

Data Structures

IT 205

Dr. Manish Khare



Lecture - 1

About Me



Dr. Manish Khare

PhD (Computer Science)

Research Interests

Image Processing

Computer Vision

Infrared Imaging

Surveillance System

Contact



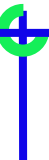
Manish Khare

Room No.: Faculty Block 4/ 4204

Extn No.: 619

Mail: manish_khare@daiict.ac.in

Introduction



➤ Course Name: Data Structures

➤ Course Id: IT205

➤ Credits: 3(L)-0(T)-0(P)-3(C)

➤ Related Course:

- Data Structure Lab (IT206)

Schedule

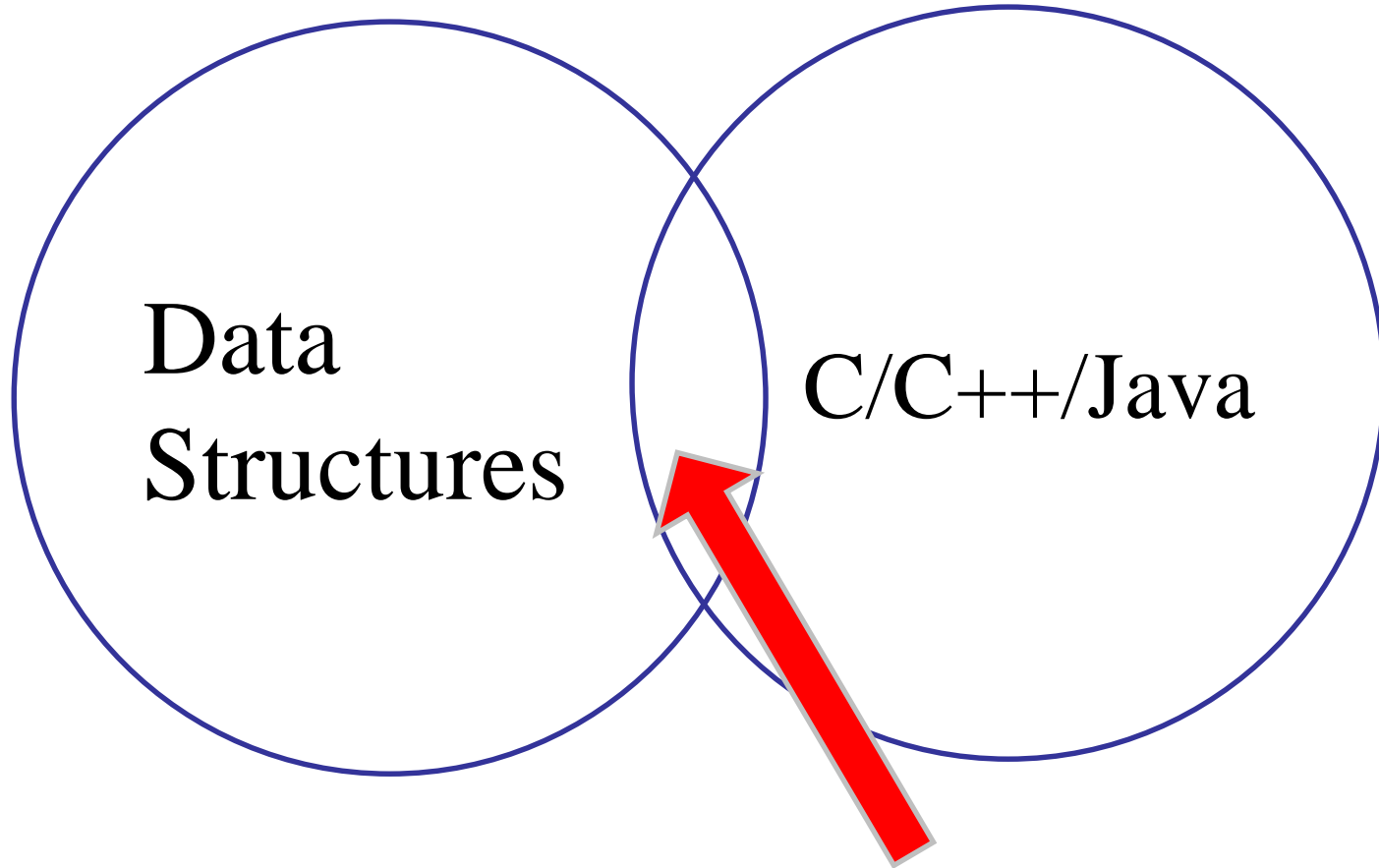


➤ Lectures

- Tuesday (9:00-9:55)
- Thursday (12:00-12:55)
- Friday (10:00-10:55)

