class Book:

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

self.title = title

self.author = author

self.isbn = isbn

class Library:

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

self.books = []

def add\_book(self, book):

self.books.append(book)

def display\_books(self):

for book in self.books:

print(f"Title: {book.title}, Author: {book.author}, ISBN: {book.isbn}")

b=Book("Atomic Habits","James Clear","isbn")

l=Library()

l.add\_book(b)

l.display\_books()

Exercise 2: Banking System

Scenario: Design a banking system to manage customer accounts.

Question: Create a BankAccount class with methods to deposit, withdraw, and check balance.

Create a Customer class that can have multiple BankAccount objects. Write methods to display

customer information along with their accounts.

class BankAccount:

def \_\_init\_\_(self, balance=0):

self.balance = balance

def deposit(self, amount):

self.balance += amount

return self.balance

def withdraw(self, amount):

if self.balance >= amount:

self.balance -= amount

return self.balance

else:

return "Insufficient funds"

def check\_balance(self):

return self.balance

class Customer:

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

self.name = name

self.account = account

def display\_customer\_info(self):

print(f"Name: {self.name}")

print("Accounts:")

for account in self.account:

print(f"Account Balance: {account.check\_balance()}")

BA=BankAccount()

BA.deposit(1000)

BA.withdraw(500)

BA.check\_balance()

C=Customer("John",[BA])

C.display\_customer\_info()

Exercise 3: E-commerce System

Scenario: You are developing an e-commerce platform.

Question: Create a Product class with attributes like name, price, and stock quantity. Create a

ShoppingCart class that can hold multiple Product objects. Implement methods to add products

to the cart and calculate the total price.

class Product:

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

self.name = name

self.price = price

self.stock\_quantity = stock\_quantity

class ShoppingCart:

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

self.products = []

def add\_product(self, product, quantity):

self.products.append((product, quantity))

def calculate\_total\_price(self):

total\_price = 0

for product, quantity in self.products:

total\_price += product.price \* quantity

return total\_price

P=Product("Apple",100,10)

S=ShoppingCart()

S.add\_product(P,2)

S.calculate\_total\_price()

Exercise 4: Vehicle Registration System

Scenario: Develop a vehicle registration system.

Question: Define a Vehicle class with attributes such as make, model, and year. Create a

Registration class that links multiple Vehicle objects to a single owner. Write methods to

display vehicle details and owner information.

class Vehicle:

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

self.make = make

self.model = model

self.year = year

class Registration:

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

self.owner = owner

self.vehicles = []

def add\_vehicle(self, vehicle):

self.vehicles.append(vehicle)

def display\_vehicle\_details(self):

print(f"Owner: {self.owner}")

for vehicle in self.vehicles:

print(f"Make: {vehicle.make}, Model: {vehicle.model}, Year: {vehicle.year}")

V=Vehicle("Toyota","Camry",2022)

R=Registration("John")

R.add\_vehicle(V)

R.display\_vehicle\_details()

Exercise 5: School Management System

Scenario: Create a school management system to manage students and courses.

Question: Define a Student class with attributes for name and age. Create a Course class that

can hold multiple Student objects. Write methods to enroll students in courses and display

course rosters.

class Student:

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

self.name = name

self.age = age

class Course:

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

self.name = name

self.students = []

def enroll\_student(self, student):

self.students.append(student)

def display\_course\_roster(self):

print(f"Course: {self.name}")

print("Students:")

for student in self.students:

print(f"Name: {student.name}, Age: {student.age}")

S=Student("John",20)

C=Course("Math")

C.enroll\_student(S)

C.display\_course\_roster()

Exercise 6: Zoo Management

Scenario: You are designing a zoo management system.

Question: Create an Animal class with attributes such as species and habitat. Create a Zoo class

that can contain multiple Animal objects. Write methods to add animals and display the list of

animals in the zoo.

class Animal:

def \_\_init\_\_(self, species, habitat):

self.species = species

self.habitat = habitat

class Zoo:

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

self.animals = []

def add\_animal(self, animal):

self.animals.append(animal)

def display\_animal\_list(self):

print("Animals in the zoo:")

for animal in self.animals:

print(f"Species: {animal.species}, Habitat: {animal.habitat}")

A=Animal("Lion","Savannah")

Z=Zoo()

Z.add\_animal(A)

Z.display\_animal\_list()

Exercise 7: Hotel Booking System

Scenario: Develop a hotel booking system.

