11/15/2024

Poorva Sharma

100934359

Assignment 2

COSC 1104

**Part 1: Identify the Problem**

## **Problem: Task Reminder Console Application**

**Problem Statement**:

Sometimes, it can be fairly difficult for someone who either has a very busy schedule or is a student to remember all the tasks he or she needs to do within a certain deadline. A simple console-based task reminder application can save him or her from forgetting an important upcoming task by enabling the user to add tasks with due dates and time and get reminders of forthcoming tasks.

**Solution Idea:**

Create a console-based Python application that will enable the user to insert activities, set the deadline, and specify at what time they want the application to remind them about the activity. The application has to store tasks in memory or, optionally, on a file for persistency; it will poll periodically on upcoming due tasks and display reminders on the console.

**Key Features:**

**Add Task:** User can add tasks along with description, deadline, and reminder time.

**View All Tasks:** User will be able to see all activities along with their deadlines.

**Explanation**

**add\_task():** Prompts the user for the task details that includes the name of the task, due date, and reminder time. The due date and reminder time will be stored in datetime objects.

**view\_tasks():** A function used to view all the created tasks with their names, due dates, and reminder times.

**Main Loop:** Program presents user with menu and performs actions based on the user's input until the user decides to quit. The program, every now and then, will check if user has set any reminders.

**Usage**

**Add Task:** User can add multiple tasks, specifying due date with reminders.

**View Task List:** User is able to view listed tasks with their respective deadlines.

The console application keeps you exactly on top of the deadline, controls the tasks in real time.

## **Problem: Automated File Backup Console Application**

**Problem Statement:**

Keeping timely backups of critical data, such as code scripts, configuration files, and personal documents, creates the best measures for data security and recovery. In many cases, it's tedious to perform a backup manually; hence, it is often ignored. The consequences of not making a backup are massive data loss due to unintended deletion, hardware failure, or corruption. This would automate regular backups to a local or even external directory for those who deal with important files daily.

It can be further improved by developing an automated file backup system, which will periodically copy the vital files to a specific backup location without user interference. This would enable users to carry on with their confident work while their data is being safeguarded in the background.

**Solution Suggestion:**

This backup could be automated through a Python-based console application, which gives one an interface for defining a "source" directory with files that need to be backed up and a "backup" directory where these should be saved. This program periodically copies all the files from the source directory into the backup directory; meanwhile, timestamped folders can be created for holding copies of backups. Timestamping here can be quite useful if one also does some versioning of files.

**This solution would achieve the following:**

**Minimize manual labor:** It automates the backup process whereby users no longer need to remember backing up files.

**Make recovery easier:** The timestamped backups would allow files to be retrieved from previous dates.

**Increase data security:** The frequent backups ensure that data loss will not be permanent.

**Principal Characteristics:**

**Configuring Source and Backup Directories:** It provides a way to specify the source directory to be backed up and the backup directory where all the files are to be copied.

**Timestamped Backups:** Every backup is stored in a uniquely timestamped folder inside the backup directory, so each backup version can be accessed.

**Auto Scheduling:** It runs at an interval, say every 24 hours, and does the backups for the day without user intervention.

**File Preservation:** Using metadata-preserving methods for the files it backs up allows it to keep the same properties, modification times included.

**Complexity of Implementation:**

**Difficulty:** Relatively easy to medium. The script would be based mainly on the standard Python libraries: os, shutil, time, and datetime for handling respectively the file operations, directory creation, and interval scheduling.

Libraries:

**nos** and **shutil** for the creation of directories and file handling.

**datetime** for creating timestamped backup folders.

**time** for setting intervals for automated backups.

**Solution's Value:**

This is a real find for users who have to work with critical files and would not want to face accidental data loss. Because this utility continuously makes regular backups in the background, it allows users to always have relatively recent file versions safely stored, without depending on cloud services or manual backups.

**Part 3: Reflect on the Solution**

* **Did you pick a suitably challenging problem to solve? Or was it too easy, or too hard?**

The Task Reminder Console Application project has been both fun and an informative one to work on. I chose this project because it has a real-life practical purpose, calling for a moderate level of complexity, thus allowing me to create a tool that could be useful in managing my tasks and deadlines. The project wasn't exactly complicated but required me to dive into libraries used less often, such as datetime for handling dates and times. Altogether, the problem chosen felt well-suited for my current programming skills: challenging enough to push me but not overwhelming.

* **What was the most challenging aspect of solving this problem?**

The most complicated part for me was real-time reminders handling. It was some thinking and experimenting to perform a continuous check for reminders in a way without blocking other users' interactions and take into consideration that the reminders have to not be shown more than once. I tracked shown reminders with a set, thus avoiding notification duplicates.

* **How did you test your completed product? How confident are you that it works reliably?**

The following are some of the scenarios I ran to test my program: adding tasks with reminders a few minutes apart, then watching to see that reminders were shown as expected. I am rather confident that the program works reliably, although more extensive testing would be in place to further solidify this confidence, especially if the application is to handle a large number of tasks or reminders over time.

* **Is there something you would still like to add to this, or something it makes you want to try next?**

In the future, I would like to add features such as editing and deleting of tasks and maybe a logging system to see what has already been done. Overall, I liked this assignment because I could do something useful while learning more about the libraries of Python, file handling, and time management in an application. It was rewarding to see how such small steps combined into something useful, and I certainly hope to build on this experience in projects to come. Also maybe I can try saving tasks in a separate file.