**Programming Project 2:** Based on the available ingredients, what recipes can be prepared?

**Objectives**

1. Create, read, and modify files.
2. Apply the control structure and basic concepts of lists and dictionaries to provide accurate solutions to problems that require basic programming.
3. Gain experience implementing applications using layers of increasing coding complexity.
4. Gain experience solving a real-life problem by understanding the requirements provided by the user.

**Due Date:** November 9, 2020, at 11:55 pm.

**Instructions**

For Project 1, you developed an interactive program that asks the user to input the current ingredients and asked for the desired recipe from the menu. You validated if the chosen recipe could be cooked and updated the pantry after successfully cooked a meal from the menu. In Phase 2, you had to select a modification to allow the user:

1. Increase the inventory in the pantry when there were not enough ingredients.
2. Add a new recipe with its corresponding ingredients.
3. Define the desired amount of servings for the chosen meal.

We will provide you our solution for Project 1, including a working implementation of all three modification options. This program will be your base code for Project 2.

**The objective** of Project 2 is to modify Project 1 code to display only the recipes that can be prepared, based on the available ingredients in the pantry. Also, you will change the program to use files, dictionaries, and a list of dictionaries instead of predefined lists for the recipes.

***For this project, you will be working individually. You should start by studying the application’s architecture in the base code and modifying the project to comply with the minimum requirements.***

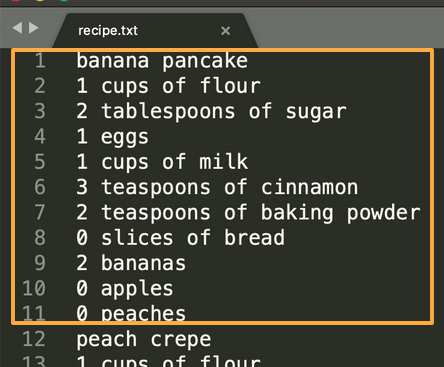
This Project will have **four tasks**, and they will be **submitted separately** on Moodle using CodeRuner. For full credit, you must develop the program’s required features by reusing as little code as possible (avoid code redundancy and duplication) , and following the project’s structure. You would need to modify the given program to perform the assigned tasks, but do not change the format of the output (order, appearance, or contents of the output messages (except when specifically noted in the Task)).

*Submission guidelines:*

1. You will submit individual programs to perform each task in Moodle using CodeRuner. It is highly recommended that you test your program in your preferred IDE before it is entered in CodeRuner. You will have **one** opportunity to run the code; it will be programmed to assign a grade of zero if you try to check the code again. **Please make sure you submit the final version of your code in CodeRuner.**
2. **After submitting your program, you are required to answer some questions corresponding to the project evaluation. Failing to complete the evaluation form will result in a grade of F.**
3. **NO PROJECTS WITH COMPILER ERRORS WILL BE ACCEPTED**.

**Description:**

**Task 1:**

Create a **new**  program to create a file called **recipe.txt** that will contain the name of each recipe and its ingredients. Your program should use the recipes defined at the beginning of the *base code* (the lists representing each recipe). You need to save the name of the recipe with the corresponding amount of required ingredients and its units. The information in the file must be in the following format:

<name of recipe1>

<ingredients of recipe1>

<name of recipe2>

<ingredients of recipe2>

<name of recipe3>

<ingredients of recipe3>

...

