1. **Task List Manager\*: Develop a Python program to manage a task list using lists and tuples,including adding, removing, updating, and sorting tasks.**

# Define the task list

tasks = []

# Function to add a task

def add\_task(description, priority):

tasks.append((description, priority))

print(f'Task "{description}" with priority {priority} added.')

The line print(f'Task "{description}" with priority {priority} added.') uses an **f-string** (formatted string literal) in Python to print a message that includes the values of variables description and priority.

Here's a breakdown:

* f'...' : The f before the quotation marks indicates that this is a formatted string.
* {description} and {priority}: These are placeholders inside the string. When the string is evaluated, the current values of description and priority variables will be inserted in these positions.
* print(...): This function outputs the specified message to the console.

# Function to remove a task

def remove\_task(description):

global tasks

tasks = [task for task in tasks if task[0] != description]

print(f'Task "{description}" removed.')

Absolutely! Let's break it down:

1. **List Comprehension:**

python

tasks = [task for task in tasks if task[0] != description]

* + This line uses a **list comprehension** to create a new list of tasks. It iterates through each task in the existing tasks list.
  + task for task in tasks: This part means "include task in the new list for each task in the old tasks list".
  + if task[0] != description: This is a conditional statement. It means "include the task in the new list only if the first element of the task (task[0]) is not equal to the description variable".
  + The result is a new tasks list that excludes tasks whose first element matches the description variable.

# Function to update a task

def update\_task(description, new\_description, new\_priority):

for i, task in enumerate(tasks):

if task[0] == description:

tasks[i] = (new\_description, new\_priority)

print(f'Task "{description}" updated to "{new\_description}" with priority {new\_priority}.')

break

else:

print(f'Task "{description}" not found.')

Sure! Let's break down the update\_task function step by step:

1. **Function Definition:**

python

def update\_task(description, new\_description, new\_priority):

* + This defines a function named update\_task that takes three parameters: description, new\_description, and new\_priority.

1. **For Loop with Enumeration:**

python

for i, task in enumerate(tasks):

* + The for loop iterates over the tasks list. The enumerate function provides both the index (i) and the task (task) for each iteration.

1. **Condition to Check Task Description:**

python

if task[0] == description:

* + This if statement checks if the first element of the current task (task[0]) matches the description parameter.

1. **Updating the Task:**

python

tasks[i] = (new\_description, new\_priority)

* + If the description matches, the task at index i is updated with the new\_description and new\_priority.

1. **Print Statement for Update Confirmation:**

python

print(f'Task "{description}" updated to "{new\_description}" with priority {new\_priority}.')

* + This prints a message confirming that the task has been updated. The f-string (f'...') includes the values of the description, new\_description, and new\_priority variables.

1. **Breaking Out of the Loop:**

python

break

* + The break statement exits the for loop once the task has been updated.

1. **Else Clause for Task Not Found:**

python

else:

print(f'Task "{description}" not found.')

* + If the loop completes without finding the task, the else clause executes, printing a message indicating that the task with the given description was not found.

# Function to sort tasks based on priority

def sort\_tasks():

global tasks

tasks.sort(key=lambda task: task[1])

print('Tasks sorted by priority.')

print (tasks)

# Function to display all tasks

def display\_tasks():

if not tasks:

print('No tasks to display.')

else:

for task in tasks:

print(f'Task: {task[0]}, Priority: {task[1]}')

def main():

while True:

print("\nTask Manager Menu:")

print("1. Add Task")

print("2. Remove Task")

print("3. Update Task")

print("4. Sort Tasks by Priority")

print("5. Display Tasks")

print("6. Exit")

choice = input("Enter your choice (1-6): ")

if choice == '1':

description = input("Enter task description: ")

priority = int(input("Enter task priority: "))

add\_task(description, priority)

elif choice == '2':

description = input("Enter task description to remove: ")

remove\_task(description)

elif choice == '3':

description = input("Enter task description to update: ")

new\_description = input("Enter new task description: ")

new\_priority = int(input("Enter new task priority: "))

update\_task(description, new\_description, new\_priority)

elif choice == '4':

sort\_tasks()

elif choice == '5':

display\_tasks()

elif choice == '6':

print("Exiting Task Manager.")

break

else:

print("Invalid choice. Please try again.")

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

main()

The if \_\_name\_\_ == "\_\_main\_\_": idiom is used to check whether a Python file is being run as the main program or if it is being imported as a module in another program. When the file is run directly, the code within the if block will execute.

def main():

print("Hello, world!")

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

main()

**Output:**

Task Manager Menu:

1. Add Task

2. Remove Task

3. Update Task

4. Sort Tasks by Priority

5. Display Tasks

6. Exit

Enter your choice (1-6):

**b) Create a Python code to demonstrate the use of sets and perform set operations (union, intersection, difference) to manage student enrollments in multiple courses / appearing for multiple entrance exams like CET, JEE, NEET etc.**

# Define sets of students enrolled in different entrance exams

cet\_students = {'Alice', 'Bob', 'Charlie', 'David'}

jee\_students = {'Eve', 'Frank', 'Charlie', 'David'}

neet\_students = {'Grace', 'Heidi', 'Charlie', 'David'}

# Union: Students who are enrolled in any of the exams

all\_students = cet\_students.union(jee\_students).union(neet\_students)

print("Students enrolled in any exam (CET, JEE, NEET):")

print(all\_students)

# Intersection: Students who are enrolled in all the exams

common\_students = cet\_students.intersection(jee\_students).intersection(neet\_students)

print("\nStudents enrolled in all exams (CET, JEE, NEET):")

print(common\_students)

