CSC 102
LECTURE 1
Introduction to the core concepts of computing: Problems and problem-solving.

PROF ARIBISALA & DR. ZUBAIR ADAM

LAGOS STATE UNIVERSITY, OJO LAGOS

OVERVIEW

- Introduction to the core concepts of computing: Problems and problem-solving.
- The identification of problems and types of problems (routine problems and non-routine problems).
- Method of solving computing problems (introduction to algorithms and heuristics).
- Solvable and unsolvable problems.
- Solution techniques of solving problems (abstraction, analogy, brainstorming, trial and error, hypothesis
- Testing, reduction, literal thinking, means end analysis, method of focal object, morphological analysis, research, root cause analysis, proof, divide and conquer).
- General Problem-solving process.
- Solution formulation and design: flowchart, pseudocode, decision table, decision tree. Implementation, evaluation and refinement.
- Programming in C++, Python etc

Computing

- **What is Computing?**
  - **Definition**: The process of using computer technology to complete a task. It involves the design, development, and use of computer systems, software, and networks for the processing of data and information.
- **Key Areas**:
  - **Computer Science**: Theoretical foundations, algorithms, and computing systems.
  - **Information Technology**: Practical applications and systems management.
  - **Software Engineering**: Development and maintenance of software.
  - **Data Science**: Extraction of insights and knowledge from data using statistics, machine learning, and data visualization.
  - **Cybersecurity**: Protection of computer systems and networks from information disclosure, theft, or damage.

