

Course: PARADIGMS AND COMPUTER PROGRAMMING FUNDAMENTALS (PCPF)



Course Instructor

Mrinmoyee Mukherjee B.E (Electronics), M.E (EXTC), PhD (Pursuing)

Assistant Professor

Department of Information Technology

St. Francis Institute of Technology

email: mrinmoyeemukherjee@sfit.ac.in

Academic Year: 2023-24 (Odd Semester)



OUTLINE OF SYLLABUS

Module	Contents
1	Introduction to programming paradigms and core language design issues
2	Imperative Paradigm: Data abstraction in object orientation
3	Declarative programming paradigm: Functional programming
4	Declarative programming paradigm: Logic programming
5	Alternative paradigm: Concurrency
6	Alternative paradigm: Scripting Languages

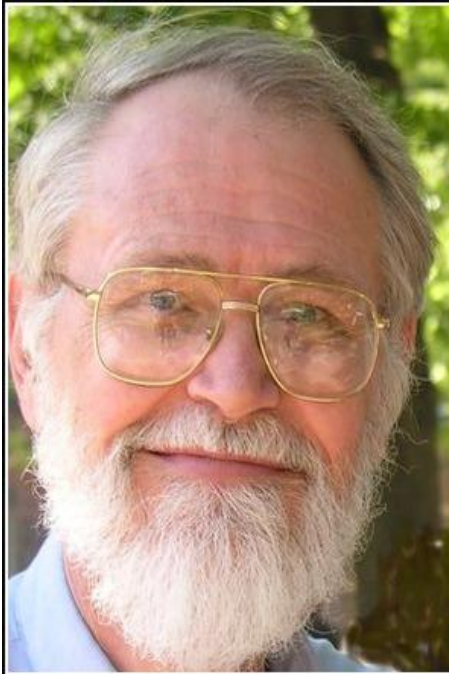
OUTLINE OF UNIT-1

Sub-Unit	Contents
1.1	Introduction to different programming paradigms
1.2	Names, Scopes, Bindings, Scope Rules, Storage Management
1.3	Type Systems, Type checking, Equality testing, and assignment
1.4	Subroutine and control abstraction, Stack layout, calling sequence, parameter passing
1.5	Generic subroutines and modules, Exception handling, co-routines and events

1.1-INTRODUCTION TO DIFFERENT PROGRAMMING PARADIGMS

*Any fool can write code that a computer can understand.
Good programmers write code that human's can understand*
-Martin Fowler

(Martin Fowler is a British software developer, author and international public speaker on software development, specializing in object-oriented analysis and design, UML, patterns)



Controlling complexity is the
essence of computer programming.

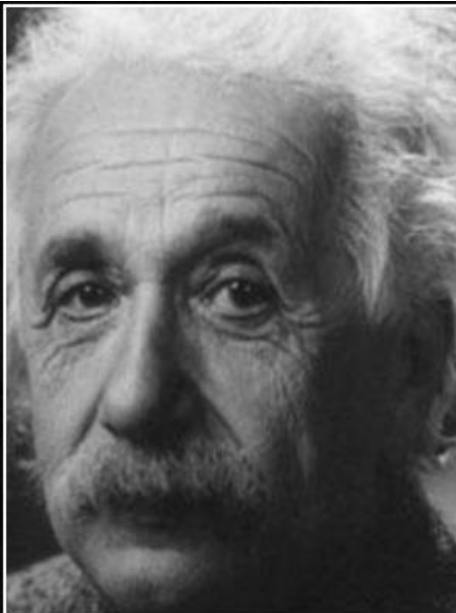
— *Brian Kernighan* —

AZ QUOTES

1. When programming , complexity is always the enemy
2. Managing complexity is a programmer's main concern



Choosing the
right
programming
paradigms



Out of complexity, find simplicity!

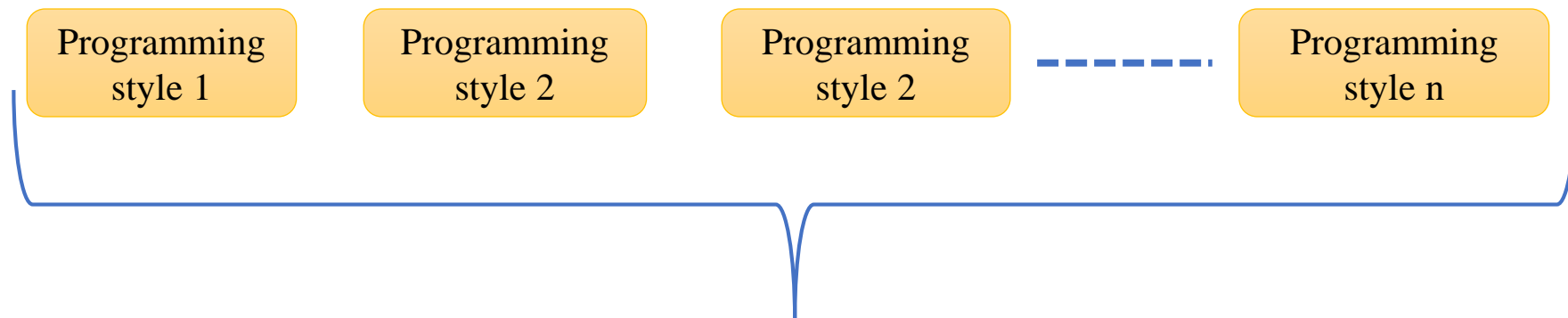
— *Albert Einstein* —

AZ QUOTES



What is Programming Paradigm?