➤ @LT-3

About the Course



Data Structures

Tentative Course Outline

➤ **Module 1: Preliminaries, Analysis Tools, Dictionary Operations (10)**

- Representation of data on a computer, data types & array and linked list representations ways of representing programs and associated data on computers,
- Notion of the running time of an algorithm, Recurrences, Parameters of performance,
- Find, Max, Min, Successor, Predecessor (query operations); Insert, Delete (modify operations)

Tentative Course Outline

➤ **Module 2: List Data, Sorting, order statistics (13)**

- Stacks, queues, variants implementation using arrays and linked lists,
- Comparison based sorting algorithms, Other sorting algorithms, lower bounds for comparison-based sorting algorithms Best-case, worst-case and average-case running times. Quicksort, Heap Sort, insertion sort, bubble sort etc.
- Maximum and minimum elements of a set, Finding median, searching for an element of a given rank, finding the rank of a given element, ranks of a subset of elements, Maintaining rank information for a dynamic set

Tentative Course Outline



➤ Module 3: Tree and Graph (17)

- Heaps, Binary search trees (BST) heights of BST,
- Red Black trees, AVL Trees, 2,3,4-trees, B Trees,
- Representation using adjacency matrices and adjacency lists
Graph searching algorithms BFS and DFS, Spanning tree

Text and References

- Fundamental of Data Structures: E. Horowitz, S. Sahani
- Fundamental of Data Structures in C: E. Horowitz, S. Sahani, A. Freed
- Fundamental of Data Structures in C++: E. Horowitz, S. Sahani, D. Mehta
- Data Structures, Algorithms, and Applications in Java: S. Sahani
- Data Structures, Algorithms and Applications in C++: S. Sahani
- Data Structures using C and C++: Y. Langsam, M.J. Augenstein, A. Tenenbaum
- Fundamentals of Computer Algorithms: E. Horowitz, S. Sahani, S. Rajasekaran
- Introduction to Algorithm: T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein

Course Folder



Lecture Folder:

\\10.100.56.21\Lecture\Manish Khare\IT205 - Data
Structures

Components and Evaluation

Sl. No	Component	Weightage
1	Quiz / Assignment /Attendance	20%
2a	In Sem I	15%
2b	In Sem II	15%
2c	End Sem	50%


Some Important Points

- NO mobile phones / laptops Use in Class
- Be Punctual
- Be Regular, Avoid Backlog
- Attendance – class
- timely submission of assignments
- NO late submissions will be allowed
- NO repeat quiz, in-semester exams
- Suggestions and feedback welcome



Questions???

Disclaimer !

- 
- This course will not teach you about
 - Any specific software development technologies like .NET, Java, C, C++
 - The course assumes students exposed to C programming
 - Learn concepts and then problem solving...

Data Structures



➤ What is Data Structures?

What is Data Structures?

- A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.
- Data Structure is a systematic way to organize data in order to use it efficiently. Following terms are the foundation terms of a data structure.
- Interface – Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
- Implementation – Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure

Data Structures



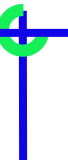
➤ What is Data Structures?

➤ Characteristics of a Data Structures?

Characteristics of a Data Structures?

- **Correctness** – Data structure implementation should implement its interface correctly.
- **Time Complexity** – Running time or the execution time of operations of data structure must be as small as possible.
- **Space Complexity** – Memory usage of a data structure operation should be as little as possible.


Data Structures

- 
- What is Data Structures?
 - Characteristics of a Data Structures?
 - Why do we need to learn data structures?

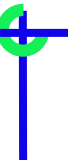
Why do we need to learn data structures?

