Recipes and Ingredients: Shopamania
COOK1001

Assignment 1
Semester 1, 2023
CSSE1001/CSSE7030
Updated: 03/03/23 15:44

Due date: 24 March 2023, 16:00 GMT+10

1 Introduction

You are a single parent with four hungry children and the weekly grocery shop is becoming difficult. Luckily, you know a bit of Python and can write a program that generates a shopping list based on what recipes are going to be prepared that week.

2 Getting Started

Download `a1.zip` from Blackboard — this archive contains the necessary files to start this assignment. Once extracted, the `a1.zip` archive will provide the following files:

`a1.py` *This is the only file you will submit* and is where you write your code. *Do not* make changes to any other files.

`constants.py` *Do not modify or submit this file*, this file also contains some constants that will be useful in your implementation. In addition to these, you are encouraged to create your own constants in `a1.py` where possible. This is where you will find some recipes that can be added to the recipe book as well.

Note: You are *not* permitted to add any import statements to `a1.py`. Doing so will result in a deduction of up to 100% of your mark.

3 Terminology

In the context of this assignment:

`recipe_name: str` represents the name of a recipe. **Note:** Recipe names should only contain lowercase letters from the English alphabet without any numbers or special characters.

`recipe: tuple[str, str]` is a `tuple` containing two `str`'s. The first `str` in the `tuple` represents the name of the recipe. The second `str` contains all the ingredients in the form of `'amount measure ingredient_name'`, e.g. `'115 g Nuttelex'` which represents 115 grams of Nuttelex. When multiple ingredients are present, they are separated by a *single comma.* Note that there is **no** space. e.g. `'115 g Nuttelex,50 g sugar'` have a look at `constants.py` for examples.

`recipes: list[tuple[str, str]]` is a `list` of recipes. Each recipe follows the format described above. Duplicates **can** exist.

`ingredient_detail: tuple[float, str, str]` , is a `tuple` containing information of a single ingredient. `(amount, measure, ingredient name)`, e.g. `(115.0, 'g', 'Nuttelex')`.

`cookbook` is a collection of all available recipes and you may use whatever type you feel is appropriate for it. Duplicates do **not** exist in the cookbook.

4 Interaction

This section provides an overview of the interaction. Where prompts and outputs are not explicitly mentioned in this section, please see Section 5.

4.1 Interaction loop

At the beginning of the interaction, the user is prompted with the message

```
Please enter a command: 
```

(note the trailing space) to choose a command. Once a command is entered the user should be prompted again. The valid commands are outlined in Table 1.

Throughout the interaction, the user may add to or remove from the cookbook. They can also add to or remove from the list of recipes as well. **You may assume that your program will not be tested for invalid inputs** that do not match the expected commands in Table 1. The prompt should repeat until the user quit by entering the `"Q"` or `"q"` action.

Input	**Description**
`"H"` or `"h"`	Display a help message.
`"Q"` or `"q"`	Quit.
`"add {recipe}"`	adds a recipe to the list of recipes.
`"rm {recipe}"`	removes a recipe from the collection.
`"rm {ingredient_name} {amount}"`	removes an ingredient from the shopping list.
`"g"` or `"G"`	generates a shopping list based on the list of recipes.
`"ls"`	list all recipes in shopping cart.
`"ls -a"`	list all available recipes.
`"ls -s"`	display shopping list.
`"mkrec"`	creates a recipe.

Table 1: Potential command that user can choose. `{recipe}` denotes that the *name* of recipe should be provided. Values not surrounded by braces should be taken as string literals.

5 Implementation

This section outlines functions you are **required** to implement in your solution (in `a1.py` only). You are awarded marks for the number of tests passed by your functions when they are tested *independently.* Thus an incomplete assignment with *some* working functions may well be awarded more marks than a complete assignment with faulty functions. Your program must operate *exactly* as specified. In particular, your program's output must match *exactly* with the expected output. Your program will be marked automatically so minor differences in output (such as whitespace or casing) *will* cause tests to fail resulting in a *zero mark* for that test.