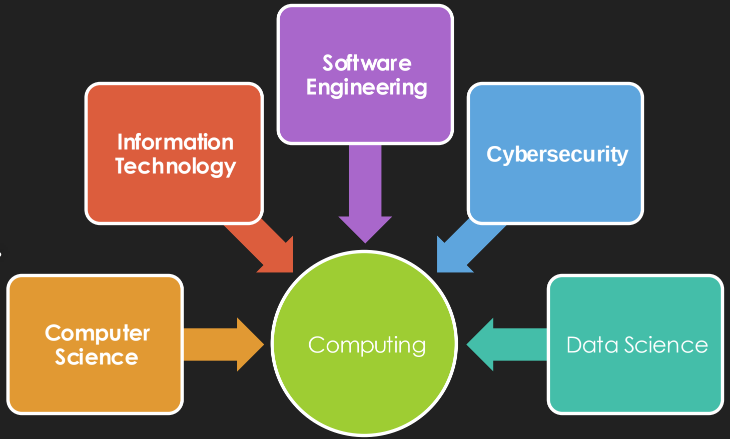


Figure 1: Key areas of Computing

History of Computing

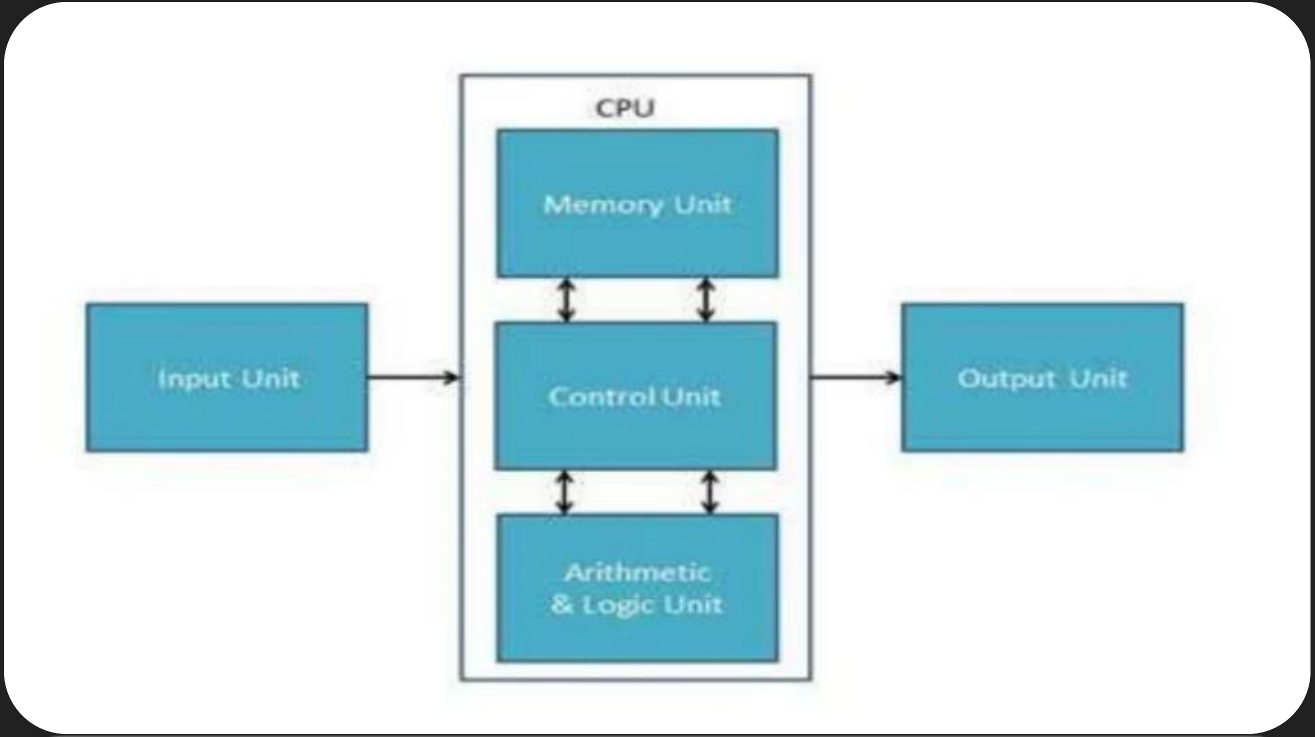
- Evolution of Computers
  - **Mechanical Devices**: Abacus, mechanical calculators (Pascaline, Leibniz wheel)
  - **Electromechanical Devices**: IBM Mark I, relay-based machines
  - **Electronic Computers**: ENIAC, UNIVAC
  - **Modern Digital Systems**: Microprocessors, personal computers, smartphones
- Key Milestones and Pioneers
  - **Charles Babbage**: Concept of the Analytical Engine
  - **Alan Turing**: Turing machine, father of theoretical computer science
  - **John von Neumann**: von Neumann architecture
  - **Bill Gates & Steve Jobs**: Personal computing revolution

History of Computer
Generations of computers

S/N	Generation	Component Used
1	First Generation (1946-1954)	Vacuum tubes
2	Second Generation (1955-1965)	Transistors
3	Third Generation (1968-1975)	Integrated Circuits (IC)
4	Fourth Generation (1976-1980)	Very Large Scale Integrated Circuits (VLSI)
5	Fifth Generation (1980 – till today)	Ultra Scale Integrated Circuits (ULSI) Micro Processor (SILICON CHIP)

BASIC COMPUTER COMPONENTS

- All types of computers follow a same basic logical structure and perform the following five basic operations for converting raw input data into information useful to their users.
- Input Unit
- CPU (Central Processing Unit)
  - Memory Unit
  - Control Unit
  - Arithmetic and Logic Unit
- Output Unit



Hardware and Software

- **Difference Between Hardware and Software**
  - **Hardware**: Physical components of a computer
  - **Software**: is any set of instructions that tells the hardware what to do. It is what guides the hardware and tells it how to accomplish each task
- **Types of Software**
  - **System Software**: Operating systems, utility programs
  - **Application Software**: Word processors, spreadsheets, games
  - **Programming Languages**: C, Java, Python

