UNIT-1

APP

Page No.
Date

Programming Paradigm & it's Hierarchy

ⓐ PROGRAMMING PARADIGM

- way of classifying & organising programming languages.
- method to solve some problem or do some task.
- Each programming paradigm has it's own set of rules, techniques & concepts that guide the development process.

★ It's a fundamental approach or style of programming that guides the process of designing, structuring & implementing computer programs

★ It's all abt. diff. approaches used to find a software soln to the problem.

Programming Paradigm Hierarchy

- Imperative
  - Procedural (P)
  - Object-Oriented (O)
  - Parallel Processing (P)
- Declarative
  - DBMS (Database processing) (D)
  - Logic (L)
  - Functional (F)

Programming & it's Elements

→ A programming language is made up of specific terms or directions used to create some type of o/p, such as websites, apps, & other software.

→ • Languages like JS (scripting languages), Python & Java are often used by websites for a variety of purposes.
• C++ is used just abt everywhere to make things like desktop apps, games, & more.

☆ Elements of Programming

(i) Variables
(ii) I/P & O/P Statements
(iii) Conditionals (if ... else)
(iv) loops (for, while, do-while)
(v) Functions & Methods.

☆ Problem Analysis & Sol^ns

✓ 1st approach: Procedural Programming
✓ 2nd approach: Object-Oriented "
✓ 3rd approach: Logical "
✓ 4th approach: Functional "

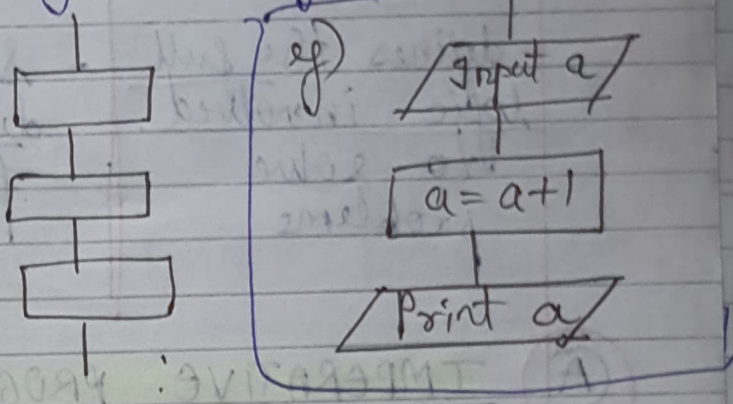
P
O
L
F

(Imperative)
(Declarative)

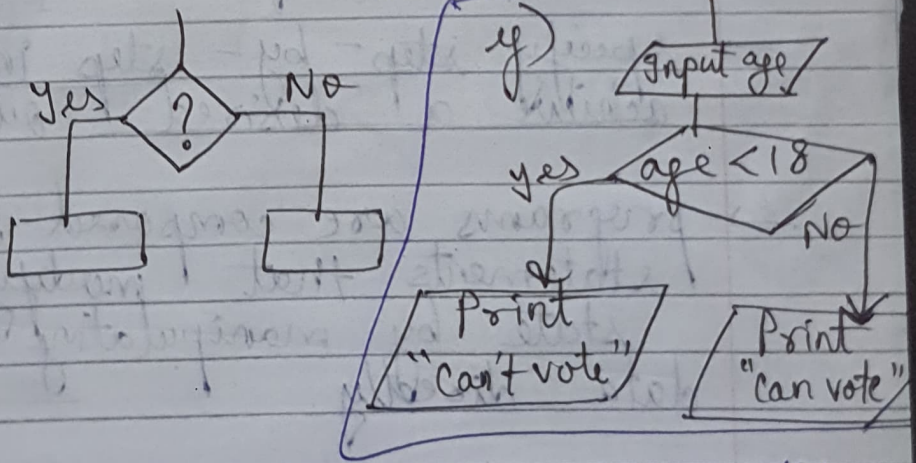
Böhm - Jacopini Theorem (1960s)

→ An algo. can be written using only 3 kind of statements.