Each function is accompanied with some examples for usage to help you *start* your own testing. You should also test your functions with other values to ensure they operate according to the descriptions.

5.1 Required Functions

The following functions **must** be implemented in `a1.py`. They have been *listed in order of increasing difficulty.* It is *highly recommended* that you do not begin work on a later function until each of the preceding functions can *at least* behave as per the shown examples. You may implement additional functions if you think they will help with your logic or make your code easier to understand.

Your first task should be writing function headers for each of these required functions into `a1.py`. For example, you could write

```
def get_recipe_name(recipe: tuple[str, str]) -> str
    """ Return ...

    >>> get_recipe_name(...)
    """
    return
```

in your file and then add usage examples and docstrings based on what you read below. Then completing the assignment means filling in each function properly as is done with ShiFoo.

5.1.1 `num_hours() -> float`

This function should return the number of hours you estimate you spent (or have spent so far) on the assignment, as a *float.* You will not be marked differently for spending more or less time on the assignment. The purpose of this function is to enable you to verify that you understand how to submit to Gradescope as soon as possible, and to allow us to gauge difficulty level of this assignment in order to provide the best possible assistance.

Ensure this function passes the relevant test on Gradescope *as soon as possible.* If the Gradescope tests have been released, you must ensure this function passes the relevant test before seeking help regarding Gradescope issues for any of the later functions. See Section 7.3 for instructions on how to submit your assignment to Gradescope.

5.1.2 `get_recipe_name(recipe: tuple[str, str]) -> str`

Returns the name of the recipe.

Example:

```
    >>> get_recipe_name(('chocolate peanut butter banana shake',
            '1 large banana,240 ml almond milk'))
    'chocolate peanut butter banana shake'
```

```
>>> get_recipe_name(('cinnamon rolls',
        '480 ml almond milk,170 g Nuttelex'))
'cinnamon rolls'
```

5.1.3 parse_ingredient(raw_ingredient_detail: str) -> tuple[float, str, str]

Return the ingredient breakdown from the details amount, measure, and ingredient.

Example:

```
>>> parse_ingredient('0.5 tsp coffee granules')
(0.5, 'tsp', 'coffee granules')

>>> parse_ingredient('1 large banana')
(1.0, 'large', 'banana')
```

5.1.4 create_recipe() -> tuple[str, str]

Return a recipe in the `tuple[str, str]` format after a series of prompting. The recipe name is prompted first followed by continuous ingredient prompting until an empty string is entered (enter or return key press with no text).

Example:

```
>>> create_recipe()
    Please enter the recipe name: peanut butter
    Please enter an ingredient: 300 g peanuts
    Please enter an ingredient: 0.5 tsp salt
    Please enter an ingredient: 2 tsp oil
    Please enter an ingredient:
('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')
```

5.1.5 recipe_ingredients(recipe: tuple[str, str]) -> tuple[tuple[float, str, str]]

Return the ingredients of a recipe amount, measure, and ingredient. This transforms a given recipe from the string form into the tuples form.

Example:

```
>>> recipe_ingredients(('peanut butter',
    '300 g peanuts,0.5 tsp salt,2 tsp oil'))
((300.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'))
```

5.1.6 add_recipe(new_recipe: tuple[str, str], recipes: list[tuple[str, str]]) -> None

Add a given recipe, `new_recipe`, into the list of recipes. *Hint:* This function doesn't return anything. Recall list mutability.

Example:

```
>>> recipes = []
>>> recipe = ('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')
>>> add_recipe(recipe, recipes)
>>> recipes
[('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')]
>>> add_recipe(recipe, recipes)
>>> recipes
[('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil'), ('peanut butter',
'300 g peanuts,0.5 tsp salt,2 tsp oil')]
```

5.1.7 `find_recipe(recipe_name: str, recipes: list[tuple[str, str]]) -> tuple[str, str] | None`

Return a recipe or `None`. This function should attempt to find the recipe by the given recipe name within the list of recipes. If the recipe can not be found then this function should return `None`.

