Practical No 5

Aim

Develop a Python program to manage a task list using lists and tuples, including adding, removing, updating, and sorting tasks.

Apparatus / Software Required

- Python Interpreter (Python 3.14.2)

Theory

In Python, **list** and **tuple** are collection data types used to **store multiple value**s in a single variable.
In this practical, both list and tuple are used together to manage a task list.

A task list means storing daily activities like *Study*, *Exercise*, *Assignment*, etc., along with their status (*Pending* or *Completed*).

Use of List in This Practical

A **list** is used to store all tasks.

Why list is used?

- **List** can store multiple tasks
- List is **mutable**, so tasks can be **added**, **removed**, **updated**, and **sorted**
- List maintains order

List in Python

A list is a collection of items stored in a single variable.

Features of List

- Lists are ordered
- Lists are mutable (can be changed)
- Lists allow duplicate values
- Lists can store multiple data types
- Represented using square brackets [ ]

Example:

```
task_list = []
```

Here, `task_list` stores all tasks.

Use of Tuple in This Practical

A **tuple** is used to store one task.
Each task has fixed information:

1. Task name
2. Task status

Why tuple is used

- Tuple is immutable, so task details remain safe
- Tuple keeps task data structured

Tuple in Python

A **tuple** is similar to a list but with some differences.

Features of Tuple

- Tuples are **ordered**
- Tuples are **immutable** (cannot be changed)
- Represented using **round brackets ( )**
- Used to store **fixed data**

Example:

```
task = ("Study Python", "Pending")
```

This represents one task.

Storing All Tasks

All tasks are stored as tuples inside a list.

Example:

```
task_list = [
    ("Study Python", "Pending"),
    ("Exercise", "Completed")
]
```

Here:

- **List** → stores all tasks
- **Tuple** → stores one task

Operations on List :

1.Adding New Tasks

To add a new task, a new tuple is created and added to the list using `append()`.

Example:

```
task_list.append(("Complete Assignment", "Pending"))
```

This shows how new tasks are added to the task list.

2.Updating Tasks

Tuples cannot be changed directly because they are immutable.
So, to update a task, the old tuple is replaced with a new tuple at the same position.

Example:

```
task_list[0] = ("Study Python", "Completed")
```

This updates the task status from *Pending* to *Completed*.

3.Removing Tasks

To remove a task, the `remove()` method is used.

Example:

```
By using Value - task_list.remove(("Exercise", "Completed"))
```

Or

```
By using index - del task_list[1]
```

This removes the selected task from the list.

4. Sorting Tasks

Tasks are sorted in **alphabetical order based on task name** to keep the task list organized.

- The `sort()` method sorts the task list **permanently**.

```
task_list.sort()
```

- The `sorted()` function returns a **new sorted list** without changing the original list.

```
sorted_tasks = sorted(task_list)
```

Algorithm

1. Start
2. Create an empty list named `task_list` to store all tasks.
3. Create tasks as tuples, where each tuple contains:
   - Task name
   - Task priority number (1, 2, 3, … indicating order of execution)
4. Store all task tuples inside the `task_list`.
5. Add a new task by creating a tuple and inserting it into the list using `append()`.
6. Remove a task from the list using `del` by specifying the index position.
7. Update a task by replacing the old tuple with a new tuple at the same index to change its priority number.
8. Sort the task list based on task priority or task name using:
   - `sort()` to modify the original list
   - or `sorted()` to create a new sorted list
9. Display the final task list.
10. Stop

Program Code

Output

Conclusion