Object Oriented Programming (Using Python)

UNIT- I (Part-1)

Introduction to Object Oriented Programming Paradigms:

- Programming paradigms
- Advantages of OOP
- Comparison of OOP with Procedural Paradigms

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

Programming Paradigms

Programming Paradigms

- Paradigm means organization principle, also known as model.
- Programming paradigm/model is a style of building the structure and elements of computer programs.
- The following are some of the programming models:
  1. Imperative Programming Model
  2. Functional Programming Model
  3. Procedural Programming Model
  4. Event-Driven Programming Model
  5. Logic Programming Model
  6. Object-Oriented Programming Model

Imperative Programming Model

- Imperative programming is a programming paradigm where the program describes a sequence of steps to perform a computation.
- This paradigm is based on the concept of statements that change the program's state.
- Examples of imperative programming languages include C, C++, and Java.

Imperative Programming Model

Strengths:

- **Control:** Imperative programming provides a high level of control over the program's behavior, allowing the programmer to precisely specify the steps to perform a computation.
- **Efficiency:** Imperative programming languages are often compiled directly to machine code, which can result in high performance.
- **Flexibility:** Imperative programming allows the programmer to use low-level features, such as direct memory access, to implement performance-critical algorithms.
- **Familiarity:** Imperative programming is a widely-used programming paradigm and is familiar to most programmers.

Imperative Programming Model

Weaknesses:

- **Complexity:** Imperative programming can be complex and difficult to read and understand, especially for large programs with many interacting components.
- **Maintenance:** Imperative programming can be difficult to maintain due to its tight coupling between data and behavior.
- **State management:** Imperative programming requires the programmer to manage the state of the program manually, which can lead to bugs and errors.
- **Lack of abstraction:** Imperative programming can be limited in terms of abstraction, making it difficult to write reusable code and maintain a high level of modularity.

Functional Programming Model

- Functional Programming Model divides a problem into a set of functions. These functions provide the main source of logic in the program.
- Functions take input parameters and produce outputs.
- Examples of functional programming languages include Haskell, Lisp, and Scala.
- **Python** provides functional programming techniques like lambda, map, reduce and filter.
- In this model computation is treated as evaluation of mathematical functions.

Functional Programming Model

- Example: To get factorial value of a number or n^{th} Fibonacci number, we can use the following functions:

factorial (n) = 1	if n == 0
factorial (n) = n * factorial (n-1)	if n > 0
fibo (n) = 0	if n = 0
fibo (n) = 1	if n = 1
fibo (n) = fibo (n-2) + fibo (n-1)	if n > 0

- The output value of a function depends only on its arguments, so calling a function with the same value for an argument always produces the same result. As a result, it is a good fit for parallel execution.

Functional Programming Model

- No function can have any side effects on other variables (state remains unaltered).
- Functional programming model is often called a 'declarative' programming paradigm as programming is done with expressions or declarations instead of statements.

Functional Programming Model

Strengths:

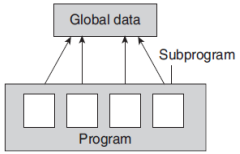
- **Immutability:** Functional programming encourages the use of immutable data structures, which can simplify program logic and make code easier to reason about.
- **Modularity:** Functional programming promotes modularity by allowing functions to be composed and reused across the codebase.
- **Parallelism:** Functional programming can be easily parallelized, allowing for efficient use of multi-core CPUs.
- **Declarative style:** Functional programming uses a declarative style, which focuses on the "what" rather than the "how" of computation. This can lead to more concise and understandable code.

Functional Programming Model

Weaknesses:

- **Complexity:** Functional programming can be complex and difficult to understand, especially for programmers who are used to imperative programming.
- **Performance:** Functional programming can have performance issues due to the need to create and manipulate many small objects.
- **Limited side effects:** Functional programming avoids side effects, which can limit the range of problems that can be solved using this paradigm.
- **Limited I/O:** Functional programming can be limited in terms of I/O capabilities, which can be a disadvantage for some software applications.

Procedural Programming Model



- Procedural programming model solves the problem by implementing one statement (a procedure) at a time. Thus it contains explicit steps that are executed in a specific order.
- It also uses functions, but these are not mathematical functions like the one used in functional programming.
- Functional programming focuses on expressions, where as procedural programming focuses on statements.
- The statements do not have values and instead modify the state of some conceptual machine.
- Examples of procedural programming languages include FORTRAN, Pascal, and C.

