**Task No. 1:** Create a program that recursively searches for files with a specific extension in a given directory.

**Solution:**

import os

def search\_files\_with\_extension(directory, target\_extension):

found\_files = []

# Walk through the directory tree

for root, dirs, files in os.walk(directory):

for file in files:

if file.endswith(target\_extension):

found\_files.append(os.path.join(root, file))

return found\_files

if \_\_name\_\_ == "\_\_main\_\_":

directory\_to\_search = input("Enter the directory to search: ")

extension\_to\_find = input("Enter the target file extension (e.g., .txt, .jpg): ")

if not os.path.exists(directory\_to\_search):

print("Directory not found.")

else:

found\_files = search\_files\_with\_extension(directory\_to\_search, extension\_to\_find)

if found\_files:

print(f"Found {len(found\_files)} files with the extension '{extension\_to\_find}':")

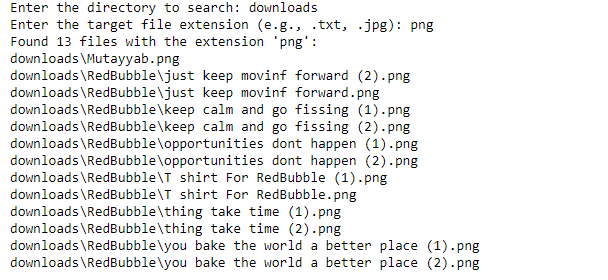
for file in found\_files:

print(file)

else:

print(f"No files with the extension '{extension\_to\_find}' found in the directory.")

**Output:**



**Task No. 2:** Implement Fibonacci series using recursive function.

**Solution:**

def fibonacci\_recursive(n):

if n <= 0:

return 0

elif n == 1:

return 1

else:

return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2)

# Example usage:

n\_terms = 10 # Change this to the number of terms you want in the sequence

if n\_terms <= 0:

print("Please enter a positive integer.")

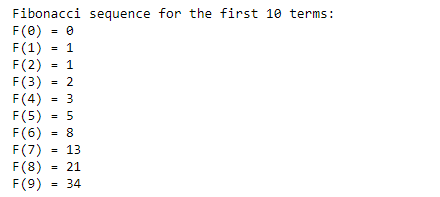
else:

print(f"Fibonacci sequence for the first {n\_terms} terms:")

for i in range(n\_terms):

print(f"F({i}) = {fibonacci\_recursive(i)}")

**Output:**



**Task No. 3:** Implement Undo Mechanism Using Appropriate ADT.

**Solution:**

class UndoStack:

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

self.stack = []

def push(self, state):

self.stack.append(state)

def pop(self):

if not self.is\_empty():

return self.stack.pop()

else:

return None

def is\_empty(self):

return len(self.stack) == 0

undo\_stack = UndoStack()

document\_state = "Initial Content"

while True:

action = input("Enter an action (Type 'undo' to undo or 'exit' to exit): ")

if action == "exit":

break

elif action == "undo":

previous\_state = undo\_stack.pop()

if previous\_state is not None:

document\_state = previous\_state

print(f"Undo: Document state is now '{document\_state}'")

else:

print("Nothing to undo.")

else:

undo\_stack.push(document\_state)

document\_state = action

print(f"Document state is now '{document\_state}'")

**Output:**

A screenshot of a computer code

Description automatically generated

**Task No. 4:** Identify ADT used in Print Job Management and implement its code as well.

**Solution:**

import queue

def print\_job\_management():

print\_queue = queue.Queue()

while True:

print("Options:")

print("1. Submit a print job")

print("2. Print a job")

print("3. Exit")

choice = input("Enter your choice: ")

if choice == "1":

job = input("Enter the print job: ")

print\_queue.put(job)

print(f"Print job '{job}' has been added to the queue.")

elif choice == "2":

if not print\_queue.empty():

job = print\_queue.get()

print(f"Printing: {job}")

else:

print("No print jobs in the queue.")

elif choice == "3":

break

else:

