Computer Programing (CP)

Lecture # 1 & 2

Introduction of Programming Languages

Programming Language Concepts

- What is a programming language?
- Why are there so many programming languages?
- What are the types of programming languages?
- Does the world need new languages?

What is a Programming Languages

- A **programming language** is a formal **language** that specifies a set of instructions that can be used to produce various kinds of output. **Programming languages** generally consist of instructions for a computer.

- **Programming languages** can be used to create programs that implement specific algorithms.

What is a Programming Language?

A **programming language** is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. The term **programming language** usually refers to high-level **languages**, such as BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal.

A programming language is a tool for developing executable models for a class of problem domains.

Why Are There So Many Programming Languages

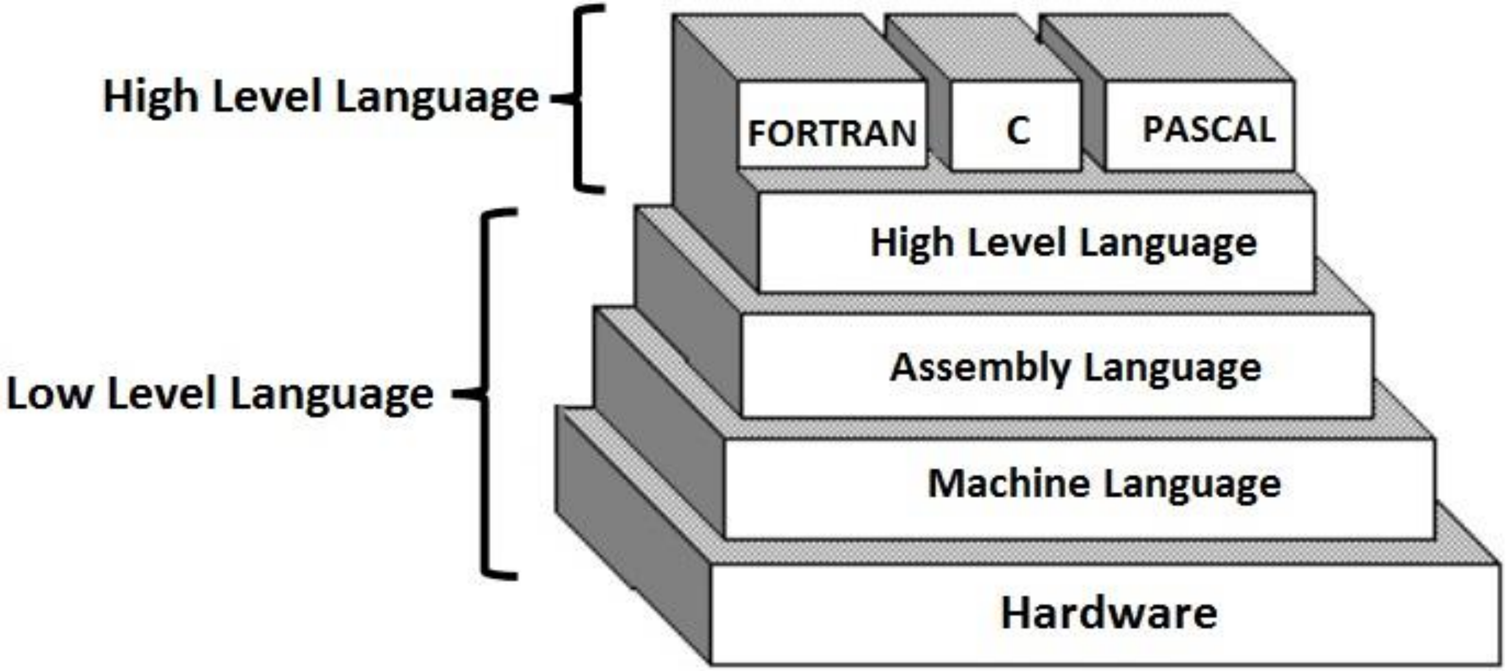
- Why does some people speak French?
  Programming languages have evolved over time as better ways have been developed to design them.