Example:

```
>>> recipes = [('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')]
>>> find_recipe('peanut butter', recipes)
('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')
>>> find_recipe('cinnamon rolls', recipes)
>>> print(find_recipe('cinnamon rolls', recipes))
None
```

5.1.8 `remove_recipe(name: str, recipes: list[tuple[str, str]]) -> None`

Remove a recipe from the list of recipes given the name of a recipe. If the recipe name does not match any of the recipes within the list of recipes then nothing happens.

Example:

```
>>> recipes = [('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil'),
('cinnamon rolls', '480 ml almond milk,115 g Nuttelex,50 g sugar,7 g
active dry yeast,5.5 cup flour,1 tsp salt,170 g Nuttelex,165 g brown
sugar,2 tbsp cinnamon,160 g powdered sugar,30 ml almond milk,0.5 tsp
vanilla extract')]
>>> remove_recipe('brownie', recipes)
>>> recipes
    [('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil'), ('cinnamon
    rolls', '480 ml almond milk,115 g Nuttelex,50 g sugar,7 g active dry
    yeast,5.5 cup flour,1 tsp salt,170 g Nuttelex,165 g brown sugar,2 tbsp
    cinnamon,160 g powdered sugar,30 ml almond milk,0.5 tsp vanilla
    extract')]

>>> remove_recipe('cinnamon rolls', recipes)
>>> recipes
    [('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')]
```

5.1.9 `get_ingredient_amount(ingredient: str, recipe: tuple[str, str]) -> tuple[float, str] | None`

Return the amount and measure of a certain ingredient as a `tuple[float, str]` given the ingredient name as a `str` and a recipe. If the ingredient doesn't exist then nothing happens.

Example:

```
>>> recipe = ('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')
>>> get_ingredient_amount('peanuts', recipe)
(300.0, 'g')
>>> get_ingredient_amount('soy beans', recipe)
```

5.1.10 `add_to_shopping_list(ingredient_details: tuple[float, str, str], shopping_list: list[tuple[float, str, str] | None]) -> None`

Add an ingredient to the shopping list which could be empty or could contain tuples of ingredient details. If the ingredient being added already exist within the shopping list then the *amount* should be combined. If the ingredient does not exist then it can be added without any calculations.
Note: It can be assumed that the *measure* is consistent for all ingredients of the same name. In addition, `ingredient_details` contains all the information about the ingredient being added to the shopping list. Also, the order does not matter.

Example:

```
>>> shopping_list = [(300.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'),
                     (2.0, 'tsp', 'oil')]

>>> add_to_shopping_list((1000.0, 'g', 'tofu'), shopping_list)

>>> shopping_list
[(300.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(1000.0, 'g', 'tofu')]

>>> add_to_shopping_list((1200.0, 'g', 'peanuts'), shopping_list)

>>> shopping_list
[(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(1000.0, 'g', 'tofu')]

>>> add_to_shopping_list((8000.0, 'g', 'tofu'), shopping_list)

>>> shopping_list
[(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(9000.0, 'g', 'tofu')]
```

5.1.11 `remove_from_shopping_list(ingredient_name: str, amount: float, shopping_list: list) -> None`

