Emmanuel Cabili Cuyugan
387094
29/01/2024

In Partial Fulfilment of The University of Law's:
MSc - Computer Science - Software Development - Online FT - Sep 23 Start - October 2023
Assessment 2: Coursework Portfolio

Project Title: Python Task Manager

Abstract:

The Python Task Manager is a task management application, functioning as a comprehensive demonstration of the my skill in Python programming, problem-solving ability, and learning journey. Through the use of a modular and stylized approach from previous applications that I developed, this application provides a more feature-rich experience and comprehensive navigation system. Also built in are powerful external libraries, decorators, custom notification system, and data persistence via csv.

Introduction

The Python Task Manager is a task management application was developed building on knowledge of existing functions and explores new, powerful libraries, decorators, notification systems, and logging techniques.

Users can utilise this application by installing several external modules to run the full application and begins by entering tasks. From there, the system will automatically send notifications to users for upcoming, due, and overdue tasks. Beyond that, users can sort through their data by displaying tasks through sort by category, sort by due date, and search by text string. Finally, users can delete individual task lines from their list once complete.

This application serves as a powerful CLI task manager that can be further built upon with even more tools.

Requirements: Standard across Python projects:

This project seeks to assess proficiency in Python programming, problem-solving skills, and the ability to create a functional software solution. Key Requirements identified and attempted include:

Displaying Development Environment Competence:

This program has been built using a development environment to implement functioning code. Writing on VSCode, I have attempted to use the environment to ensure code modularity, readability, and scalability within the program for a seamless development experience.

Utilization of Software Versioning Tools:

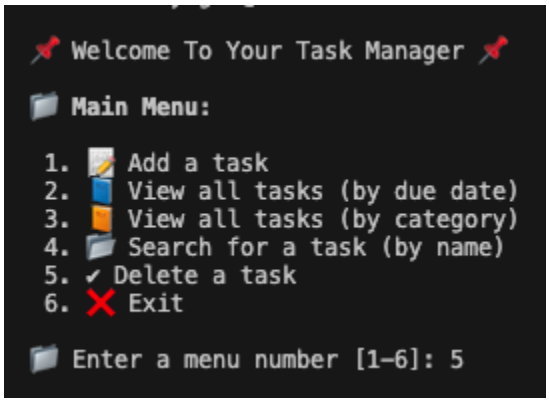
GitHub has been selected as a software versioning tool to showcase proficiency and understanding in deployment and version control. This project has been deployed to GitHub, accessible through my, Emmanuel's, GitHub account, @emmcygn

Unique Requirements:

Code Modularity:

Wanting to take code modularity further, several basic functions were used from a previous submission, my Python Budget Tracker. Using the basic read/write function base and the category selection base, I was able to build the basic frame of this application that got user input and wrote to a file.

Expounding on this further, I implemented a menu system that allowed a user to switch between functions and to serve as the home function after each sub function was performed. This allowed for modular additions of the display by category + due date, the text string search, and the line by line delete system.



To build the additional functions, several modular external modules were used including datetime, calendar, and more.

User Input and Error Handling:

Further building my knowledge from using loops, I introduced the decorators error_handler, log_function_call, a traceback function, and a debug mode. Using this powerful set of tools allows a more comprehensive reception to any errors that may be encountered, appropriate

informative responses to user input errors, and the ability to debug and trace errors within the application.

```python
# MAIN FUNCTION (Define the main function for the task manager)
DEBUG = False  # Set to False in production  # Added debug flag

@error_handler  # Apply the error_handler decorator to the main function
@log_function_call  # Apply the log_function_call decorator to the main function
def main():
    # Configure the logging module
    logging.basicConfig(
        filename="task_manager.log",
        level=logging.DEBUG if DEBUG else logging.INFO,  # Use DEBUG level if in debug mode
        format="%(asctime)s:%(levelname)s:%(message)s",
    )
```

Notification System:

To build the notification system, I utilised the external module notifypy to handle push notifications. This called for pip installation and several nested functions to display custom notifications at certain actions or conditions met.

Some difficulty was met in using the correct nested functions, and creating custom sounds and notification icons (for windows users) was an enjoyable pursuit to add functionality and polish to this application. One such example is that notifications for the due tasks that appear daily come with a custom disc loading sound that imitates the loading of a disc from a Playstation 3.

