# TASK 1:

import string

import random

class PasswordGenerator:

    def \_\_init\_\_(self, length=12):

        self.length = length

        self.character\_sets = ["upper", "lower", "digits", "special"]

    def set\_length(self, length):

        self.length = length

    def set\_character\_sets(self, character\_sets):

        self.character\_sets = character\_sets

    def generate\_password(self):

        password = ""

        all\_characters = {

            "upper": string.ascii\_uppercase,

            "lower": string.ascii\_lowercase,

            "digits": string.digits,

            "special": string.punctuation

        }

        while len(password) != self.length:

            for char\_set in self.character\_sets:

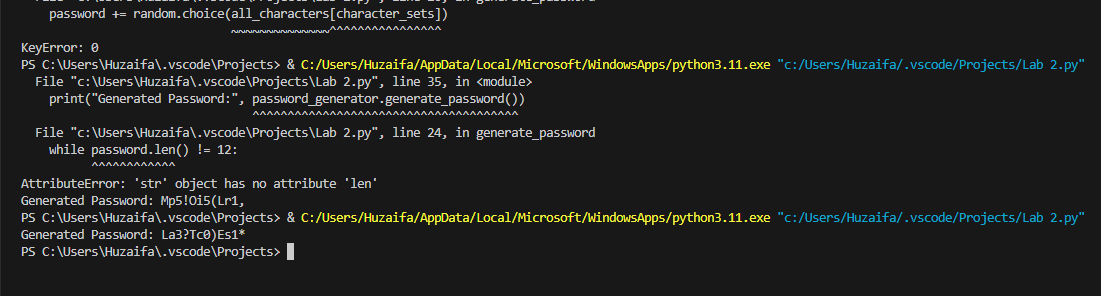
                password += random.choice(all\_characters[char\_set])

        return password

password\_generator = PasswordGenerator()

print("Generated Password:", password\_generator.generate\_password())

# OUTPUT:



# TASK 2:

class ListOperations:

    def \_\_init\_\_(self, initial\_list=None):

        self.my\_list = initial\_list

    def add\_element(self, element):

        self.my\_list.append(element)

    def remove\_element(self, element):

        if element in self.my\_list:

            self.my\_list.remove(element)

        else:

            print(f"{element} not found in the list.")

    def find\_maximum(self):

        if not self.my\_list:

            print("List is empty.")

            return None

        return max(self.my\_list)

    def find\_minimum(self):

        if not self.my\_list:

            print("List is empty.")

            return None

        return min(self.my\_list)

initial\_list = [5, 3, 8, 2, 7]

list = ListOperations(initial\_list)

list.add\_element(10)

print("List after adding 10:", list.my\_list)

list.remove\_element(8)

print("List after removing 8:", list.my\_list)

max\_value = list.find\_maximum()

print("Maximum value in the list:", max\_value)

min\_value = list.find\_minimum()

print("Minimum value in the list:", min\_value)

# OUTPUT:

# 

# TASK 4:

import random

from datetime import datetime, timedelta

class LibrarySystem:

    def \_\_init\_\_(self):

        self.students = {}

    def register\_student(self, name, roll\_number):

        if roll\_number in self.students:

            print("Student with the given roll number already exists.")

        else:

            self.students[roll\_number] = {'name': name, 'books': {}}

            print(f"Hi {name}! You have been successfully registered with roll number {roll\_number}.")

    def display\_categories(self):

        print("\nBook Categories:")

        print("1. Fiction")

        print("2. Science Fiction")

        print("3. Mystery")

        print("4. History")

        print("5. Science")

    def display\_books(self, category):

        categories = {

            1: ["Fiction Book 1", "Fiction Book 2", "Fiction Book 3"],

            2: ["Sci-Fi Book 1", "Sci-Fi Book 2", "Sci-Fi Book 3"],

            3: ["Mystery Book 1", "Mystery Book 2", "Mystery Book 3"],

            4: ["History Book 1", "History Book 2", "History Book 3"],

            5: ["Science Book 1", "Science Book 2", "Science Book 3"]

        }

        if category in categories:

            print(f"\nBooks in the {categories[category]} category:")

            for i, book in enumerate(categories[category], start=1):

                print(f"{i}. {book}")

        else:

            print("Invalid category.")

    def borrow\_book(self, roll\_number, category, book\_index):

        if roll\_number in self.students:

            student = self.students[roll\_number]

            book\_categories = {

                1: "Fiction",

                2: "Science Fiction",

                3: "Mystery",

                4: "History",

                5: "Science"

            }

            if category in book\_categories:

                selected\_book = book\_categories[category][book\_index - 1]

                student['books'][selected\_book] = datetime.now() + timedelta(days=random.randint(7, 21))

                print(f"\n{student['name']}, you have successfully borrowed {selected\_book}.")

                print(f"Please return the book by {student['books'][selected\_book]}")

            else:

                print("Invalid category.")

        else:

            print("Student not found.")

library = LibrarySystem()

name = input("Enter your name: ")

roll\_number = input("Enter your roll number: ")

library.register\_student(name, roll\_number)

if roll\_number in library.students:

    library.display\_categories()

    selected\_category = int(input("Enter the number of the book category you are interested in: "))

    library.display\_books(selected\_category)

    selected\_book\_index = int(input("Enter the number of the book you want to borrow: "))

    library.borrow\_book(roll\_number, selected\_category, selected\_book\_index)

# OUTPUT:

# 

# TASK 5:

class Graph:

    def \_\_init\_\_(self):

        self.adjacency\_list = {}

    def add\_edge(self, vertex, edge):

        if vertex not in self.adjacency\_list:

            self.adjacency\_list[vertex] = []

        self.adjacency\_list[vertex].append(edge)

    def print\_graph(self):

        for vertex, edges in self.adjacency\_list.items():

            print(f"{vertex} -> {', '.join(edges)}")

graph = Graph()

graph.add\_edge('A', 'B')

graph.add\_edge('A', 'C')

graph.add\_edge('A', 'D')

graph.add\_edge('B', 'E')

graph.add\_edge('B', 'F')

graph.add\_edge('E', 'G')

graph.add\_edge('F', 'H')

graph.print\_graph()

# OUTPUT:

# 

# TASK 6:

class Graph:

    def \_\_init\_\_(self, vertices):

        self.vertices = vertices

        self.adjacency\_matrix = [[0] \* (vertices + 1) for \_ in range(vertices + 1)]

    def add\_edge(self, start, end, weight):

        if 1 <= start <= self.vertices and 1 <= end <= self.vertices:

            self.adjacency\_matrix[start][end] = weight

            self.adjacency\_matrix[end][start] = weight  # Assuming it's an undirected graph

    def print\_graph(self):

        for row in self.adjacency\_matrix[1:]:

            print(" ".join(map(str, row[1:])))

graph = Graph(6)

graph.add\_edge(6, 5, 9)

graph.add\_edge(6, 1, 14)

graph.add\_edge(6, 3, 2)

graph.add\_edge(5, 4, 6)

graph.add\_edge(3, 4, 11)

graph.add\_edge(3, 1, 9)

graph.add\_edge(3, 2, 10)

graph.add\_edge(1, 2, 7)

graph.add\_edge(2, 4, 15)

graph.print\_graph()

# OUTPUT: