TECHWINGSYS™

5th Floor, Usnaz Tower, Church Landing Rd, near Medical Trust Hospital, Pallimukku, Kochi, Kerala 682016

TECHWINGSYS

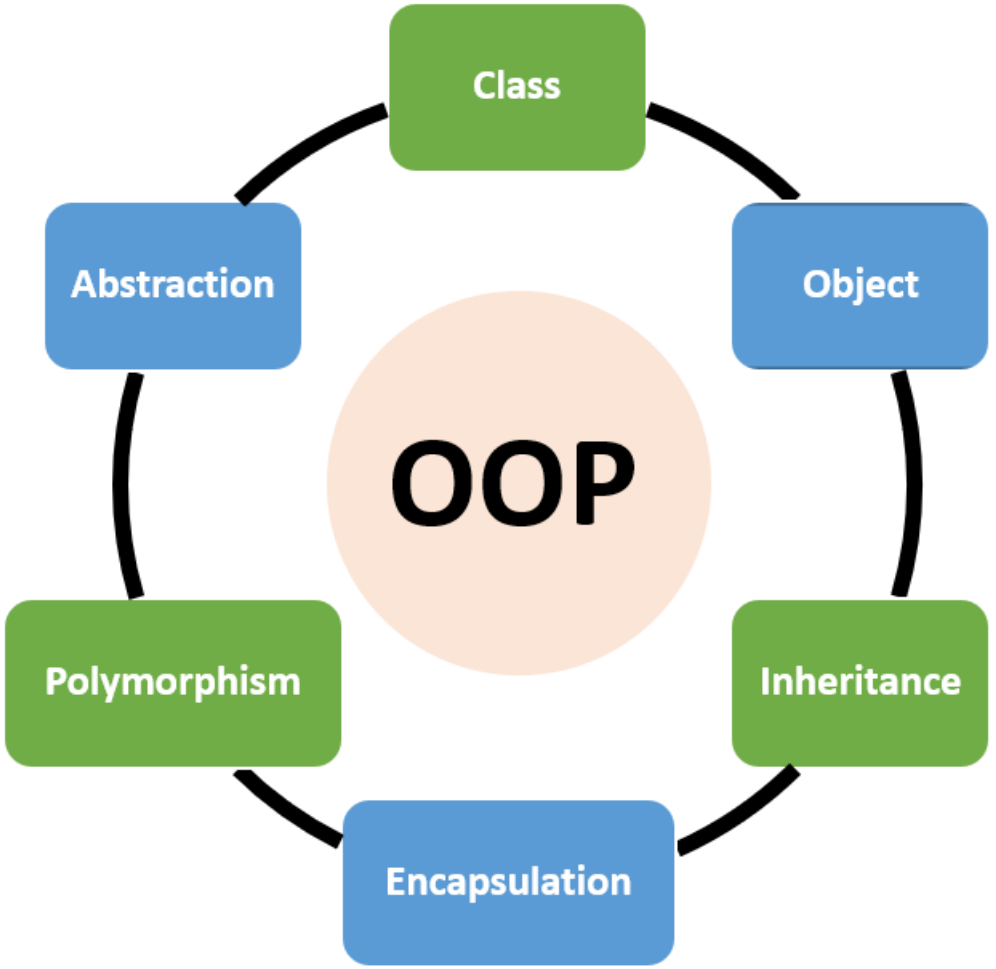
Object-oriented programming (OOP)

TECHWINGSYS

What is an OOPs in Python?

Object-Oriented Programming (OOPs) is a programming paradigm based on the concept of objects. It is a way of organizing and structuring code that allows for code reuse and efficient problem solving. In Python, OOPs is used to create classes and objects that can be used to store and manipulate data.

Python is an object-oriented language, meaning it allows users to create and use objects. Objects are defined by their attributes and behaviors, and can interact with other objects. This makes it easier to develop complex applications with fewer lines of code.

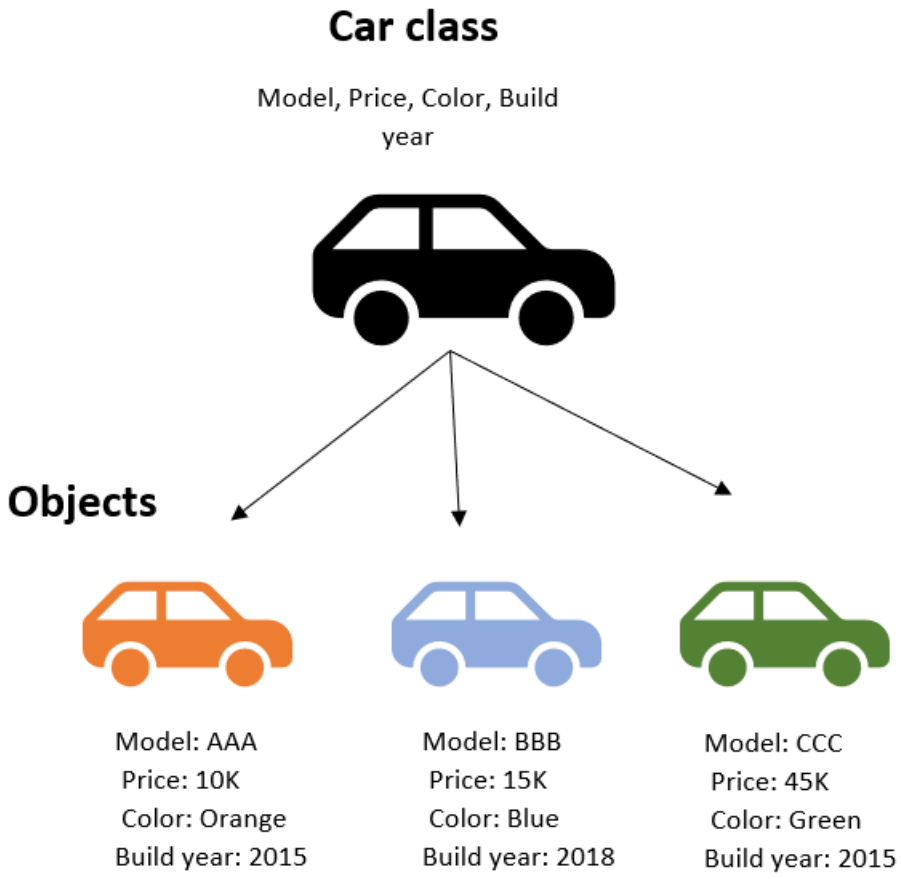


TECHWINGSYS

Examples of OOPs in Python

One example of OOPs in Python is creating classes and objects. A class is a template for creating objects, and objects are instances of a class. Objects can contain data, such as attributes, and methods, which are functions that can be used to manipulate the data.

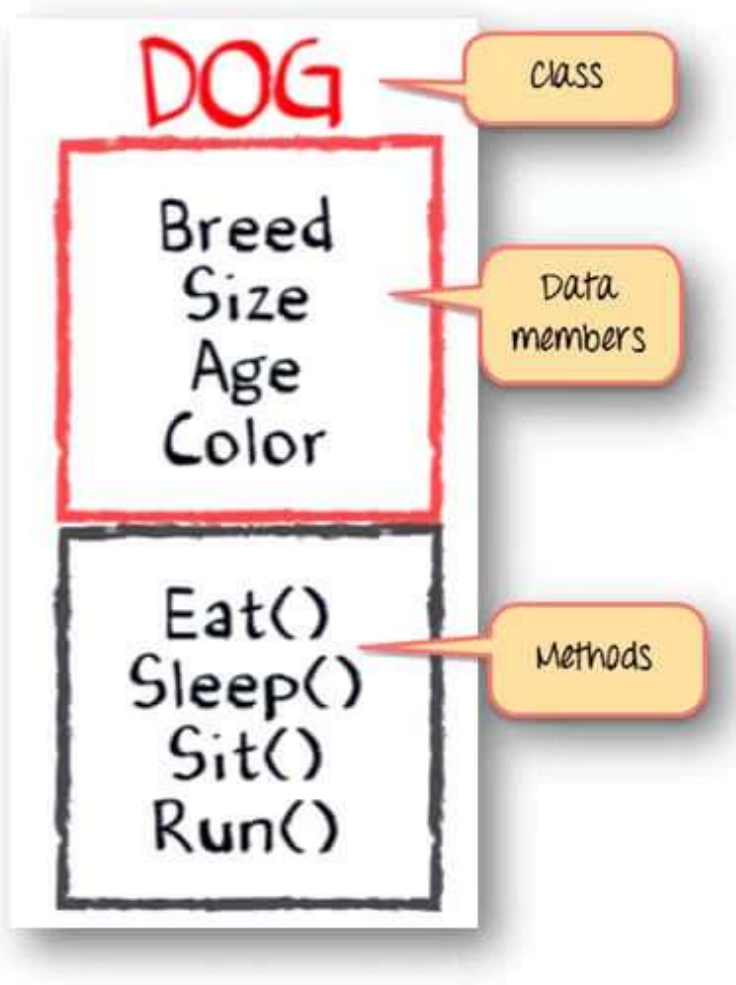
Another example of OOPs in Python is creating inheritance. Inheritance allows one class to inherit the attributes and methods of another class, which makes it easier to create complex applications with fewer lines of code.