- Programs are comprised of two things: data and algorithms. The algorithms describe the way the data is to be transformed. The reason for learning about data structures is because adding structure to our data can make the algorithms much simpler, easier to maintain, and often faster.
- As applications are getting complex and data rich, there are three common problems that applications face now-a-days.
- **Data Search** – Consider an inventory of 1 million (10^6) items of a store. If the application is to search an item, it has to search an item in 1 million (10^6) items every time slowing down the search. As data grows, search will become slower.
- **Processor speed** – Processor speed although being very high, falls limited if the data grows to billion records.
- **Multiple requests** – As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

Data Structures

- 
- What is Data Structures?
 - Characteristics of a Data Structures?
 - Why do we need to learn data structures?
 - What is Data Types?

What is Data Types?



➤ A data type (in computing terms) is a set of data values having predefined characteristics. You will have encountered data types during computer programming classes. Example data types would be integer, floats, string and characters. Generally, in all programming languages we have a limited number of such data types. The range of values that can be stored in each of these data types is defined by the language and the computer hardware that you are using the high level language on.

➤ **Scalar Data types**


A simple (scalar) data type consists of a collection of ordered values and a set of operations that can be performed on those values. The C, C++ and Java programming languages refer to scalar types as primitive data types.

A scalar variable can hold only one piece of data at a time. So in C, C++ and Java scalar data types include int, char, float and double, along with others. Scalar variables of the same type can be arranged into ascending or descending order based on the value.

➤ **Structured data types**

Structured data types hold a collection of data values. This collection will generally consist of the primitive data types. Examples of this would include arrays, records (structs), classes and files. These data types, which are created by programmers, are extremely important and are the building block of data structures. So what exactly is a data structure?

Data Structures

- 
- What is Data Structures?
 - Characteristics of a Data Structures?
 - Why do we need to learn data structures?
 - What is Data Types?
 - Basic Terminology in Data Structures?

Basic Terminology in Data Structures?

- **Data** – Data are values or set of values.
- **Data Item** – Data item refers to single unit of values.
- **Group Items** – Data items that are divided into sub items are called as Group Items.
- **Elementary Items** – Data items that cannot be divided are called as Elementary Items.
- **Attribute and Entity** – An entity is that which contains certain attributes or properties, which may be assigned values.
- **Entity Set** – Entities of similar attributes form an entity set.
- **Field** – Field is a single elementary unit of information representing an attribute of an entity.
- **Record** – Record is a collection of field values of a given entity.
- **File** – File is a collection of records of the entities in a given entity set.



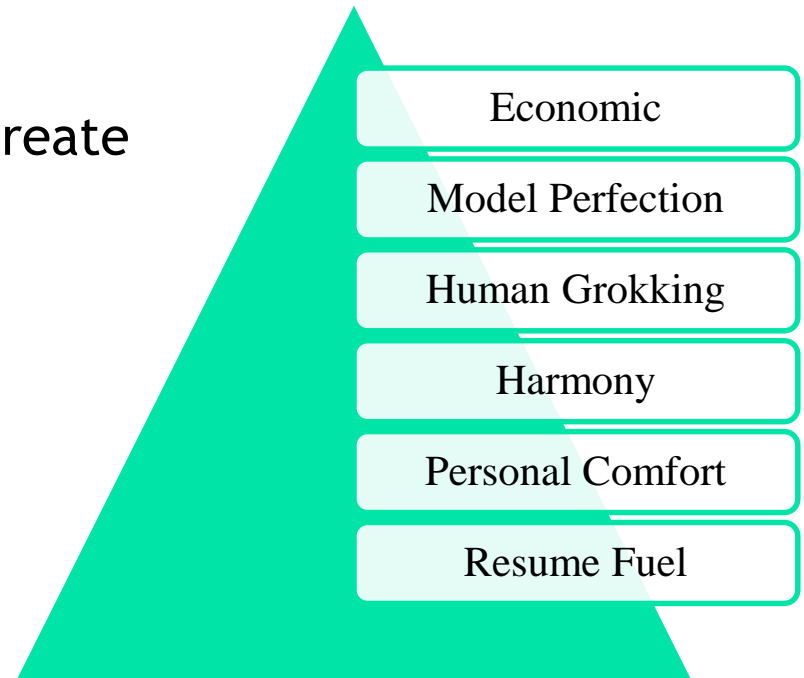
*What is the Purpose of | Why we do
Programming?*

Arguments...

