FOOD DIARY APP

T1A3 Terminal App
Karla Bucoy
2022 OCT STND

AGENDA

INTRODUCTION

The purpose of the Food Diary app is to allow users to track their daily meals.
The app is targeted for users who want to form a daily habit around recording their meals, and ensuring they stay on track with their food and nutrition goals.
Users can input today's and yesterday's meals in the app and view their food diary in table format on the screen. Users also have the opportunity to edit today's and yesterday's meals in case they have made a mistake or omitted information.

Tech Stack:

- Python source code, using the PEP8 styling convention written in Visual Studio Code.
- Python libraries:
    - Pandas for creating dataframes and reading and writing to the .csv diary file.
    - Datetime for calling and returning today's and yesterday's date.
    - Prettytable for listing additional user options and to display the diary in table format in the terminal window.
    - Termcolor for coloured text to highlight important messages (green - confirmations, blue - instructions, red - errors).
- Markdown ReadMe file written in Visual Studio Code.
- GitHub for managing source code changes and tracking and controlling versions of the source code.
- Bash .sh script to execute the program.

APP WALKTHROUGH: first time user

Enter today's meals

- User receives welcome message and instructions.
- There is no historical data for first time users.
- User is prompted to input today's meals in order of breakfast, lunch, dinner and snack.
- Users cannot enter empty meal values. If the user enters an empty value, they are prompted to enter that specific meal until the condition is met.
- Meal input is saved as a dictionary and displayed on the screen, in addition to a confirmation message that today's meals have been saved.

```
Welcome to your food diary where you can easily track your meals each day.
Enter today's meals - breakfast, lunch, dinner and snack - when prompted.
Enter breakfast: oats
Enter lunch: sandwich
Enter dinner: soup
Enter snack: fruit
{'date': ['17-12-22'], 'breakfast': ['oats'], 'lunch': ['sandwich'], 'dinner': ['soup'], 'snack': ['fruit']}
Your meals have been saved.
```

APP WALKTHROUGH: first time user

Enter yesterday's meals

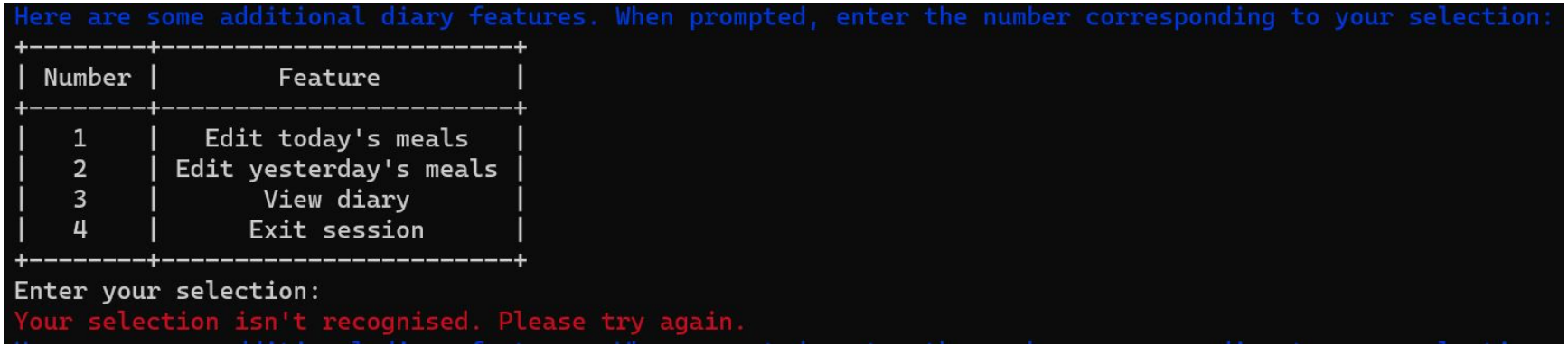
- Because there is no historical data for first time users, the user is prompted to input yesterday's meals in order of breakfast, lunch, dinner, snack.
- Users cannot enter empty meal values. If the user enters an empty value, they are prompted to enter that specific meal until the condition is met.
- Meal input is saved as a dictionary and displayed on the screen, in addition to a confirmation message that today's meals have been saved.

```
You didn't input yesterday's meals. When prompted, enter yesterday's meals - breakfast, lunch, dinner, snack.
Enter yesterday's breakfast: eggs
Enter yesterday's lunch: steak and fries
Enter yesterday's dinner: sushi
Enter yesterday's snack: smoothie
{'date': ['16-12-22'], 'breakfast': ['eggs'], 'lunch': ['steak and fries'], 'dinner': ['sushi'], 'snack': ['smoothie']}
Your meals have been saved.
```