- First programming languages were developed in the 1950s

Since then thousands of languages have been developed

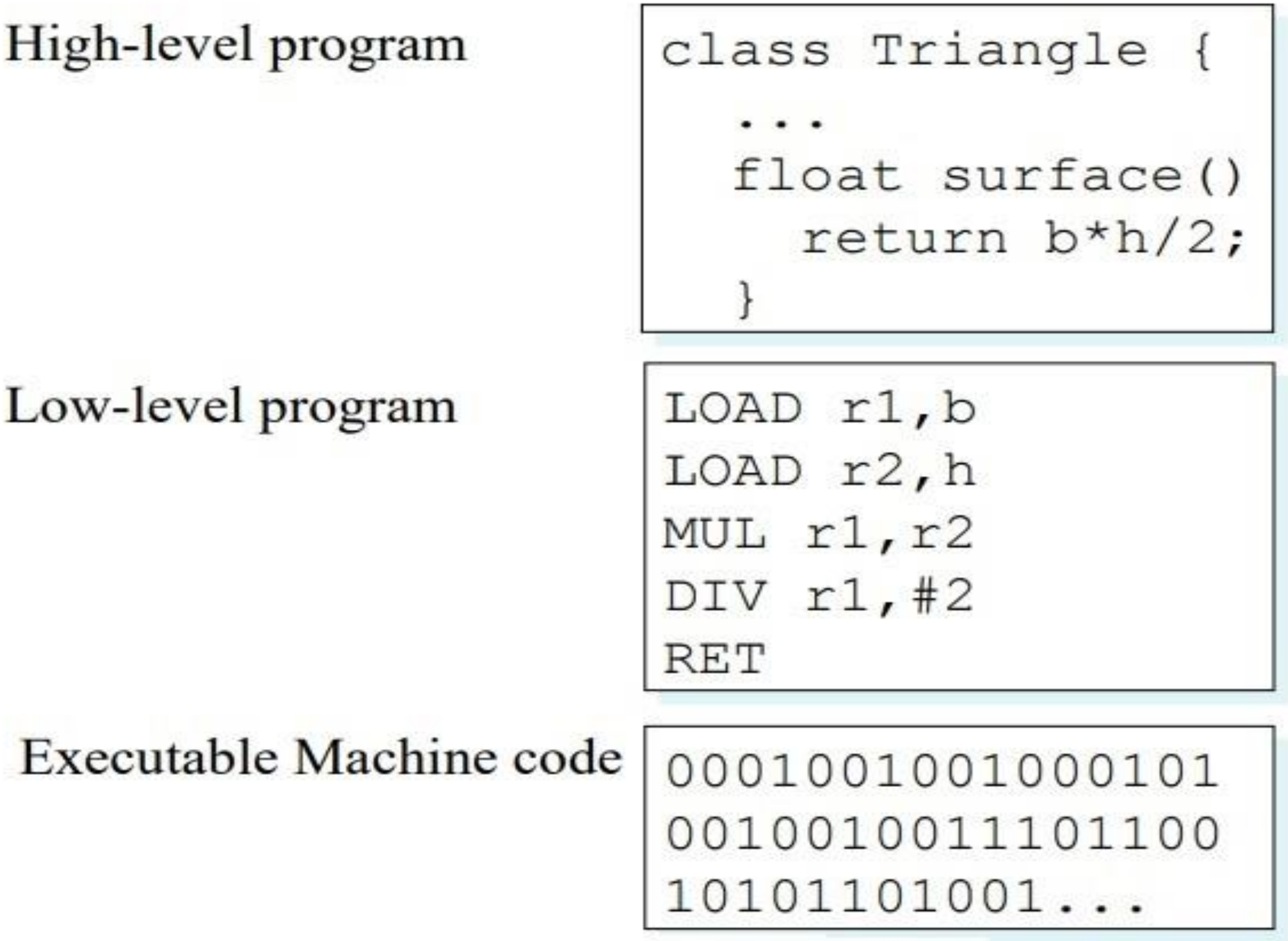
- Different programming languages are designed for different types of programs.