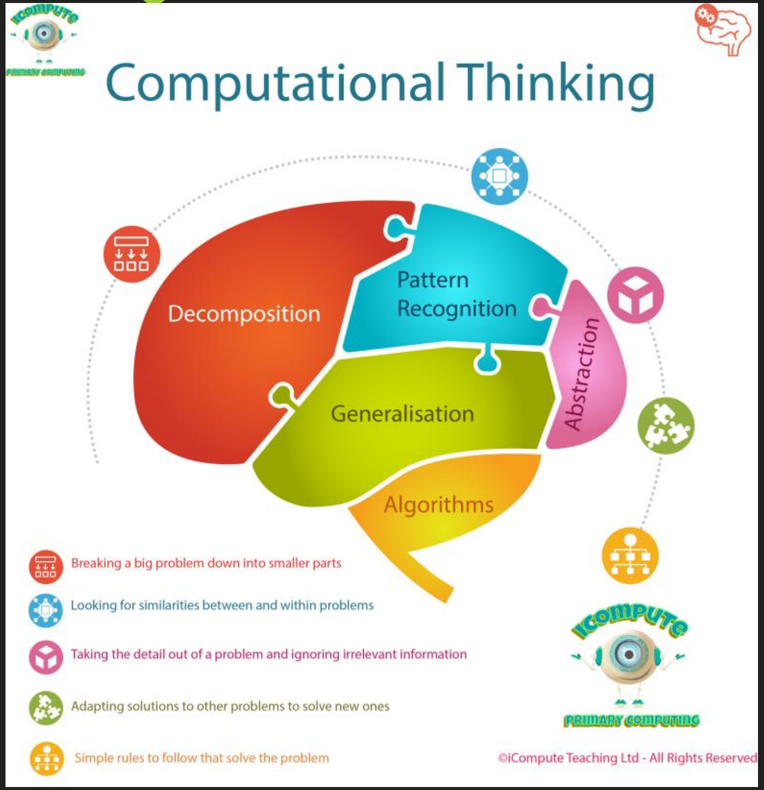
Computing Operations

- Computer is an electronic machine that performs the following four basic operations: –
  - Input
  - Process
  - Output
  - Store

COMPUTING ENTITIES

- **Data** : It is the collection of raw facts, figures & symbols.
  - Ex : Names of students and their marks in different subjects listed in random order.
- **Information** : It is the data that is processed & presented in an organized manner.
  - Ex : When the names of students are arranged in alphabetical order, total and average marks are calculated & presented in a tabular form, it is information.
- **Program** : Set of instructions that enables a computer to perform a given task.

Computational Thinking



iCompute
Primary Computing
Computational Thinking
Decomposition
Pattern Recognition
Abstraction
Generalisation
Algorithms
Breaking a big problem down into smaller parts
Looking for similarities between and within problems
Taking the detail out of a problem and ignoring irrelevant information
Adapting solutions to other problems to solve new ones
Simple rules to follow that solve the problem
©iCompute Teaching Ltd - All Rights Reserved

- **Computing focuses on using computer science techniques and devices to solve real world problems.**
- **Computational Thinking**: A problem-solving method that draws on concepts fundamental to computer science. It involves breaking down complex problems into smaller, manageable parts, recognizing patterns, abstracting essential details, designing algorithms, and analyzing the efficiency of solutions.

Tools for Problem Solving

- Programming Languages: Python, Java, C++
- Integrated Development Environments (IDEs): Visual Studio, PyCharm, Eclipse
- Version Control Systems: Git, GitHub, Bitbucket
- Debugging Tools: GDB, WinDbg
- Simulation Software: MATLAB, Simulink
- Data Analysis Tools: R, Pandas (Python), SQL
- Optimization Tools: Linear programming solvers (e.g., CPLEX, Gurobi)
- Visualization Tools: Tableau, Power BI, Matplotlib

Problem-Solving Strategies

- Decomposition: Breaking down complex problems into manageable parts
  - Example: Breaking down the task of creating a website into design, development, and testing phases.
- Pattern Recognition: Identifying similarities and patterns
  - Example: Recognizing patterns in data to detect fraud.
- Abstraction: Focusing on important information, ignoring irrelevant details
  - Example: Creating a simplified model of a car to focus on its fuel efficiency.
- Algorithm Design: Creating step-by-step solutions
  - Example: Designing an algorithm for sorting a list of numbers.
- Debugging: Identifying and fixing errors in code
  - Example: Fixing syntax errors in a Python script.
- Simulation and Modeling: Using models to represent and test solutions
  - Example: Using a simulation to predict weather patterns.
- Iteration: Repeating processes to refine and improve solutions
  - Example: Iteratively testing and improving a software application.

Applications of Computing

- It is safe to say that computing application in all works of life.