Definition: A programming paradigm is a style or a “way” of programming. Some languages make it easy to write programs in some paradigms but others do not.



You have different “styles”/ “ways”/ **“Paradigms”** of programming to tackle the issues of handling complexity

ARE
CONCRETE

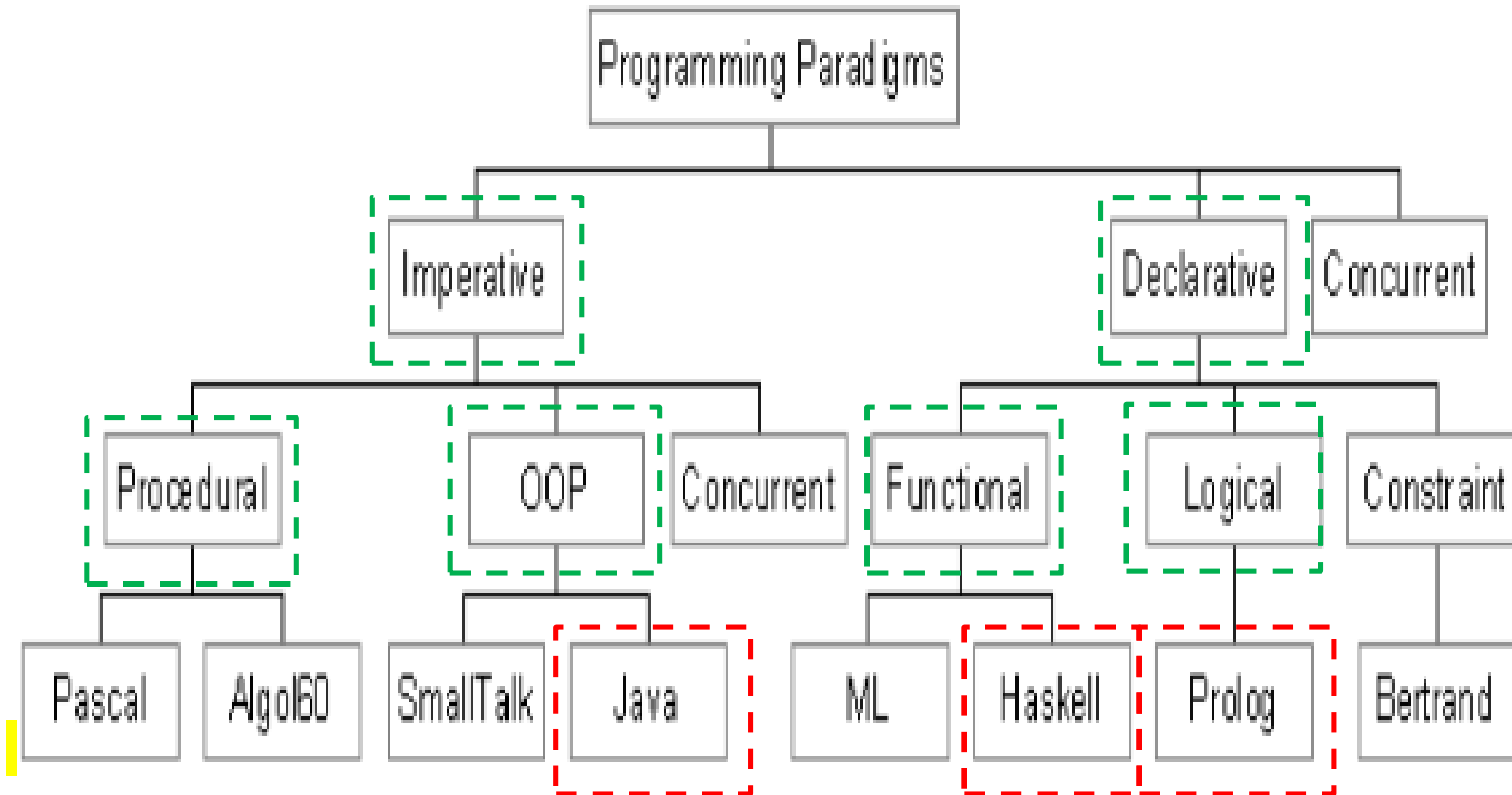
Programming
Languages



Programming
Paradigms

ARE WAY OF
DOING

Diagram of Programming Paradigms



Features of Good Programming Language

- Clear, simple
- More expressive
- Orthogonal
- Support of abstraction
- Ease of implementation
- Ease of program verification
- Programming environments
- Portability of programs
- Naturalness for the application
- Low cost of implementation and usage

Why So Many Languages?