Remove a certain amount of an ingredient, with the given `ingredient_name`, from the `shopping_list`. If the ingredient exists in the `shopping_list` then the `amount` given as the parameter of this func-

tion should be subtracted from the amount that exists in the `shopping_list`. The ingredient should be removed from the shopping list altogether if the amount goes below 0.
Example:

```
>>> shopping_list = [(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'),
    (2.0, 'tsp', 'oil'), (9000.0, 'g', 'tofu'), (100.0, 'g', 'sugar'),
    (50.0, 'g', 'tomato sauce'), (120.0, 'g', 'rice'),
    (920.0, 'g', 'ice cream')]

>>> remove_from_shopping_list('ice cream', 500.0, shopping_list)

>>> shopping_list
[(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(9000.0, 'g', 'tofu'), (100.0, 'g', 'sugar'), (50.0, 'g', 'tomato
sauce'), (120.0, 'g', 'rice'), (420.0, 'g', 'ice cream')]

>>> remove_from_shopping_list('sugar', 500.0, shopping_list)

>>> shopping_list
[(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(9000.0, 'g', 'tofu'), (50.0, 'g', 'tomato sauce'), (120.0, 'g',
'rice'), (420.0, 'g', 'ice cream')]

>>> remove_from_shopping_list('ice cream', 9000.0, shopping_list)

>>> shopping_list
[(1500.0, 'g', 'peanuts'), (0.5, 'tsp', 'salt'), (2.0, 'tsp', 'oil'),
(9000.0, 'g', 'tofu'), (50.0, 'g', 'tomato sauce'), (120.0, 'g',
'rice')]
```

5.1.12 generate_shopping_list(recipes: list[tuple[str, str]]) -> list[tuple[float, str, str]]

Return a list of ingredients, `(amount, measure, ingredient_name)`, given a list of recipes.

Example:

```
>>> shopping_list = generate_shopping_list([PEANUT_BUTTER,
        MUNG_BEAN_OMELETTE])

>>> shopping_list
[(300.0, 'g', 'peanuts'), (1.0, 'tsp', 'salt'), (3.0, 'tsp', 'oil'),
(1.0, 'cup', 'mung bean'), (0.75, 'tsp', 'pink salt'), (0.25, 'tsp',
'garlic powder'), (0.25, 'tsp', 'onion powder'), (0.125, 'tsp',
'pepper'), (0.25, 'tsp', 'turmeric'), (1.0, 'cup', 'soy milk')]

>>> shopping_list = generate_shopping_list([PEANUT_BUTTER, PEANUT_BUTTER,
        MUNG_BEAN_OMELETTE])

>>> shopping_list
[(600.0, 'g', 'peanuts'), (1.5, 'tsp', 'salt'), (5.0, 'tsp', 'oil'),
(1.0, 'cup', 'mung bean'), (0.75, 'tsp', 'pink salt'), (0.25, 'tsp',
'garlic powder'), (0.25, 'tsp', 'onion powder'), (0.125, 'tsp',
'pepper'), (0.25, 'tsp', 'turmeric'), (1.0, 'cup', 'soy milk')]
```

5.1.13 `display_ingredients(shopping_list: list[tuple[float, str, str]]) -> None`

Print the given shopping list in any order you wish. CSSE7030 students must display the shopping list alphabetically based on the name of the ingredients. See example in Appendix.

Note: The amount of spaces changes depending on how long the longest text is. The order does not matter.

Example:

```
>>> display_ingredients([(1.0, 'large', 'banana'), (0.5, 'cup', 'ice'),])
| 1.0 | large  | banana |
| 0.5 |  cup   | ice    |

>>> display_ingredients([(1.0, 'large', 'banana'),
        (2.0, 'tbsp', 'peanut butter'),
        (2.0, 'pitted', 'dates'),
        (1.0, 'tbsp', 'cacao powder'),
        (240.0, 'ml', 'almond milk'),
        (0.5, 'cup', 'ice'),
        (1.0, 'tbsp', 'cocao nibs'),
        (1.0, 'tbsp', 'flax seed')])
|   1.0 |  large  | banana         |
|   2.0 |  tbsp   | peanut butter  |
|   2.0 | pitted  | dates          |
|   1.0 |  tbsp   | cacao powder   |
| 240.0 |   ml    | almond milk    |
|   0.5 |   cup   | ice            |
|   1.0 |  tbsp   | cocao nibs     |
|   1.0 |  tbsp   | flax seed      |
```