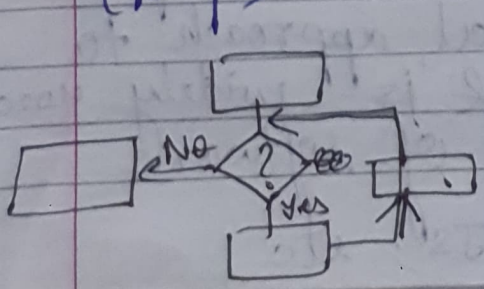
- **Sequence:** Executing subprograms order-wise.

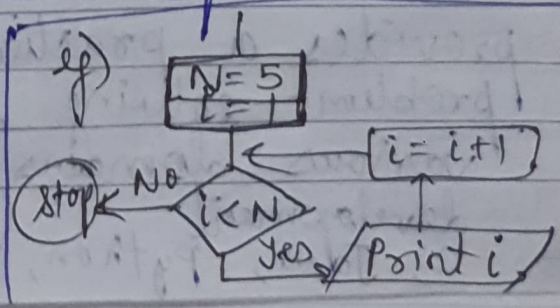


- **Selection: (Decision)** Execute one of 2 subprograms acc. to the value of boolean expression



- **Iteration: (Loops)** Executing a subprogram until a Boolean expression is true.





Programming Paradigm Hierarchy in Detail

Imperative v/s Declarative

Imperative	Declarative
- How to execute	- What to execute
- Detailed control flow	- Not Detailed control flow.
- defines the full logic involved to solve problems	- Solves the problem BUT does NOT define full logic for solving it.

(A) IMPERATIVE PROGRAMMING PARADIGM

→ HOW a program should accomplish a task by providing a sequence of instructions.

→ specify step-by-step instructions to achieve a desired outcome.

→ programs are composed of a series of statements that modify the program's state by manipulating variables & data directly.

→ It also provides modularity (reusable functions)

→ provides a practical approach to problem solving & is widely used in various domains of software development

→ C, Java, Python, JS, etc.

Types of Imperative Programming Paradigm

(1) Procedural Programming Paradigm

- focuses on the concept of fns / routines which are a sequence of instructions that perform specific tasks.
- dividing a program into smaller, reusable fns & organising them in a hierarchical manner to solve a problem.

eg → C, C++, Java, Python, Pascal, etc.

Code (Java) →

```
class A
{
    static void getdata(int x, int y)
    {
        int c = x+y;
        System.out.print(c);
    }
    public static void main(String[] args)
    {
        getdata(10,20);
    }
}
```

O/P → 30

P.T.O. →

encourages a more natural representation of real-world concepts in the code.

② Object-Oriented Programming Paradigm

→ program is written as a collection of classes & objects which are meant for communication.

- ⊙ Classes: Blueprints of objects, methods & attributes.
- ⊙ Objects: Are instances of a class.
- ⊙ Methods: Describe the behaviour of an object.
- ⊙ Attributes: Represent the state of an object.

■ OOPs concepts
- ~ Inheritance
- ~ Abstraction
- ~ Polymorphism
- ~ Encapsulation

y Code → (Java)

```
class A
{
    public void disp()
    {
        System.out.print("Welcome");
    }
}
class B
{
    public static void main(String[] args)
    {
        A obj = new A();
        obj.disp();
    }
}
```

O/P → Welcome

P (3) PARALLEL Processing Paradigm

⇒ performing multiple tasks parallelly.
→ increases speed & efficiency by dividing a larges task into smaller sub-tasks that can be executed concurrently.
→ these smaller sub-tasks are executed simultaneously on multiple processors, cores, or threads.
The results from each sub-task are then combined to obtain the final result.

eg) NEST (one of the oldest one), C++, Java, Python, etc (with some of library fns)

eg Code →

```
public class ParallelProcessingExample
{
    public static void main (String [] args)
    {
        Thread t1 = new Thread (() -> {
            System.out.println ("Task 1 is executing
                                 in parallel");
            // Task-1 logic here
        });
        Thread t2 = new Thread (() -> {
            System.out.println ("Task-2 is executing
                                 in parallel");
            // task-2 logic here
        });
```

```
t1.start();
t2.start();

try{
    t1.join();
    t2.join();
}
catch(InterruptedException e)
{
    e.printStackTrace();
}

// Proceed with the main thread after
// Task 1 & Task 2 have completed
}
}
```

O/P → Task 1 is executing in parallel
Task 2 is executing in parallel

(B) DECLARATIVE Programming Paradigm

→ focuses on WHAT to execute.

