

Meal Planner

TIA3: TERMINAL APP

By Kat Mountford

Key Features

1. Add a new recipe
2. View all recipes
3. Choose how many days to meal plan for
4. Generate a meal plan
5. Optionally regenerate a meal plan
6. Save the meal plan

```
Hey, welcome to Meal Planner! Please choose from one of the actions below to get started
```

```
What would you like to do?
```

```
a = Add a new recipe
```

```
v = View all recipes
```

```
n = Start a new meal plan
```

```
q = quit
```

Adding recipes

There are two ways to add a new recipe to the recipes list (a CSV file called recipes.csv)

1. Add a new recipe via the main **user menu** in the application:

```
Hey, welcome to Meal Planner! Please choose from one of the actions below to get started

What would you like to do?
a = Add a new recipe
v = View all recipes
n = Start a new meal plan
q = quit
a
What is the name of the recipe?: Banana Split
What are the ingredients of the recipe? (For 1 serving): 2 Banana's, Cream, Sprinkles
How many calories in this recipe? (For 1 serving): 675
Your new recipe 'Banana Split' has successfully been added!
```

```
recipes.csv M x
Users > kat > Documents > ca-projects > python > meal-planner > recipes.csv
275 Chai Latte,"Chai Tea, Soy Milk",210
276 Protein Smoothie,"Proten powder, milk, yoghurt, water",315
277 Banana Smoothie,"Banana, Milk, Ice",220
278 Banana Split,"2 Banana's, Cream, Sprinkles",675
279
```

```
# Function to add recipe to CSV file
def add_recipe_to_CSV(new_recipe):
    # Append dictionary to the existing CSV file (all recipes)
    with open(r'recipes.csv', 'a', newline='') as f:
        fieldnames = ['title', 'ingredients', 'calories']
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writerow(new_recipe)

    # Confirmation for the user - recipe successfully added
    print(f"Your new recipe '{new_recipe['title']}' has successfully been added!")
```

Appends the new entry to recipes.csv using csv.DictWriter

Adding recipes cont.

2. Add a new recipe via the **commandline** (through using argparse)

```
[kat@Kats-MacBook-Air meal-planner % python3 main.py --title "Ravioli Carbonara" --ingredients "Ravioli, Carbonara Sauce, Bacon" --calories 812  
Your new recipe 'Ravioli Carbonara' has successfully been added!
```

recipes.csv M X

Users > kat > Documents > ca-projects > python > meal-planner > recipes.csv

```
277 Banana Smoothie,"Banana, Milk, Ice",220  
278 Banana Split,"2 Banana's, Cream, Sprinkles",675  
279 Ravioli Carbonara,"Ravioli, Carbonara Sauce, Bacon",812
```

Successfully **appends** to recipes.csv in the same way as previous process

Checks if any values have been passed from the cli before running the function to add the recipe – so it doesn't hinder the standard running of the application

```
# Argparse to add recipe direct from the commandline  
parser = argparse.ArgumentParser(description="Meal Planner: Meal planning made simple")  
  
parser.add_argument("--title", help="Name of the recipe")  
parser.add_argument("--ingredients", help="Ingredients for 1 serving")  
parser.add_argument("--calories", type=int, help="Calories for 1 serving")  
  
args = parser.parse_args()  
  
# Check if a recipe has been added from the command line  
if args.title or args.ingredients or args.calories:  
    recipes.add_recipe_from_cli(args) # Call the add recipe via cli function and pass the args
```

View all recipes & navigation

Aside from accessing the CSV externally, the user can view all recipes **printed to their screen** within the application by pressing 'v' on the user menu:

```
Ham and Cheese Omelette: 2 Eggs, 50g Ham, 30g Cheddar Cheese. Calories: 740
Tuna Cucumber Salad: 100g Tuna, 1/2 Cucumber. Calories: 630
Turkey Cheese Melt: 2 Slices Whole Wheat Bread, 50g Sliced Turkey, 30g Swiss Cheese. Calories: 790
Chai Latte: Chai Tea, Soy Milk. Calories: 210
Protein Smoothie: Proten powder, milk, yoghurt, water. Calories: 315
Banana Smoothie: Banana, Milk, Ice. Calories: 220
Banana Split: 2 Banana's, Cream, Sprinkles. Calories: 675
Ravioli Carbonara: Ravioli, Carbonara Sauce, Bacon. Calories: 812
```

```
What would you like to do?
a = Add a new recipe
v = View all recipes
n = Start a new meal plan
q = quit
```