<name of recipeN>

<ingredients of recipeN>

**Fig.1** Output file for Task 1.

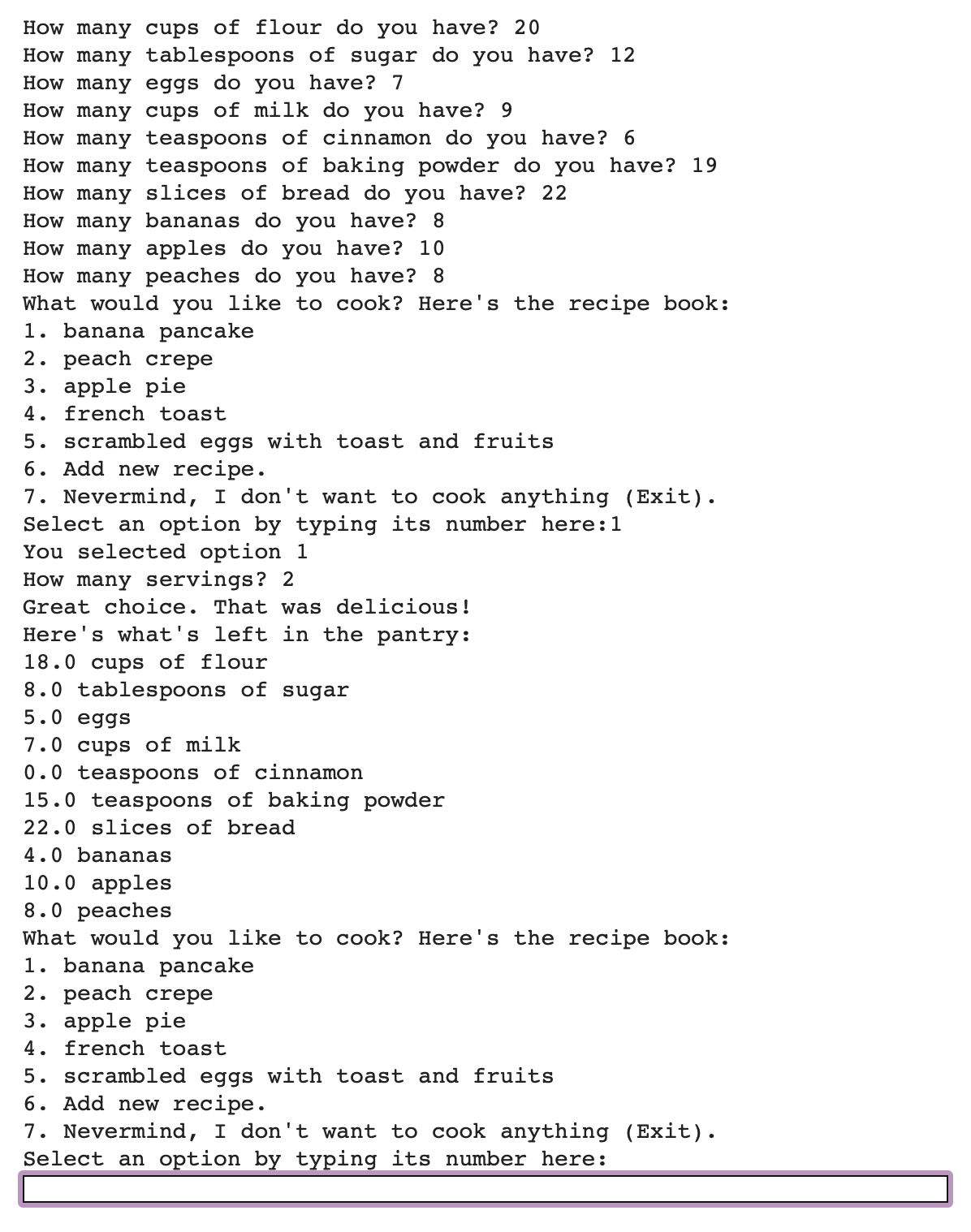
**Task 2:**

Remove the lists defined at the beginning of the *base code*. Modify the *base code* to read the information of each recipe from the file (recipe.txt) created on Task 1 . You have to use **dictionaries** to save **each recipe**, where the key will be the name of the ingredient with its corresponding unit. For example:

{   
 "cups of flour": 12,  
 "tablespoons of sugar": 23,  
 "eggs": 24,  
 ...   
}

The **menu** variable will now change from a "list of lists" to a **list of dictionaries**. The **pantry** also has to be a **dictionary** (same structure as the above example). The **menu\_list** may stay as a list; this list contains the name of each recipe.