Procedural Programming Model

Strengths:

- **Simplicity:** Procedural programming is relatively simple and easy to understand, especially for programmers who are new to the field.
- **Performance:** Procedural programming can have good performance due to the lack of overhead associated with creating and manipulating objects.
- **Flexibility:** Procedural programming allows for flexible program structure and flow control.
- **Debugging:** Procedural programming can be easier to debug than other programming paradigms due to the lack of complex data structures.

Procedural Programming Model

Weaknesses:

- **Limited modularity:** Procedural programming can have limited modularity due to the lack of encapsulation and abstraction.
- **Limited reusability:** Procedural programming can have limited reusability due to the lack of encapsulation and abstraction.
- **Limited scalability:** Procedural programming can have limited scalability due to the lack of encapsulation and abstraction.
- **Limited expressiveness:** Procedural programming can be limited in terms of expressiveness, making it difficult to write complex software.

Event-Driven Programming Model

- Event-driven programming is a programming paradigm where the program responds to events such as user input, mouse clicks, and network requests.
- This paradigm is based on the concept of event handlers that are triggered by events.
- Examples of event-driven programming languages include JavaScript, Python, and Ruby.

Event-Driven Programming Model

Strengths:

- **Responsiveness:** Event-driven programming is highly responsive and can handle events as they occur in real-time.
- **Modular design:** Event-driven programming promotes a modular design, where different components of the program can respond to different events.
- **User-friendly:** Event-driven programming can be user-friendly because it can provide a graphical user interface (GUI) to interact with the program.
- **Scalability:** Event-driven programming can be highly scalable, especially in distributed systems where multiple events can be handled simultaneously.

Event-Driven Programming Model

Weaknesses:

- **Complexity:** Event-driven programming can be complex and difficult to understand, especially for programmers who are new to the paradigm.
- **Debugging:** Event-driven programming can be difficult to debug because of the non-linear flow of the program.
- **Performance:** Event-driven programming can have performance issues due to the need to constantly check for events and respond to them.
- **Difficulty in testing:** Event-driven programming can be difficult to test, especially for applications that require a high level of integration testing.

Logic Programming Model

- Logic programming is a programming paradigm where the program is composed of a set of logical rules and facts.
- In this paradigm, the program is based on the concept of inference rules that manipulate logical expressions.
- Examples of logic programming languages include Prolog, Mercury, and Oz.

Logic Programming Model

Strengths:

- **Declarative programming:** Logic programming is a declarative programming paradigm that allows programmers to focus on what they want to achieve rather than how to achieve it.
- **Natural language representation:** Logic programming languages such as Prolog use a natural language-like syntax that can make the programs easier to read and understand.
- **Non-determinism:** Logic programming can handle non-deterministic problems, where there are multiple possible solutions for a given problem.
- **Symbolic computation:** Logic programming can handle symbolic computation, which is useful for applications such as natural language processing and artificial intelligence.

Logic Programming Model

Weaknesses:

- **Limited applicability:** Logic programming is not well-suited for all types of problems, and may be less efficient than other programming paradigms for some tasks.
- **Limited expressiveness:** Logic programming can be limited in terms of expressiveness, making it difficult to write complex software.
- **Limited debugging:** Logic programming can be difficult to debug because the program execution is not always straightforward and can be influenced by the order in which rules are applied.
- **Limited performance:** Logic programming can have performance issues due to the need to perform extensive search operations to find solutions to problems.

Object-Oriented Programming Model

- This model mimics the real world by creating inside the computer a min-world of objects.
- A program divided into smaller units. These units are called classes and objects.
- In a University system objects can be Vice-Chancellor, Professor, non-teaching staff, students, courses, semesters, examinations, etc.
- Each object has a state (values) and behavior (interface/methods).
- Objects get state and behavior based on the class from which it created.
- Objects interact with one another by sending messages to each other, i.e. by calling each other's interface methods.
- Examples: C++, Java, Apex, Go, Move, Swift,.....etc.

