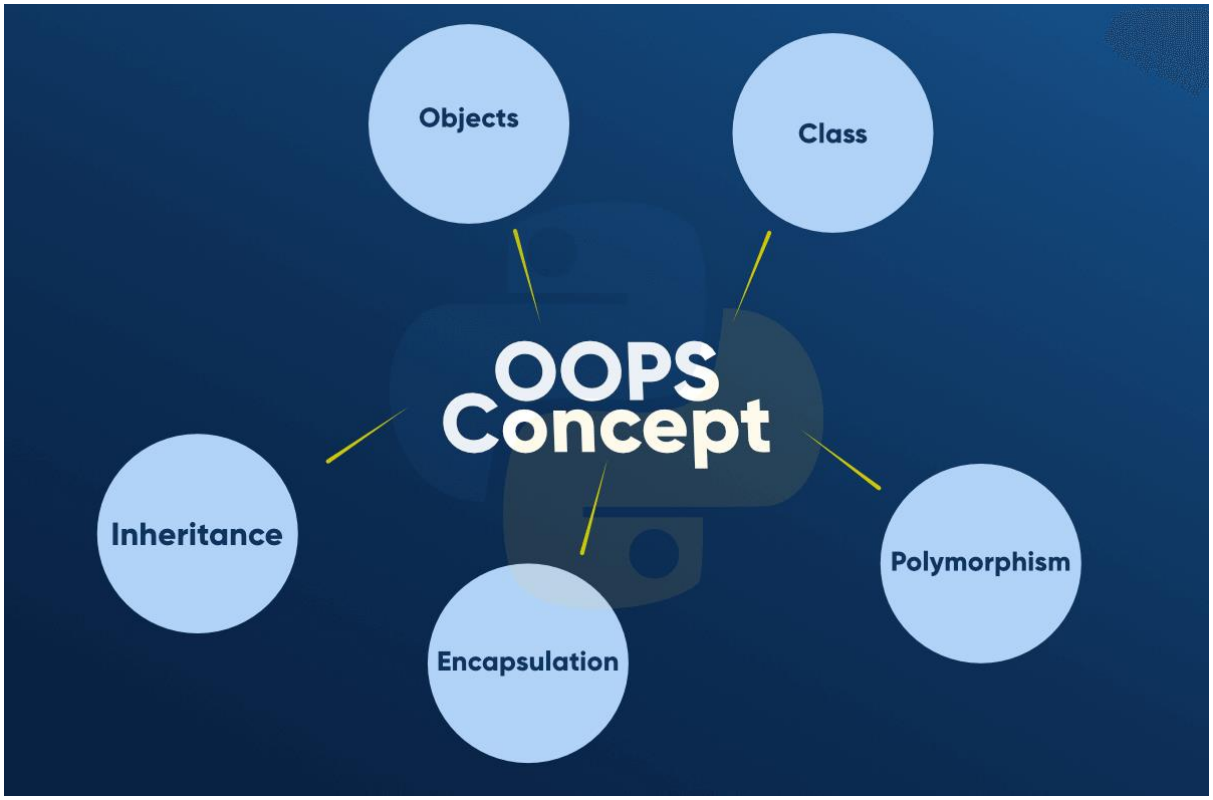
Python OOPs Concept

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Main Concepts of Object-Oriented Programming (OOPs)

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance



Class

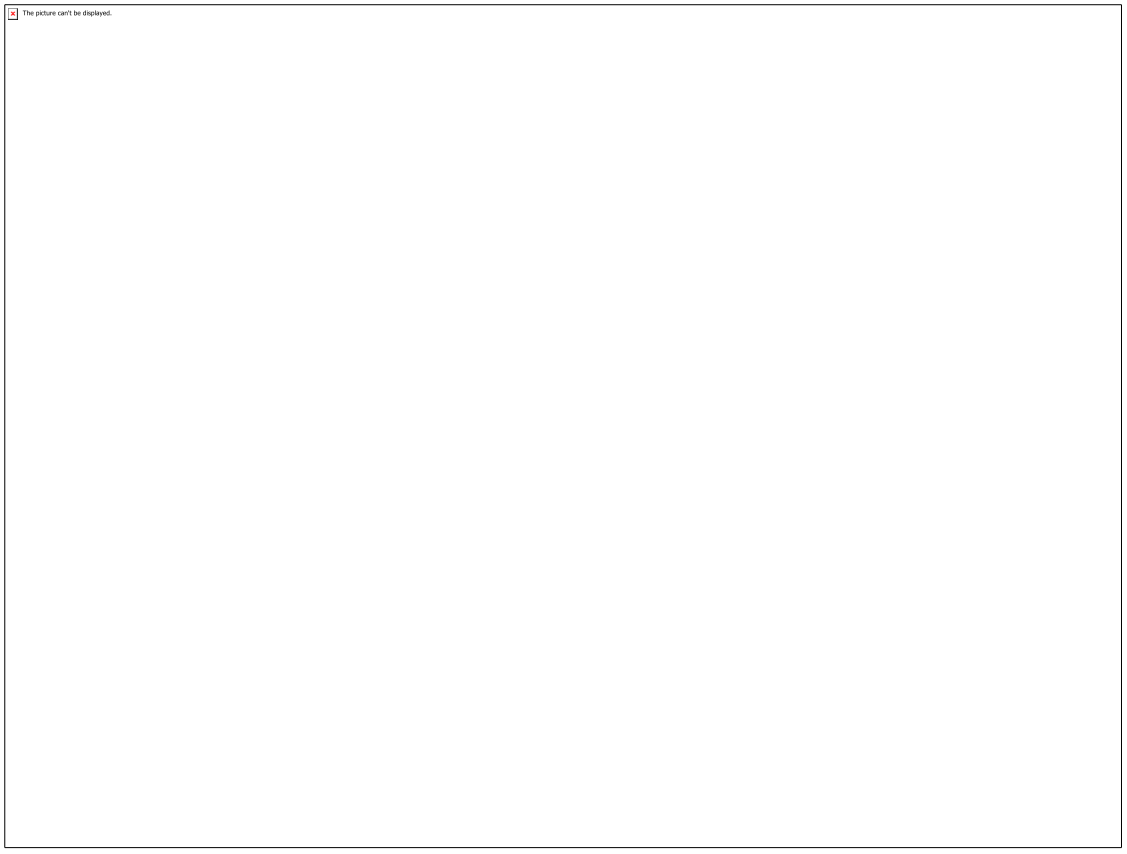
A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Objects

