# Class:

"""In Object-Oriented Programming (OOP), a class is a blueprint or template for creating objects (instances).

It defines the properties (called attributes or fields) and behaviors (called methods) that the objects created from the class will have.

A class encapsulates data and functionality together."""

# Attributes:

"""Attributes (Fields): These are variables that hold data related to the object.

They represent the state or properties of the object."""

# Method:

"""Methods: These are functions defined inside the class that operate on the object.

They define the behavior or actions the object can perform."""

# Object:

"""Objects (Instances): Objects are individual instances of a class.

Each object has its own copy of the class's attributes and methods."""

# Constructor:

"""Constructor (\_\_init\_\_ method): This special method is called when an object is created from the class

and is used to initialize the object's attributes."""

# Instance variables:

"""Instance variables are defined in the class constructor (the \_\_init\_\_() method), usually with the self keyword.

They are used to store the state or data unique to each object of the class.

You can access and modify instance variables through an object, but they are not shared among other instances of the same class."""

#Instance method:

"""Takes self as the first parameter: This refers to the instance of the class and allows access to instance variables and other methods.

Can modify instance variables: Instance methods can read and modify the state of the object (the instance variables).

Called on an object: You need to create an instance of the class to call the instance method."""

"""1. Create a class called Person with attributes name and age. Define a method to display the person's details.

Instantiate an object and print the details."""

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display(self):

print(f"Name is {self.name} and age is {self.age}")

# Instantiate an object of the Person class

x = Person("shreya", 30)

# Use the display method to print the details

x.display()

"""2. Write a class Car with attributes make, model, and year.

Use a constructor to initialize these attributes when creating an object."""

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

x = Car("Hatchback", "Glanza", 2020)

print(x.make)

print(x.model)

print(x.year)

"""3. Create a class BankAccount with attributes account\_holder and balance.

Include an instance method deposit() to add money to the balance. Create an object and perform a deposit operation. """

"""The class name should follow the convention of using BankAccount without the underscore.

It is better to make the deposit method accept an amount as an argument instead of relying on user input within the method, for better flexibility.

In this version, the deposit amount is passed as an argument to deposit() instead of taking input within the method."""

class Bank\_Account:

def \_\_init\_\_(self, account\_holder, balance):

self.account\_holder = account\_holder

self.balance = balance

def deposit(self, amount):

self.balance += amount

print(f"Deposited {amount} into your account. Your new balance is {self.balance}")

x = Bank\_Account("shreya", 2000)

x.deposit(1000)

"""4. Define a class Student with instance variables name and grade. Create an instance method

change\_grade() to modify the grade value for the student object. """

class Student:

def \_\_init\_\_(self, name, grade):

self.name = name

self.grade = grade

def change\_grade(self):

self.grade = input("Enter the new grade: ")

print(f" Grade changed to {self.grade}")

x = Student("shreya", "B")

x.change\_grade()

"""5. Create a class Book with attributes title and author. Write a constructor to initialize these attributes.

Create two objects of Book and print their details."""

class Book:

def \_\_init\_\_(self, title, author):

self.title = title

self.author = author

b1 = Book("Lord of Rings", "Tolkien")

b2 = Book("Harry Potter", "Rowling")

print(f" Book1: title is {b1.title} and author is {b1.author}")

print(f" Book2: title is {b2.title} and author is {b2.author}")

"""6. Define a class Laptop with attributes brand and price. Create three

objects for different laptop brands and display their prices using an instance method. """

class Laptop:

def \_\_init\_\_(self, brand, price):

self.brand = brand

self.price = price

def display\_price(self):

print(f" price of Laptop of brand {self.brand} is {self.price}")

l1 = Laptop("Lenovo", 35000)

l2 = Laptop("HP", 45000)

l3 = Laptop("Acer", 25000)

l1.display\_price()

l2.display\_price()

l3.display\_price()

"""7. Create a class Employee with attributes name and salary. Assign an object to two different

reference variables and demonstrate that changes through one reference reflect in the other."""

class Employee:

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

e1 = Employee("shreya", 50000)

e2 = e1

e2.salary = 60000

print(f" Employee1 name is {e1.name} and salary is {e1.salary}")

print(f" Employee2 name is {e2.name} and salary is {e2.salary}")

"""8. Write a class Rectangle with attributes length and width. Include methods to calculate the area and perimeter.

