# Recipes

Imagine you have a unique cookbook with different recipes. Each recipe comes with its own unique name and a list of ingredients, each with its own specific quantity. Now, let's embark on an imaginative journey to discover all the possibilities of ingredient combinations for each recipe, aiming to unearth the most tantalizing and delightful taste experiences. This code is designed to assist in organizing and generating recipe files. It provides a convenient way to extract dish names and ingredients from a recipe data file and create individual recipe files for each dish. The script utilizes regular expressions to parse the recipe data, allowing for flexibility in handling different file formats. With customizable formatting options, users can specify the title, separators, and text alignment for the generated recipe files. It includes functionality to extract recipe names and ingredients from a text file, and also to generate recipe files with different combinations of ingredients.

## Features

- Extract names of dishes from a recipes data file.
- Extract ingredients and their weights for each dish.
- Generates combinations of ingredients for different recipes.
- Generate recipe files with different combinations of ingredients.
- Allows the chef to rate each combination based on taste, texture, and looks.
- Calculates the total rating for each combination.
- Generates a chef scorecard file with the ratings for all combinations.
- Provides a display of the chef's scorecard using Tkinter.

## Usage

1. Ensure that the recipe data file "recipes_data.txt" is present in the "Recipes" folder otherwise update the `file_path` variable in the `__main__.py` with the correct path to your recipes data file.
2. Ensure you have a recipes data file in the specified format.

```
[Recipe Name]:
------------------------------------------------------------------
[Ingredient 1] = [Quantity 1], [Quantity 2], [Quantity 3], ...
[Ingredient 2] = [Quantity 1], [Quantity 2], [Quantity 3], ...
[Ingredient 3] = [Quantity 1], [Quantity 2], [Quantity 3], ...
.
.
.
------------------------------------------------------------------
...
```

3. Modify the main() function in the `__main__.py` file if needed.
4. Run the program by executing the following command:

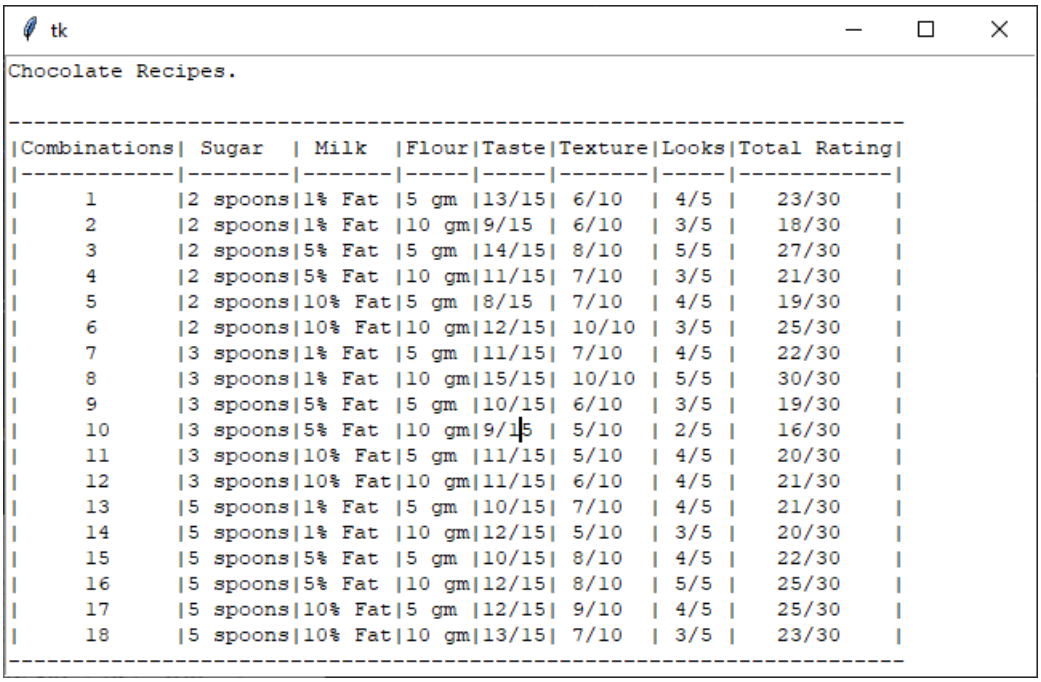
```
python __main__.py
```

5. The program will generate separate recipe files in the "Recipes" folder, each containing various combinations of ingredients for a specific recipe.
6. By default, the output format of recipe file is as:

```
[Recipe Name].

-------------------------------------------------------------------------------------------------------
|  Combinations  |     [Ingredient 1]      |      [Ingredient 2]      |     [Ingredient 3]      | ...
|----------------|-------------------------|--------------------------|-------------------------| ...
|       1        |      [Quantity 1]       |       [Quantity 1]       |      [Quantity 1]       | ...
|       2        |      [Quantity 1]       |       [Quantity 1]       |      [Quantity 2]       | ...
|       3        |      [Quantity 1]       |       [Quantity 2]       |      [Quantity 3]       | ...
|       4        |      [Quantity 1]       |       [Quantity 2]       |      [Quantity 4]       | ...
.
.
.
-------------------------------------------------------------------------------------------------------
```

