To-Do List Application Report

Introduction

What is your application?

The To-Do List App is a simple task management tool that allows users to add, edit, delete, save and load the tasks. The app is implemented in Python using the “Tkinter” library for the GUI.

How to run the program?

To run the application, execute the `main.py` file using Python. Please be sure that you have a Python installed on your computer.

How to use the program?

Add Task: Enter a task in the input field and click "Add Task".

Edit Task: Select a task from the list, enter the new task name in the input field and click "Edit Task".

Delete Task: Select a task from the list and click "Delete Task".

Save Tasks: Click "Save Tasks" to save the current tasks to file.

Load Tasks: Click "Load Tasks" to load tasks from file.

Body/Analysis

Functional Requirements

OOP Principles

Polymorphism

Demonstrated through method overriding in the AdvancedToDoListApp class.

```
class AdvancedToDoListApp(ToDoListApp):
    new *
    def _add_task(self):
        task = self._task_input.get()
        if task:
            if messagebox.askyesno( title: "Add Task",  message: f"Are you sure you want to add the task: {task}?"):
                new_task = TaskFactory.create_task(task)
                self.task_manager.add_task(new_task)
                self._tasks_list_box.insert(tk.END,  *elements: new_task)
                self._task_input.delete( first: 0, tk.END)
```

Abstraction

Implemented through methods that hide the implementation details and provide a simple interface for task management.

```
class AdvancedToDoListApp(ToDoListApp):
    new *
    def _add_task(self):
        task = self._task_input.get()
        if task:
            if messagebox.askyesno( title: "Add Task",  message: f"Are you sure you want to add the task: {task}?"):
                new_task = TaskFactory.create_task(task)
                self.task_manager.add_task(new_task)
                self._tasks_list_box.insert(tk.END,  *elements: new_task)
                self._task_input.delete( first: 0, tk.END)
```

Inheritance

Used to create AdvancedToDoListApp from ToDoListApp.

```
class AdvancedToDoListApp(ToDoListApp):
```

Encapsulation

Protected members and methods (prefixed with an underscore) are used to encapsulate the functionality and data.

```
        if file_path:
            with open(file_path, "r") as f:
                for line in f:
                    task = line.strip()
                    if task:
                        self.task_manager.add_task(task)
                        self._tasks_list_box.insert(tk.END, *elements: task)
```

Design Patterns

Singleton Pattern

Ensures that only one instance of “TaskManagerSingleton” exists.

```
class TaskManagerSingleton:
    """Singleton for managing tasks."""
    _instance = None

    new *
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(TaskManagerSingleton, cls).__new__(cls)
            cls._instance.tasks = []
        return cls._instance
```

Factory Method Pattern

Used to create tasks.

```
class TaskFactory:
    """Factory Method for creating tasks."""
    2 usages  new *
    @staticmethod
    def create_task(task_name):
        return task_name
```

Reading from and Writing to Files

The application includes functionalities to save tasks to a file and load tasks from a file.

```
1 usage  new *
def _save_tasks(self):
    file_path = filedialog.asksaveasfilename(defaultextension=".txt")
    if file_path:
        with open(file_path, "w") as f:
            for task in self.task_manager.get_tasks():
                f.write(task + "\n")

1 usage  new *
def _load_tasks(self):
    self._tasks_list_box.delete( first: 0, tk.END)
    self.task_manager.tasks.clear()
    file_path = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
    if file_path:
        with open(file_path, "r") as f:
            for line in f:
                task = line.strip()
                if task:
                    self.task_manager.add_task(task)
                    self._tasks_list_box.insert(tk.END,  *elements: task)
```

Unit Testing

Core functionalities are tested using the unittest framework.

```python
class TestTaskManager(unittest.TestCase):
    new *
    def setUp(self):
        self.task_manager = TaskManagerSingleton()

    new *
    def tearDown(self):
        TaskManagerSingleton._instance = None  # Reset the singleton instance for each test

    new *
    def test_add_task(self):
        self.task_manager.add_task("Task 1")
        self.assertIn( member: "Task 1", self.task_manager.get_tasks())

    new *
    def test_edit_task(self):
        self.task_manager.add_task("Task 1")
        self.task_manager.edit_task( index: 0,  new_task: "Task 1 Edited")
        self.assertIn( member: "Task 1 Edited", self.task_manager.get_tasks())
        self.assertNotIn( member: "Task 1", self.task_manager.get_tasks())

    new *
    def test_delete_task(self):
        self.task_manager.add_task("Task 1")
        self.task_manager.delete_task(0)
        self.assertNotIn( member: "Task 1", self.task_manager.get_tasks())
```

Results and Summary

Future Prospects

Task Prioritization

We can add a feature to prioritize tasks.

Due Dates

We can include due dates for tasks and notifications.

Multiple Task Lists

We can allow managing multiple task lists.

Resources and References

- Build a Powerful Todo List App with Python and Tkinter (youtube.com)
- 1 Videoda Git & Github Öğren. (youtube.com)
- How Git Works: Explained in 4 Minutes (youtube.com)
- Python's unittest: Writing Unit Tests for Your Code – Real Python
- PEP 8 – Style Guide for Python Code | peps.python.org