COSC480 Semester 1 2023
Project Report

Michelle Visscher
84632465

1. Instructions on how to run the project.

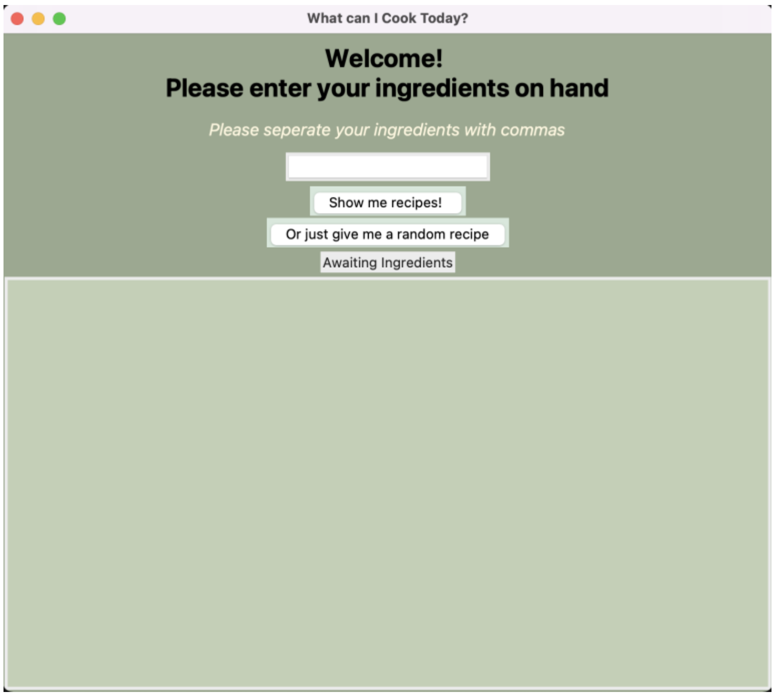
a) Installing necessary dependencies:

Please install the python libraries json, tkinter and random if this is not already done.

Please save the file 'recipes_raw_nosource_epi.json' as attached in project file to the same directory as the code.

b) Accessing the .py file:

Upon opening the .py file, please run the file to access the GUI. This should look like figure 1.



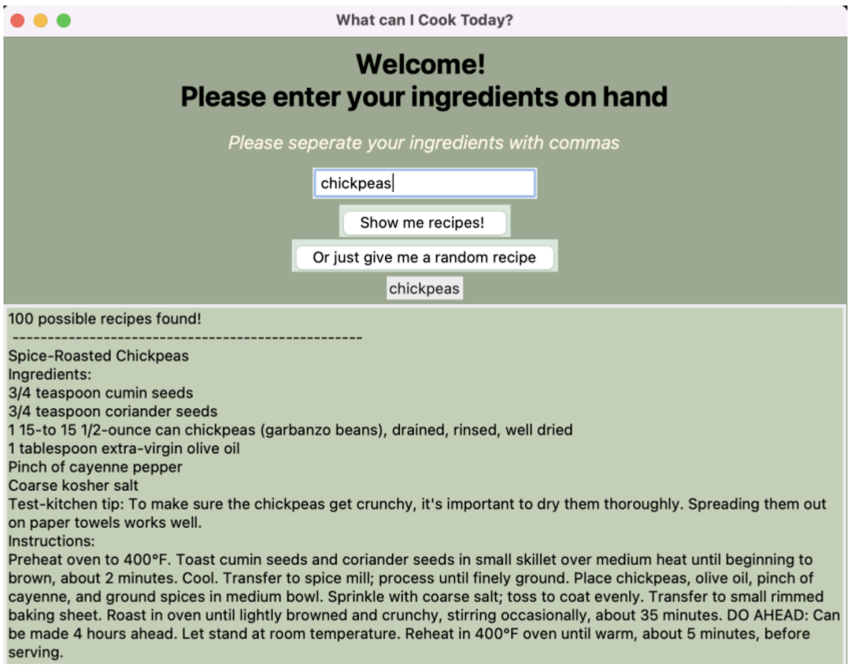
What can I Cook Today?
Welcome!
Please enter your ingredients on hand
Please seperate your ingredients with commas
Show me recipes!
Or just give me a random recipe
Awaiting Ingredients

Figure 1: GUI.

c) What can I cook today? An example:

If you would like to cook with specific ingredients, please enter these in the entry box, separated by commas. Then, click the 'Show me recipes!' button to retrieve your recipes based on your entry.