Levels of Programming Languages



Computer Language and its Types

Levels of Programming Languages



High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
10101101001...
```

What Are the Types of Programming Languages

- First Generation Languages
- Second Generation Languages
- Third Generation Languages
- Fourth Generation Languages
- Fifth Generation Languages

First Generation Languages

Machine language

- Operation code – such as addition or subtraction.
- Operands – that identify the data to be processed.
- Machine language is machine dependent as it is the only language the computer can understand.
- Very efficient code but very difficult to write.

Second Generation Languages

Assembly languages

- Symbolic operation codes replaced binary operation codes.
- Assembly language programs needed to be “assembled” for execution by the computer Each assembly language instruction is translated into one machine language instruction.
- Very efficient code and easier to write.

Third Generation Languages

Closer to English but included simple mathematical notation.

- Programs written in source code which must be translated into machine language programs called object code.
- The translation of source code to object code is accomplished by a machine language system program called a compiler.

Third Generation Languages (cont'd.)

- Alternative to compilation is interpretation which is accomplished by a system program called an interpreter.
- Common third generation languages
  - FORTRAN
  - COBOL
  - C and C++
  - Visual Basic

Fourth Generation Languages

A high level language (4GL) that requires fewer instructions to accomplish a task than a third generation language.

- Used with databases