APP WALKTHROUGH: first time user

Additional features

- Once today and yesterday's meals are inputted and saved, the user is presented with additional features: edit today's meals, edit yesterday's meals, view diary, exit session.
- Users must enter the number corresponding to their selection (per features table). If the user enters a blank value, or a value that is not one of the numbers listed in the features table, they will be prompted to enter a selection until a correct selection is entered.



APP WALKTHROUGH: first time user

Edit today's meals

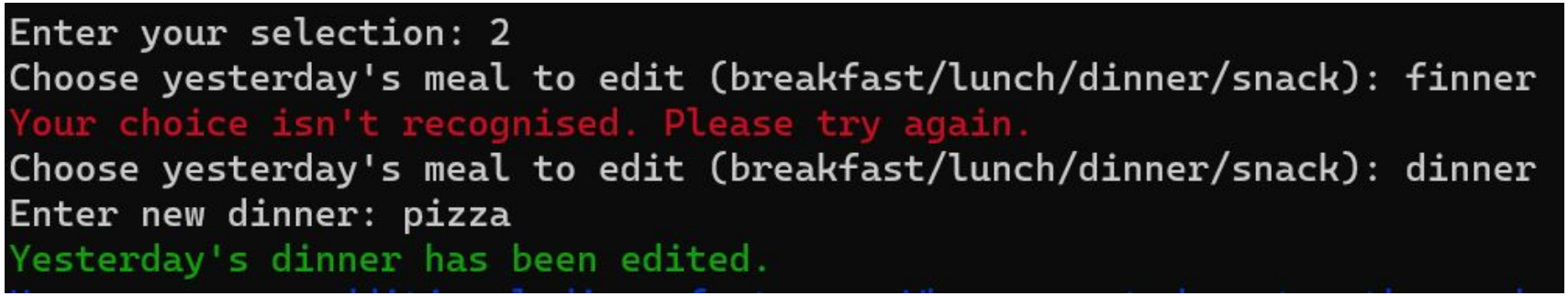
- If editing today's meals, users are prompted to enter the meal of their choice to edit.
- Users must enter the name of the meal as it appears in the prompt. If the user enters a blank value, or a value that doesn't match the meal name, they will be prompted to enter a selection until a correct selection is entered.
- Once today's meals is edited the user will receive a confirmation message, and then they will return to the additional features menu for further editing or viewing of the diary or exiting the program.

```
Enter your selection: 1
Choose today's meal to edit (breakfast/lunch/dinner/snack):
Your choice isn't recognised. Please try again.
Choose today's meal to edit (breakfast/lunch/dinner/snack):
Your choice isn't recognised. Please try again.
Choose today's meal to edit (breakfast/lunch/dinner/snack): breakfast
Enter new breakfast: smoothie
Today's breakfast has been edited.
```

APP WALKTHROUGH: first time user

Edit yesterday's meals

- If editing yesterday's meals, users are prompted to enter the meal of their choice to edit.
- Users must enter the name of the meal as it appears in the prompt. If the user enters a blank value, or a value that doesn't match the meal name, they will be prompted to enter a selection until a correct selection is entered.
- Once yesterday's meal is edited the user will receive a confirmation message, and then they will return to the additional features menu for further editing or viewing of the diary or exiting the program.