For example, say you had some chickpeas in the pantry and needed inspiration on what to do with these. The Figure 2 illustrates the results from this entry in the text box.


What can I Cook Today?
Welcome!
Please enter your ingredients on hand
Please seperate your ingredients with commas
chickpeas
Show me recipes!
Or just give me a random recipe
chickpeas

```
100 possible recipes found!
--------------------------------------------------
Spice-Roasted Chickpeas
Ingredients:
3/4 teaspoon cumin seeds
3/4 teaspoon coriander seeds
1 15-to 15 1/2-ounce can chickpeas (garbanzo beans), drained, rinsed, well dried
1 tablespoon extra-virgin olive oil
Pinch of cayenne pepper
Coarse kosher salt
Test-kitchen tip: To make sure the chickpeas get crunchy, it's important to dry them thoroughly. Spreading them out
on paper towels works well.
Instructions:
Preheat oven to 400°F. Toast cumin seeds and coriander seeds in small skillet over medium heat until beginning to
brown, about 2 minutes. Cool. Transfer to spice mill; process until finely ground. Place chickpeas, olive oil, pinch of
cayenne, and ground spices in medium bowl. Sprinkle with coarse salt; toss to coat evenly. Transfer to small rimmed
baking sheet. Roast in oven until lightly browned and crunchy, stirring occasionally, about 35 minutes. DO AHEAD: Can
be made 4 hours ahead. Let stand at room temperature. Reheat in 400°F oven until warm, about 5 minutes, before
serving.
```

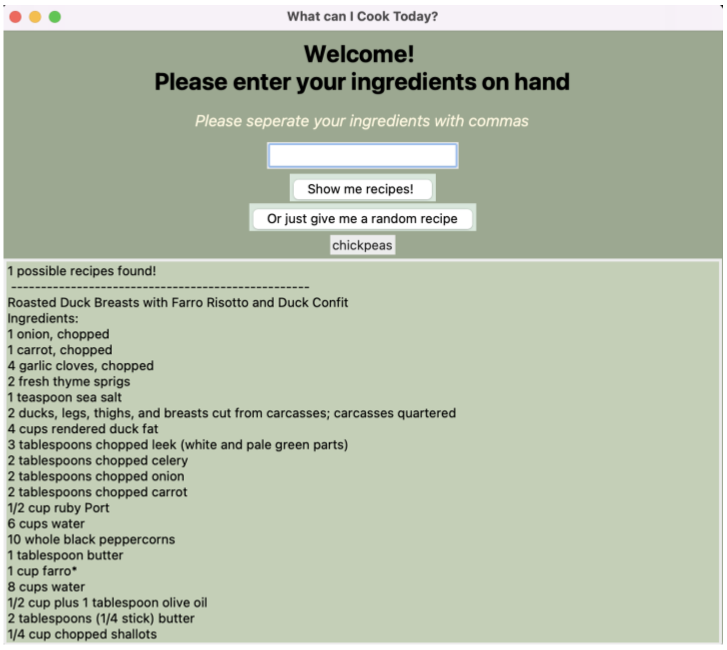
Figure 2: Show me recipes! Result.

The text box will fill with the number of possible recipes found (to a maximum of 100 recipes), followed by a dashed line indicator, recipe title, ingredients and recipe instructions as illustrated in Figure 2. Each subsequent recipe is separated by a dashed line to clearly indicate a new recipe.

If you have more ingredients or wish to change the ingredients, simply re-type your entry into the entry box and click 'show me recipes!' again. This will refresh the search results.

d) Feeling Lucky? A random recipe is given:

If you don't have any specific ingredients in mind to cook with and simply wish to randomly generate some fun new inspiration, simply click the 'Or just give me a random recipe' button. A random recipe will replace the text in the text box. This button can be re-clicked as many times as you like until you find a good recipe! Only one recipe will be returned at a time, as illustrated in Figure 3. The last ingredient given will still appear in the label box below the button.


