1 Create a Class and Object Write a Python program to create a class Car with attributes brand, model, and year. Then, create an object of the class and print its attributes.

```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

c = Car("Toyota", "Corolla", 2022)
print(c.brand, c.model, c.year)

```

```
Toyota Corolla 2022
```

2 Class with Methods Define a class Rectangle with attributes length and width. Include a method area() to calculate and return the area of the rectangle. Expected Output: r = Rectangle(5, 10) print(r.area()) # Output: 50

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

r = Rectangle(5, 10)
print(r.area())  # Output: 50

```

```
50
```

3 Using Constructor (**init**) Create a class Person with a constructor that initializes name and age. Instantiate an object and print the details.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("John", 25)
print(p.name, p.age)

```

```
John 25
```

4 Inheritance Create a class Animal with a method speak(), which prints "Animal speaks". Create a subclass Dog that overrides speak() to print "Dog barks".

Expected Output: a = Animal() a.speak() # Output: Animal speaks d = Dog() d.speak() # Output: Dog barks

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

a = Animal()
a.speak()  # Output: Animal speaks

d = Dog()
d.speak()  # Output: Dog barks

```

```
Animal speaks
Dog barks
```

5 Default and Parameterized Constructor Create a class Book with a default constructor that assigns "Unknown" to the title, and a parameterized constructor to initialize it with a given value.

Example: b1 = Book() print(b1.title) # Output: Unknown b2 = Book("Python Programming") print(b2.title) # Output: Python Programming

```
class Book:
    def __init__(self, title="Unknown"):
        self.title = title

b1 = Book()
print(b1.title)  # Output: Unknown

b2 = Book("Python Programming")
print(b2.title)  # Output: Python Programming

```

```
Unknown
Python Programming
```

6 Encapsulation (Private Variables) Create a class BankAccount with a private attribute __balance. Provide methods deposit(amount) and get_balance() to access it safely.

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance

account = BankAccount()
account.deposit(1000)
print(account.get_balance())  # Output: 1000

```

```
1000
```

7 Class with Multiple Objects Create a class Student with attributes name and grade. Create multiple student objects and print their details.

Example: s1 = Student("Alice", "A") s2 = Student("Bob", "B") print(s1.name, s1.grade) # Output: Alice A print(s2.name, s2.grade) # Output: Bob B

Generate print hello world using rot13 Close

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

s1 = Student("Alice", "A")
s2 = Student("Bob", "B")

print(s1.name, s1.grade)  # Output: Alice A
print(s2.name, s2.grade)  # Output: Bob B

```

```
Alice A
Bob B
```

Double-click (or enter) to edit

8 Polymorphism with Method Overriding Create a base class Shape with a method draw(). Create subclasses Circle and Square that override draw() to print "Drawing Circle" and "Drawing Square".

Expected Output: s1 = Circle() s2 = Square() s1.draw() # Output: Drawing Circle s2.draw() # Output: Drawing Square

```
class Shape:
    def draw(self):
        print("Drawing Shape")

class Circle(Shape):
    def draw(self):
        print("Drawing Circle")

class Square(Shape):
    def draw(self):
        print("Drawing Square")

s1 = Circle()
s2 = Square()

s1.draw()  # Output: Drawing Circle
s2.draw()  # Output: Drawing Square

```

```
Drawing Circle
Drawing Square
```

Double-click (or enter) to edit

9 Simple Getter and Setter Methods Create a class Employee with a private attribute __salary. Provide get_salary() and set_salary(amount) methods to access and modify it safely.

```
class Employee:
    def __init__(self, salary):
        self.__salary = salary

    def get_salary(self):
        return self.__salary

    def set_salary(self, amount):
        if amount > 0:
            self.__salary = amount

e = Employee(5000)
print(e.get_salary())  # Output: 5000
e.set_salary(7000)
```

```
print(e.get_salary())  # Output: 7000

```

```
5000
7000
```

10 Abstract Class (Using ABC module) Create an abstract class Vehicle with an abstract method start(). Implement subclasses Car and Bike that define start() differently.

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Car starts with a key")

class Bike(Vehicle):
    def start(self):
        print("Bike starts with a button")

c = Car()
b = Bike()

c.start()  # Output: Car starts with a key
b.start()  # Output: Bike starts with a button

```

```
Car starts with a key
Bike starts with a button
```