Your new program should return the same results as the originally provided base code; the difference will be in the data type of the variables.



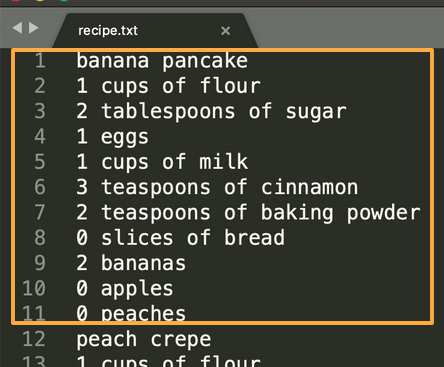
**Fig.2** Example of Task 2. The output is unchanged, only the variable types in the code have been updated.

**Task 3:**

* Make a copy of the file created in **Task 1 (recipe.txt)** and name it **recipe2update.txt**.
* Use the code developed in **task 2** and add the required commands to append new recipes to **recipe2updated.txt** file.

Your program still prints the same results as the provided code; but now you should have modified it to use dictionaries to save and process the data (task 2). In addition it will write any new recipe in the file **recipe2update.txt.**

You will run your code outside of CodeRuner to validate the file created correctly.

Example:

<name of recipe1>

<ingredients of recipe1>

<name of recipe2>

<ingredients of recipe2>

<name of recipe3>

<ingredients of recipe3>

...

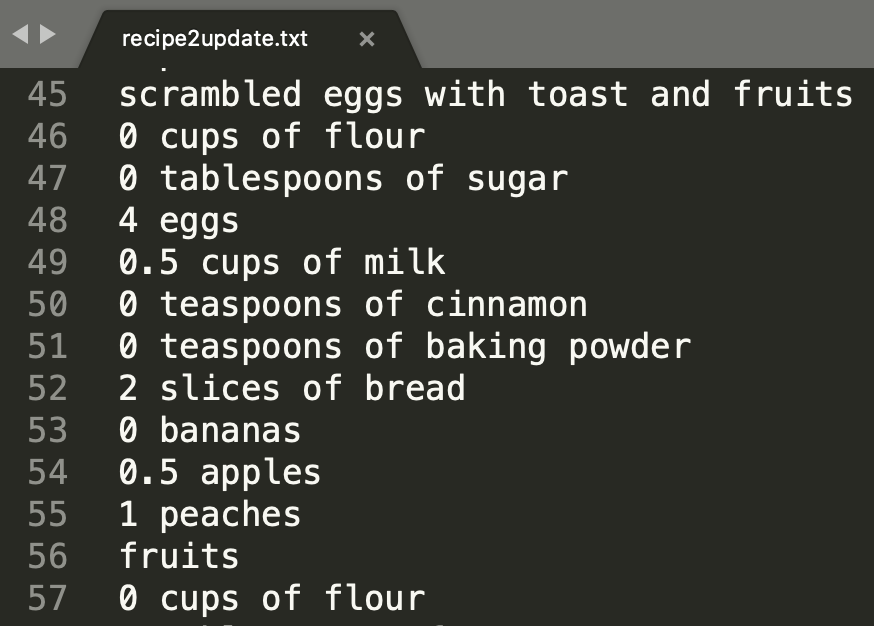
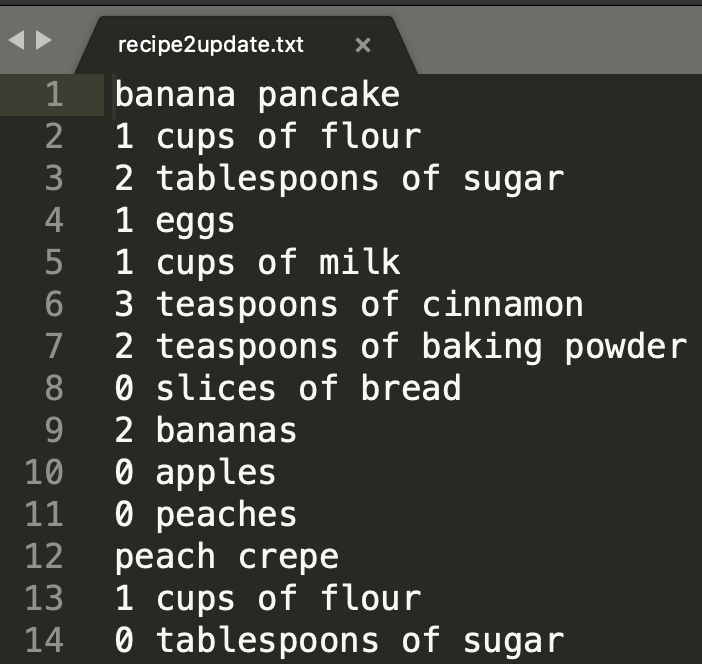
<name of recipeN>