# Difference: Students who are enrolled in CET but not in JEE

cet\_not\_jee = cet\_students.difference(jee\_students)

print("\nStudents enrolled in CET but not in JEE:")

print(cet\_not\_jee)

# Difference: Students who are enrolled in JEE but not in NEET

jee\_not\_neet = jee\_students.difference(neet\_students)

print("\nStudents enrolled in JEE but not in NEET:")

print(jee\_not\_neet)

# Difference: Students who are enrolled in NEET but not in CET

neet\_not\_cet = neet\_students.difference(cet\_students)

print("\nStudents enrolled in NEET but not in CET:")

print(neet\_not\_cet)

# Symmetric Difference: Students who are enrolled in CET or JEE, but not both

cet\_jee\_diff = cet\_students.symmetric\_difference(jee\_students)

print("\nStudents enrolled in CET or JEE, but not both:")

print(cet\_jee\_diff)

# Symmetric Difference: Students who are enrolled in JEE or NEET, but not both

jee\_neet\_diff = jee\_students.symmetric\_difference(neet\_students)

print("\nStudents enrolled in JEE or NEET, but not both:")

print(jee\_neet\_diff)

# Symmetric Difference: Students who are enrolled in CET or NEET, but not both

cet\_neet\_diff = cet\_students.symmetric\_difference(neet\_students)

print("\nStudents enrolled in CET or NEET, but not both:")

print(cet\_neet\_diff)

Output:

Students enrolled in any exam (CET, JEE, NEET):

{'Bob', 'Grace', 'Eve', 'Alice', 'David', 'Heidi', 'Charlie', 'Frank'}

Students enrolled in all exams (CET, JEE, NEET):

{'Charlie', 'David'}

Students enrolled in CET but not in JEE:

{'Bob', 'Alice'}

Students enrolled in JEE but not in NEET:

{'Eve', 'Frank'}

Students enrolled in NEET but not in CET:

{'Grace', 'Heidi'}

Students enrolled in CET or JEE, but not both:

{'Bob', 'Eve', 'Alice', 'Frank'}

Students enrolled in JEE or NEET, but not both:

{'Grace', 'Eve', 'Heidi', 'Frank'}

Students enrolled in CET or NEET, but not both:

{'Bob', 'Grace', 'Alice', 'Heidi'}

This code demonstrates the following set operations:

* **Union:** Combines all students enrolled in any of the exams (CET, JEE, NEET).
* **Intersection:** Identifies students enrolled in all the exams.
* **Difference:** Finds students enrolled in one exam but not in another.
* **Symmetric Difference:** Identifies students enrolled in either of two exams but not both.

# Define the dictionary to hold student records

students = {}

# Function to add or update a student record

def add\_or\_update\_student(student\_id, name, grades, attendance):

students[student\_id] = {'name': name, 'grades': grades, 'attendance': attendance}

print(f'Student record for {name} added/updated.')

# Function to display all student records

def display\_students():

if not students:

print('No student records available.')

else:

for student\_id, info in students.items():

avg\_grade = sum(info['grades'].values()) / len(info['grades']) if info['grades'] else 0

print(f'Student ID: {student\_id}, Name: {info["name"]}, Grades: {info["grades"]}, Attendance: {info["attendance"]}, Average Grade: {avg\_grade:.2f}')

# Function to sort students by grades

def sort\_by\_grades():

sorted\_students = sorted(students.items(), key=lambda x: sum(x[1]['grades'].values()) / len(x[1]['grades']) if x[1]['grades'] else 0, reverse=True)

for student\_id, info in sorted\_students:

avg\_grade = sum(info['grades'].values()) / len(info['grades']) if info['grades'] else 0

print(f'Student ID: {student\_id}, Name: {info["name"]}, Average Grade: {avg\_grade:.2f}, Attendance: {info["attendance"]}')

# Function to sort students by attendance

def sort\_by\_attendance():

sorted\_students = sorted(students.items(), key=lambda x: x[1]['attendance'], reverse=True)

for student\_id, info in sorted\_students:

avg\_grade = sum(info['grades'].values()) / len(info['grades']) if info['grades'] else 0

print(f'Student ID: {student\_id}, Name: {info["name"]}, Attendance: {info["attendance"]}, Average Grade: {avg\_grade:.2f}')

def main():

while True:

print("\nStudent Record Manager Menu:")

print("1. Add/Update Student Record")

print("2. Display Student Records")

print("3. Sort by Grades")

print("4. Sort by Attendance")

print("5. Exit")

choice = input("Enter your choice (1-5): ")

if choice == '1':

student\_id = int(input("Enter student ID: "))

name = input("Enter student name: ")

grades = {}

num\_subjects = int(input("Enter number of subjects: "))

for \_ in range(num\_subjects):

subject = input("Enter subject name: ")

grade = float(input(f"Enter grade for {subject}: "))

grades[subject] = grade

attendance = float(input("Enter attendance percentage: "))

add\_or\_update\_student(student\_id, name, grades, attendance)

elif choice == '2':

display\_students()

elif choice == '3':

sort\_by\_grades()

elif choice == '4':

sort\_by\_attendance()

elif choice == '5':

print("Exiting Student Record Manager.")

break

else:

print("Invalid choice. Please try again.")

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

main()

Student Record Manager Menu:

1. Add/Update Student Record

2. Display Student Records

3. Sort by Grades

4. Sort by Attendance

5. Exit

Enter your choice (1-5): 1