The purpose of programming is to automate processes

The practice of programming is one of teaching the computer to do something

The purpose of programming is to create



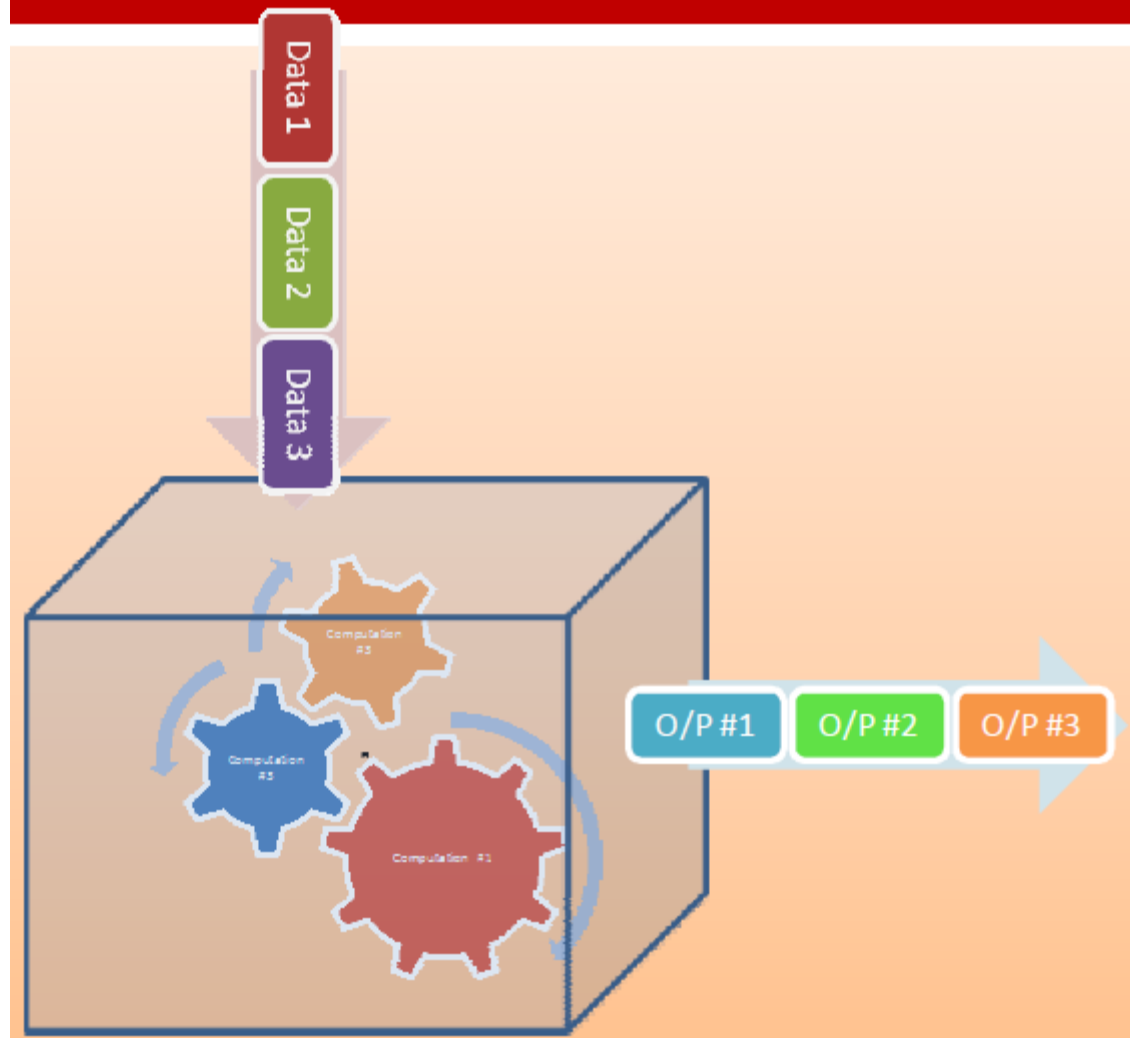
Computer Engineering is about...