After any completed action, the user is returned to the **user menu** so they can easily jump to the next task or quit the application

Choose how many days

When starting a new meal plan (User menu option = 'n'), the user can choose **how many days** they want to get planned meals for:

```
What would you like to do?
a = Add a new recipe
v = View all recipes
n = Start a new meal plan
q = quit
n
How many days would you like to get a meal plan for?:
```

This is important for setting up a meal plan in the next step – as we loop through the meal plan process for the **range** of the number of days the user provides

```
# Continuously loop until the user is happy with their meal plan
while True:
    all_meals = [] # Empty list to add meals to

    for i in range(num_days):
        day = Day(calorie_target) # Create a Day object and set the calorie range and daily calories
        day.set_meals() # Set the meals for the day
        all_meals.append(day) # Append set meals to all_meals list
```

Generating a meal plan

To generate a meal plan, a user next provides their daily **calorie target**.

At this stage a number of things happen; the first of which is that an **instance of the class Day** is created, and upon **initialisation**, the calorie target is stored and a function is called to automatically set the calories for each meal, based on the calorie target we just received.

```
How many days would you like to get a meal plan for?: 2
What is your daily calorie target? Please enter a target between 1400 - 3000 Calories: 1680
```

```
for i in range(num_days):
    day = Day(calorie_target) # Create a Day object and set the calorie range and daily calories
    day.set_meals() # Set the meals for the day
    all_meals.append(day) # Append set meals to all_meals list
```

```
class Day():
    def __init__(self, calorie_target):
        self.calorie_target = calorie_target
        self.meal_calories = {} # Store meal calories based on the users calorie target
        # Set the meal calories (based on the calorie range) each time a day is initialized
        self.set_meal_calories()
        self.todays_meals = [] # Empty list for todays meals
        self.semantic_names = ["Breakfast", "Lunch", "Dinner", "Snack 1", "Snack 2"]
        self.result = None
```


Meal calorie ranges

Based on the calorie target the user enters, this will determine what their **meal calories** are for each meal in a day.

This is both to provide the program the best chance at arriving at a meal plan within the daily calorie range for the user, and to ensure meal sizes make general sense (e.g. generally people's snacks are smaller and lower in calories than their dinners).

1. **Conditional** to check the range of the calorie target the user has entered

2. Once a match has been found, **set the calories for each meal** as a dictionary.

```
# Based on each calorie range, set the dictionary to equal the below specified calories
if 1400 <= self.calorie_target <= 1700:
    self.meal_calories = {
        "M1": 350,
        "M2": 350,
        "M3": 500,
        "M4": 150,
        "M5": 200,
    }
elif 1701 <= self.calorie_target <= 2000:
    self.meal_calories = {
        "M1": 400,
        "M2": 400,
        "M3": 600,
        "M4": 300,
        "M5": 200,
    }
```

M1 = Meal 1 (breakfast)
M2 = Meal 2 (Lunch)
M3 = Meal 3 (Dinner)
M4 = Meal 4 (Snack 1)
M5 = Meal 5 (Snack 2)

Meal calorie ranges cont.

Setting the meals in the meal plan is broadly done by looping through each of the meal calories that have been set, and then **finding a meal that matches** those calories from the recipe list and save those 'matching' meals to a new list titled `today's_meals`.

However, it's very unlikely that there will be recipe in our recipe list with the exact calories that have been set for each meal. So to account for this, I've allowed a range of ± 75 calories each side of the meal calorie that's been set – so we can get pretty close to the specified range, but allow some variation.

```
for calories in self.meal_calories.values():
    # For the value in that dict (cals),  $\pm 75$  either side
    min_cal = calories - 75
    max_cal = calories + 75
    meal_found = False

    # Iterate over the recipes to find a meal, and check it's in the calorie range.
    # While it's not, keep searching, if it is, add it to today's meals and set meal_found to True
    while meal_found == False:
        for recipe in recipes:
            # Find the 'calories' value
            value = int(recipe.get('calories'))
            if min_cal < value < max_cal: # Check value within acceptable range
```