What can I Cook Today?
Welcome!
Please enter your ingredients on hand
Please seperate your ingredients with commas
Show me recipes!
Or just give me a random recipe
chickpeas

```
1 possible recipes found!
 ----------------------------------------------
Roasted Duck Breasts with Farro Risotto and Duck Confit
Ingredients:
1 onion, chopped
1 carrot, chopped
4 garlic cloves, chopped
2 fresh thyme sprigs
1 teaspoon sea salt
2 ducks, legs, thighs, and breasts cut from carcasses; carcasses quartered
4 cups rendered duck fat
3 tablespoons chopped leek (white and pale green parts)
2 tablespoons chopped celery
2 tablespoons chopped onion
2 tablespoons chopped carrot
1/2 cup ruby Port
6 cups water
10 whole black peppercorns
1 tablespoon butter
1 cup farro*
8 cups water
1/2 cup plus 1 tablespoon olive oil
2 tablespoons (1/4 stick) butter
1/4 cup chopped shallots
```

Figure 3: Random recipe results.

When you are finished with the application, imply enter the red exit button as on a normal application.

2. The development process.

a) Background and project goals

The background of this python project lies in the absence of an available and easily accessible product from which I can quickly generate some recipe ideas based off ingredients I have on-hand, or I wish to cook with. The process of finding a recipe online can sometimes be time consuming, not only to find a good recipe but also reading through authors' lengthy introductory stories.

My main goal for the project is to make a computer application that would be able to take ingredients that the user inputs and is able to return a some suitably matching recipes based off these ingredients. Alongside this goal, I also wish to implement a random-recipe generator part of this project, where a random recipe from the dataset is called. This would allow for exploring different recipes that the user may not have thought of previously or wishes to try something new.

The specifications of this goal include:

- Finding a suitable set of recipes which can be loaded easily in python.

- Producing suitable logic to match the ingredients the user defines to ingredients in the recipes.
- Using tkinter to build a graphical user interface (GUI) that the user can easily interact with. This would include a space for the user to enter their ingredients, a button to click to access the program and return some recipes.
- If time allows, building a button to return a single random recipe.

b) <u>Code and development outline</u>

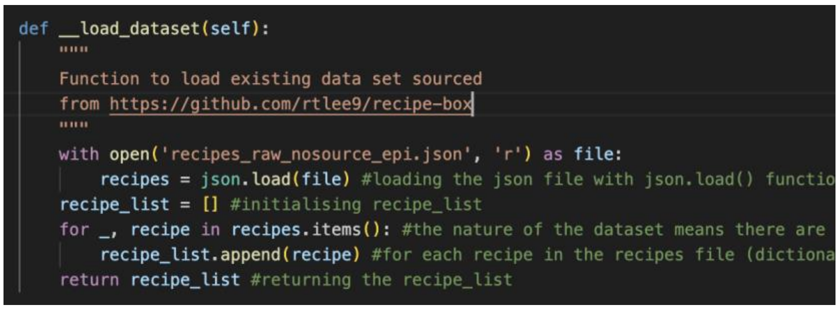
i. *<u>Finding a recipe set</u>*

Multiple avenues for sourcing suitable recipes were explored, including API usage to access web-based recipes, web-scraping a website for recipes, or finding a pre-scraped set of suitable recipes. Upon exploration and advice, finding a suitable set of pre-scraped recipes was the way to go for this project, as this took out the time-consuming aspects of web-scraping or dealing with API keys, allowing me to spend more time on the actual python project at hand.

A set of recipes was found via an online source (Lee, 2017; rtlee9, 2018). This set of web-scraped recipes had been previously procured by the author and formatted as a JSON file. The option of 3 recipe sets was available; I chose the option of the Epicurious website recipes (Epicurious) as I am familiar with their content and their offerings of interesting recipes.