APP WALKTHROUGH: first time user

View diary

- Upon selecting 'View diary' the user's food diary is displayed in table format on the screen.
- Underneath the diary table, the additional features menu and prompt is displayed allowing the user to select options for further editing or viewing of the diary or exiting the program.

```
Enter your selection: 3
Here's your food diary:
+----------+-----------+-----------------+---------+-----------+
|   date   | breakfast |      lunch      | dinner  |   snack   |
+----------+-----------+-----------------+---------+-----------+
| 17-12-22 |  smoothie |     sandwich    |  soup   |   fruit   |
| 16-12-22 |    eggs   | steak and fries | pizza   | chocolate |
+----------+-----------+-----------------+---------+-----------+
```

APP WALKTHROUGH: returning user

Additional features

- Returning users who have entered today's and yesterday's meals receive a confirmation message and are presented with the additional features menu.
- Users are then able to edit today's meals, edit yesterday's meals, view their diary or exit the program.
- The additional features menu experience is the same as the first time user additional features menu experience and utilises the same code base.

```
You've already entered your meals for the day.
Here are some additional diary features. When prompted, enter the number corresponding to your selection:
+--------+-----------------------+
| Number |        Feature        |
+--------+-----------------------+
|   1    |   Edit today's meals  |
|   2    | Edit yesterday's meals|
|   3    |       View diary      |
|   4    |      Exit session     |
+--------+-----------------------+
Enter your selection: 3
```

LOGIC & CODE

Habit formation

The app is designed to help users **form a daily habit** around recording meals.

Users can:

- Input and edit today's meals.
- Input and edit yesterday's meals.
- View their food diary.

Because users are unlikely to remember what they ate a week ago, let alone several days ago, inputs and edits are limited to two days: today and yesterday. It was unnecessary to include a function for entering and editing historic data past yesterday's date.

LOGIC & CODE

Date checks

Functions to call and return today's and yesterday's date were written using the datetime module to:

- Check whether today's and yesterday's meals have been inputted. If meals have been inputted, users receive a confirmation message. If meals haven't been inputted, users will be prompted to input their meals. For a first time user without any historical meal data, the app will create a diary.csv file and write meal input to it.
- Edit today's and yesterday's meals by locating the specific date in the diary.csv file and overwriting existing meal data in that date's row.

```
def today_date():
    from datetime import date
    date_today = date.today()
    return date_today.strftime('%d-%m-%y')
```

```
def yesterday_date():
    from datetime import date
    from datetime import timedelta
    date_today = date.today()
    date_yesterday = date_today - timedelta(days = 1)
    return date_yesterday.strftime('%d-%m-%y')
```

```
def today_check():
    import pandas as pd
    from termcolor import colored
    try:
        df = pd.read_csv('diary.csv')
        today_date() in df['date'].values is True
        print(colored("You've already entered your meals for the day.", 'green'))
    except FileNotFoundError:
        meals_dict = meal_input()
        add_input(meals_dict)
    yesterday_check()
    yesterday_date()
```

```
def yesterday_check():
    import pandas as pd
    df = pd.read_csv('diary.csv')
    if yesterday_date() in df['date'].values:
        additional_options()
        additional_selections()
    else:
        print(colored("You didn't input yesterday's meals. When prompted, enter yesterday's meals - breakfast, lunch, dinner, snack.", 'red'))
        meals_dict = yesterday_meal_input()
        add_input(meals_dict)
        additional_options()
        additional_selections()
```

LOGIC & CODE

Inputting meals

Functions to input and return today's and yesterday's meals were written to:

- Prompt users to input their meals in order of: breakfast, lunch, dinner, snack.
- Users cannot input empty meal values. If the user enters an empty value, a while loop is activated prompting the user to enter that specific meal until the condition is met.
- Meal inputs and the corresponding date are then saved in a dictionary and printed on the screen.

```
def meal_input():
    from termcolor import colored
    print(colored("Enter today's meals - breakfast, lunch, dinner and snack - when prompted.", 'blue'))
    breakast_input = ''
    while True:
        breakfast_input = input("Enter breakfast: ")
        if breakfast_input:
            break
    lunch_input = ''
    while True:
        lunch_input = input("Enter lunch: ")
        if lunch_input:
            break
    dinner_input = ''
    while True:
        dinner_input = input("Enter dinner: ")
        if dinner_input:
            break
    snack_input = ''
    while True:
        snack_input = input("Enter snack: ")
        if snack_input:
            break
    date_input = today_date
        # need error handling for input that is not a string or word, only numbers.
    breakfast_list = [breakfast_input]
    lunch_list = [lunch_input]
    dinner_list = [dinner_input]
    snack_list = [snack_input]
    date_list = [date_input()]
    meals_dict = {'date': date_list, 'breakfast': breakfast_list, 'lunch': lunch_list, 'dinner': dinner_list, 'snack': snack_list}
    print(meals_dict)
    return meals_dict
```

```
def yesterday_meal_input():
    breakast_input = ''
    while True:
        breakfast_input = input("Enter yesterday's breakfast: ")
        if breakfast_input:
            break
    lunch_input = ''
    while True:
        lunch_input = input("Enter yesterday's lunch: ")
        if lunch_input:
            break
    dinner_input = ''
    while True:
        dinner_input = input("Enter yesterday's dinner: ")
        if dinner_input:
            break
    snack_input = ''
    while True:
        snack_input = input("Enter yesterday's snack: ")
        if snack_input:
            break
    yesterday_input = yesterday_date
    breakfast_list = [breakfast_input]
    lunch_list = [lunch_input]
    dinner_list = [dinner_input]
    snack_list = [snack_input]
    yesterday_date_list = [yesterday_input()]
    meals_dict = {'date': yesterday_date_list, 'breakfast': breakfast_list, 'lunch': lunch_list, 'dinner': dinner_list, 'snack': snack_list}
    print(meals_dict)
    return meals_dict
```

LOGIC & CODE

Editing meals

Functions to edit today's and yesterday's meals and return the edits were written to:

- Prompt users to edit yesterday's and today's meal inputs.
- Users must enter the name of the meal as it appears in the prompt. If the user enters a blank value, or a value that doesn't match the meal name, a while loop is activated prompting the user to enter that specific meal until the condition is met.
- Edited meal inputs and the corresponding date are then saved in a dictionary and printed on the screen.

```
def edit_diary():
    import pandas as pd
    from termcolor import colored
    edit_today = input("Choose today's meal to edit (breakfast/lunch/dinner/snack): ")
    if edit_today == "breakfast":
        edit_breakfast = ''
        while True:
            if edit_breakfast := input("Enter new breakfast: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[today_date(), 'breakfast'] = edit_breakfast
                df.to_csv('diary.csv')
                print(colored("Today's breakfast has been edited.", 'green'))
                break
    elif edit_today == "lunch":
        edit_lunch = ''
        while True:
            if edit_lunch := input("Enter new lunch: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[today_date(), 'lunch'] = edit_lunch
                df.to_csv('diary.csv')
                print(colored("Today's lunch has been edited.", 'green'))
                break
    elif edit_today == "dinner":
        edit_dinner = ''
        while True:
            if edit_dinner := input("Enter new dinner: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[today_date(), 'dinner'] = edit_dinner
                df.to_csv('diary.csv')
                print(colored("Today's dinner has been edited.", 'green'))
                break
    elif edit_today == "snack":
        edit_snack = ''
        while True:
            if edit_snack := input("Enter new snack: "):
                df = pd.read_csv('diary.csv', index_col='date')
            df.loc[today_date(), 'snack'] = edit_snack
            df.to_csv('diary.csv')
            print(colored("Today's snack has been edited.", 'green'))
            break
    else:
        print(colored("Your choice isn't recognised. Please try again.", 'red'))
        edit_diary()
    additional_options()
    additional_selections()
```

```
def edit_yesterday_diary():
    import pandas as pd
    from termcolor import colored
    edit_yesterday = input("Choose yesterday's meal to edit (breakfast/lunch/dinner/snack): ")
    if edit_yesterday == "breakfast":
        edit_yesterday_breakfast = ''
        while True:
            if edit_yesterday_breakfast := input("Enter new breakfast: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[yesterday_date(), 'breakfast'] = edit_yesterday_breakfast
                df.to_csv('diary.csv')
                print(colored("Yesterday's breakfast has been edited.", 'green'))
                break
    elif edit_yesterday == "lunch":
        edit_yesterday_lunch = ''
        while True:
            if edit_yesterday_lunch := input("Enter new lunch: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[yesterday_date(), 'lunch'] = edit_yesterday_lunch
                df.to_csv('diary.csv')
                print(colored("Yesterday's lunch has been edited.", 'green'))
                break
    elif edit_yesterday == "dinner":
        edit_yesterday_dinner = ''
        while True:
            if edit_yesterday_dinner := input("Enter new dinner: "):
                df = pd.read_csv('diary.csv', index_col='date')
                df.loc[yesterday_date(), 'dinner'] = edit_yesterday_dinner
                df.to_csv('diary.csv')
                print(colored("Yesterday's dinner has been edited.", 'green'))
                break
    elif edit_yesterday == "snack":
        edit_yesterday_snack = ''
        while True:
            if edit_yesterday_snack := input("Enter new snack: "):
                df = pd.read_csv('diary.csv', index_col='date')
            df.loc[yesterday_date(), 'snack'] = edit_yesterday_snack
            df.to_csv('diary.csv')
            print(colored("Yesterday's snack has been edited.", 'green'))
            break
    else:
        print(colored("Your choice isn't recognised. Please try again.", 'red'))
        edit_yesterday_diary()
    additional_options()
    additional_selections()
```

