**Project Proposal**

**Project Description:**

The name of the project is "Weekly Scheduling." As the name indicates, the project helps to make weekly planners by allowing the users to create new tasks and customized one-time event and make feedbacks to the existing tasks each week. With the weekly feedback, we are able to briefly evaluate the workload of certain tasks based on the past feedbacks. With the newly calculated workload, now we can generate a new weekly calendar for the next week.

**Competitive Analysis:**

The project is similar to other planners in which its major function is to offer weekly schedules; however, the project differs from most of the existing projects in the sense that it takes in weekly feedbacks to adjust the predicted workload of each task (so it takes into account of the previous performances). In addition, the planner does not require the user to make the plan; instead, it generates the weekly calendar after the user inputs certain valid information about the task.

**Structural Plan:**

A class named "Task" is created to store the information of each task. There are three major modes (inputting task mode, giving feedback mode, and displaying calendar mode), and each mode has their own app functions using the new version of cmu\_112\_graphics. To store the existing tasks, the Task objects will be converted into strings and stored in a separate .txt file.

**Algorithmic Plan:**

The most algorithmically complex parts of the project are the storage of task information and the process of generating the new calendar.

In order to permanently store the task information, I use file I/O, converting all the task information into a string and write it in a separate file. Each time the app starts, the information will be read from the file and converted back to task objects. After MVP, I plan to add user login so that multiple .txt files could be stored and user could read task information from their corresponding file.

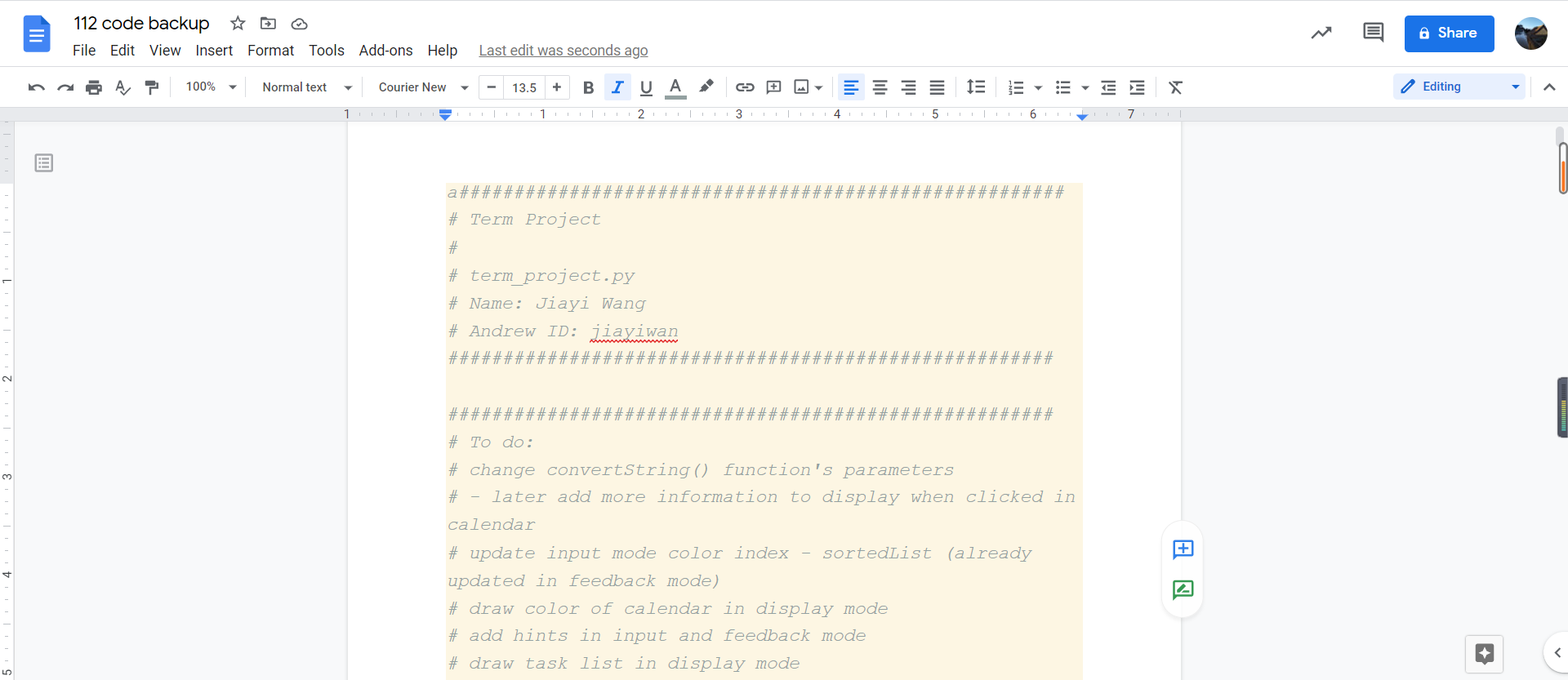
In an attempt to seek the optimized solution, with the predicted workload, deadline, and other parameters of a list of tasks, I use recursion and memoization to generate the “best” (determined by other functions) calendar. This is done by finding the best solution by examining the combined calendar of the current task and the best solution found with the rest of the tasks. Memoization helps to greatly reduce the time cost.

**Timeline Plan:**

1. Conversion between OOP and string, basic file I/O – finish by Thursday 22nd;
2. Basic UI essential for operating of the project – finish by Sunday 25th;
3. TP 1 deliverable – finish by Monday 26th;
4. Basic working structure of generating calendar – finish by Wednesday 28th;
5. Debugging of UI + generating calendar – finish by Thursday 29th;
6. TP2 demo and code wrap up – finish by Friday 30th;
7. More features (such as background and icon) of UI – finish by Sunday 2nd;
8. (if with additional time) User login and readme file – finish by Monday 3rd.

**Version Control Plan:**

I back up my code by copying and pasting the code into a google docs.



**Module List:**

I do not plan to use any module before MVP.

**TP2 Update:**

* Allow three weeks of calendar to be generated based on current predicted workload instead of only one because user, by the design of the program, are able to have an idea of what their workload might look like for the next three weeks;
* Allow customized one-time event in the input mode;
* Program automatically shifts to the next week once a user makes sure they wants to save the feedback.

**TP3 Update:**

* Better graphics, with finalized and working version of background and icon for all pages;
* Add a new ‘Settings’ page for choosing customized personal preferences – morning/night person & hours for lunch;
* Refine messages displayed in customizeMode, feedbackMode, and displayMode to provide more information.