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object.

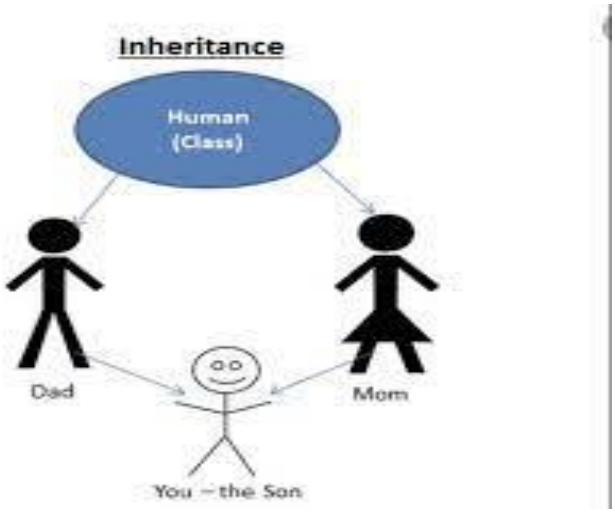


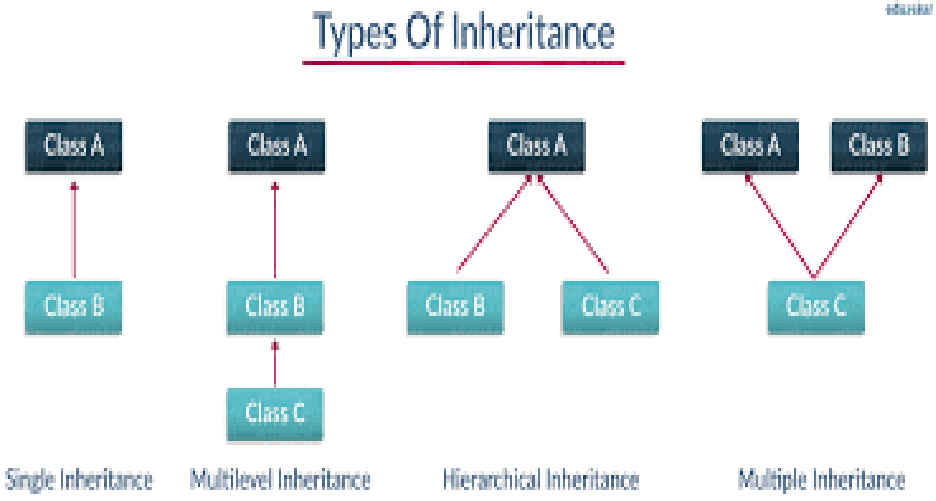
Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. The benefits of inheritance are:

- It represents real-world relationships well.
- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Example: Inheritance in Python