Healthcare	• Electronic health records, telemedicine, medical imaging
Finance	• Algorithmic trading, fraud detection, financial modeling
Education	• E-learning platforms, educational software, virtual classrooms
Entertainment	• Video games, streaming services, digital media
Transportation	• Autonomous vehicles, traffic management, logistics
Manufacturing	• Robotics, automation, supply chain management
Communication	• Social media, email, instant messaging
Science and Research	• Data analysis, simulations, computational biology

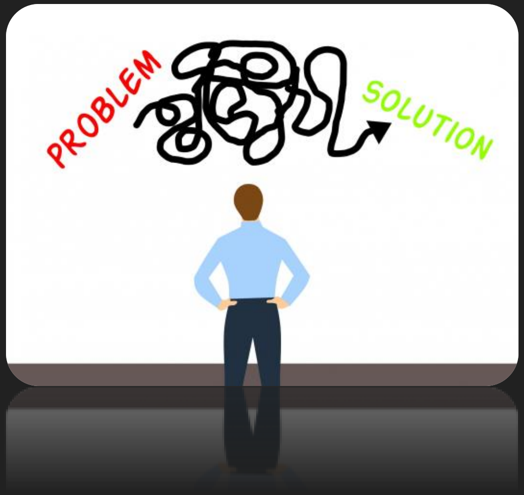
Problem-Solving in Computing

What is Problem-Solving?

- **Definition**: Problem-solving is the sequential process of analyzing information related to a given situation and generating appropriate response options.
- Problem-solving in computing involves using computational thinking, algorithms, and tools to analyze, design, and implement solutions to identified problems.

Key Aspects of Problem-Solving

- **Computational Thinking**: Applying principles like decomposition, pattern recognition, and abstraction to break down problems and devise efficient solutions.
- **Algorithm Design**: Creating step-by-step procedures or algorithms to solve specific tasks or problems.
- **Implementation**: Translating algorithms into code using programming languages and leveraging appropriate tools and technologies.
- **Evaluation and Optimization**: Testing, refining, and optimizing solutions based on performance metrics and user feedback.

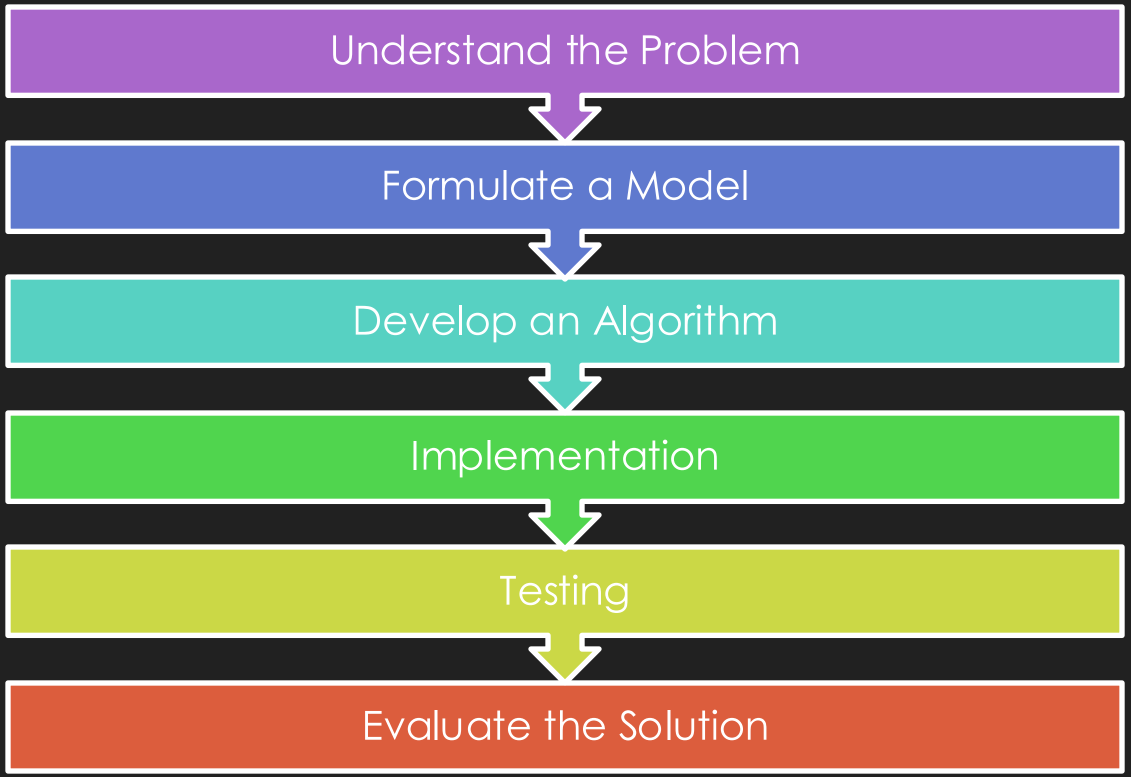


Computers as a problem solving tool

- Computer science is all about solving problems with computers (regardless of the area of study).
- Real-world or abstract world problems can be tackled with computer science.
- We need to have a standard systematic approach to solving problems.

