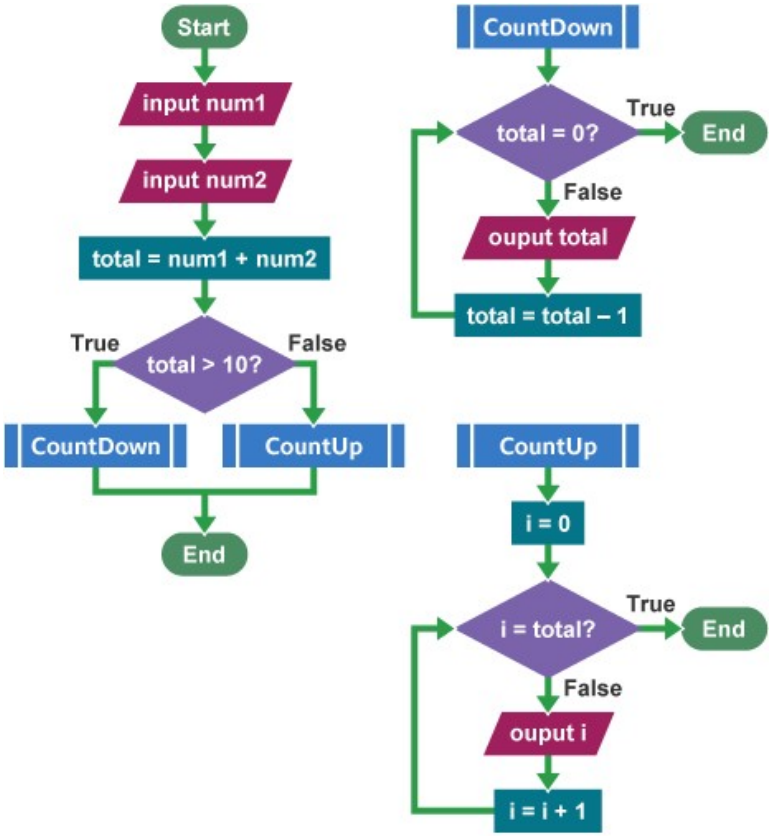




Sub-routines in Flow Charts



Sub-routines in Flow Charts



Class Assignment

S1 = [67, 78, 49, 68, 0, 0]
S2 = [87, 98, 79, 88, 0, 0]
S3 = [97, 68, 99, 58, 0, 0]

Requirement:

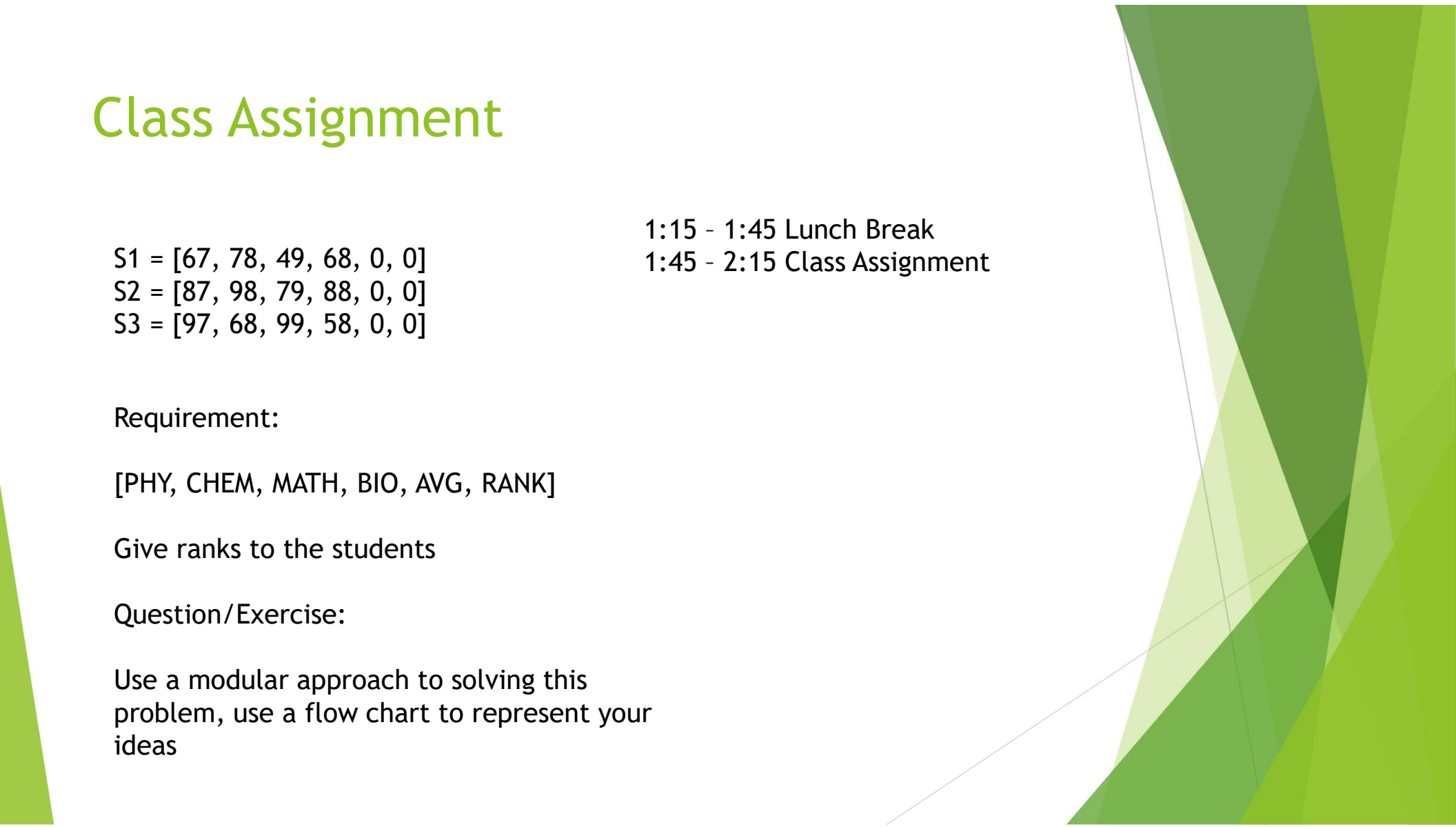
[PHY, CHEM, MATH, BIO, AVG, RANK]

Give ranks to the students

Question/Exercise:

Use a modular approach to solving this problem, use a flow chart to represent your ideas

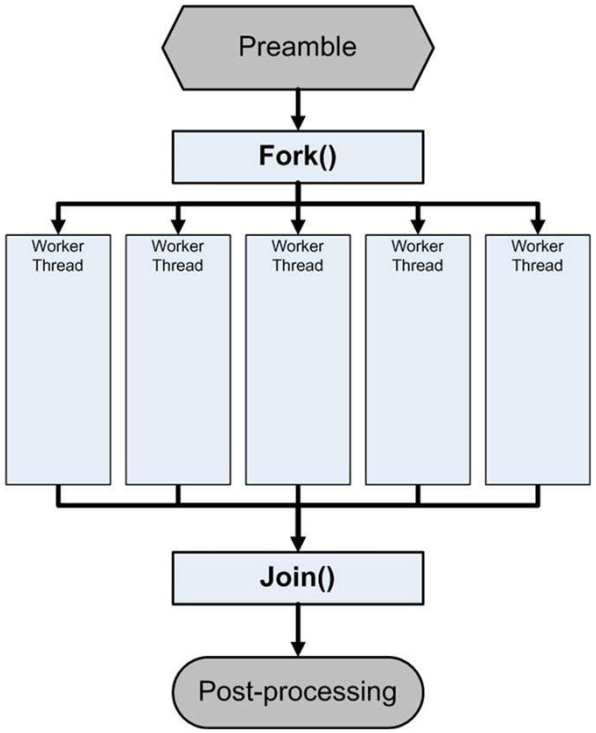
1:15 – 1:45 Lunch Break
1:45 – 2:15 Class Assignment