  - Query languages
  - Report generators
  - Forms designers
  - Application generators

Fifth Generation Languages

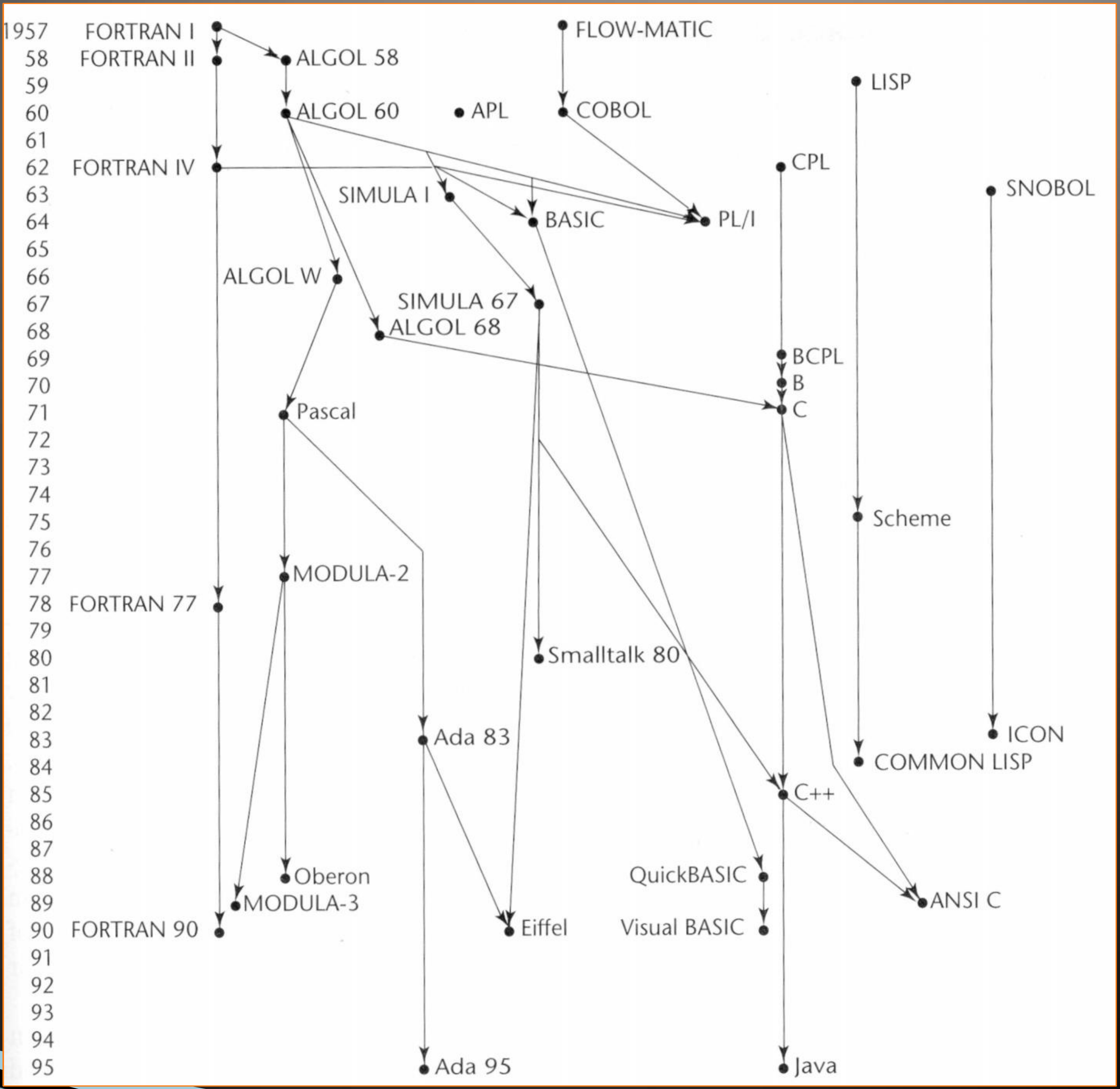
- Functional(?): Lisp, Scheme, SML
  - Also called applicative
  - Everything is a function
- Logic: Prolog
  - Based on mathematical logic
  - Rule- or Constraint-based

Beyond Fifth Generation Languages

- Though no clear definition at present, natural language programs generally can be interpreted and executed by the computer with no other action by the user than stating their question.

- Limited capabilities at present.

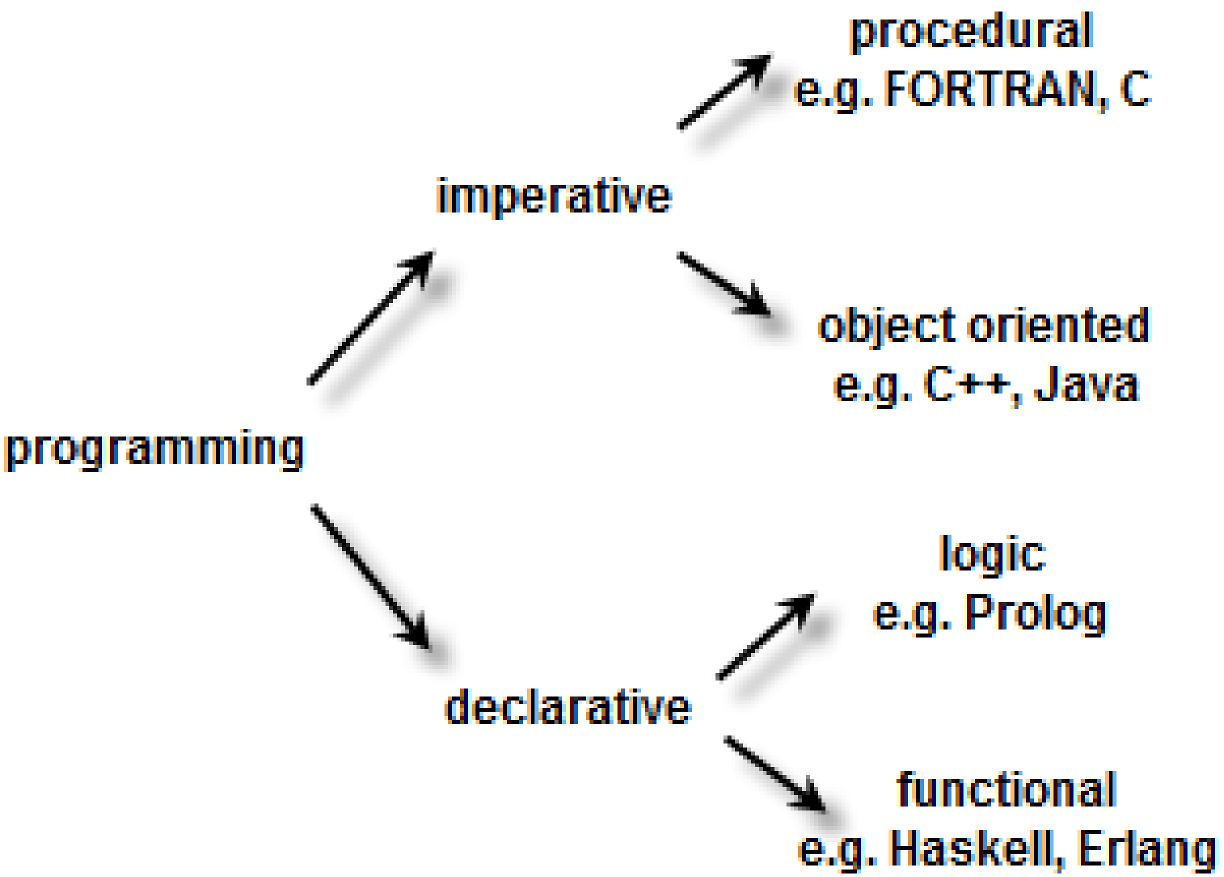
Language Family Tree



The principal paradigms

- Imperative Programming (C)
- Object-Oriented Programming (C++)
- Logic/Declarative Programming (Prolog)
- Functional/Applicative Programming (Lisp)

The principal paradigms



programming

imperative

procedural
e.g. FORTRAN, C

object oriented
e.g. C++, Java

declarative

logic
e.g. Prolog

functional
e.g. Haskell, Erlang

Imperative vs. Declarative

- Imperative programming is like giving instructions to an infant.

- Telling the “machine” **how** to do something, and as a result what you want to happen will happen.

- Functional programming is like describing your problem to a mathematician.

- Telling the “**machine**” what you would like to happen, and let the computer figure out how to do it.

Imperative vs. Declarative

Imperative: ***how*** to achieve our goal

Take the next customer from a list.
If the customer lives in Spain, show their details.
If there are more customers in the list, go to the beginning

Declarative: ***what*** we want to achieve

Show customer details of every customer living in Spain

Imperative Programming Paradigm