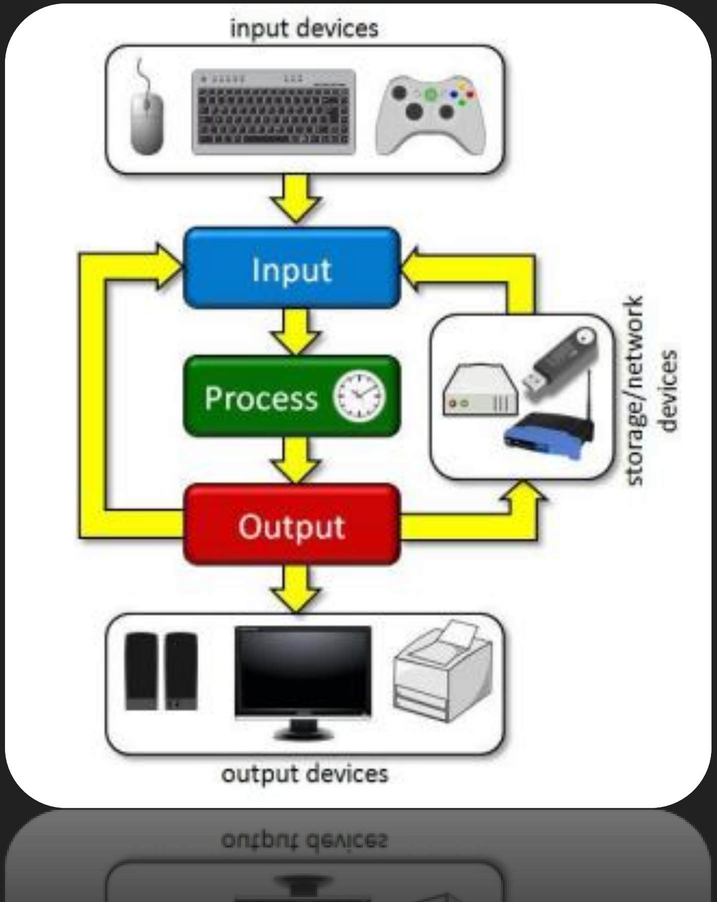
Systematic approach to solving problems

1. Understand the Problem
2. Formulate a Model
3. Develop an Algorithm
4. Implementation (Coding)
5. Test (the Program)
6. Evaluate the Solution



A problem scenario

- Consider a simple example of how the input/process/output works on a simple problem which requires calculating the average grade for all students in a class.
  1. Input: Get all the grades … perhaps by typing them in via the keyboard or by reading them from a USB flash drive or hard disk.
  2. Process: add them all up and compute the average grade.
  3. Output: Output the answer to either the monitor, to the printer, to the USB flash drive or hard disk …



Understand the Problem

- The first step to solving any problem is to make sure that you understand the problem that you are trying to solve. One needs to know:
  1. What input data/information is available?
  2. What does it represent?
  3. What format is it in?
  4. Is anything missing?
  5. Do I have everything that I need?
  6. What output information am I trying to produce?
  7. What do I want the result to look like text, a picture, a graph?
  8. What am I going to have to compute?

Understand the Problem

- **In our example**, we well understand that the input is a bunch of grades. But we need to understand the format of the grades.
- Each grade might be a number from 0 to 100 or it may be a letter grade from A+ to F. If it is a number, the grade might be a whole integer like 73 or it may be a real number like 73.42.
- We need to understand the format of the grades in order to solve the problem.
- We also need to consider missing grades. What if we do not have the grade for every student (e.g., some were away during the test) ? Do we want to be able to include that person in our average (i.e., they received 0) or ignore them when computing the average ?
- We also need to understand what the output should be. Again, there is a formatting issue. Should we output a whole or real number or a letter grade ? Maybe we want to display a pie chart with the average grade. It is our choice.
- Finally, we should understand the kind of processing that needs to be performed on the data. This leads to the next step