Here is the output again with visile spaces.

```
|␣␣␣1.0␣|␣␣large␣␣|␣banana␣␣␣␣␣␣␣␣␣|
|␣␣␣2.0␣|␣␣tbsp␣␣␣|␣peanut␣butter␣␣|
|␣␣␣2.0␣|␣pitted␣␣|␣dates␣␣␣␣␣␣␣␣␣␣|
|␣␣␣1.0␣|␣␣tbsp␣␣␣|␣cacao␣powder␣␣␣|
|␣240.0␣|␣␣␣ml␣␣␣␣|␣almond␣milk␣␣␣␣|
|␣␣␣0.5␣|␣␣␣cup␣␣␣|␣ice␣␣␣␣␣␣␣␣␣␣␣␣|
|␣␣␣1.0␣|␣␣tbsp␣␣␣|␣cocao␣nibs␣␣␣␣␣|
|␣␣␣1.0␣|␣␣tbsp␣␣␣|␣flax␣seed␣␣␣␣␣␣|
```

5.1.14 `sanitise_command(command: str) -> str`

Return a standardised command to all lower-case and no leading or trailing white spaces removing any numbers from the string.

Example:

```
>>> sanitise_command('add chocolate brownies')
'add chocolate brownies'
>>> sanitise_command('add c4hocolate Brownies')
'add chocolate brownies'
>>> sanitise_command('add chocolate Brownies   5')
'add chocolate brownies'
>>> sanitise_command('add chocolate Brownies        ')
'add chocolate brownies'
```

5.1.15 `main() -> None`

The main function should be called when the file is run, and coordinates the overall interaction. The `main` function should utilize other functions you have written. You should consider writing extra helper functions. In the provided `a1.py`, the function definition for `main` has already been provided, and the `if __name__ == __main__:` block will ensure that the code in the `main` function is run when your `a1.py` file is run. Do not call your `main` function outside of this block, and do not call any other function outside this block unless you are calling them from within the body of another function. The output from your `main` function (including prompts) must exactly match the expected output. Running the sample tests will give you a good idea of whether your prompts and other outputs are correct. See Section 6 for example usage.

Section 4 describes how the program runs. The basic steps that must be implemented by this function are as follows:

1. Prompt user for command.
2. Displays certain information depending on what command is entered.
3. Until the quit command is entered.