Question: Create a Room class with attributes like room number, type, and availability status.

Create a Hotel class that can manage multiple Room objects. Implement methods to book a room

and check room availability.

class Room:

def \_\_init\_\_(self, room\_number, room\_type, availability):

self.room\_number = room\_number

self.room\_type = room\_type

self.availability = availability

class Hotel:

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

self.rooms = []

def add\_room(self, room):

self.rooms.append(room)

def book\_room(self, room\_number):

for room in self.rooms:

if room.room\_number == room\_number and room.availability:

room.availability = False

return f"Room {room\_number} has been booked."

return f"Room {room\_number} is not available."

H=Hotel()

R=Room(101,"Deluxe",True)

H.add\_room(R)

H.book\_room(101)

Exercise 8: Movie Rental System

Scenario: Design a movie rental system.

Question: Define a Movie class with attributes such as title, genre, and rental price. Create a

RentalStore class that manages a collection of Movie objects. Write methods to rent movies

and display available movies.

class Movie:

def \_\_init\_\_(self, title, genre, rental\_price):

self.title = title

self.genre = genre

self.rental\_price = rental\_price

self.rented = False # Add rented status

class Rental\_Store:

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

self.movies = []

def add\_movie(self, movie):

self.movies.append(movie)

def rent\_movie(self, title):

for movie in self.movies:

if movie.title == title and not movie.rented:

movie.rented = True

return f"Movie '{title}' has been rented."

return f"Movie '{title}' is not available for rent."

def display\_available\_movies(self):

for movie in self.movies:

if not movie.rented:

print("Available Movies:")

print(f"Title: {movie.title}, Genre: {movie.genre}, Rental Price: {movie.rental\_price}")

else:

print(f"No Available Movies")

# Example usage

M = Movie("The Shawshank Redemption", "Drama", 19.99)

R = Rental\_Store()

R.add\_movie(M)

print(R.rent\_movie("The Shawshank Redemption")) # Output: Movie 'The Shawshank Redemption' has been rented.

R.display\_available\_movies() # Output: No available movies

Exercise 9: Fitness Tracker

Scenario: You are creating a fitness tracker application.

Question: Create a Workout class with attributes for type of exercise, duration, and calories

burned. Create a User class that can have multiple Workout objects. Implement methods to log

workouts and calculate total calories burned.

class Workout:

def \_\_init\_\_(self, type\_of\_exercise, duration, calories\_burned):

self.type\_of\_exercise = type\_of\_exercise

self.duration = duration

self.calories\_burned = calories\_burned

class User:

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

self.name = name

self.workouts = []

def add\_workout(self, workout):

self.workouts.append(workout)

def calculate\_total\_calories\_burned(self):

total\_calories = 0

for workout in self.workouts:

total\_calories += workout.calories\_burned

return total\_calories

W=Workout("Running",60,200)

U=User("John")

U.add\_workout(W)

U.calculate\_total\_calories\_burned()

Exercise 10: Task Management System

Scenario: Develop a task management system.

Question: Define a Task class with attributes such as title, description, and status. Create a

Project class that can hold multiple Task objects. Write methods to add tasks, update their

status, and display all tasks in the project.

class Task:

def \_\_init\_\_(self, title, description, status):

self.title = title

self.description = description

self.status = status

class Project:

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

self.name = name

self.tasks = []

def add\_task(self, task):

self.tasks.append(task)

def update\_task\_status(self, task\_title, new\_status):

for task in self.tasks:

if task.title == task\_title:

task.status = new\_status

def display\_tasks(self):

print(f"Project: {self.name}")

for task in self.tasks:

print(f"Title: {task.title}, Description: {task.description}, Status: {task.status}")

T=Task("Task 1","Description 1","Pending")

P=Project("Project 1")

P.add\_task(T)

P.update\_task\_status("Task 1","Completed")

P.display\_tasks()