Instantiate an object and calculate the area and perimeter. """

class Rectangle:

def \_\_init\_\_(self, length, width):

self.length = length

self.width = width

def method(self):

area = self.length \* self.width

perimeter = 2 \* (self.length + self.width)

return area, perimeter

r1 = Rectangle(5,6)

area, perimeter = r1.method() # The method returns both the area and perimeter, which are then stored in the variables area and perimeter.

print(f" Area of Rectangle is {area} and Perimeter is {perimeter}")

"""9. Create a class Phone with attributes brand and battery\_life. Write a method use\_phone() that

reduces the battery life by 10%. Create an object and use this method to show changes in battery\_life."""

class Phone:

def \_\_init\_\_(self, brand, battery\_life):

self.brand = brand

self.battery\_life = battery\_life

def use\_phone(self):

self.battery\_life -= (self.battery\_life \* 0.1)

print(f" brand is {self.brand} and reduced battery life is {self.battery\_life:.2f}") # The :.2f in the print statement formats the battery life to two decimal places for better readability.

x = Phone("Oneplus", 100)

x.use\_phone()

"""10.Define a class House with an attribute rooms. Create multiple objects of the House class and

demonstrate the object references by printing the memory addresses of each object. """

# id():

"""The id() function returns the memory address of the object, which helps demonstrate that each object is distinct in memory.

Each object (h1, h2, and h3) has a different memory address, confirming they are separate instances."""

class House:

def \_\_init\_\_(self, rooms):

self.rooms = rooms

h1 = House(4)

h2 = House(3)

h3 = House(2)

print(f" memory address of house1 is {id(h1)} of {h1.rooms} rooms")

print(f" memory address of house2 is {id(h2)} of {h2.rooms} rooms")

print(f" memory address of house3 is {id(h3)} of {h3.rooms} rooms")

"""11.Write a class Employee with attributes name and salary, where salary has a default value.

Create objects with and without the salary value and print their attributes. """

class Employee:

def \_\_init\_\_(self, name, salary=25000):

self.name = name

self.salary = salary

e1 = Employee("shreya", 50000)

e2 = Employee("phani")

print(e1.name)

print(e1.salary)

print(e2.name)

print(e2.salary)

"""12.Create a class Circle with attributes radius. Write an instance method calculate\_area() that

returns the area of the circle. Create an object and call the method to get the result."""

import math

class Circle:

def \_\_init\_\_(self, radius):

self.radius = radius

def calculate\_area(self):

area = math.pi \* (self.radius \*\* 2)

return area

c = Circle(5)

area = c.calculate\_area()

print(f"area of a circle is {area:.2f}")

"""13.Define a class Dog with attributes name and energy\_level. Write methods to increase and decrease the energy\_level.

Create an object and manipulate the energy level using the methods."""

class Dog:

def \_\_init\_\_(self, name, energy\_level):

self.name = name

self.energy\_level = energy\_level

def increase\_energy(self):

self.energy\_level += 10

print(f" {self.name} increased energy level is {self.energy\_level}")

def decrease\_energy(self):

self.energy\_level -= 5

print(f" {self.name} energy level is decreased to {self.energy\_level}")

d = Dog("bonky", 20)

# Manipulate the energy level using the methods

d.increase\_energy()

d.decrease\_energy()

"""14.Create two classes Person and Job. The Person class has a name and the Job class has a job\_title.

Write a method that takes a Person object and a Job object and assigns the job title to the person."""

class Person:

def \_\_init\_\_(self,name):

self.name = name

class Job:

def \_\_init\_\_(self, job\_title):

self.job\_title = job\_title

"""15.Write a class Product with attributes name and price. In the constructor,

validate that the price is a positive number. Raise a ValueError if the

price is negative. Create objects with valid and invalid prices."""

class Product:

def \_\_init\_\_(self, name, price):

if price > 0:

self.name = name

self.price = price

else:

raise ValueError("price must be positive number")

p1 = Product("pen", 10)

p3 = Product("pencil", 5)

print(f"{p1.name} and {p1.price}")

print(f"{p3.name} and {p3.price}")

try:

p2 = Product("eraser", -10)

except ValueError as e:

print(e)