- Evolution.
- Special purposes /Application Specific
- Personal preference

Language Groups

- Multi-purpose languages
 - Scala, C#, Java, C++, C
 - Haskell, SML, Scheme, LISP
 - Perl, Python, Ruby
- Special-purpose languages
 - UNIX shell
 - SQL
 - LATEX

History of PLs

- 1951–55: Experimental use of expression compilers.
- 1956–60: **FORTRAN**, **COBOL**, **LISP**, **Algol 60**.
- 1961–65: APL notation, Algol 60 (revised), **SNOBOL**, CPL.
- 1966–70: APL, SNOBOL 4, FORTRAN 66, **BASIC**, **SIMULA**, Algol 68, Algol-W, **BCPL**.
- 1971–75: Pascal, PL/1 (Standard), **C**, **Scheme**, **Prolog**.
- 1976–80: **Smalltalk**, Ada, FORTRAN 77, **ML**.
- 1981–85: Smalltalk-80, **Prolog**, Ada 83.
- 1986–90: **C++**, SML, **Haskell**.
- 1991–95: Ada 95, TCL, Perl.
- 1996–2000: **Java**.
- 2000–05: C#, **Python**, Ruby, Scala.

IMPERATIVE PROGRAMMING

#(first do this then do that.....)

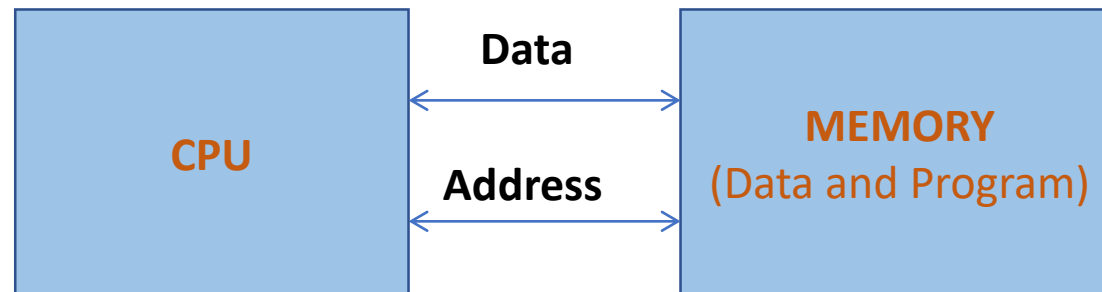
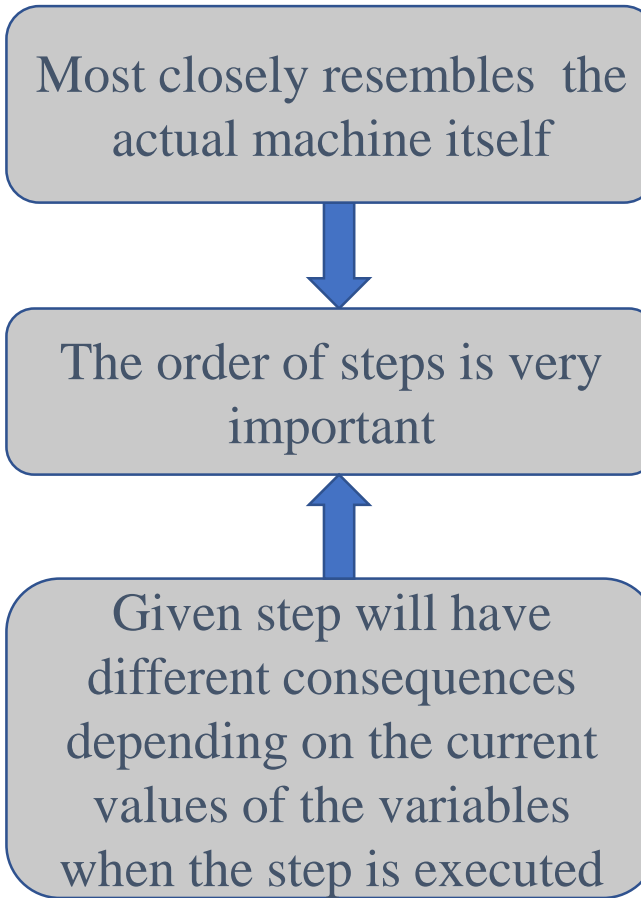
- The imperative paradigm is the oldest and the most popular programming paradigm
- Based on the von Neumann architecture of computers (<https://www.javatpoint.com/von-neumann-model>)
- Imperative programs define sequences of commands/statements for the computer that change a program state (i.e., set of variables)
 - Commands are stored in memory and executed in the order found
 - Commands retrieve data, perform a computation, and assign the result to a memory location
- The hardware implementation of almost all computers is imperative
- Machine code which is naïve to the computer hardware is written in imperative style