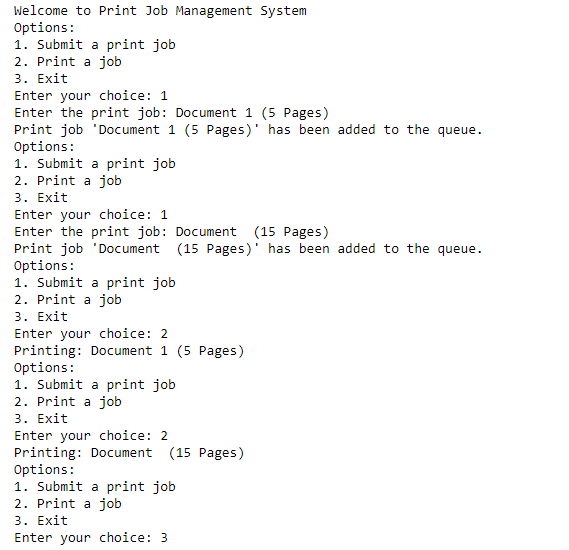
print("Invalid choice. Please choose 1, 2, or 3.")

if \_\_name\_\_ == "\_\_main\_\_":

print("Welcome to Print Job Management System")

print\_job\_management()

**Output:**



**Task No. 5:** You are working on a social network analysis project and need to implement various set operations to analyze relationships between users. Your task is to implement a program that performs the following functionalities using the Set ADT:

**Friends of Friends:** Given a user, find their friends of friends (i.e., users who are friends with at least one of their friends but not directly connected to the user).

**Mutual Friends:** Given two users, find their mutual friends (i.e., users who are friends with both users).

**Friend Suggestions**: Given a user, find friend suggestions by identifying users who are friends with at least two of the user's friends but are not yet connected to the user.

**Solution:**

class SocialNetwork:

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

# Create a dictionary to store the relationships between users.

# The keys are user IDs, and the values are sets of their friends.

self.users = {}

def add\_user(self, user\_id):

# Initialize a new user with an empty set of friends.

self.users[user\_id] = set()

def add\_friend(self, user\_id, friend\_id):

# Add a friend to the user's friend set.

if user\_id in self.users and friend\_id in self.users:

self.users[user\_id].add(friend\_id)

self.users[friend\_id].add(user\_id)

def friends\_of\_friends(self, user\_id):

if user\_id not in self.users:

return set() # User doesn't exist, return an empty set.

user\_friends = self.users[user\_id]

friends\_of\_friends\_set = set()

for friend in user\_friends:

friends\_of\_friends\_set.update(self.users[friend])

# Remove the user and their direct friends from the set.

friends\_of\_friends\_set.discard(user\_id)

friends\_of\_friends\_set.difference\_update(user\_friends)

return friends\_of\_friends\_set

def mutual\_friends(self, user1, user2):

if user1 not in self.users or user2 not in self.users:

return set() # User(s) don't exist, return an empty set.

mutual\_friends\_set = self.users[user1].intersection(self.users[user2])

return mutual\_friends\_set

def friend\_suggestions(self, user\_id):

if user\_id not in self.users:

return set() # User doesn't exist, return an empty set.

user\_friends = self.users[user\_id]

friend\_suggestions\_set = set()

for friend in user\_friends:

for suggested\_friend in self.users[friend]:

if suggested\_friend != user\_id and suggested\_friend not in user\_friends:

friend\_suggestions\_set.add(suggested\_friend)

return friend\_suggestions\_set

if \_\_name\_\_ == '\_\_main\_\_':

social\_network = SocialNetwork()

social\_network.add\_user(1)

social\_network.add\_user(2)

social\_network.add\_user(3)

social\_network.add\_user(4)

social\_network.add\_friend(1, 2)

social\_network.add\_friend(2, 3)

social\_network.add\_friend(3, 4)

social\_network.add\_friend(4, 1)

# Friends of Friends

print("Friends of Friends for User 1:", social\_network.friends\_of\_friends(1))