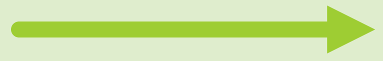
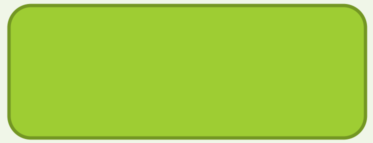
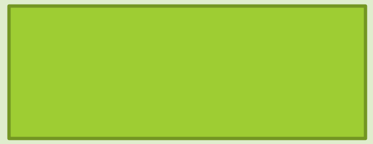
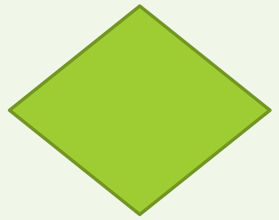
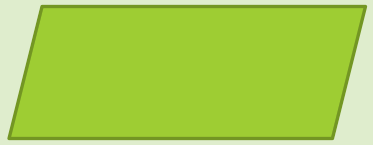
Formulate a model

- Now we need to understand the processing part of the problem.
- To achieve that we break down the problem into smaller problems that require some kind of simple mathematical computations to process the data.
- **In our example**, we are going to compute the average of the incoming grades. So, we need to know the model (or formula) for computing the average of a bunch of numbers. If there is no such “formula”, we need to develop one
- Assuming that the input data is a bunch of integers or real numbers x1,x2,…,xn representing a grade percentage, we can use the following computational model: Average = (x1 + x2 + x3 + … + xn) / n where the result will be a number from 0 to 100.

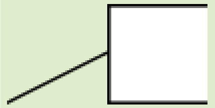
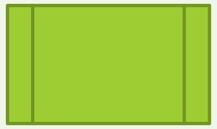
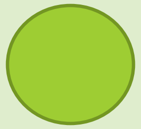
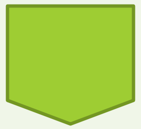
Develop an Algorithm

- Now that we understand the problem and have formulated a model, it is time to come up with a precise plan of what we want the computer to do.
- **An algorithm** is a precise sequence of instructions for solving a problem.
- To develop an algorithm, we need to represent the instructions in some way that is understandable to a person who is trying to figure out the steps involved. Two commonly used representations for an algorithm is by using
  - Pseudo code, or
  - Flow charts.
- **Pseudocode** is a simple and concise sequence of English-like instructions to solve a problem
- **Flowchart** is simply a graphical representation of steps in sequential order and is widely used in presenting the flow of algorithms.

Flowchart

ANSI/ISO Shape	Name	Decription
	Flowline (Arrowhead)	Shows the process's order of operation. A line coming from one symbol and pointing at another. Arrowheads are added if the flow is not the standard top-to-bottom, left-to right.
	Terminal	Indicates the beginning and ending of a program or sub-process. Represented as an oval or a rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
	Process	Represents a set of operations that changes value, form, or location of data. Represented as a rectangle.
	Decision	Shows a conditional operation that determines which one of the two paths the program will take. The operation is commonly a yes/no question or true/false test. Represented as a diamond (rhombus).
	Input/Output	Indicates the process of inputting and outputting data, as in entering data or displaying results. Represented as a rhomboid.

Flowchart

ANSI/ISO Shape	Name	Decription
	Annotation (Comment)	Indicating additional information about a step in the program. Represented as an open rectangle with a dashed or solid line connecting it to the corresponding symbol in the flowchart.
	Predefined Process	Shows named process which is defined elsewhere. Represented as a rectangle with double-struck vertical edges.
	On-page Connector	Pairs of labeled connectors replace long or confusing lines on a flowchart page. Represented by a small circle with a letter inside.
	Off-page Connector	A labeled connector for use when the target is on another page. Represented as a home plate-shaped pentagon.

Pseudocode Vs Flowchart

Consider the following example (from Wikipedia) of solving the problem of a broken lamp. To the right is an example of a flow chart, while to the left is an example of pseudocode for solving the same problem:

Pseudocode

1. IF lamp works, go to step 7.
2. Check if lamp is plugged in.
3. IF not plugged in, plug in lamp.
4. Check if bulb is burnt out.
5. IF blub is burnt, replace bulb.
6. IF lamp doesn't work buy new lamp.
7. Quit ... problem is solved.