- State maintain through Variables
- Computations are performed through a guided sequence of steps, in which these variables are referred to or changed.
- The order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed.

  - the imperative paradigm most closely resembles the actual machine itself, so the programmer is much closer to the machine;
  - because of such closeness, the imperative paradigm was the only one efficient enough for widespread use until recently.

Imperative Programming Paradigm

Advantages

- efficient;
- close to the machine;
- popular;
- familiar.

Disadvantages

- Side effects also make debugging harder;
- Abstraction is more limited than with some paradigms;
- Order is crucial, which doesn't always suit itself to problems.

Logical Programming Paradigm

- The Logical Paradigm takes a declarative approach to problem-solving.
- Various logical assertions about a situation are made
- A logical program is divided into three sections:
- a series of definitions/declarations that define the problem domain
- statements of relevant facts
- statement of goals in the form of a query

Advantages:

- The system solves the problem, so the programming steps themselves are kept to a minimum;
- Proving the validity of a given program is simple.

Functional Programming Paradigm

- The Functional Programming paradigm views all subprograms as functions in the mathematical sense-informally.
- They take in arguments and return a single solution.
- The solution returned is based entirely on the input, and the time at which a function is called has no relevance.

Functional Programming Paradigm

Advantages

- The following are desirable properties of a functional language:
