Object-Oriented Programming (OOP) Concepts in Python

## 1. Class and Object

A class is a blueprint for creating objects. An object is an instance of a class.

class Person:  
 def \_\_init\_\_(self, name, age): # constructor  
 self.name = name  
 self.age = age  
  
 def greet(self): # method  
 print(f"Hello, my name is {self.name} and I am {self.age} years old.")  
  
# Creating an object  
person1 = Person("Alice", 30)  
person1.greet()

## 2. Inheritance

Inheritance allows a class to inherit attributes and methods from another class.

class Animal:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def speak(self):  
 print(f"{self.name} makes a sound.")  
  
class Dog(Animal):  
 def speak(self):  
 print(f"{self.name} barks.")  
  
dog = Dog("Buddy")  
dog.speak()

## 3. Encapsulation

Encapsulation means restricting access to methods and variables. This is done using access modifiers.

class BankAccount:  
 def \_\_init\_\_(self, balance):  
 self.\_\_balance = balance # private variable  
  
 def deposit(self, amount):  
 self.\_\_balance += amount  
  
 def get\_balance(self):  
 return self.\_\_balance  
  
account = BankAccount(1000)  
account.deposit(500)  
print(account.get\_balance())

## 4. Polymorphism

Polymorphism allows the same method to behave differently in different classes.

class Bird:  
 def fly(self):  
 print("Bird can fly")  
  
class Penguin(Bird):  
 def fly(self):  
 print("Penguins can't fly")  
  
def flying\_test(bird):  
 bird.fly()  
  
b = Bird()  
p = Penguin()  
  
flying\_test(b)  
flying\_test(p)

## 5. Abstraction

Abstraction hides complex implementation details and shows only the essential features.

from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
 @abstractmethod  
 def start\_engine(self):  
 pass  
  
class Car(Vehicle):  
 def start\_engine(self):  
 print("Car engine started.")  
  
car = Car()  
car.start\_engine()