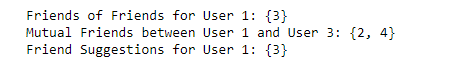
# Mutual Friends

print("Mutual Friends between User 1 and User 3:", social\_network.mutual\_friends(1, 3))

# Friend Suggestions

print("Friend Suggestions for User 1:", social\_network.friend\_suggestions(1))

**Output:**



**Task No. 6:** Implement a gradebook analysis program using the Map ADT. The program should read a CSV file containing student names and their corresponding grades. The CSV file will have two columns: "Student Name" and "Grade". The program should analyze the grades and provide the following functionalities:

* Calculate the average grade for all students.

**Solution:**

import csv

# Open the CSV file

with open('Grade.csv', 'r') as file:

# Create a CSV reader object

reader = csv.reader(file)

# Skip the header row

next(reader)

# Create a dictionary to store the grades for each student

grades = {}

# Read each row in the CSV file

for row in reader:

# Get the student name and grade

student\_name, grade = row

# Check if the student is already in the dictionary

if student\_name in grades:

# Add the grade to the student's list of grades

grades[student\_name].append(int(grade))

else:

# Create a new list of grades for the student

grades[student\_name] = [int(grade)]

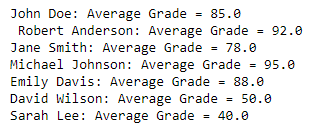
# Calculate the average grade for each student and print the results

for student\_name, student\_grades in grades.items():

average\_grade = sum(student\_grades) / len(student\_grades)

print(f'{student\_name}: Average Grade = {average\_grade}')

**Output:**



* Determine the student with the highest grade.

**Solution:**

import csv

# Open the CSV file

with open('Grade.csv', 'r') as file:

# Create a CSV reader object

reader = csv.reader(file)

# Skip the header row

next(reader)

# Initialize variables to store the highest grade and the student with the highest grade

highest\_grade = float('-inf')

highest\_student = ''

# Read each row in the CSV file

for row in reader:

# Get the student name and grade

student\_name, grade = row

# Convert the grade to a float

grade = float(grade)

# Check if the grade is higher than the current highest grade

if grade > highest\_grade:

# Update the highest grade and the student with the highest grade

highest\_grade = grade

highest\_student = student\_name

# Print the student with the highest grade

print(f'The student with the highest grade is {highest\_student} with a grade of {highest\_grade}.')

**Output:**



* Determine the student with the lowest grade.

**Solution:**

import csv

# Open the CSV file

with open('Grade.csv', 'r') as file:

# Create a CSV reader object

reader = csv.reader(file)

# Skip the header row

next(reader)

# Initialize variables to store the lowest grade and the student with the lowest grade

lowest\_grade = float('inf')

lowest\_student = ''

# Read each row in the CSV file

for row in reader:

# Get the student name and grade

student\_name, grade = row

# Convert the grade to a float

grade = float(grade)

# Check if the grade is lower than the current lowest grade

if grade < lowest\_grade:

# Update the lowest grade and the student with the lowest grade

lowest\_grade = grade

lowest\_student = student\_name

# Print the student with the lowest grade

print(f'The student with the lowest grade is {lowest\_student} with a grade of {lowest\_grade}.')

**Output:**



* Count the number of students who received a grade above a specified threshold.

**Solution:**

import csv

# Open the CSV file

with open('Grade.csv', 'r') as file:

# Create a CSV reader object

reader = csv.reader(file)

# Skip the header row

next(reader)

# Initialize a variable to store the number of students who received a grade above the threshold

num\_above\_threshold = 0

threshold = 80

# Read each row in the CSV file

for row in reader:

# Get the student name and grade

student\_name, grade = row

grade = int(grade)

# Check if the grade is above the threshold

if grade > threshold:

# Increment the number of students who received a grade above the threshold

num\_above\_threshold += 1

print(f'{num\_above\_threshold} students received a grade above {threshold}.'

**Output:**