- The high level of abstraction, especially when functions are used, suppresses many of the details of programming
- Thus removes the possibility of committing many classes of errors;
- The lack of dependence on assignment operations, allowing programs to be evaluated in many different orders.
- This evaluation order independence makes function-oriented languages good candidates for programming massively parallel computers;

Object Oriented Programming Paradigm

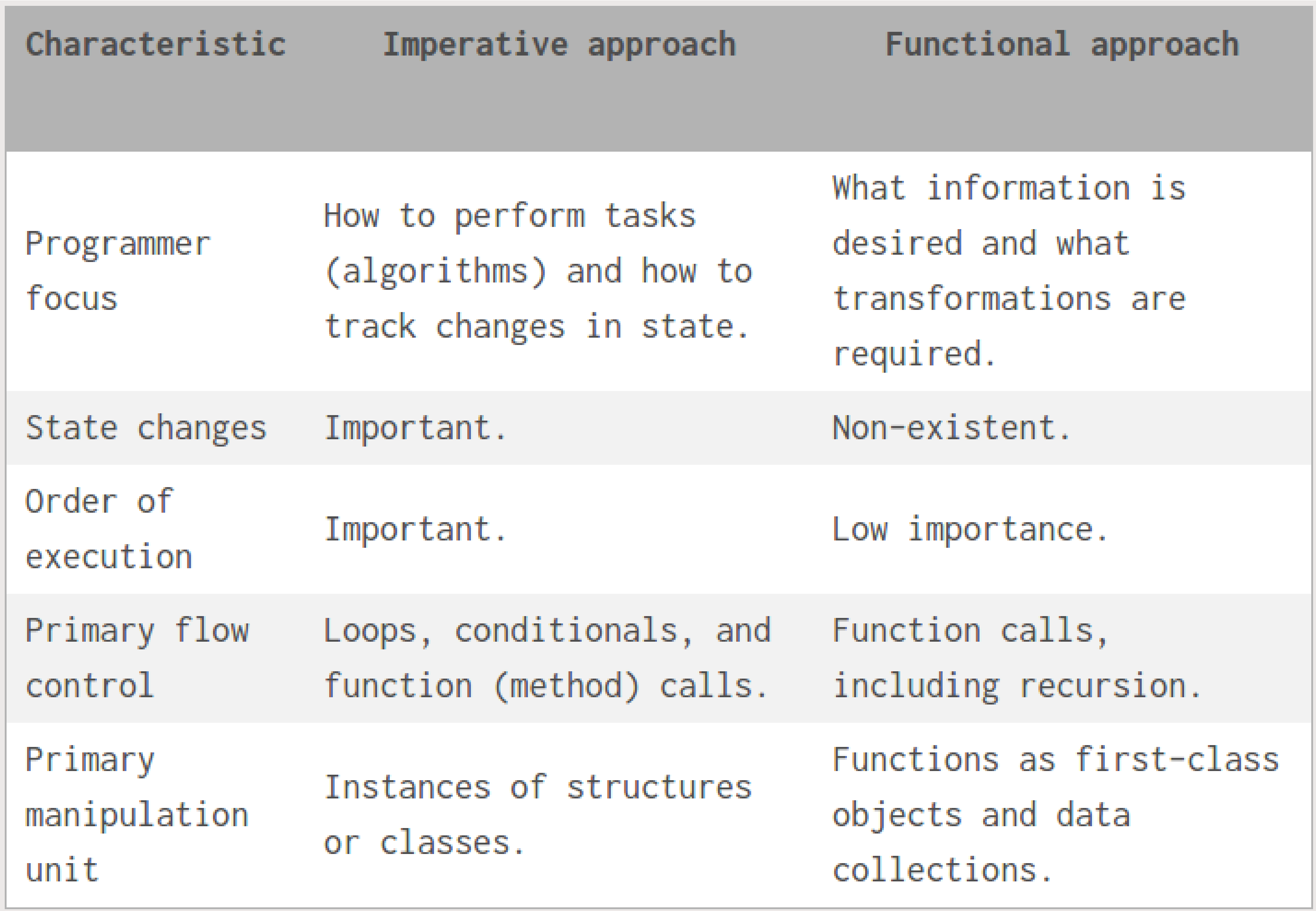
- Object Oriented Programming (OOP) is a paradigm in which real-world objects are each viewed as separate entities having their own state which is modified only by built in procedures, called methods.
- Objects are organized into classes, from which they inherit methods and equivalent variables. The object-oriented paradigm provides key benefits of reusable code and code extensibility.
- OOP paradigm enhances the code security and reusability

- Class: Blueprint of an object, Comprehensive datatype, Detailed definition of an object
- Object: Every thing in this world is an object. Object is a collection of attributes and behaviors

Object Oriented Programming Paradigm

Key Features/ Pillars of OOP

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism



Characteristic	Imperative approach	Functional approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.	What information is desired and what transformations are required.
State changes	Important.	Non-existent.
Order of execution	Important.	Low importance.
Primary flow control	Loops, conditionals, and function (method) calls.	Function calls, including recursion.
Primary manipulation unit	Instances of structures or classes.	Functions as first-class objects and data collections.