Searching for recipes

I also realised it was inefficient to continuously open and close the csv in order to search for a meal through each loop, so alternatively at the beginning of `set_meals()` a **copy of all the recipes** in the CSV is made and **stored in a new list**. This list is also **shuffled** to ensure that users are served a variety of meals each time rather than potentially getting the same ones over and over for a given calorie range.

```
# Function to find and select meals for the specified calorie_ranges
def set_meals(self):
    attempts = 0 # Num of attempts to set a valid meal plan
    meal_completed = False
    recipes = self.shuffle_recipes()
```

```
def shuffle_recipes(self):
    recipe_list = get_recipes() # Get all recipes (as a dictionary)
    # Shuffle recipes so it's a random one being returned
    random.shuffle(recipe_list)
    return recipe_list
```

```
for recipe in recipes:
    # Find the 'calories' value
    value = int(recipe.get('calories'))
    if min_cal < value < max_cal: # Check value within acceptable range
        # Append to todays meals
        self.todays_meals.append(recipe)
        # Remove from the recipe list (so as to not duplicate on the same day)
        recipes.remove(recipe)
        meal_found = True
        break # Exit the loop
```

Recipes are also removed from the list once they've been 'found' and added to `todays_meals`, to increase variation and prevent duplicate meals across multiple days.

The set_meals() function

With a shuffled recipe list and meal_calories set, for each of the meals, we can loop through the list of recipes and **search** for recipes where the **calories for that recipe** are **within the meal_calories range**

for Loop to iterate through each meal

If calories in the recipe are within the range of the meal_calories – append the meal to today's meals and move onto the next meal by setting meal_found = True

```
for calories in self.meal_calories.values():
    # For the value in that dict (cals), +/- 75 either side
    min_cal = calories - 75
    max_cal = calories + 75
    meal_found = False

    # Iterate over the recipes to find a meal, and check it's in the calorie range.
    # While it's not, keep searching, if it is, add it to today's meals and set meal_found to True
    while meal_found == False:
        for recipe in recipes:
            # Find the 'calories' value
            value = int(recipe.get('calories'))
            if min_cal < value < max_cal: # Check value within acceptable range
                # Append to todays meals
                self.todays_meals.append(recipe)
                # Remove from the recipe list (so as to not duplicate on the same day)
                recipes.remove(recipe)
                meal_found = True
                break # Exit the loop

        if meal_found == False:
            meal_found = True
            raise FindRecipeError("\nSorry, we were unable to find a suitable meal for you with the current
            calorie target and available recipes. Please consider one of the following options:\n1. Adjust
            your daily calorie target to a different target.\n2. Add more recipes to your collection.")
```

Nested while Loop to continue

searching for a match until a meal is found

Nested for loop in order to iterate through each recipe in the list to find a matching meal

If the program has cycled through all recipes in the list and not found a match, raise an exception. Still set meal_found = True in order to break the loop and continue onto the next meal (where a match can still be made)

The set_meals() function cont.

Once we have today's meals completed (1 for each meal – 5 meals total), then another check occurs to sum the calorie values across all meals; and compare if the calorie total is close to the calorie target the user set initially. If it's **within ± 75 of the users calorie target, then it's accepted.**

```
# Check that meal total is acceptable close to the users target calories
total_calories = 0
# Sum each meals calories together to find the total
for meal in self.todays_meals:
    total_calories += int(meal['calories'])

# Set the range to be acceptable based on the daily calories goal
daily_min_cal = self.calorie_target - 75
daily_max_cal = self.calorie_target + 75

# If within target, mark meal as completed
if daily_min_cal < total_calories < daily_max_cal:
    meal_completed = True

# Otherwise, start again
else:
    self.todays_meals = []
    self.shuffle_recipes()
    attempts += 1
```

If it's not within the acceptable range, then the process is **repeated** with new meals until the summed calories **are** within an acceptable range.

Each time resetting todays_meals and getting a refreshed list of shuffled recipes.

The set_meals() function cont.