ii. *<u>Importing files into python project</u>*

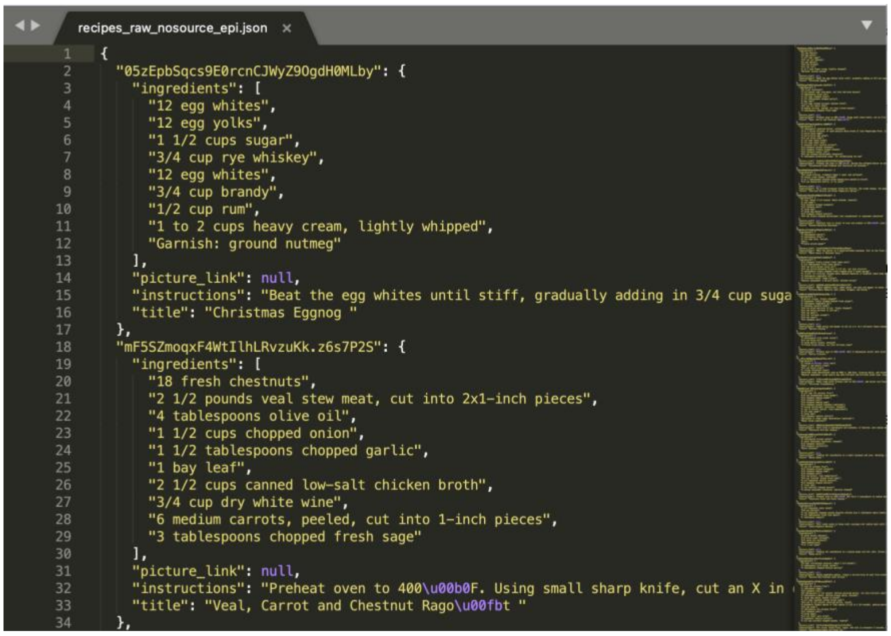
The structuring of the file in JSON format made importing this file into python a relatively simple task, as the JSON library and load() function allows for easy loading of JSON files, as shown in figure 4.



```
def __load_dataset(self):
    """
    Function to load existing data set sourced
    from https://github.com/rtlee9/recipe-box
    """
    with open('recipes_raw_nosource_epi.json', 'r') as file:
        recipes = json.load(file) #loading the json file with json.load() functio
    recipe_list = [] #initialising recipe_list
    for _, recipe in recipes.items(): #the nature of the dataset means there are
        recipe_list.append(recipe) #for each recipe in the recipes file (dictiona
    return recipe_list #returning the recipe_list
```

<u>Figure 4: Loading the dataset</u>

The nature of the JSON file (figure 5) shows that the entire recipe set is contained in nested dictionaries, wherein the first key of this dictionary is a random set of numbers and letters identifying each recipe. This initial key is of no relevance for us. The second 'key' in this nested dictionary is the ingredients, followed by the key picture link, key instructions, and finally the key title of the recipe.

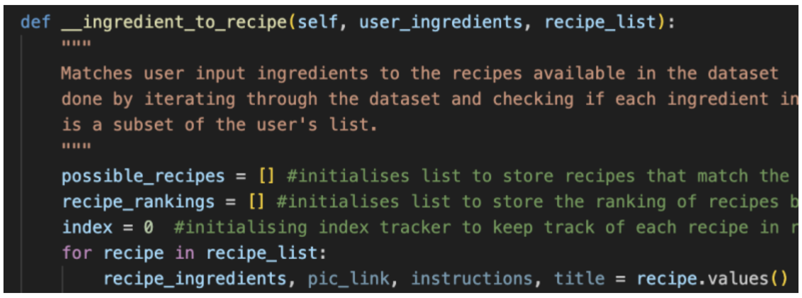


recipes_raw_nosource_epi.json

```
{
  "05zEpbSqcs9E0rcnCJWyZ9OgdH0MLby": {
    "ingredients": [
      "12 egg whites",
      "12 egg yolks",
      "1 1/2 cups sugar",
      "3/4 cup rye whiskey",
      "12 egg whites",
      "3/4 cup brandy",
      "1/2 cup rum",
      "1 to 2 cups heavy cream, lightly whipped",
      "Garnish: ground nutmeg"
    ],
    "picture_link": null,
    "instructions": "Beat the egg whites until stiff, gradually adding in 3/4 cup suga
    "title": "Christmas Eggnog "
  },
  "mF5SZmoqxF4WtIlhLRvzuKk.z6s7P2S": {
    "ingredients": [
      "18 fresh chestnuts",
      "2 1/2 pounds veal stew meat, cut into 2x1-inch pieces",
      "4 tablespoons olive oil",
      "1 1/2 cups chopped onion",
      "1 1/2 tablespoons chopped garlic",
      "1 bay leaf",
      "2 1/2 cups canned low-salt chicken broth",
      "3/4 cup dry white wine",
      "6 medium carrots, peeled, cut into 1-inch pieces",
      "3 tablespoons chopped fresh sage"
    ],
    "picture_link": null,
    "instructions": "Preheat oven to 400\u00b0F. Using small sharp knife, cut an X in
    "title": "Veal, Carrot and Chestnut Rago\u00fbt "
  },
```