Object-Oriented Programming Model

The striking features of OOP include the following

- The programs are data centered.
- Programs are divided in terms of objects and not procedures.
- Functions that operate on data are tied together with the data.
- Data is hidden and not accessible by external functions.
- New data and functions can be easily added as and when required.
- Follows a bottom-up approach for problem solving.

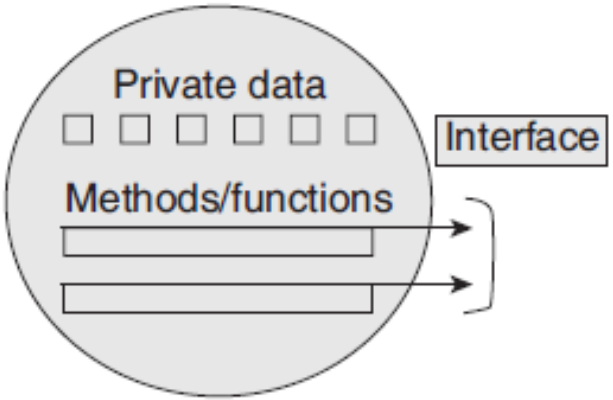
Classes, Objects, and Methods

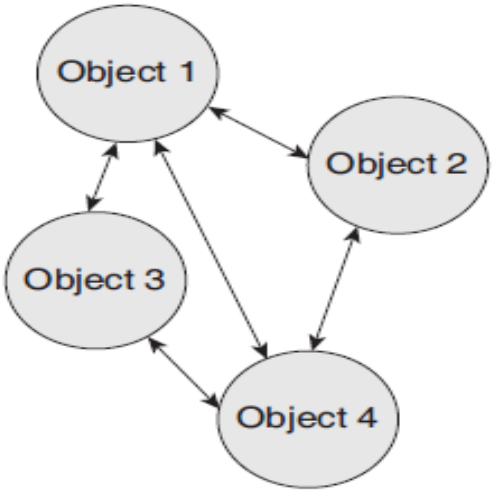
- A class is used to describe something in the world, such as occurrences, things, external entities, and so on.
- A class provides a template or a blueprint that describes the structure and behavior of a set of similar objects.
- Once we have the definition for a class, a specific instance of the class can be easily created.
- A class can have multiple instances or objects.
- Every object contains some data and procedures. They are also called methods.

Classes, Objects, and Methods

- A method is a function associated with a class. It defines the operations that the object can execute when it receives a message.
- In OOP language, only methods of the class can access and manipulate the data stored in an instance of the class (or object).
- Two objects can communicate with each other through messages. An object asks another object to invoke one of its methods by sending it a message.

Classes, Objects, and Methods





Object of a program interact by sending messages to each other

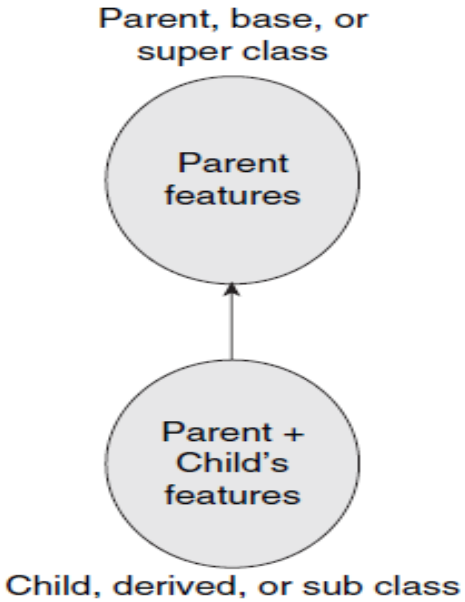
Inheritance

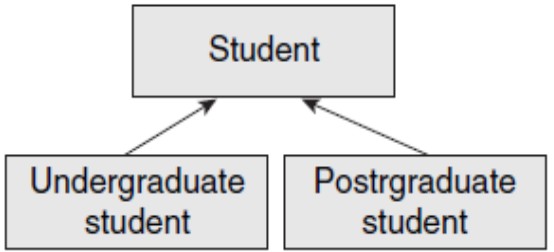
- Inheritance is a concept of OOP in which a new class is created from an existing class. The new class, often known as a sub-class, contains the attributes and methods of the parent class.
- The new class, known as sub-class or derived class, *inherits* the attributes and behavior of the pre-existing class, which is referred to as super-class or parent class.
- The inheritance relationship of sub- and super classes generates a hierarchy.
- Therefore, inheritance relation is also called 'is-a' relation.