IMPERATIVE PROGRAMMING

Program: Sum of first 5 natural numbers in C

```
#include<stdio.h>

int main( )
{
int sum=0;
sum+=1;
sum+=2;
sum+=3;
sum+=4;
sum+=5;
printf("The sum is: %d/n", sum);
return 0;
}
```





Are there any **CENTRAL** elements of imperative paradigms

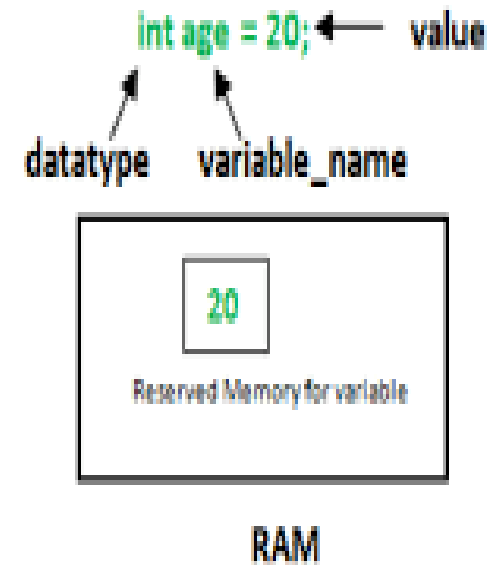
YES!!! YES!!!!

Assignment Statement

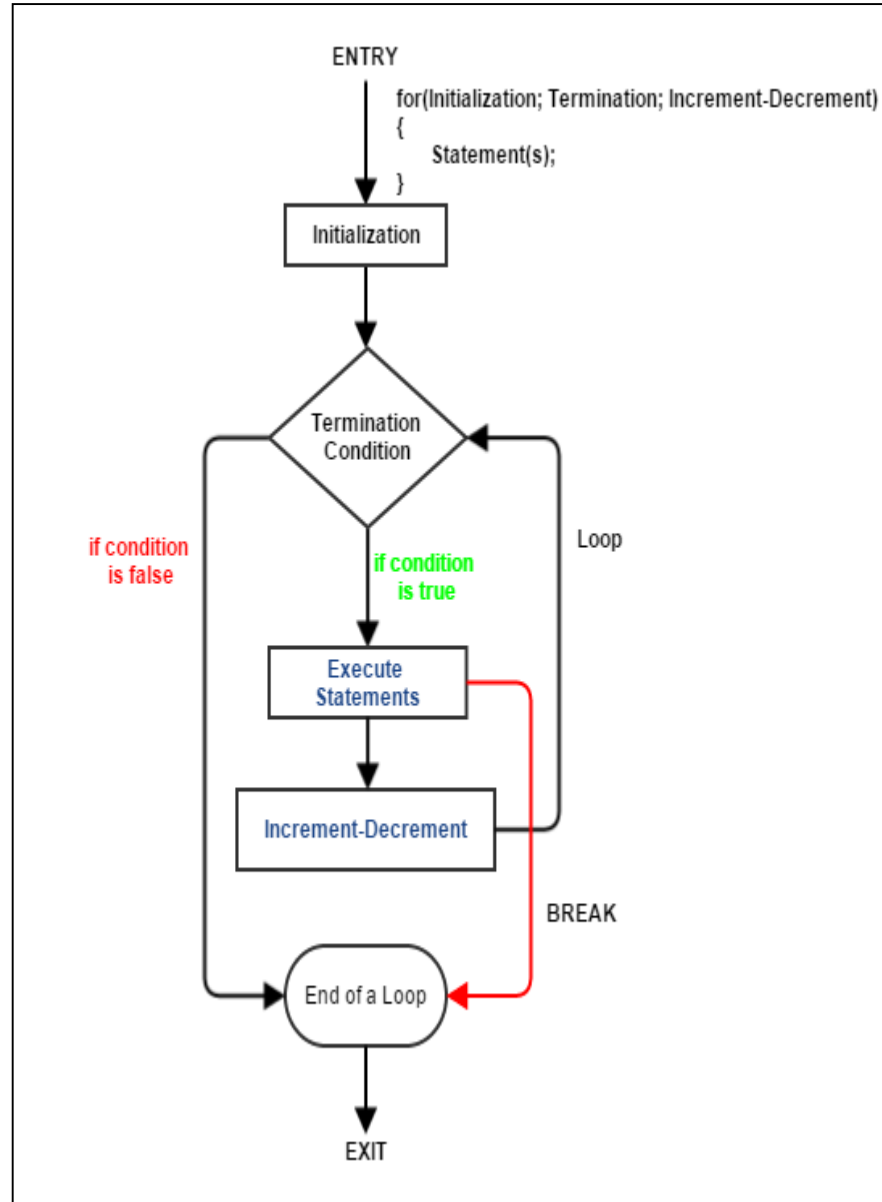
- The general syntax:
`<target_var> <assign_operator> <expression>`
- The assignment operator:
 - `=` FORTRAN, BASIC, PL/I, C, C++, Java.
 - `:=` ALGOLs, Pascal, Ada.
- Simple assignment:
 - `a = b;`
 - `a = b = c;`
 - Suppose **a**, **b**, and **c** are integers.
 - In C, the integer value of **c** is assigned to **b**, which is in turn assigned to **a** (**multiple targets**).

