Joanna Huang

About the Recipe Organizer

I enjoy cooking and currently use a Google sheet (see here) to record all the recipe URLs I test and manually create shopping lists whenever I planned to cook a recipe. Thus, this project was the perfect opportunity to refine this process.

This recipe organizer contains four classes:

1. Ingredient (with attributes: name, unit, amount)
2. Recipe (with attributes: name, source, categories, notes, rating, level, ingredients, time and serving size)
3. MyFridge (with Ingredient attributes plus an additional inventory attribute)
4. RecipeOrganizerController (controls the functions of the program and possesses a recipecollection attribute that creates a list of all recipe inputs from a txt file)

Users can perform the following tasks:

- Add recipes to their recipe collection (which is saved at the end of the program to a txt file. Previous recipes are reloaded into the program each time program is run)
- Search for recipes by category or by ingredient
- Check their fridge inventory for specific recipes in the collection
- Update their fridge inventory with new items after a trip to the store
- Create shopping lists for selected recipes

Challenges Faced

I faced two main challenges with this project: 1) Determining the scope of project 2) Saving the data from each session. When writing the proposal for this project, the components of the Recipe class seemed very straightforward. However, as I started creating the program and its functions, I realized how tedious it could be to define a recipe object. On the first level, a recipe possesses its own attributes (9 in the case of my program). However, not only could I easily increase that number (i.e. splitting Est. Time to Prep Time and Cooking Time or adding Equipment Required), I also needed to consider the rabbit hole of options for the attributes of each recipe attribute (i.e. allowing entries for servings to be a range, adding more preparation details for ingredients (minced onions versus sliced)). In the end, I decided to record the most basic aspects of each attribute in an exact amount.

The second challenge I faced was figuring out how to save the data from each session. This all happened in the last week before the project was due, as we had just learned about reading and writing into files. I started with csv files, as I felt more familiar with it than any other file format. I spent one day rewriting the functions of the program to deal with the strings that were being loaded from the csv file. After an epiphany that I should instead recreate the data into objects of the program to save myself from the trouble of rewriting every function, I spent another day trying to create my program's objects from the strings of the csv file. Finally, I

decided to try json, which was way easier for calling each part of the data and converting everything back into objects.

Notes for Testing Program

Saving User Input: The program only writes the data to a file IF the user enters 6 on the main menu to end the program. If the program is ended in any other way, the data from that session will not be saved.

Accessing Recipe Database: Please make sure the recipedata.txt and fridgedata.txt files are in the same folder as RecipeOrganizer.py file. The program will not run if the txt files are unavailable as there needs to be at least one item in each file for the program to run.

Recipe Entry: At this time, the program only allows for adding recipes and adding ingredients. In another version, I'd love to add the ability to edit or delete objects from the existing databases. Right now, that option is unavailable. Thus if you make a mistake when entering the information (i.e. misspelling a name, a unit), there is no way to change it.

In terms of how to use the program, I tried to make every prompt and output as clear as possible. Since I regularly look at recipes, I'm quite familiar with what a recipe consists of without clarification. Thus, to ensure clarity for the user, I had a few of my friends test the program and added more details to my prompts when they were unclear on what their input should look like. Furthermore, I tried to block invalid entries with more guidelines on what the program expects.