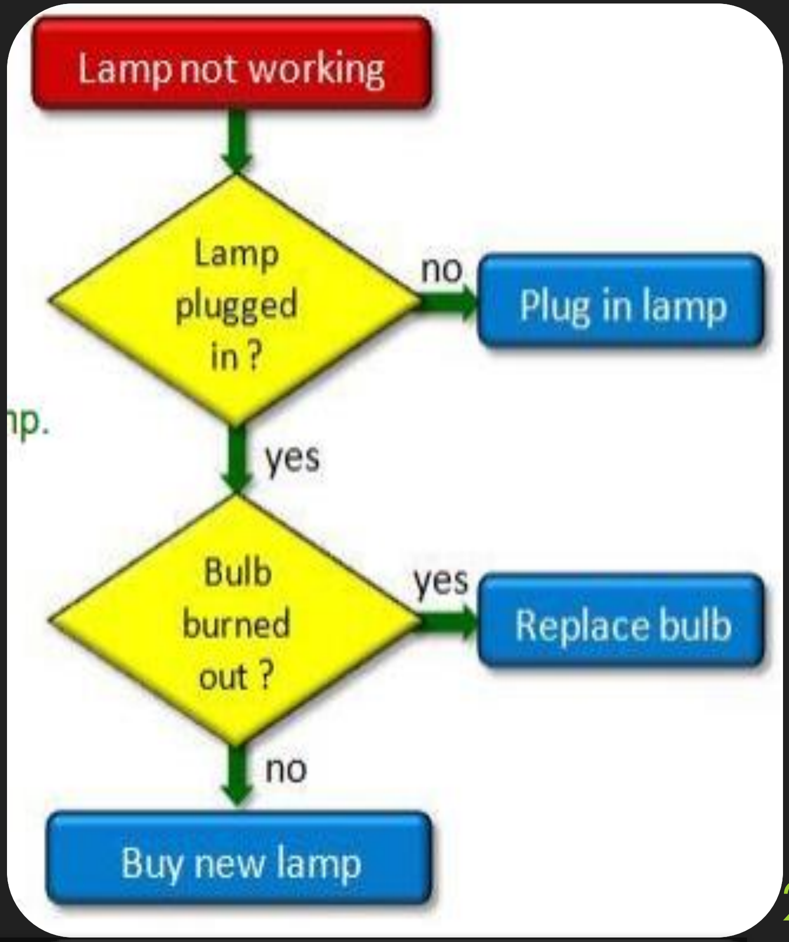


Figure 1: Flowchart

Implementation (Write the Program)

- Writing a program is often called "writing code" or “implementing an algorithm”.
- The code (or source code) is the program itself.
- The code is written by using a programming language that is best suited for the job.
- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.
- Some examples of programming language are BASIC, C, C++, COBOL, Java, FORTRAN, Ada, Python (etc).

Algorithm (Pseudocode) to Program

S/N	Pseudocode	Processing code (i.e., program)
1.	set the sum of the grade values to 0. int sum = 0;	int sum = 0;
2.	load all grades x1 … xn from file.	byte[] x = loadBytes("numbers");
3.	repeat n times{	for (int i=0; i <x.length; i++)
4.	get grade x_i add x_i to the sum }	sum = sum + x[i];
5.	compute the average to be sum / n.	int avg = sum / x.length
6.	print the average.	print(avg);

Test the Program

- Once you have a program written that compiles, you need to make sure that it solves the problem that it was intended to solve and that the solutions are correct.
- Therefore the program is run in order to evaluate the compiled instructions.
- After running the program, if all is well, then a correct output displayed.
- It is possible however, that the program works correctly for some set of data input but not for all.
- If the output of the program is incorrect,
  - it is possible that the algorithm was not converted properly into a proper program.
  - It is also possible that the algorithm itself is flawed.
  - Or some instructions where written and performed out of sequence.
- Whatever happened, such problems with your program are known as bugs.

Test the Program

- ***Bugs*** are problems/errors with a program that cause it to stop working or produce incorrect or undesirable results.
- One should fix as many bugs in ones program as possible.
- To find bugs effectively, the program must be evaluated with many test cases (called a test suite).
- It is also a good idea to have a third-party test ones program because they may think up situations or input data that the writer may never have thought of.
- The process of finding and fixing errors in your code is called ***debugging*** and it is often a very time-consuming "chore" when it comes to being a programmer.
- If one takes time to carefully follow problem solving steps 1 through 3, this should greatly reduce the amount of bugs in ones programs and it should make debugging much easier.

Discussion