→ describing the logic & properties of a computation without explicitly specifying the control flow.

→ main focus is achieving the end result.

→ straight-forward & to the pt. while writing the program code.

Types of Declarative Programming Paradigm

① DATABASE Processing Paradigm

→ involves the storage, retrieval, manipulation & ~~manipulation~~ management of data in a structured manner using DBMS.

eg) SQL (structural Query Language), MongoDB, etc

eg code (SQL) →

```
→ create database SRM;
→ use SRM;
→ create table Student (name varchar[30],
  id int, course varchar[20],
  phone_no int);
→ insert into Student values ("Seema",
  101, "BTech", 9920467839);
→ select * from Student;
```

O/P →

Student

name	id	course	phone_no
Seema	101	BTech	9920467839

(2) LOGIC Programming Paradigm

→ Based on logic & control
- logic → facts & rules
- control → order of rules

→ A logic program is a collection of logical propositions & questions.

y) PROLOG (Programming in Logic)

→ use logical inference & backtracking to find all possible answers based on the available facts & rules.

eg code → If x is bird or an airplane then x has wings.
Tweedy is a bird.
Does Tweedy have wings? "Yes."

use the concepts of AI, machine learning models

P.T.O. →

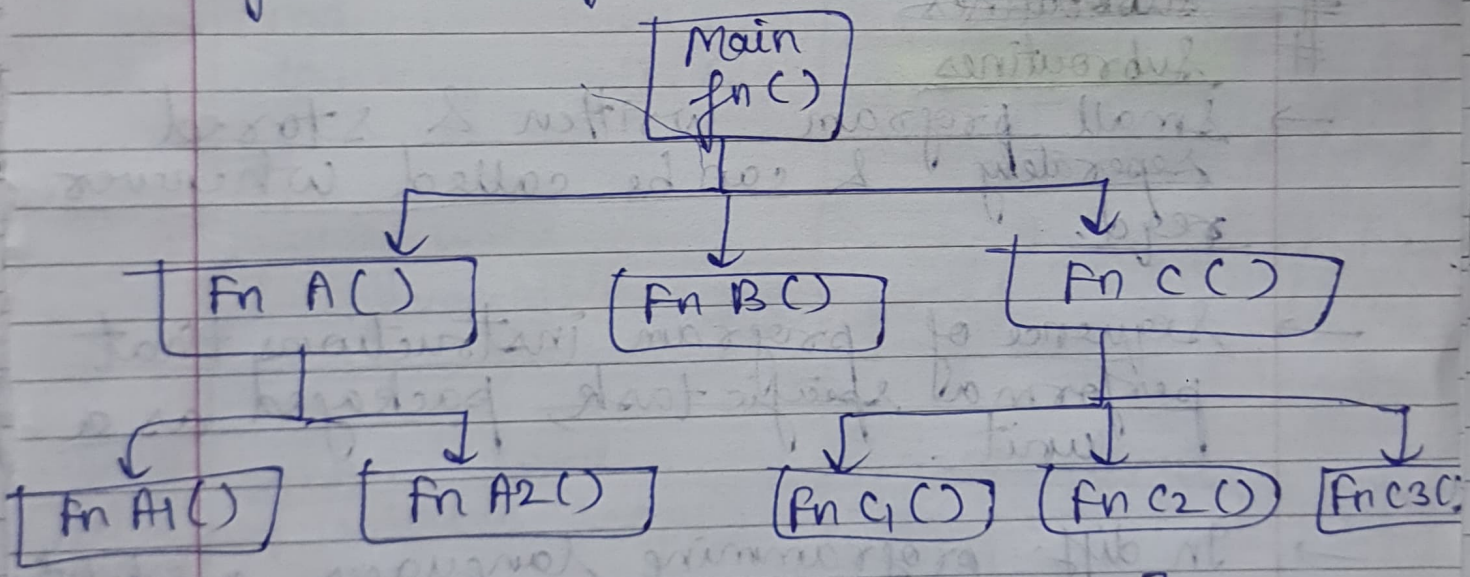
(3) Functional Programming Paradigm

→ Programs are written using fns (blocks of codes) intended to behave like mathematical fns.

→ discourages changes in the value of var's throught the assignment.

→ makes a great deal with recursion.

→ eg → Js, Python, Scala, etc.



eg code → (Python)

```
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared).
```

P.T.O. →

Multi-Paradigm

→ Programming languages that support more than one programming paradigm.

↓

allow the program code to implement more than one paradigm.

eg → Java, Python, etc.

~~# Subroutines~~

Subroutines

→ Small program written & stored seperately & can be called whenever reqd.

→ Sequence of program instructions that perform a specific task, packaged as a unit.

→ In diff. programming languages, a subroutine may be called a procedure, function, routine, method or subprogram.

P.T.O →

Memory Allocation

→ process of allocating / assigning memory.

	STATIC (STACK)	DYNAMIC (HEAP)
1)	Assigning the memory space during compile-time	1) Assigning the memory space during run time.
2)	It operates in LIFO (last In - First - Out) manner, resembling a stack of plates.	2) Whenever we create any object it's created in the heap & reference var. of that object is created in the stack.