Variables

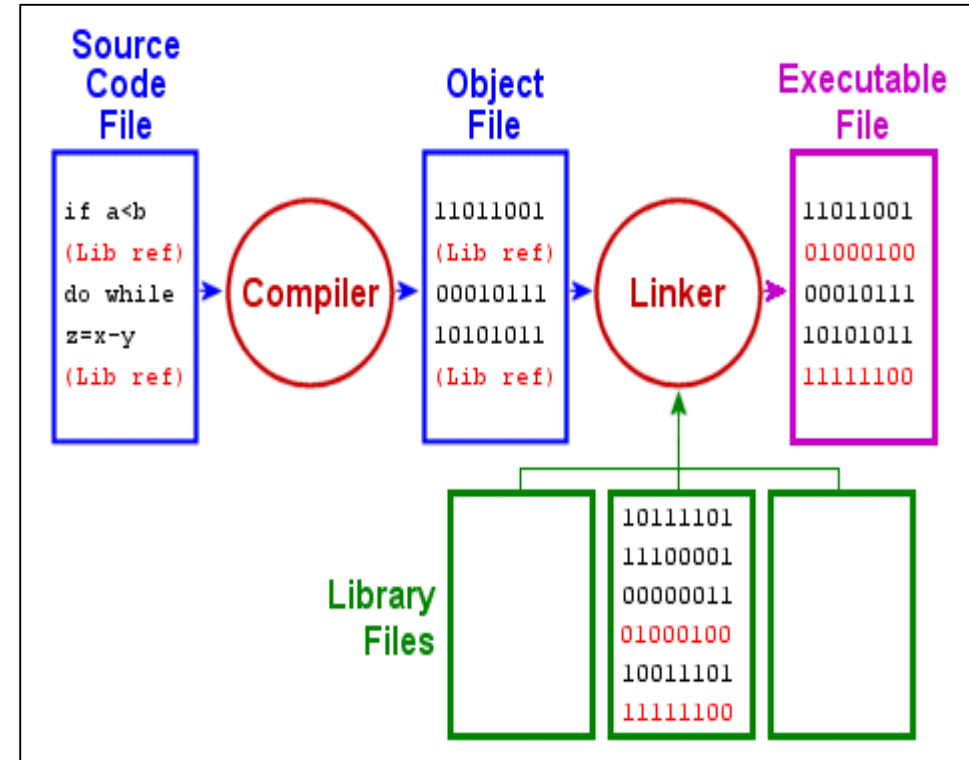
Variables in C++



Flow control



Step by step execution



Examples of imperative programming language (languages that follow imperative programming paradigm)

C: Developed by Dennis Ritchie and ken Thompson

FORTAN: Developed by John Backus from IBM

Basic: Developed by John Kemeny and Thomas E Kurtz

PROCEDURAL PROGRAMMING

(for repetitive job)

- Procedural programming is a **refinement** of the imperative paradigm adding subroutines (or procedures)
- Procedures can be used the same way that built-in commands are used (allows re-usability)

?? What is a procedure

- Set of subroutines
- May or may not return a value

In a program for drawing shapes, the program could ask the user what shape to draw. The instructions for drawing a square could be captured in a **procedure**.

The algorithm for this action could be a set of tasks, such as these:

Repeat the next two steps four times:

Draw a line of length n .

Turn right by 90 degrees.

If this were a computer program, this set of instructions could be given the name '*square*' and this sequence would be executed by **running** (calling) that **procedure**.