Figure 5: JSON file containing recipes

To extract the relevant information from this dictionary, these values were unpacked from each recipe in the imported recipe list, as evidenced in figure 6.



```
def __ingredient_to_recipe(self, user_ingredients, recipe_list):
    """
    Matches user input ingredients to the recipes available in the dataset
    done by iterating through the dataset and checking if each ingredient in
    is a subset of the user's list.
    """
    possible_recipes = [] #initialises list to store recipes that match the
    recipe_rankings = [] #initialises list to store the ranking of recipes b
    index = 0  #initialising index tracker to keep track of each recipe in r
    for recipe in recipe_list:
        recipe_ingredients, pic_link, instructions, title = recipe.values()
```

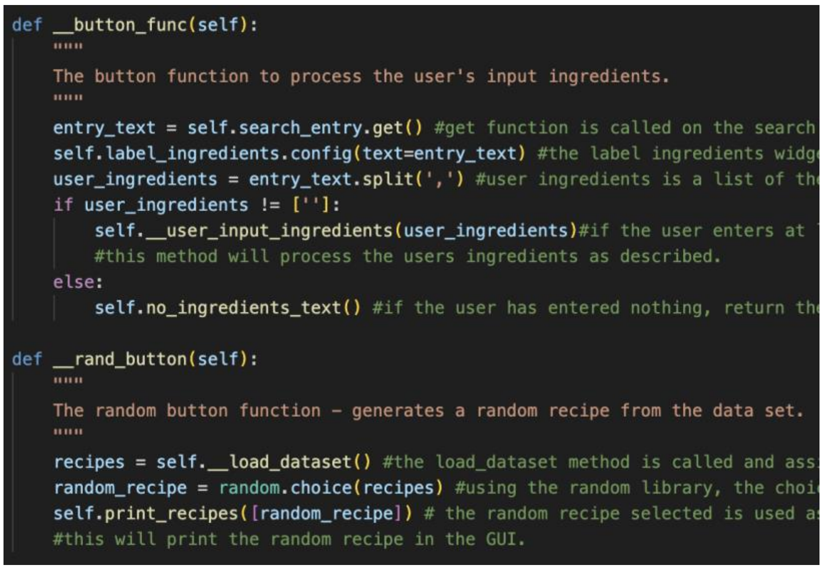
Figure 6: Unpacking nested dictionary.

iii. *Matching ingredients*

The logic behind matching user-defined ingredients to recipe ingredients has been through many iterations. The challenges behind this can be found in section 2.d (below). The current function logic to match ingredients uses a rank-based system which matches the user's input ingredients with the recipes in the dataset by iterating through each recipe and checking if each user ingredient is in a recipe's ingredients. This method tracks the recipes which match in a score-based system, and then ranks the recipes based on the number of highest matching recipes, sorts these from highest to lowest, and returns a list of only the top-matching ingredients, to a maximum of 100 recipes (many thousands of recipes could be found using this functionality, so limiting this is crucial for usability in a reasonable time). This final function, __ingredient_to_recipe() takes user_ingredients and the recipe_list as parameters, and uses these to find matching recipes. I found this rank system to be the most effective use when having such a large dataset.

iv. *Building GUI with tkinter*

The structure of the GUI that the user will interact with was built with the python library Tkinter. Tkinter is python's standard GUI. The GUI was initialised with the tkinter Tk() function, which sets up the primary window. This window is used as the master window on which following widgets are placed. The welcoming title label was initialised and set using the grid() method, which allows placement of objects in a grid-like fashion, where columns and rows are specified. This was the easiest way of centring the labels and other widgets in the window, and meant that objects could each be placed below one another. An instructional label was placed below the title label, informing the user how to enter their ingredients into the entry box (separated by commas).
The entry box was initialised using the Entry() method, which sets an entry box for the user. Below this, the Button() method was used to place a button below the entry box which the user could click. The button would then follow the command 'button_func', which will initiate the main matching code to return recipes to the user. Another button was placed below this to give the user the option of returning a random recipe from the dataset following the 'rand_button' function. The respective functionality of these buttons can be observed in figure 7. The 'button_func' initially processes the user's entry (the user's ingredients). The method processes the text from the entry box and splits with a comma delimiter. This method then checks that if a string has been entered, the method user_input_ingredients is called which will process the user's ingredients and print these in the text box in the GUI. If no text was entered, the method will instead call to a function which will simply print an error message (figure 8).