Concurrency and Parallelism

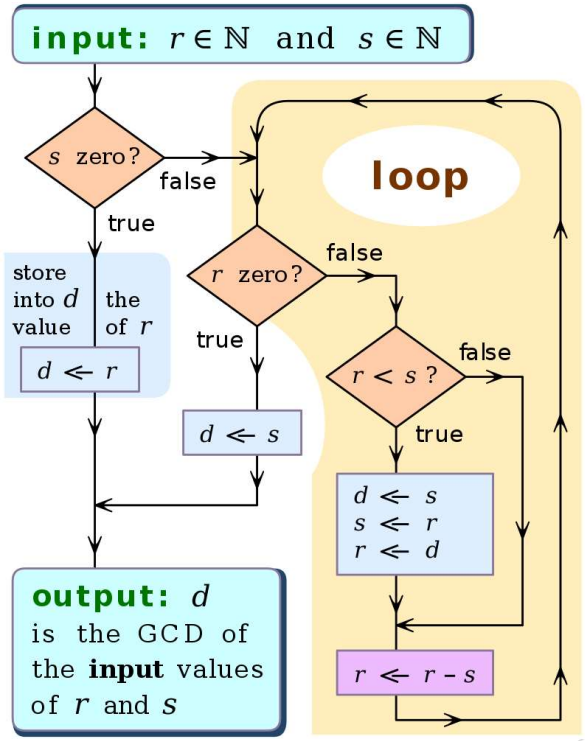
Features	Concurrency	Parallelism
Basic	It is the task of simultaneously running and managing numerous computations.	It is the task of executing numerous computations at the same time.
Achieve through	It is achieved via the interleaving of processes on the CPU.	It is achieved by the use of several processors.
Control flow	The non-deterministic control flow approach is called concurrency.	It is a deterministic control flow technique.
Processing Units Required	A single processing unit may be utilized to implement concurrency.	It may not be achieved with a single processing unit.
Usage	It deals with many things simultaneously.	It does multiple things simultaneously.
Make use of	It uses context Switching.	It uses multiple processors to run several processes.
Debugging	Debugging is very complex in concurrency.	Debugging is also complex in parallelism but simpler than concurrency.
Benefits	It increased the amount of work completed at a time.	It improved throughput and computational speed-up.
Examples	It executes several apps simultaneously.	It executes a web crawler on the cluster.

Flow Chart


Preamble
Fork()
Worker Thread
Worker Thread
Worker Thread
Worker Thread
Worker Thread
Join()
Post-processing

Algorithm

- A set of rules or instructions to be followed in calculations or other problem-solving operations - a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation.