Example of computing the factorial of a number:

IMPERATIVE

```
unsigned int n = 5;  
unsigned int result = 1;  
while(n > 1) {  
    result *= n;  
    n--;  
}
```

Introduce
procedure,
have return
type

PROCEDURAL

Forming a procedure

```
int factorial(unsigned int n)  
{  
    unsigned int result = 1;  
    while(n > 1) {  
        result *= n;  
        n--;  
    }  
    return result;  
}
```

Returning a value

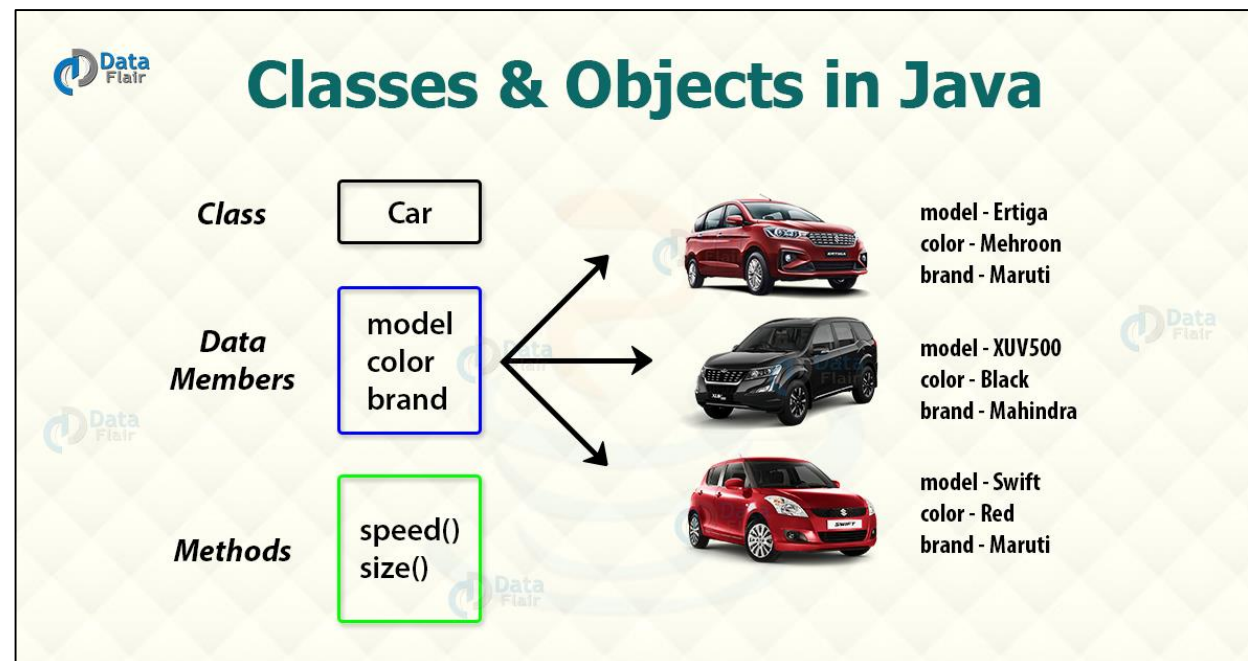
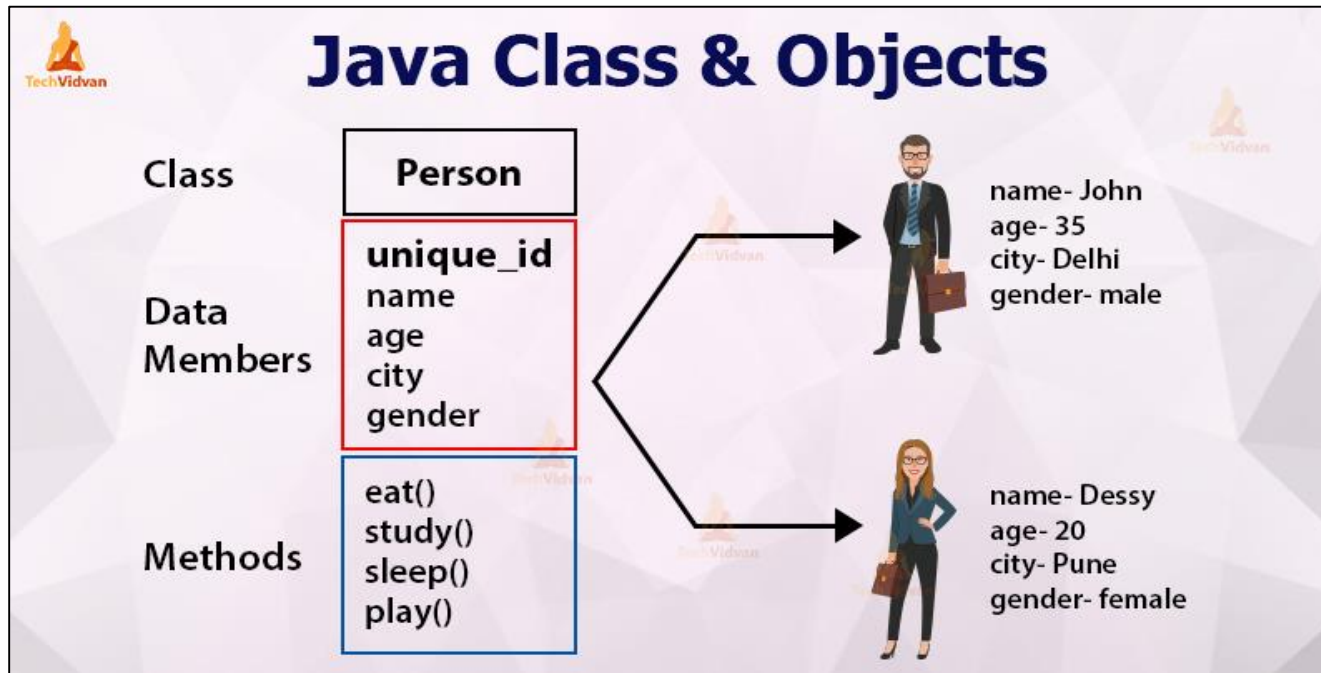
Procedure will be called from main/home