The number of attempts at getting a meal plan within an acceptable range is tracked and incremented each time the program has to re-try

```
# Function to find and select meals for the specified calorie_ranges
def set_meals(self):
    attempts = 0 # Num of attempts to set a valid meal plan
    meal_completed = False
    recipes = self.shuffle_recipes()

    try:
        # While todays meals is incomplete
        while meal_completed == False and attempts < 1000:
            try:
                for calories in self.meal_calories.values():
                    # For the value in that dict (cals), +- 75 either side
                    min_cal = calories - 75
                    max_cal = calories + 75
                    meal_found = False
```

Variable to store the number of attempts made

Increment attempts upon re-try

Other code between here

```
        # Otherwise, start again
    else:
        self.todays_meals = []
        self.shuffle_recipes()
        attempts += 1

    except FindRecipeError as e:
        print(e)

    if attempts >= 1000:
        raise FindRecipeError("\nSorry, we were unable to find a suitable meal plan for you with the current calorie target and available recipes. Please consider one of the following options:\n1. Adjust your daily calorie target to a different target.\n2. Add more recipes to your collection.\n3. Try generating a meal plan for a different number of days.")

    except FindRecipeError as e:
        print(e)
```

The program is allowed a maximum of 1000 retries; if it exceeds this then it's assumed that there are no valid combinations to meet the calorie_target and an exception is raised to prevent an infinite loop

Displaying the meal plan

Assuming a valid combination of meals is found that is within the acceptable range to the users calorie target, then the meals are displayed to the user

Meal information, broken down into each meal; with semantic meal names for easy readability

Total calories for the day

Day number

How many days would you like to get a meal plan for?: 2
What is your daily calorie target? Please enter a target between 1400 - 3000 Calories: 1680

Here are your daily meal plans:

Day 1 Meal Plan:

Breakfast: Chicken and Dumplings (390 calories)
Ingredients: 200g Chicken,Dumplings,Carrots,Peas

Lunch: Cherry Vanilla Parfait (330 calories)
Ingredients: 1 Cup Greek Yogurt,50g Dried Cherries,1tbs Vanilla Extract

Dinner: Sausage and Egg Breakfast Wrap with Cheese (570 calories)
Ingredients: 2 Whole Wheat Tortillas,2 Scrambled Eggs,50g Sausage,30g Cheese

Snack 1: Strawberry Spinach Salad (160 calories)
Ingredients: 150g Baby Spinach,100g Strawberries,2tbs Balsamic Vinaigrette

Snack 2: Veggie Stir-Fry (245 calories)
Ingredients: 200g Tofu,150g Broccoli,100g Bell Peppers,2tbs Soy Sauce

The total calories for the day is: 1695

Day 2 Meal Plan:

Breakfast: Peanut Butter Energy Balls (390 calories)
Ingredients: 2 Energy Balls,3tbs Peanut Butter,1 Cup Rolled Oats

Lunch: Tuna Poke Bowl (350 calories)
Ingredients: 150g Ahi Tuna,Sushi Rice,Avocado,Soy Sauce

Dinner: Garlic Butter Shrimp Scampi (480 calories)
Ingredients: 150g Shrimp,150g Linguine Pasta,Garlic,Butter

Snack 1: Baked Cinnamon Apple (220 calories)
Ingredients: 1 Apple,1tbs Cinnamon,1tbs Brown Sugar

Snack 2: Yogurt and Cucumber Slices (210 calories)
Ingredients: 1 Cup Yogurt,100g Cucumber Slices

The total calories for the day is: 1650

What do you think of these meals?
Enter 's' to save them or 'n' to generate a new meal plan:

Optional regeneration

The user is then asked if they're happy with the meals and can choose to either save them or generate a new meal plan. If opting to regenerate, the user can additionally optionally reset their daily calorie target

```
What do you think of these meals?
Enter 's' to save them or 'n' to generate a new meal plan: n
Do you want to change your daily calorie target of 1680 calories? Enter 'y' to change or 'n' to keep your current target: y
Regenerating meal plan...
What is your daily calorie target? Please enter a target between 1400 - 3000 Calories:
```

```
# Check if user is happy with their meals
day.result = input("What do you think of these meals?\nEnter 's' to save them or 'n' to generate a new meal plan: ").
lower()

if day.result == 's':
    if day.save_meal_plan(all_meals): # Save the meals to a file
        break

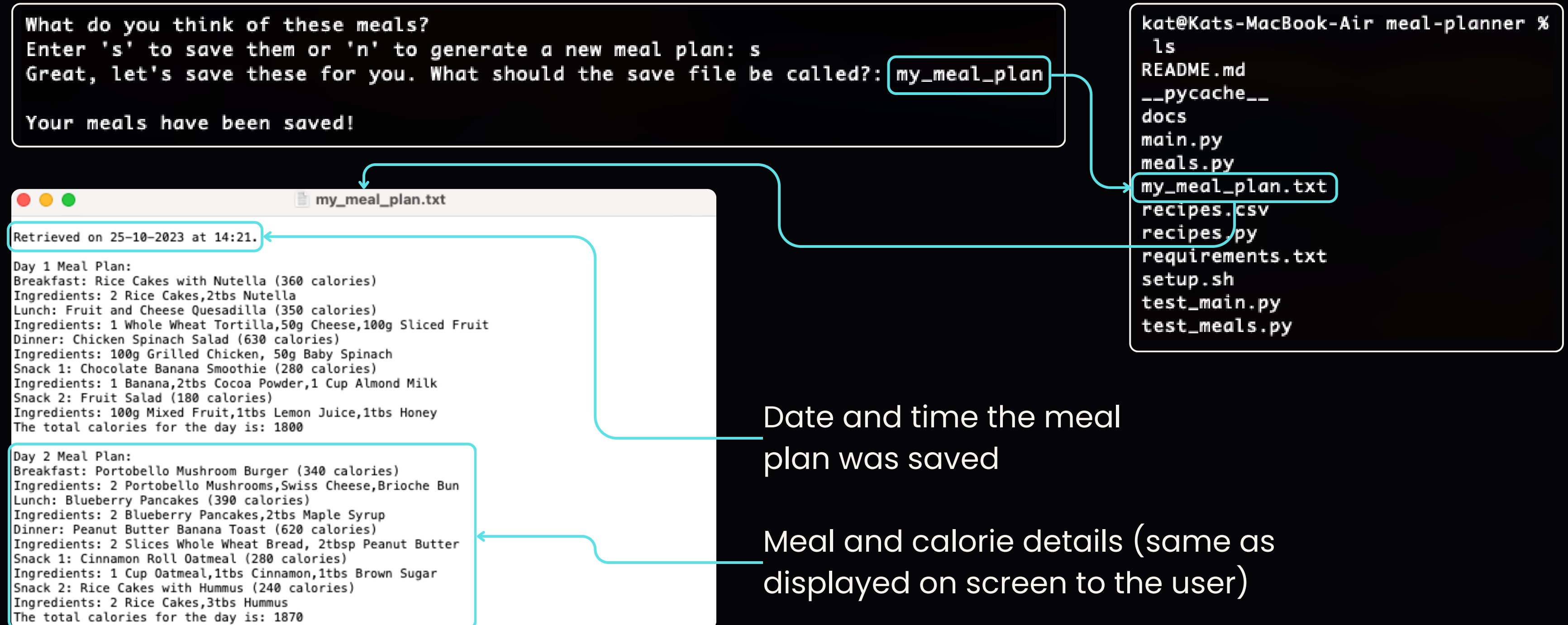
elif day.result == 'n':
    calorie_change_input = input(f"Do you want to change your daily calorie target of {day.calorie_target} calories?
Enter 'y' to change or 'n' to keep your current target: ")
    if day.check_calorie_change(calorie_change_input): # Check if user wants to change calorie target is True
        calorie_target_input = input(f"Regenerating meal plan...\nWhat is your daily calorie target? Please enter a
target between {CALORIE_MIN} - {CALORIE_MAX} Calories: ")
        calorie_target = get_calorie_target(calorie_target_input)

else:
    raise InvalidInputError("Invalid input. Please enter 's' to save or 'n' to generate a new meal plan")
```

Nested if statement, so only if the user wants to regenerate their plan AND chooses to reset their target, will they be prompted to set a new calorie_target, otherwise the meal plan will be regenerated using the original calorie goal

Saving the meal plan

Assuming the user is happy with their meal plan, they can then choose to save it. This will save the meal plan to a text file in the source directory for them, with a name of their choosing.



Thoughts & Challenges

Favourite Parts

- Learning how to write scripts! Never done this before so was fun to learn
- Researching and trying python packages I'd not yet heard of; there's so much you can do with them!

Challenges

- Initially not having written easily testable code and so having to refactor in order to improve testability
- Initially a very steep learning curve wrapping my head around complex logic and things like loops within loops within loops