Chareteristics

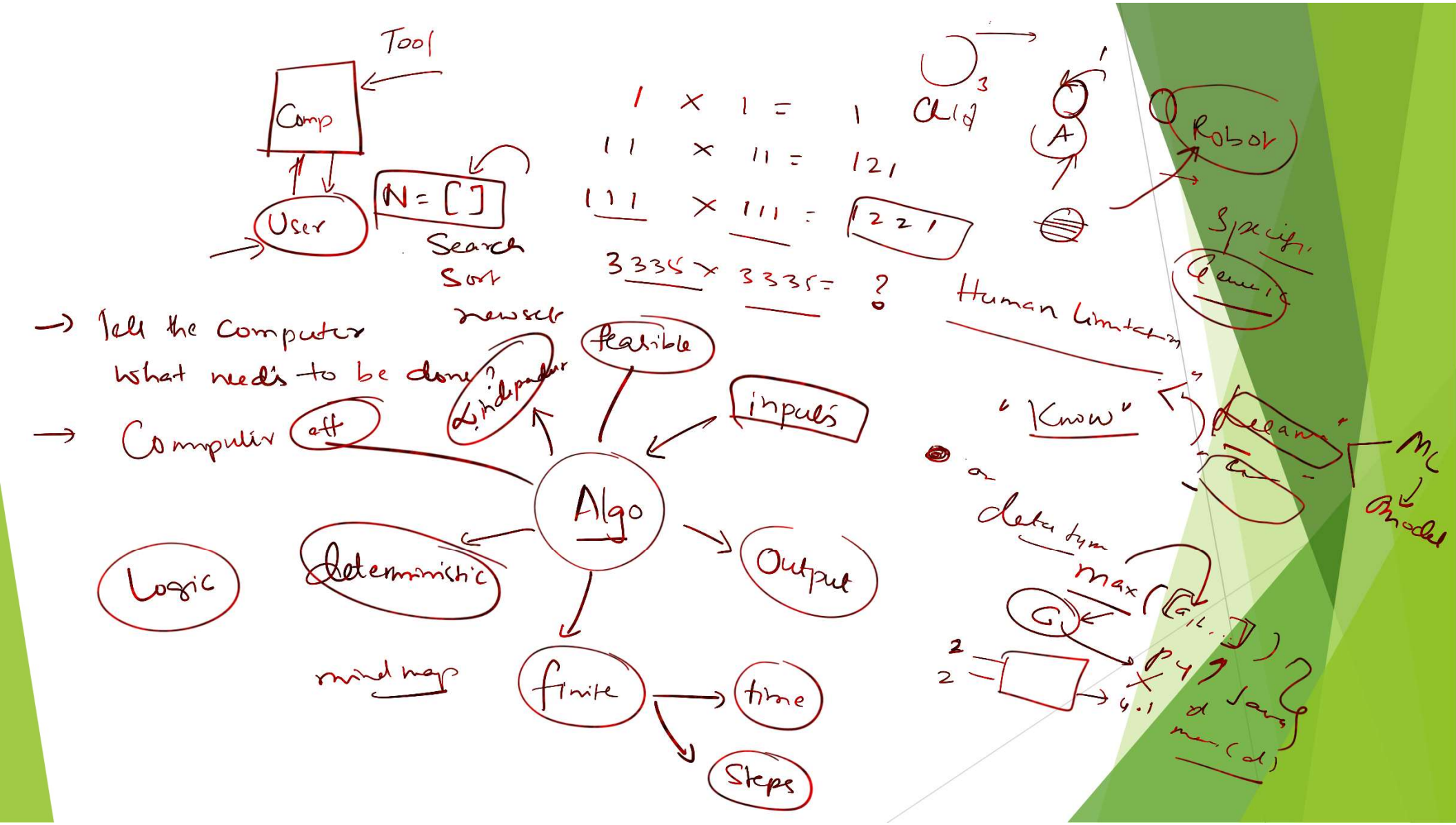
- Clear and Unambiguous: The algorithm should be unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- Well-Defined Inputs: If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- Well-Defined Outputs: The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- Finite-ness: The algorithm must be finite, i.e. it should terminate after a finite time.
- Feasible: The algorithm must be simple, generic, and practical, such that it can be executed with 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 the same, as expected.

Charecteristics

- Input: An algorithm has zero or more inputs. Each that contains a fundamental operator must accept zero or more inputs.
- Output: An algorithm produces at least one output. Every instruction that contains a fundamental operator must accept zero or more inputs.
- Definiteness: All instructions in an algorithm must be unambiguous, precise, and easy to interpret. By referring to any of the instructions in an algorithm one can clearly understand what is to be done. Every fundamental operator in instruction must be defined without any ambiguity.
- Finiteness: An algorithm must terminate after a finite number of steps in all test cases. Every instruction which contains a fundamental operator must be terminated within a finite amount of time. Infinite loops or recursive functions without base conditions do not possess finiteness.
- Effectiveness: An algorithm must be developed by using very basic, simple, and feasible operations so that one can trace it out by using just paper and pencil.

Properties

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work.


Tool
Comp
User
N = [ ]
Search
Sort
1 × 1 = 1
11 × 11 = 121
111 × 111 = 1221
3335 × 3335 = ?
Child
A
Robot
Specific
Human Limitation
→ Tell the computer what needs to be done
→ Compiler
Algo
Logic
Deterministic
Output
Steps
finite
time
mind map
feasible
input
"Know"
Learn
ML
Model
data type
max
Java

Assignment

- Flowchart Creation - Design a flowchart that outlines the logic for a user login process.
- It should include conditional paths for successful and unsuccessful login attempts, and a loop that allows a user three attempts before locking the account.

Assignment

- Function Design and Modularization - Create a document that describes the design of two modular functions:
  - one that returns the **factorial of a number**, and another that calculates the nth Fibonacci number.
  - Include pseudocode and a brief explanation of how modularity in programming helps with code reuse and organization.