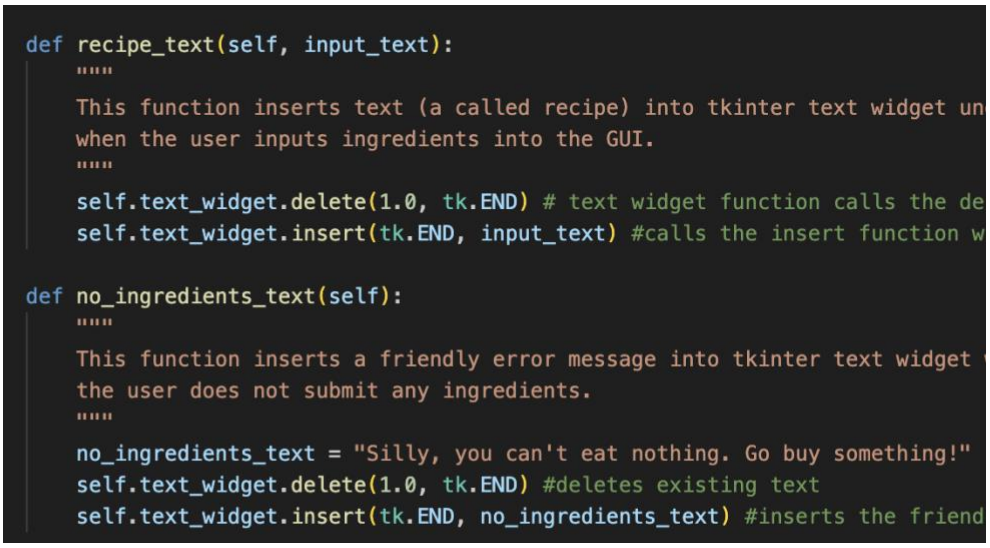
```
def __button_func(self):
    """
    The button function to process the user's input ingredients.
    """
    entry_text = self.search_entry.get() #get function is called on the search
    self.label_ingredients.config(text=entry_text) #the label ingredients widg
    user_ingredients = entry_text.split(',') #user ingredients is a list of th
    if user_ingredients != ['']:
        self.__user_input_ingredients(user_ingredients)#if the user enters at
        #this method will process the users ingredients as described.
    else:
        self.no_ingredients_text() #if the user has entered nothing, return th

def __rand_button(self):
    """
    The random button function - generates a random recipe from the data set.
    """
    recipes = self.__load_dataset() #the load_dataset method is called and ass
    random_recipe = random.choice(recipes) #using the random library, the choi
    self.print_recipes([random_recipe]) # the random recipe selected is used a
    #this will print the random recipe in the GUI.
```

Figure 7: Button functionality.

The random functionality works by calling the recipes from the dataset, choosing a random recipe via the random.choice() function and printing this recipe.

Finally, a text widget was initialised for the recipe text to be placed in. This text widget is set to wrap the text according to each word. Figure 8 demonstrates the code used to access this text widget and to import text when called upon. The text in the text widget is first deleted and then inserts new text every time it is called up (every time the user would click the button).



```
def recipe_text(self, input_text):
    """
    This function inserts text (a called recipe) into tkinter text widget un
    when the user inputs ingredients into the GUI.
    """
    self.text_widget.delete(1.0, tk.END) # text widget function calls the de
    self.text_widget.insert(tk.END, input_text) #calls the insert function w

def no_ingredients_text(self):
    """
    This function inserts a friendly error message into tkinter text widget 
    the user does not submit any ingredients.
    """
    no_ingredients_text = "Silly, you can't eat nothing. Go buy something!"
    self.text_widget.delete(1.0, tk.END) #deletes existing text
    self.text_widget.insert(tk.END, no_ingredients_text) #inserts the friend
```

Figure 8: Text loading in the text widget.

v. *Class RecipeApp*

The Tkinter GUI was initially built separately from the underlying 'backend' code for proof of concept. Once I was comfortable with the GUI, the GUI and backend were integrated together using an object-oriented style of programming, creating the main class RecipeApp(). This style allowed for clean encapsulation of the code in one class and will improve future scalability of the project.

c) Goals reached and minimum viable product

Most of the project goals initially set out in the proposal and specifications have been met. The project has been successful in having a useable GUI for the user to interact with in a simple manner and has both functions of returning recipes which match to user ingredients and a random recipe function, which was successfully implemented using python's Random library. The GUI itself is functional, easy to use and relatively aesthetic, making this project successful in meeting its minimum viable product goals. I am personally very pleased with the project, particularly with the random recipe feature, which is a fun addition and allows for easy access to cooking inspiration.