Solving problems by **writing programming solutions**

- Not just any solution but the best possible solution of the problem
 - a. Characteristics of the best solution
 - b. Efficient (performance or speed)
 - c. Cost effective
 - d. Scalable
 - e. Useful
 - f. Correct
 - g. Robust
 - h. Secure
 - i. Modifiable (change)

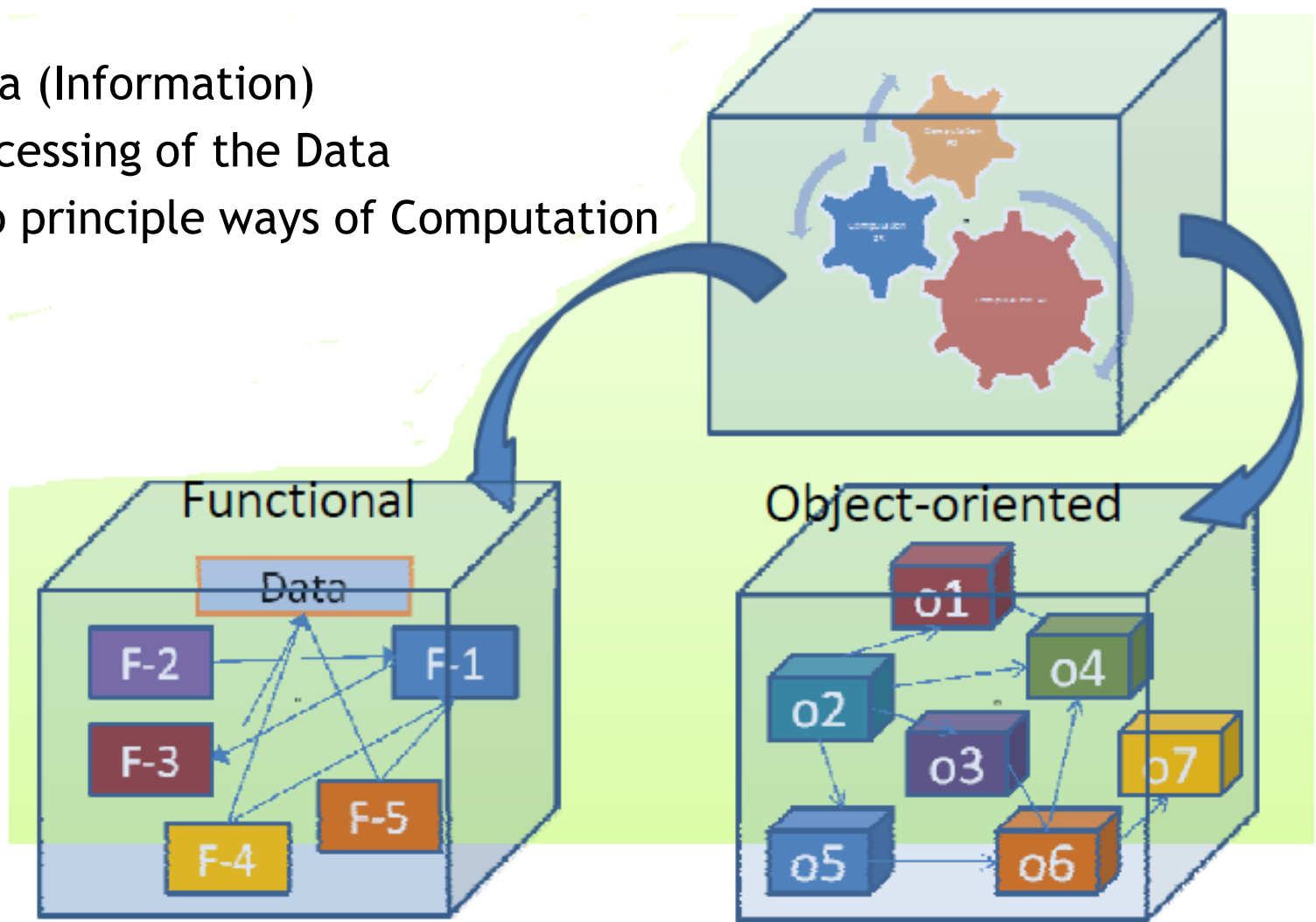
A Programming Solution

- Inputs
 - Keyboard (user)
 - Mouse (user)
 - Network (system)
 - Devices (system)
 - Other programs
- Computations
- Outputs
 - Monitor
 - Network
 - Devices
 - Other programs
- GUI