6 Example Interaction

The following section provides extended instances of user interaction to demonstrate completed program behaviour.
Example 1

```
Please enter a command: h
    H or h: Help
    mkrec: creates a recipe, add to cook book.
    add {recipe}: adds a recipe to the collection.
    rm {recipe}: removes a recipe from the collection.
    rm -i {ingredient_name} {amount}: removes ingredient from shopping list.
    ls: list all recipes in shopping cart.
    ls -a: list all available recipes in cook book.
    ls -s: display shopping list.
    g or G: generates a shopping list.
    Q or q: Quit.
Please enter a command: H
    H or h: Help
    mkrec: creates a recipe, add to cook book.
    add {recipe}: adds a recipe to the collection.
    rm {recipe}: removes a recipe from the collection.
    rm -i {ingredient_name} {amount}: removes ingredient from shopping list.
    ls: list all recipes in shopping cart.
    ls -a: list all available recipes in cook book.
    ls -s: display shopping list.
    g or G: generates a shopping list.
    Q or q: Quit.
Please enter a command: ls
No recipe in meal plan yet.
Please enter a command: ls -a
chocolate peanut butter banana shake
chocolate brownies
seitan
cinnamon rolls
peanut butter
omelette
Please enter a command: g
Please enter a command: add peanut butter
Please enter a command: ls
[('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil')]
Please enter a command: g
| 300.0 |  g   | peanuts  |
|   0.5 | tsp  | salt     |
|   2.0 | tsp  | oil      |
Please enter a command: add cinnamon rolls
Please enter a command: add peanut butter
Please enter a command: ls
[('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil'), ('cinnamon rolls', '480
ml almond milk,115 g Nuttelex,50 g sugar,7 g active dry yeast,5.5 cup flour,1 tsp
salt,170 g Nuttelex,165 g brown sugar,2 tbsp cinnamon,160 g powdered sugar,30 ml
almond milk,0.5 tsp vanilla extract'), ('peanut butter', '300 g peanuts,0.5 tsp
salt,2 tsp oil')]
Please enter a command: g
| 600.0 |   g   | peanuts           |
|   2.0 |  tsp  | salt              |
|   4.0 |  tsp  | oil               |
| 510.0 |  ml   | almond milk       |
| 285.0 |   g   | Nuttelex          |
|  50.0 |   g   | sugar             |
|   7.0 |   g   | active dry yeast  |
|   5.5 |  cup  | flour             |
| 165.0 |   g   | brown sugar       |
|   2.0 | tbsp  | cinnamon          |
| 160.0 |   g   | powdered sugar    |
|   0.5 |  tsp  | vanilla extract   |
Please enter a command: ls -s
| 600.0 |   g   | peanuts           |
|   2.0 |  tsp  | salt              |
|   4.0 |  tsp  | oil               |
| 510.0 |  ml   | almond milk       |
| 285.0 |   g   | Nuttelex          |
|  50.0 |   g   | sugar             |
|   7.0 |   g   | active dry yeast  |
|   5.5 |  cup  | flour             |
| 165.0 |   g   | brown sugar       |
|   2.0 | tbsp  | cinnamon          |
| 160.0 |   g   | powdered sugar    |
|   0.5 |  tsp  | vanilla extract   |
Please enter a command: rm peanut butter
Please enter a command: g
| 510.0 |  ml   | almond milk       |
| 285.0 |   g   | Nuttelex          |
|  50.0 |   g   | sugar             |
|   7.0 |   g   | active dry yeast  |
|   5.5 |  cup  | flour             |
|   1.5 |  tsp  | salt              |
| 165.0 |   g   | brown sugar       |
|   2.0 | tbsp  | cinnamon          |
| 160.0 |   g   | powdered sugar    |
|   0.5 |  tsp  | vanilla extract   |
| 300.0 |   g   | peanuts           |
|   2.0 |  tsp  | oil               |
Please enter a command: rm peanut butter
Please enter a command: g
| 510.0 |  ml   | almond milk       |
| 285.0 |   g   | Nuttelex          |
|  50.0 |   g   | sugar             |
|   7.0 |   g   | active dry yeast  |
|   5.5 |  cup  | flour             |
|   1.0 |  tsp  | salt              |
| 165.0 |   g   | brown sugar       |
|   2.0 | tbsp  | cinnamon          |
| 160.0 |   g   | powdered sugar    |
|   0.5 |  tsp  | vanilla extract   |
Please enter a command: ls
[('cinnamon rolls', '480 ml almond milk,115 g Nuttelex,50 g sugar,7 g active dry
yeast,5.5 cup flour,1 tsp salt,170 g Nuttelex,165 g brown sugar,2 tbsp cinnamon,160
g powdered sugar,30 ml almond milk,0.5 tsp vanilla extract')]
Please enter a command: rm -i sugar 100
Please enter a command: ls -s
| 510.0 |  ml   | almond milk       |
| 285.0 |   g   | Nuttelex          |
|   7.0 |   g   | active dry yeast  |
|   5.5 |  cup  | flour             |
|   1.0 |  tsp  | salt              |
| 165.0 |   g   | brown sugar       |
|   2.0 | tbsp  | cinnamon          |
| 160.0 |   g   | powdered sugar    |
|   0.5 |  tsp  | vanilla extract   |
Please enter a command: q
```

Note that the ordering of the display does not matter.