d) Challenges faced and resolutions

i. Logic and dataset structure

One major challenge faced (and is still ongoing) is that of the logic behind matching the recipes to the user defined ingredients. Many iterations of this logic were engaged with, initially using python's set data type and logic to test if the user defined ingredients were in the set of ingredients for each recipe. This process used a set notation to return recipes which had their set of ingredients match the user defined ingredients. This process did not function as desired however, because the match of ingredients would not be exactly correct. For example, if the user had given the ingredient 'milk', then only recipes with a string matching 'milk' would be returned. This caused problems because often the ingredient string would be '1/4 cup milk' and thus would not return a match.
Further issues with this process made this a non-viable approach, mainly stemming from the fact that the dataset chosen to work with (Epicurious) was very large and contained an estimated 20,000 recipes (60mb). This meant that any loops needed to check if an ingredient was in each recipe would take an impractical amount of time to process.
As a result of there being so many recipes, there is also an issue with there being many results for each user-defined ingredient, with a common ingredient like 'butter' returning thousands of recipes. To address this, the current logic which matches ingredients uses a rank-based system which matches the user's input ingredients with the recipes in the dataset by iterating through each recipe and checking if each user ingredient is in a recipe's ingredients. This method then ranks the recipes based on the number of matching recipes, sorts these from highest to lowest and returns a list of only the top-matching ingredients, to a maximum of 100 recipes. Further work could be done to find a more elegant solution to this problem.

ii. Tkinter

A challenge of this project was working with and learning how to use Tkinter. An earlier iteration of the Tkinter GUI used the more common place() method to set objects in the Tkinter window, however difficulties were experienced as I tried to insert a picture (application logo) and scrollbar into this window, ruining the centred placement and aesthetic of the main objects in the application. Instead, a more simple approach was used whereby the objects were set as via grid() method. This simplified version improves upon earlier iterations in both overall look of the GUI and useability, by including only title labels, entry box, buttons and returning text box, making for a clean interface.

iii. Classes – OO programming

After discussions and following recommendations, an object-oriented programming approach was taken to refactor the code into a readable and clean system. This came as a welcome challenge and allowed for further exploration with this style of programming. A class RecipeApp was built, wherein the initialisation function served as the basis for setting up the Tkinter GUI, and the additional methods were added in an integrated as needed.

e) *Future developments and improvements*

For future development and improvement on the current python project, there are several planned implementations to improve the user experience and functioning of the code.

- Firstly, the existing dataset will be improved by cleaning the 'instructions' section. As the dataset currently stands, the instruction section is unfortunately duplicated in one string format. To address this, a python script could be produced which checks if the string is duplicated and remove the second duplicated part of the string.

- Adjustments are also planned for the logic section of the code, where user-defined ingredients are matched to respective recipes. As discussed previously, the nature of this large dataset means that there are many likely matches between ingredients, thus returning many possible recipes. These adjustments could either include changes to the logical matching of the code, and improvements to the dataset itself. Ideally, this project would hold my own personal recipes, so it's a consideration to re-do this dataset for personal use.

- Changes to improve the code itself are also planned implementations, where some refactoring of code may be necessary to improve on useability in the future.

- Pagination would ideally be implemented to improve the user experience of the GUI. This would remove the endless scrolling aspect of the GUI and make certain recipes easier to find, with say 5 recipes per page.

- As a stretch goal, this recipe app could also hold and 'remember' a set of ingredients the user may have (like a pantry) and be able to return all possible recipes one could cook with these ingredients. A shopping list addition (alerts to say when things have run out etc) is also possible for future iterations. Finally, cloud compatibility would be implemented with this project so that the user could access this project as a web application.

References:

Epicurious. “Epicurious – Recipes, Menu Ideas, Videos & Cooking Tips.” Epicurious, 2023, https://www.epicurious.com.

Lee, Ryan. “Recipe Box | Eight Portions.” Eight Portions, Ryan Lee, 14 Mar. 2017, https://eightportions.com/datasets/Recipes/.

rtlee9. “GitHub - Rtlee9/Recipe-Box: Scrape Lists of Recipes.” GitHub, 22 Mar. 2018, https://github.com/rtlee9/recipe-box.