Class

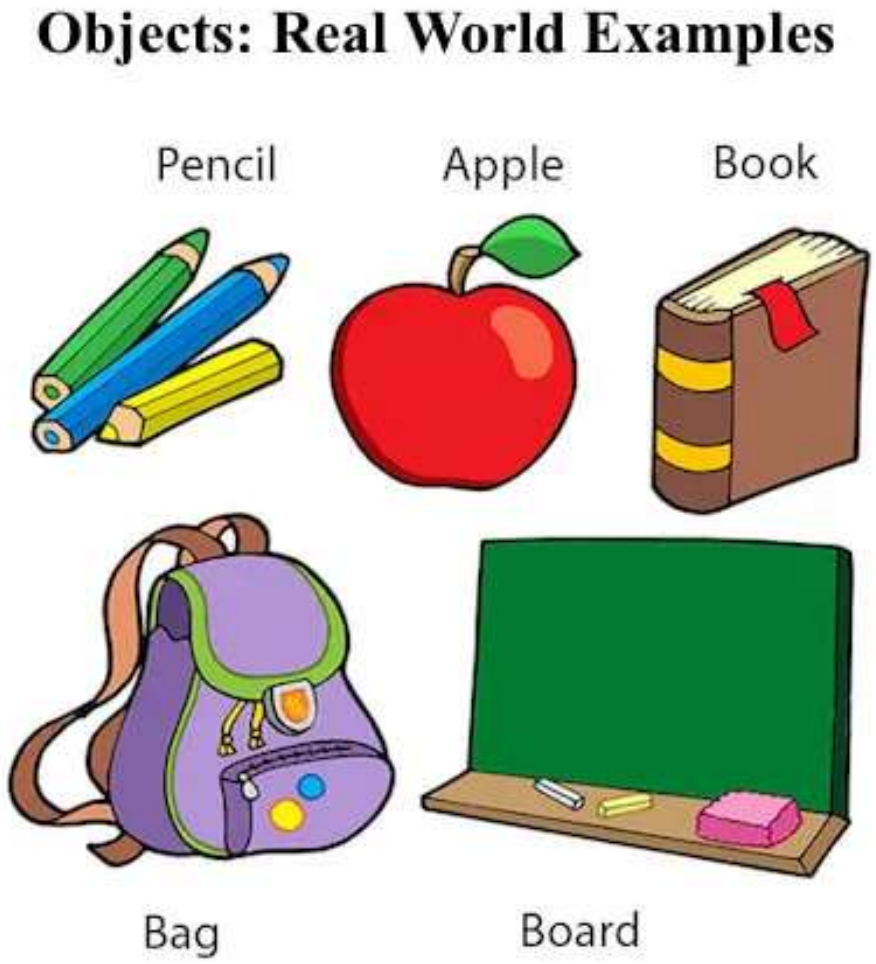
A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.



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. The number 12 is an object, the string “Hello, world” is an object, a list is an object that can hold other objects, and so on.



```
class employee:
    name="rahul"
    designation="SE"
    salary=30000
    def fun1(self):
        print("Employee Name: ",self.name)
        print("Designation: ",self.designation)
        print("Salary: ",self.salary)
obj=employee()
obj.fun1()
```

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.

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.

TECHWINGSYS

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
    def start(self):
        print("The vehicle is starting.")

class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.num_doors = num_doors
    def drive(self):
        print(f"The {self.make} {self.model} is driving.")

class Motorcycle(Vehicle):
    def __init__(self, make, model, year, num_wheels):
        super().__init__(make, model, year)
        self.num_wheels = num_wheels
    def ride(self):
        print(f"The {self.make} {self.model} is riding.")


car = Car("Toyota", "Corolla", 2022, 4)
car.start()
car.drive()

motorcycle = Motorcycle("Harley-Davidson", "Sportster", 2021, 2)
motorcycle.start()
motorcycle.ride()
```

TECHWINGSYS

Abstraction in OOP with Python

Abstraction is a key concept in object-oriented programming (OOP) that allows you to model complex systems and interactions using simpler, high-level representations. It involves hiding the internal details of a system from the user and exposing only the relevant information and functionality. Data Abstraction in Python can be achieved through creating abstract classes.

Abstraction in Python allows for code to be organized in a logical manner and makes it easier to understand. It also allows for the creation of abstract classes, which can be used to create new objects. This makes the code more efficient, as it can be used in different scenarios and can be easily modified when needed.

Polymorphism in Python

Polymorphism is a concept in OOPs that allows objects to have different behaviors depending on the context. In Python, polymorphism is used to create objects that can be used in different ways.

1.Overloading:

Method overloading allows a class to define multiple methods with the same name but with different parameters. This means that methods with the same name can behave differently depending on the number or type of arguments passed to them. In Python, method overloading can be achieved by defining methods with different numbers or types of arguments.

2.Overriding:

Method overriding occurs when a subclass provides a different implementation for a method that is already defined in its parent class. This allows the subclass to provide a specialized behavior that is specific to the subclass. In Python, method overriding is achieved by defining a method with the same name and signature in the subclass.

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.