Example 2

```
Please enter a command: add peanut butter
Please enter a command: g
| 300.0 |  g   | peanuts  |
|   0.5 | tsp  | salt     |
|   2.0 | tsp  | oil      |
Please enter a command: rm -i salt 1
Please enter a command: ls -s
| 300.0 |  g   | peanuts  |
|   2.0 | tsp  | oil      |
Please enter a command: add 123123123PEANUT butter
Please enter a command: ls
[('peanut butter', '300 g peanuts,0.5 tsp salt,2 tsp oil'), ('peanut butter',
'300 g peanuts,0.5 tsp salt,2 tsp oil')]
Please enter a command: g
| 600.0 |  g   | peanuts  |
|   1.0 | tsp  | salt     |
|   4.0 | tsp  | oil      |
Please enter a command: q
```

7 Assessment and Marking Criteria

This assignment assesses course learning objectives:

1. apply program constructs such as variables, selection, iteration and sub-routines,
2. read and analyse code written by others,
3. read and analyse a design and be able to translate the design into a working program, and
4. apply techniques for testing and debugging.

7.1 Functionality

Your program's functionality will be marked out of a total of 6 marks. Your assignment will be put through a series of tests and your functionality mark will be proportional to the number of tests you pass. You will be given a *subset* of the functionality tests before the due date for the assignment.

You may receive partial marks within each section for partially working functions, or for implementing only a few functions.

You need to perform your *own* testing of your program to make sure that it meets *all* specifications given in the assignment. Only relying on the provided tests is likely to result in your program failing in some cases and you losing some functionality marks. Note: Functionality tests are automated, so string outputs need to match *exactly* what is expected.
Your program must run in the Python interpreter (the IDLE environment). Partial solutions will be marked but if there are errors in your code that cause the interpreter to fail to execute your program, you will get zero for functionality marks. If there is a part of your code that causes the interpreter to fail, comment out the code so that the remainder can run. Your program must

run using the Python 3.10 interpreter. If it runs in another environment (e.g. Python 3.8 or PyCharm) but not in the Python 3.10 interpreter, you will get zero for the functionality mark.

7.2 Code Style

The style of your assignment will be assessed by a tutor. Style will be marked according to the style rubric provided with the assignment. The style mark will be out of 4.

The key consideration in marking your code style is whether the code is easy to understand. There are several aspects of code style that contribute to how easy it is to understand code. In this assignment, your code style will be assessed against the following criteria.

- Readability
  - Program Structure: Layout of code makes it easy to read and follow its logic. This includes using whitespace to highlight blocks of logic.
  - Descriptive Identifier Names: Variable, constant, and function names clearly describe what they represent in the program's logic. Do not use Hungarian Notation for identifiers. In short, this means do not include the identifier's type in its name, rather make the name meaningful (e.g. employee identifier).
  - Named Constants: Any non-trivial fixed value (literal constant) in the code is represented by a descriptive named constant (identifier).
- Algorithmic Logic
  - Single Instance of Logic: Blocks of code should not be duplicated in your program. Any code that needs to be used multiple times should be implemented as a function.
  - Variable Scope: Variables should be declared locally in the function in which they are needed. Global variables should not be used.
  - Control Structures: Logic is structured simply and clearly through good use of control structures (e.g. loops and conditional statements).
- Documentation:
  - Comment Clarity: Comments provide meaningful descriptions of the code. They should not repeat what is already obvious by reading the code (e.g. `# Setting variable to 0`). Comments should not be verbose or excessive, as this can make it difficult to follow the code.
  - Informative Docstrings: Every function should have a docstring that summarises its purpose. This includes describing parameters and return values (including type information) so that others can understand how to use the function correctly.
  - Description of Logic: All significant blocks of code should have a comment to explain how the logic works. For a small function, this would usually be the docstring. For long or complex functions, there may be different blocks of code in the function. Each of these should have an in-line comment describing the logic.

7.3 Assignment Submission

You must submit your assignment electronically via Gradescope (`https://gradescope.com/`). You **must** use your UQ email address which is based on your student number (e.g. s4123456@student.uq.edu.au) as your Gradescope submission account.