<ingredients of recipeN>

**<name of new recipe>**

**<ingredients of new recipe>**

**Fig.1** Output file for Task 1.



**Fig.3** Output file for Task 3.

**Task 4:**

Modify your program developed in **Task 3** to verify and show the recipes that you can prepare based on the recipes menu and the available ingredients in the pantry. This means that the program does not show the recipes for which the ingredients are not available/enough in the pantry.  **Hint:** To perform this task, you may want to modify the existing functions or create a new one.

You will also need to adjust where the program asks the number of servings -- it must be requested before selecting the recipe; that way, you’ll be able to confirm if the available ingredients are enough. The program developed in task 3 requests the number of servings *after* selecting a valid option from the menu. Now it should ask the *number of servings* *beforehand,* to check if there are enough ingredients for the entered number of servings of the available recipes.

Example:

How many cups of flour do you have? 3

How many tablespoons of sugar do you have? 5

How many eggs do you have? 2

How many cups of milk do you have? 3

How many teaspoons of cinnamon do you have? 2

How many teaspoons of baking powder do you have? 1

How many slices of bread do you have? 5

How many bananas do you have? 5

How many apples do you have? 5

How many peaches do you have? 5

How many servings? 1

What would you like to cook? Here's the recipe book:

1. peach crepe

2. apple pie

3. Add new recipe.

4. Nevermind, I don't want to cook anything (Exit).

Select an option by typing its number here:

**Grading Rubric**

**Functionality (70 pts):** The code can be executed and provides the correct solution to the assigned tasks using the control structures, function, list, dictionaries, tuples, and files accurately. If the code does not execute, it will receive 0.

* Task 1 [10 pts]
  + Create the file **recipes.txt** following the specified format.
* Task 2 [10 pts]
  + Read the file **recipes.txt** and store the values as specified.
  + The program runs correctly; your program should return the same results as the base code.
* Task 3 [20 pts]
  + Add new recipes to the file **recipes2updated.txt** following the specified format.
  + The program runs correctly; your program should return the same results as the base code. We will run your code outside CodeRuner to validate the file is created correctly.
* Task 4 [30 pts]
  + The program runs correctly; your program displays only the recipe menu with the available options, based on the pantry ingredient. The program should not show recipes without enough ingredients as available options.

**Design and efficiency (15pts):** The project follows the original design of the given code

results, and the repository.

* Not efficient at all. [0 pts]
* Slightly inefficient [8 pts]
* Only a couple of inefficient remarks [15 pts]

**Legibility/Style (5pts):**

* Not documented at all or none conventions were followed [0 pts]
* Half of the code is documented [2.5 pts]
* More than 90% of the code is documented and/or only a few to none convention were broken [5 pts]

**Description of the solution design (10pts):**  Provide details about the evaluated strategies to provide the final code implementation during the evaluation form.

**Total: 100pts**