Inheritance

- A sub-class not only has all the states and behaviors associated with the super-class but has other specialized features (additional data or methods) as well.
- The main advantage of inheritance is the ability to reuse the code.
- When we want a specialized class, we *do not have* to write the entire code for that class from scratch. We can inherit a class from a general class and add the specialized code for the sub-class.

Inheritance





Polymorphism, Containership and Reusability

- Polymorphism refers to having several different forms. It is related to methods.
- Polymorphism is a concept that enables the programmers to assign a different meaning or usage to a method in different contexts.
- Polymorphism can also be applied to operators.
- For example, we know that operators can be applied only on basic data types that the programming language supports.
- Therefore, `a + b` will give the result of adding `a` and `b`. If `a = 2` and `b = 3`, then `a + b = 5`. When we overload the `+` operator to be used with strings, then `Fraction1 + Fraction2` adds two fractional numbers and returns the result.

Polymorphism, Containership and Reusability

- Containership is the ability of a class to contain object(s) of one or more classes as member data.
- For example, class One can have an object of class Two as its data member. This would allow the object of class One to call the public functions of class Two. Here, class One becomes the container, whereas class Two becomes the contained class.
- Containership is also called composition.

Polymorphism, Containership and Reusability

- Reusability means developing codes that can be reused either in the same program or in different programs.
- Python gives due importance to building programs that are reusable.
- Reusability is attained through inheritance, containership, and polymorphism.

Data Abstraction and Encapsulation

- Data abstraction refers to the process by which data and functions are defined in such a way that only essential details are revealed and the implementation details are hidden.
- The main focus of data abstraction is to separate the interface and the implementation of a program.

Data Abstraction and Encapsulation

- Data encapsulation, also called data hiding, is the technique of packing data and functions into a single component (class) to hide implementation details of a class from the users.
- Users are allowed to execute only a restricted set of operations (class methods) on the data members of the class.
- Therefore, encapsulation organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.

Merits/Advantages of OOP Languages

Merits/Advantages of OOP Languages

- Elimination of redundant code through inheritance (by extending existing classes).
- Higher productivity and reduced development time due to reusability of the existing modules.
- Secure programs as data cannot be modified or accessed by any code outside the class.
- Real world objects in the problem domain can be easily mapped objects in the program.
- A program can be easily divided into parts based on objects.

Merits/Advantages of OOP Languages

- The data-centered design approach captures more details of a model in a form that can be easily implemented.
- Programs designed using OOP are expandable as they can be easily upgraded from small to large systems.
- Message passing between objects simplifies the interface descriptions with external systems.
- Software complexity becomes easily manageable.
- With polymorphism, behavior of functions, operators, or objects may vary depending upon the circumstances.

Merits/Advantages of OOP Languages

- Data abstraction and encapsulation hides implementation details from the external world.
- OOP enables programmers to write easily extendable and maintainable programs.
- OOP supports code reusability to a great extent.

Demerits/Disadvantages of OOP Languages

- Programs written using object oriented languages have greater processing overhead as they demand more resources.
- Requires more skills to learn and implement the concepts.
- Beneficial only for large and complicated programs.
- Even an easy to use software when developed using OOP is hard to be build.
- OOP cannot work with existing systems.
- Programmers must have a good command in software engineering and programming methodology.

Applications of OOP Languages

- Designing user interfaces such as work screens, menus, windows, and so on.
- Real-time systems
- Simulation and modelling
- Compiler design
- Client server system
- Object oriented databases
- Object oriented distributed database

Applications of OOP Languages

- Parallel programming
- Decision control systems
- Office automation systems
- Hypertext and hypermedia
- Computer-aided design (CAD) systems
- Computer-aided manufacturing (CAM) systems
- Computer animation

Applications of OOP Languages

- Developing computer games
- Artificial intelligence—expert systems and neural networks
- Networks for programming routers, firewalls, and other devices

Comparison of OOP with Procedural Paradigms

Comparison of OOP with Procedural Paradigms

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
Objects	Functions/Procedures
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of Access modifiers ‘public’, private’, protected’	Doesn’t use Access modifiers
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance
Modularity	Modular design through functions