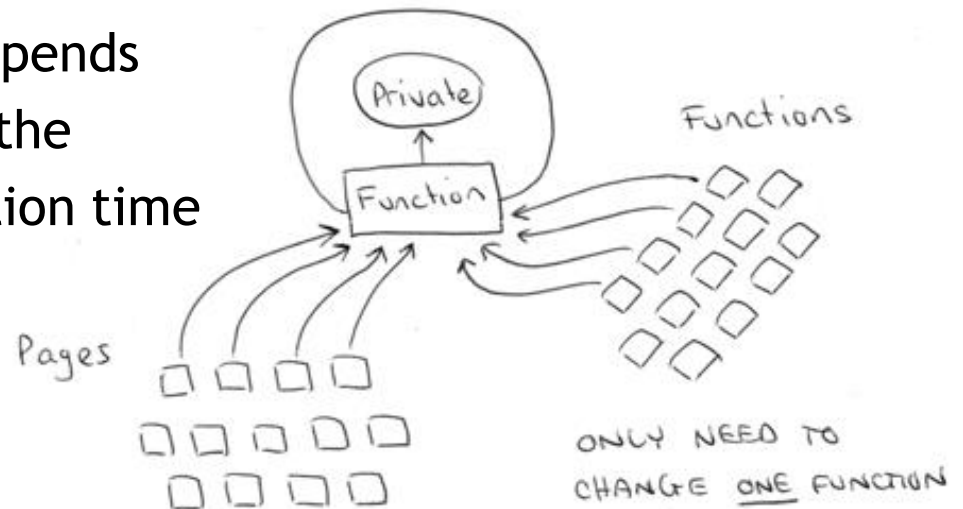
Computation

- Data (Information)
- Processing of the Data
- Two principle ways of Computation



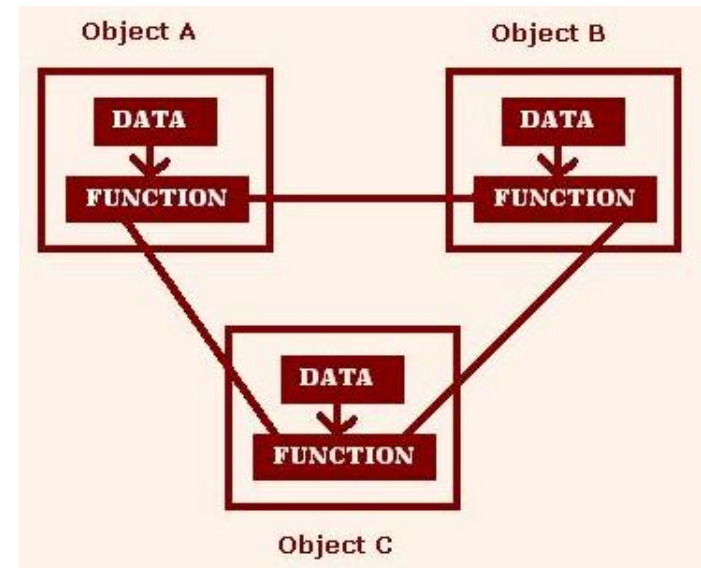
Functional

- Basic elements of functional programming are:
- Variables in the mathematical sense (named values)
- Functions in the mathematical sense (mapping values to values)
- The result of a function depends only on its arguments and the Context at function definition time