When you login to Gradescope you may be presented with a list of courses. Select CSSE1001/CSSE7030. You will see a list of assignments. Choose **Assignment 1**. You will be prompted to choose a file to upload. The prompt may say that you can upload any files, including zip files. You **must** submit your assignment as a single Python file called `a1.py` (use this name – all lower case), and *nothing* else. Your submission will be automatically run to determine the functionality mark. If you submit a file with a **different name**, the tests will **fail** and you will get **zero** for functionality. Do **not** submit **any** sort of archive file (e.g. zip, rar, 7z, etc.).

Upload an initial version of your assignment *at least* one week before the due date. Do this even if it is just the initial code provided with the assignment. If you are unable access Gradescope, contact the course helpdesk (csse1001@helpdesk.eait.uq.edu.au) *immediately.* Excuses, such as you were not able to login or were unable to upload a file will not be accepted as reasons for granting an extension.

When you upload your assignment it will run a **subset** of the functionality autograder tests on your submission. It will show you the results of these tests. It is your responsibility to ensure that your uploaded assignment file runs and that it passes the tests you expect it to pass.

Late submissions of the assignment will **not** be marked. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment *well* before the submission deadline of 20:00. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment.

In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details of how to apply for an extension.

Requests for extensions must be made **before** the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted via my.UQ. You must retain the original documentation for a minimum period of six months to provide as verification, should you be requested to do so.

7.4 Plagiarism

This assignment must be your own individual work. By submitting the assignment, you are claiming it is entirely your own work. You **may** discuss general ideas about the solution approach with other students. Describing details of how you implement a function or sharing part of your code with another student is considered to be **collusion** and will be counted as plagiarism. You **may not** copy fragments of code that you find on the Internet to use in your assignment. Use of ChatGPT is strictly prohibited.

Please read the section in the course profile about plagiarism. You are encouraged to complete *both* parts A and B of the academic integrity modules *before* starting this assignment. Submitted assignments will be electronically checked for potential cases of plagiarism.

7.5 Appendix

Example for `display_ingredients()` with visible spaces. The amounts are right justified, measures are center justified, and ingredient names are left justified. You can create this by finding the longest string in a given column, then going from there.

```
|␣240.0␣|␣␣␣ml␣␣␣␣|␣almond␣milk␣␣␣␣|
|␣␣␣1.0␣|␣␣tbsp␣␣␣|␣cocao␣nibs␣␣␣␣␣|
|␣␣␣2.0␣|␣pitted␣␣|␣dates␣␣␣␣␣␣␣␣␣␣|
|␣␣␣1.0␣|␣␣tbsp␣␣␣|␣flax␣seed␣␣␣␣␣␣|
|␣␣␣0.5␣|␣␣␣cup␣␣␣|␣ice␣␣␣␣␣␣␣␣␣␣␣␣|
|␣␣␣2.0␣|␣␣tbsp␣␣␣|␣peanut␣butter␣␣|
```

Example for `display_ingredients()` master level task with visible spaces.

```
>>> display_ingredients([(1.0, 'large', 'banana'),
(2.0, 'tbsp', 'peanut butter'),
(2.0, 'pitted', 'dates'),
(1.0, 'tbsp', 'cacao powder'),
(240.0, 'ml', 'almond milk'),
(0.5, 'cup', 'ice'),
(1.0, 'tbsp', 'cocao nibs'),
(1.0, 'tbsp', 'flax seed')])
| 240.0 |   ml    | almond milk    |
|   1.0 |  large  | banana         |
|   1.0 |  tbsp   | cacao powder   |
|   1.0 |  tbsp   | cocao nibs     |
|   2.0 | pitted  | dates          |
|   1.0 |  tbsp   | flax seed      |
|   0.5 |   cup   | ice            |
|   2.0 |  tbsp   | peanut butter  |
```

7.6 Update log

In the **Terminology** section: The second term should say `recipe`, not `recipe_name`