1. What Is Object-Oriented Programming in Python?

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects.

For instance, an **object** could represent a person with properties like a name, age, and address and behaviors such as walking, talking, breathing, and running. Or it could represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.

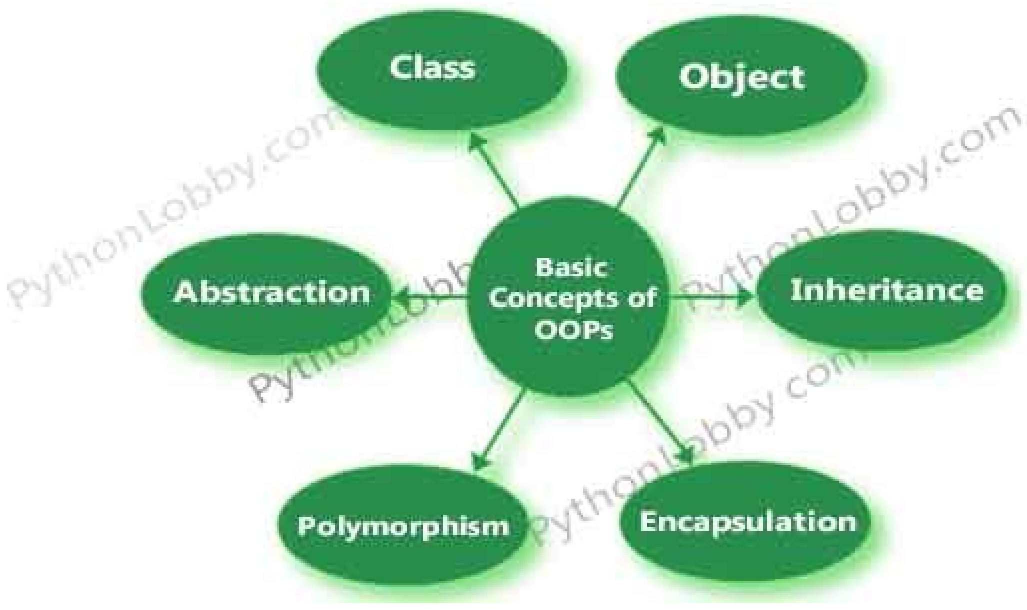
Put another way, object-oriented programming is an approach for modeling concrete, real-world things, like cars, as well as relations between things, like companies and employees, students and teachers, and so on. OOP models real-world entities as software objects that have some data associated with them and can perform certain functions.

Classes vs Instances

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

A class is a blueprint for how something should be defined. It doesn't actually contain any data. The Dog class specifies that a name and an age are necessary for defining a dog, but it doesn't contain the name or age of any specific dog.

While the class is the blueprint, **an instance** is an object that is built from a class and contains real data. An instance of the Dog class is not a blueprint anymore. It's an actual dog with a name, like Miles, who's four years old.



- **<u>Inheritance</u>**

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.

Types of Inheritance –

Single Inheritance:

Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.

Multilevel Inheritance:

Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

Hierarchical Inheritance:

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

Multiple Inheritance:

Multiple level inheritance enables one derived class to inherit properties from more than one base class.

- **<u>Polymorphism</u>**

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.

-Method overloading: in same class, method names are same, last function will be called (LIFO)

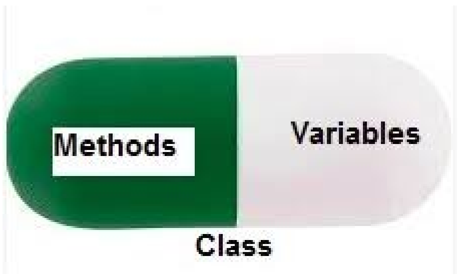
-Method overriding: same method and argument but different class. When you required same method many times...

- <u>**Encapsulation**</u>

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.

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

Encapsulation in Python



- <u>**Data Abstraction**</u>

It hides the unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

Data Abstraction in Python can be achieved through creating abstract classes.

- **<u>Advantages of using OOP</u>**

1. It helps us to write clean code which resembles a real world problem's solution. Thus, O.O.P. helps us to solve real world problems in much easier way.
2. It boosts easy maintenance, reusability and update the existing code base.
3. Partial updates are easy to do. In other hand, modularity helps us to add or improvise existing functionalities and check their dependencies in much easier way.
4. Frameworks are built around the OOP structure which helps us to render any component into our codebase without writing the entire structure of the framework. We can just create objects and use their methods to accomplish our task.