Particular sticking points included correct setup of the check_due_date and send_notification functions, at which considerable time was allocated to sort.

```python
# Define a function to check due dates and send notifications
def check_due_dates(task_file_path):
    # Read tasks from the file and check due dates
    tasks = read_tasks_from_file(task_file_path)

    # TODAY Reminders
    today = datetime.date.today()
    due_today = [task for task in tasks if task.due_date == today]

    # TOMORROW Reminders
    tomorrow = today + datetime.timedelta(days=1)
    due_tomorrow = [task for task in tasks if task.due_date == tomorrow]

    # OVERDUE Reminders
    # Get tasks with due dates before today
    due_overdue = [task for task in tasks if task.due_date < today]

    # Send notifications for tasks due today
    for task in due_today:
        send_notification(task)

    # Send notifications for tasks due tomorrow
    for task in due_tomorrow:
        send_notification(task, is_reminder=True)

    # Send notifications for overdue tasks
    for task in due_overdue:
        send_notification(task, is_overdue=True)
```

```python
def send_notification(task, is_reminder=False, is_overdue=False):
    try:
        global notification_sent
        # Initialize pygame mixer if it hasn't been initialized
        if not pygame.mixer.get_init():
            pygame.mixer.init()

        # Set audio only once for the set of notifications
        if not notification_sent:
            notification_sent = True
            pygame.mixer.music.load(sound_file_path)
            pygame.mixer.music.play()

        # Your code for sending notifications goes here
        # For example, you can use the Notify class from notifypy
        notification = Notify()
        notification.icon = icon_path

        if is_reminder:
            notification.title = "Task Reminder"
        elif is_overdue:
            notification.title = "Task Overdue"
            notification.message = f"Task '{task.name}' is overdue!"
        else:
            notification.title = "Task Due Today"

        notification.message = (
            f"📆 Task '{task.name}' is due tomorrow!"
            if is_reminder
            else f"💢 Task '{task.name}' is overdue!"
            if is_overdue
            else f"📍 Task '{task.name}' is due today!"
        )

        # Set a custom timeout (in milliseconds)
        notification.timeout = 3000

        notification.send()
    except Exception as e:
        print(f"📛 Error sending notification: {e}")
```

Delete by line function:

Wanting to create a more powerful solution for deleting individual entries via the command line, I utilised a framework that enumerated each entry by an integer starting from 1, allowing for a line by line selection for line by line deleting.

Given the irreversible nature of deleting an entry, I also included a notification system to provide a UX boost, playing a crumpling paper noise as the line is deleted.

Data Storage:

The application creates a file named tasks.csv within the same folder as the application file. The application then writes to this csv file after input. Used in tandem with the delete by line function, this file can now be largely left untouched as file i/o can be handled within the main application.

Class Setup:

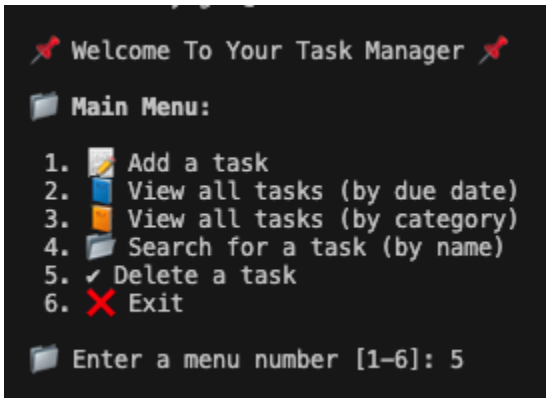
Classes are setup within another file in the folder, task.py. This allows for additional modular control over the files and trims on the size of the main application.

PyGame Dialogue Suppression:

The module pygame was utilised to limit notification sound loops within the application. This means that whenever multiple notifications are triggered simultaneously, the notification sound only plays once whilst each pop up notification still appears individually and sequentially.

This module had a pesky and pervasive print to the terminal whenever the application was run, and it took some time searching how to suppress the dialogue print to leave a clean terminal. This was achieved by importing the os module and running

```
os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = "hide" # Hide pygame support prompt
```



Testing

Phase	**Input**	**Intended Output**	**Actual Output**
Load Libraries, Suppress Pygame Print, and Style Functions	Run application	All modules load	All Modules Load
Notifications Display	Run Application	Displays Notifications on computer	Displays Notifications on computer
Enter Menu number	Type in menu number	Proceeds to next step by calling the function assigned to each number	Calls the function assigned to each number
Enter Menu number	INCORRECTLY type in menu number	Prompts a reentry of integer between 1-6	Prompts a reentry of integer between 1-6
Add Task Function	Input information to	Information is	Information is

	add a task	received, error handling is running, object is created	received, error handling is running, object is created
Write object to csv	Create object	Write object to csv	Write object to csv
Read Lines	Input from select category step and object creation	Read objects in CSV	Read objects in CSV
View All Tasks by due date/category	Select menu item 2 and/or 3	Function returns tasks sorted appropriately with errors handled	Function returns tasks sorted appropriately with errors handled
Search for a task	Select Menu Item 4 and input text string	Displays lines that contain the text string	Displays lines that contain the text string
Delete a task	Select menu item 5 and view tasks. Input corresponding line number	Displays all tasks by reading file and displays with a corresponding integer. Deletes line from user input of an integer. Saves remaining lines back to CSV	Displays all tasks by reading file and displays with a corresponding integer. Deletes line from user input of an integer. Saves remaining lines back to CSV
Close Application	Select menu item 6	Terminates Application	Terminates Application

Conclusion

Building from the first application and with an understanding and appreciation for modular functions, I was able to build in far more complex functions, source the correct resources to study how to implement the functions, and to push my troubleshooting skills further.

Moving forward, I intend on creating even more data views and ways to segment, to utilise SQL to save data to instead of a CSV, and to create a settings menu item that will allow user saved custom controls for notifications, data views, and more.