Examples of procedural programming language (languages that follow procedural programming paradigm)

C:	Developed by Dennis Ritchie and ken Thompson
C++:	Developed by Bjarne Stroustrup at Bell Labs
JAVA:	Developed by James Gosling at Sun Microsystems
ColdFusion:	Adobe, Joseph J. Allaire, Macromedia

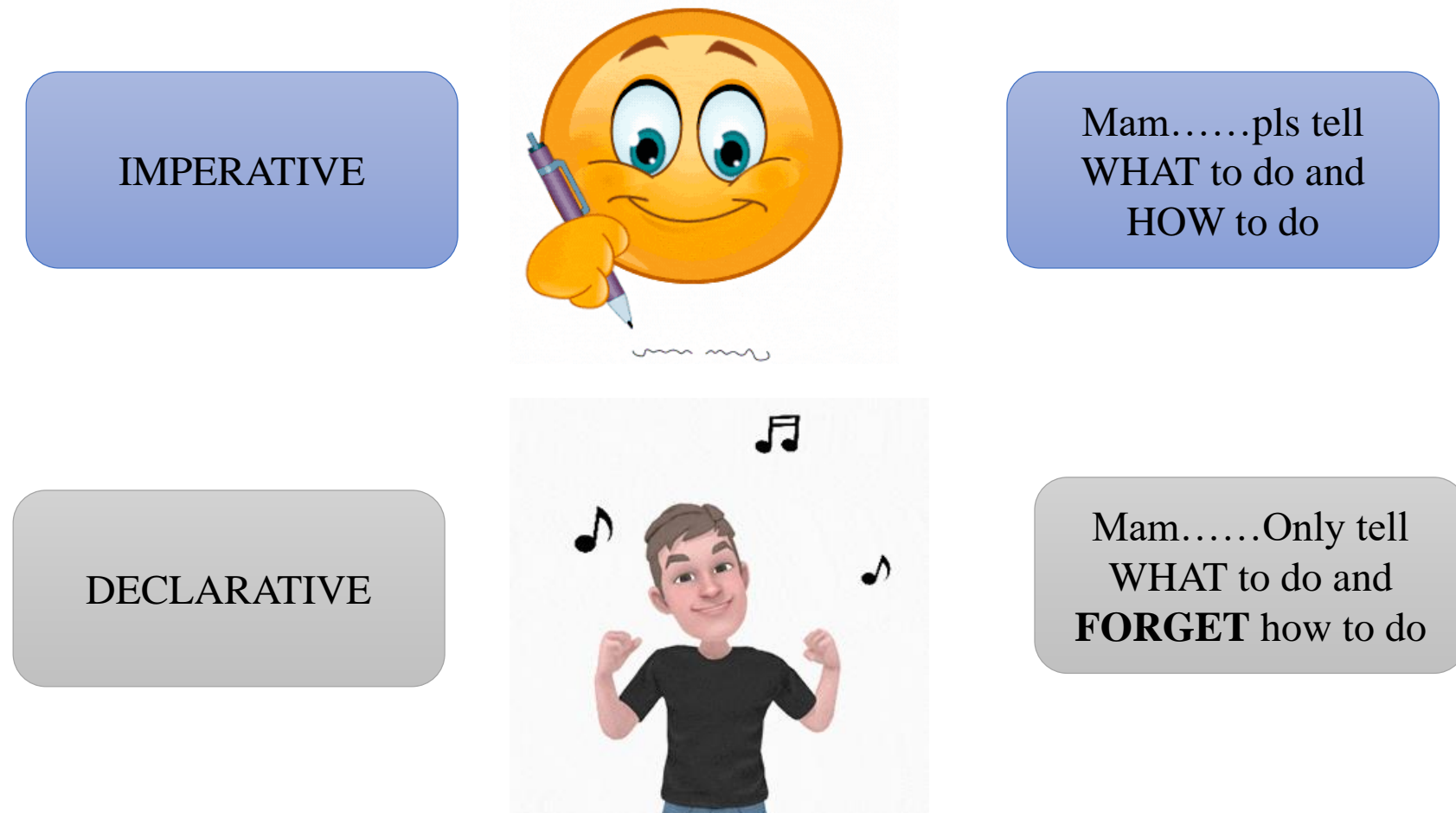
OBJECT ORIENTED PROGRAMMING PARADIGM

- An approach to the solution of problems in which all computations are performed in the context of objects
- The program is written as a collection of classes and objects.
- The smallest and the basic entity is object
- Emphasis is on data rather than procedure
- Methods that operate on the data of an object is tied together in the data structure
- Data is hidden and cannot be accessed by external function
- Objects may communicate with each other through methods
- Follows bottom up approach in program design
- **Ruby, Java, C++, Python, Simula**