Object-Oriented

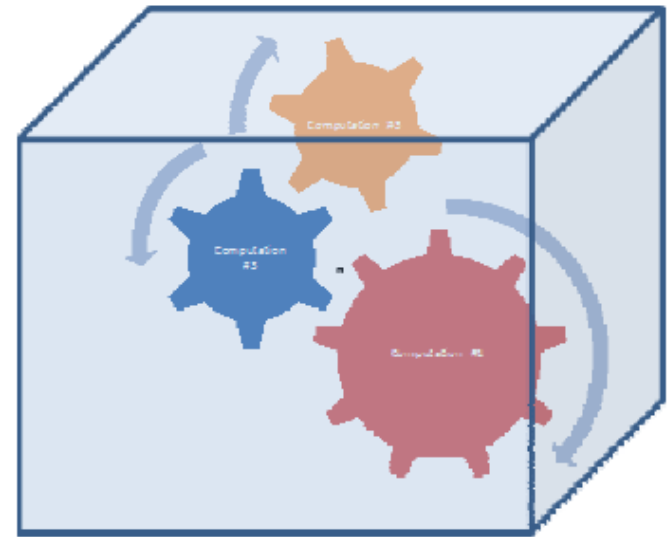
- Basic elements of Object-Oriented Programming (OOP) are:
- Objects (data) with - typically - non-constant state
- Variables are storage locations in memory that may hold different values at different times.
- Methods (operations) on objects (data) that may or may not change the objects state.
- OOP is basically imperative programming with an additional *level of abstraction* regarding data and functions.



Improving Computation

Data Structures

- Choosing suitable data organization
- Having efficient data storing and retrieving mechanism



Solution Characteristics

✓ Efficient (performance or speed)
✓ Cost effective
✓ Scalable

× Useful
× Correct
× Robust
× Secure
× Modifiable

Problem
+
Language
+
Data Structure



Algorithm




➤ Definition of Algorithm?

Definition of Algorithm?

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.
- From the data structure point of view, following are some important categories of algorithms –
 - Search – Algorithm to search an item in a data structure.
 - Sort – Algorithm to sort items in a certain order.
 - Insert – Algorithm to insert item in a data structure.
 - Update – Algorithm to update an existing item in a data structure.
 - Delete – Algorithm to delete an existing item from a data structure.

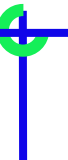
Algorithm

- 
- Definition of Algorithm?
 - Characteristics of an Algorithm?

Characteristics of an Algorithm?

- Not all procedures can be called an algorithm. An algorithm should have the following characteristics –
- **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** – An algorithm should have 0 or more well-defined inputs.
- **Output** – An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** – Algorithms must terminate after a finite number of steps.
- **Feasibility** – Should be feasible with the available resources.
- **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

Algorithm

- 
- Definition of Algorithm?
 - Characteristics of an Algorithm?
 - How to Write an Algorithm?

How to Write an Algorithm?

- There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.
- As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.
- We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution

Algorithm Example

➤ Design an algorithm to add two numbers and display the result.

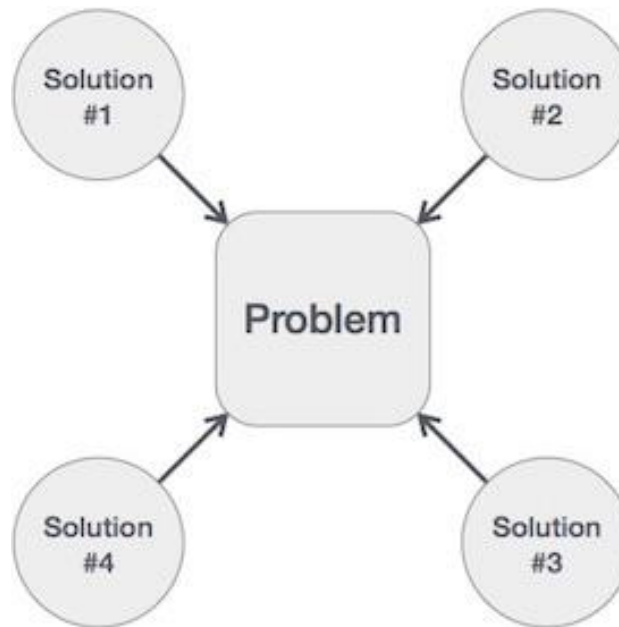
- step 1 – START
- step 2 – declare three integers a, b & c
- step 3 – define values of a & b
- step 4 – add values of a & b
- step 5 – store output of step 4 to c
- step 6 – print c
- step 7 – STOP

Algorithm Example

- step 1 – START ADD
- step 2 – get values of a & b
- step 3 – $c \leftarrow a + b$
- step 4 – display c
- step 5 – STOP

Algorithm

- We design an algorithm to get a solution of a given problem.
A problem can be solved in more than one ways.



Algorithm Analysis