3)	mainly used for local variables, reference var's, fn calls.	3) objects in heap have global access.
4) (Data lifetime)	short - lived (variable scope)	4) long - lived (can persist beyond fn scope)
5) (Size)	Smaller	5) Larger
6) (Access speed)	Faster	6) Slower
7) (Memory Mngmt)	Automatic (push/pop)	7) Manual (allocation & de-allocation)
8) (Concurrency)	NOT suitable for large data or concurrent tasks.	8) Suitable for larger data & parallel tasks.

Dynamically Dispatched Message Calls

→ are a concept in OOP languages where the appropriate method to be executed is determined at run-time based on the actual type of the object being referenced, rather than the type known at compile-time.

→ allows for "polymorphic" behaviours, where a subclass can override the method of it's superclass, & the appropriate method gets invoked based on the object's actual class.

eg code → (Java)

```
class Animal
{
  public void makeSound()
  {
    System.out.println("Animal makes a sound");
  }
}
class Dog extends Animal
{
  public void makeSound()
  {
    System.out.println("Dog barks!");
  }
}
```

P.T.O →

```
class Cat extends Animal
{
  public void makeSound()
  {
    System.out.println("Cat meows!");
  }
}
public class Main
  {
    public static void main(String[] args)
    {
      Animal animal1 = new Dog();
      Animal animal2 = new Cat();

      animal1.makeSound();
      animal2.makeSound();
    }
  }
```

O/P → Dog barks!
Cat meows!

Method Call Overheads

→ encompasses the additional time & resources reqd. when invoking a method or fn in a program.

P.T.O →

→ includes tasks such as call setup, parameter passing, control transfer, method execution, & post-call cleanup.

→ applies to any type of method or fn call, whether it is a direct procedure call or an indirect call.

★ Direct Procedure call overheads

→ subset of method call overheads.

→ refers to the overhead associated with invoking a fn or procedure directly, without any indirection or dynamic resolution.

→ focuses on the overhead arising from factors like context switching, parameter passing, stack mngmt, call/return instructions, & potential disruptions in ~~caching~~ caching & branch prediction mechanisms

cost or burden incurred by the program when executing the call

Parallel computing

(Same as Parallel Processing Programming Paradigm)

P.T.O.

Object Serialization

→ process of converting a data object (a combinat^{n} of code & data represented within a region of data storage) into a series of bytes that saves the state of an object in an easily tramittable form.