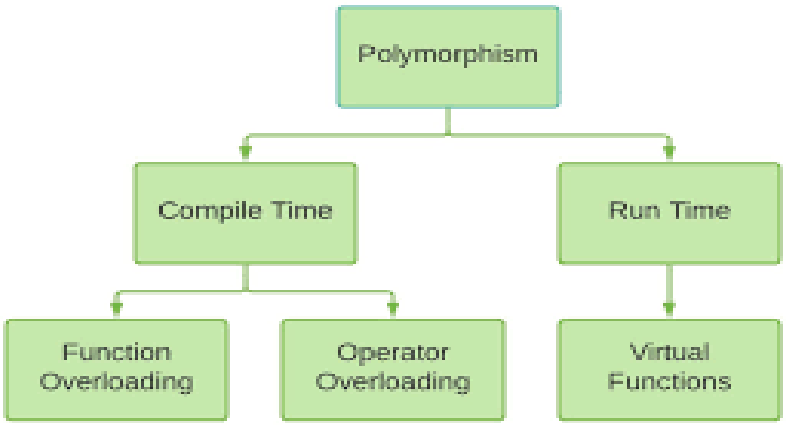


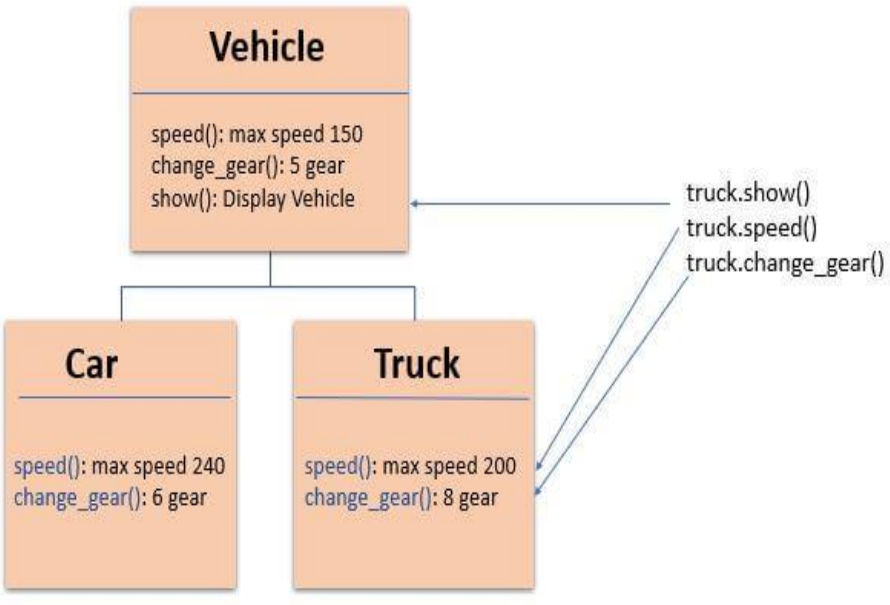


Polymorphism

Polymorphism simply means having many forms. For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

Example: Polymorphism in Python





Method overridden in Car and Truck class

Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

modifiers Access

Access modifiers limit access to the variables and functions of a class. Python uses three types of access modifiers; they are - *private*, *public* and *protected*.

Public members

Public members are accessible anywhere from the class. All the member variables of the class are by *default public*.

Protected members

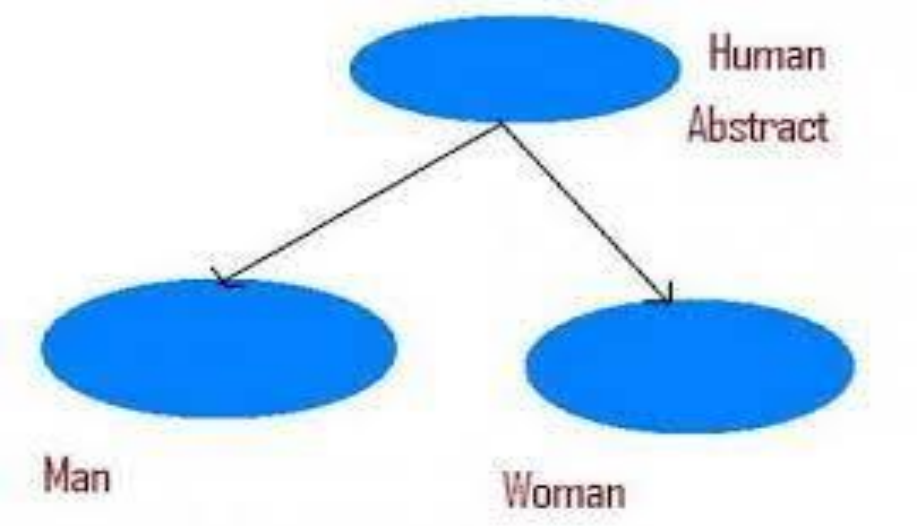
Protected members are accessible within the class and also available to its sub-classes. To define a protected member, prefix the member name with a *single underscore "_"*.

Private members

Private members are accessible within the class. To define a private member, prefix the member name with a *double underscore "__"*.

Abstraction

Abstraction in python is defined as a process of handling complexity by hiding unnecessary information from the user. This is one of the core concepts of object-oriented programming (OOP) languages. That enables the user to implement even more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden background/back-end complexity.



Importance:

Abstraction provides a programmer to hide all the irrelevant data/process of an application in order to reduce complexity and increase the efficiency of the program. Now, we can start learning how we can achieve abstraction using the Python program.

Achieving Abstraction in Python:

In Python, abstraction can be achieved by having/using abstract classes and methods in our programs.