7. The program will prompt you to enter the name of a recipe to rate the combinations.
8. For each combination, enter the taste, texture, and looks ratings when prompted. The total rating will be calculated automatically.
9. Press the Enter key to continue rating the next combination.
10. Once you have finished rating all combinations, a chef scorecard file "chef_scorecard.txt" will be generated in the "Recipes" folder.
11. A window will pop up automatically and display the scorecard. For instance:




# Methods

## main.py

The main() function is defined, which serves as the main entry point (boot function) for the program.

```
from recipes import *
from display import *

file_path = "Recipes/recipes_data.txt"

def main():
    recipes = Recipes(
        file_path, output_file_path = "Recipes/",
        serial_title = "Combinations", column_separator = "|",
        separator = "-", text_position = "center", left_padding = 0,
        right_padding = 0, column_width = 0
    )
    recipes.get_recipe_names()
    recipes.get_ingredients()
    recipes.get_recipes()
    recipes.try_recipe()

    display(recipes.scorecard_file)

if __name__ == "__main__":
    main()
```

## recipes.py

1. `__init__(self, recipes_file_name, output_file_path, serial_title, column_separator, separator, text_position, left_padding, right_padding, column_width)`
    - Constructor method that initializes the `Recipes` object with various parameters, such as file paths, separators, padding, and column width.
    - `recipes_file_name` : The name of the file recipes_data.txt containing the recipe data.
    - `output_file_path` : The path where the generated recipe files and scorecard file will be saved. By default it is `"Recipes/"` .
    - `serial_title` : The title to be used for the serial number column in the generated recipe files. By default it is `"Combinations"` .
    - `column_separator` : The character used as the column separator in the generated recipe files. By default it is `"|"` .
    - `separator` : The character used as the separator between rows in the generated recipe files. By default it is `"-"` .
    - `text_position` : The alignment of the text in the generated recipe files (left, center, right). By default it is `"center"` .
    - `left_padding` : The number of spaces to add on the left side of each cell in the generated recipe files. By default its value is `"0"` .
    - `right_padding` : The number of spaces to add on the right side of each cell in the generated recipe files. By default its value is `"0"` .
    - `column_width` : The maximum width of each column in the generated recipe files. If set to 0, it adjusts dynamically based on the content length. By default its value is `"0"` .
2. `get_recipe_names(self)`

    - Reads the recipes_data.txt file and extracts the names of the dishes. Returns a list of recipe names.
3. `get_ingredients(self)`
    - Extracts the ingredients for each recipe from the recipes_data.txt file and organizes them in a dictionary. Returns a dictionary with recipe names as keys and ingredient names with their possible weights as values.
4. `get_aligned_text(self, text, spacing)`
    - Sets the alignment of the text and returns aligned text based on the specified alignment (left, center, right) and padding.
5. `get_separators(self, max_cell_lengths, number_of_cells, add_column_separation=True)`
    - Generates a separator line based on the maximum cell lengths and number of cells in a table. If `add_column_separation` is True, it includes column separators; otherwise, it only includes separators between rows.
6. `get_combinations(self, ingredients, number_of_cells)`
    - Generates all possible combinations of ingredients for each recipe and divides them into chunks of size number_of_cells. Returns a list of combinations.
7. `write_combinations(self, file, headers, ingredients)`
    - Writes the combinations of ingredients for each recipe to a file. Formats the content with proper alignment and separators.
8. `get_recipes(self)`
    - Generates separate text files for each recipe, containing all possible combinations of ingredients. Uses the `write_combinations` method to write the combinations to the files.
9. `try_recipe(self)`
    - Allows the user to try each recipe combination and rate it for taste, texture, and looks. Collects the ratings and generates a chef's scorecard file.
    - The scorecard is user controlled.

### display.py

1. `display(file_name)`
    - Opens a window and displays the chef's scorecard text file. Reads the content of the file and inserts it into a `Text` widget in a tkinter window.
2. `open_file(file_name)`
    - Opens the chef's scorecard text file and reads its content.

## Requirements

The `re` module, which stands for regular expressions, is part of the standard library in Python. It is available by default when you install Python, and you do not need to install any additional packages or libraries to use it.

To use the "re" module in your Python code, you can simply import it at the beginning of your script or interactive session like this:

```
import re
```

The `tkinter` module is another important part of the standard library in Python. It provides a powerful and easy-to-use interface for creating graphical user interfaces (GUIs) in Python. Just like the "re" module, "tkinter" is available by default when you install Python, and you don't need to install any additional packages or libraries to use it. Just import it at the beginning of your script like this:

```
import tkinter as tk
```

`Python 3.11.3`

## Learn More

To learn more about regular expression and tkinter, take a look at the following resources:

- Python Regular Expression HOWTO
- Python `re` Documentation
- Python `tkinter` Documentation