DECLARATIVE PROGRAMMING PARADIGM

- The style expresses the logic of a computation without talking about its control flow
- It defines what needs to be accomplished by the program without defining how it needs to be implemented



IMPERATIVE: Provides instructions for assembly



DECLARATIVE: Provides a picture of finished piece as a template



The more sophisticated the application, the greater the danger that the code becomes so convoluted that it can only be read by the developer who originally wrote it

Examples of declarative programming paradigm

Prolog : Developed by by Alain Colmerauer at the University of Aix-Marseille, France

Haskell : First proposed by Philip Wadler and Stephen Blott

Miranda : Developed by David Turner Research Software Ltd.

Advantages:-

- Short, efficient code
- Easy optimization as implementation is controlled by an algorithm

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

```
main :: IO ()
main = print (sum [1..10000])
```


FUNCTIONAL PROGRAMMING

In functional programming we write the function exactly as mathematical function

$$f(x)=x+1$$

$f(x) = x + 1$

Constant Value

$f(2) = 2 + 1 = 3$ $f(5) = 5 + 1 = 6$

```
var a = 100;      ← Immutable for lifetime
```



```
var b = a + 20    ← 'a' Remains Unchanged
```

Program can easily work on Multi-Core
and Multi-Threaded systems

```
int function(int x)
{
    f(x) => x + 1
    return x + 1;
}
```

Immutable means non modifiable, mutable means modifiable

LOGIC PROGRAMMING PARADIGM

- Logic programming refers loosely to
 - The use of facts and rules to represent information
 - The use of deduction to answer queries

Lets understand with example

Overlap(X,Y):-member (M,X), member (M,Y)

In the words, lists X and Y overlap if there is some M that is a member of both X and Y.

Kowalski illustrates the division of labor in logic programming by writing the informal equation

$$\text{ALGORITHM} = \text{LOGIC} + \text{CONTROL}$$

Here logic refers to the facts and rules specifying what the algorithm does, and control refers to how the algorithm can be implemented by applying the rules in a particular order.

We (The programmers) supply the logic part and the programming language supplies the control

- Programs are written in language of some LOGIC
- Execution of a logic program is a theorem proving process; that is computation is done by logic inferences
- Prolog (PROgramming in LOGic) is a representative programming language

Here logic refers to the facts and rules specifying what the algorithm does



- A **logic** is a language. It has syntax and semantics. More than a language, it has inference rules .
- **Syntax:** The rules about how to form formulas, this is usually the easy part of a logic
- **Semantics:** About meaning carried by the formulas, mainly in terms of a logical consequences
- **Inference rules:** describes the correct way to derive conclusions

REVISION

1. **Imperative:** what to do and how to do
2. **Procedural:** Refinement of imperative, introduces the concept of procedures
3. **Object-Oriented Programming:** Solution of problems in which all computations are performed in the context of objects
4. **Declarative:** Only tell what to do, forget how to do
5. **Logic:** Use facts and rules to represent information and deduce answer

Low Level and High Level Language

Low-level language	High-level language
It is a machine-friendly language, i.e., the computer understands the machine language, which is represented in 0 or 1.	It is a user-friendly language as this language is written in English words, and easily understood by humans.
The low-level language is slow , takes more time to execute.	It executes at a faster pace.
It requires the assembler to convert the assembly code into machine code.	It requires the compiler to convert the high-level language instructions into machine code.
The machine code cannot run on all machines, so it is not a portable language.	The high-level code can run all the platforms, so it is a portable language.
It is memory efficient and better performance	It is less memory efficient
Debugging and maintenance are not easier in a low-level language.	Debugging and maintenance are easier in a high-level language.

References

1. Michael L Scott, “ Programming Language Pragmatics”, Third edition, Elsevier publication (Chapter-1)
2. Ravi Sethi, “ Programming Languages-concepts and constructs”, Pearson Education (Chapter-1)

Web References

1. NPTEL Online Video resources- Lecture-01

<http://www.nptelvideos.in/2012/11/principles-of-programming-languages.html>

2. Stanford University Online lectures- Lecture-01 and Lecture-02

<https://www.youtube.com/watch?v=Ps8jOj7diA0&list=PL9D558D49CA734A02>