LOGIC & CODE

Saving meals to .CSV file

A function to save meal data was written using Pandas:

- Convert the meals dictionary created from the meal input and edit functions into a Pandas dataframe and save the dataframe as a .csv file (‘diary.csv’) for easy access and editing.
- The function first checks if a ‘diary.csv’ file already exists, and handles a FileExistsError using a try except statement:
  - If the file doesn’t exist, the function creates the file and saves the meal data to the file.
  - If the file exists, the function saves the meal data to the file.
- Users then receive a confirmation that the meals have been saved to the diary.

```
def add_input(meals_dict):
    import pandas as pd
    from termcolor import colored
    df = pd.DataFrame (meals_dict)
    try:
        df.to_csv('diary.csv', index=False, header=True, mode="x")
    except FileExistsError:
        df.to_csv('diary.csv', index=False, header=False, mode="a")
    print(colored("Your meals have been saved.", 'green'))
    return df
```

LOGIC & CODE

Displaying the diary

A function to display the food diary using PrettyTable was created to convert diary.csv data into an ASCII table to be displayed on the terminal screen.

```
def view_diary():
    from termcolor import colored
    print(colored("Here's your food diary:", 'blue'))
    from prettytable import PrettyTable
    table_view = PrettyTable()
    from prettytable import from_csv
    fp = open("diary.csv", "r")
    table_view = from_csv(fp)
    print(table_view)
    additional_options()
    additional_selections()
```

LOGIC & CODE

Additional app features

Functions to display and activate additional app features (edit today's meals, edit yesterday's meals, view diary, exit) were created to:

- Display and prompt users to select additional app features in an easy to read table using PrettyTable.
- Users must enter the number corresponding to their selection (per features table). An if-elif-else statement handles the scenario if the user enters a blank value, or a value that is not one of the numbers listed in the features table, advising them the input is not recognised, and to try again until a correct selection is entered.
- Additional app features are called after meals are inputted or edited, or the diary is viewed.

```
def additional_options():
    from termcolor import colored
    print(colored("Here are some additional diary features. When prompted, enter the number corresponding to your selection: ", 'blue'))
    from prettytable import PrettyTable
    options_table = PrettyTable (["Number", "Feature"])
    options_table.add_row(["1", "Edit today's meals"])
    options_table.add_row(["2", "Edit yesterday's meals"])
    options_table.add_row(["3", "View diary"])
    options_table.add_row(["4", "Exit session"])
    print(options_table)
    additional_selections()
```

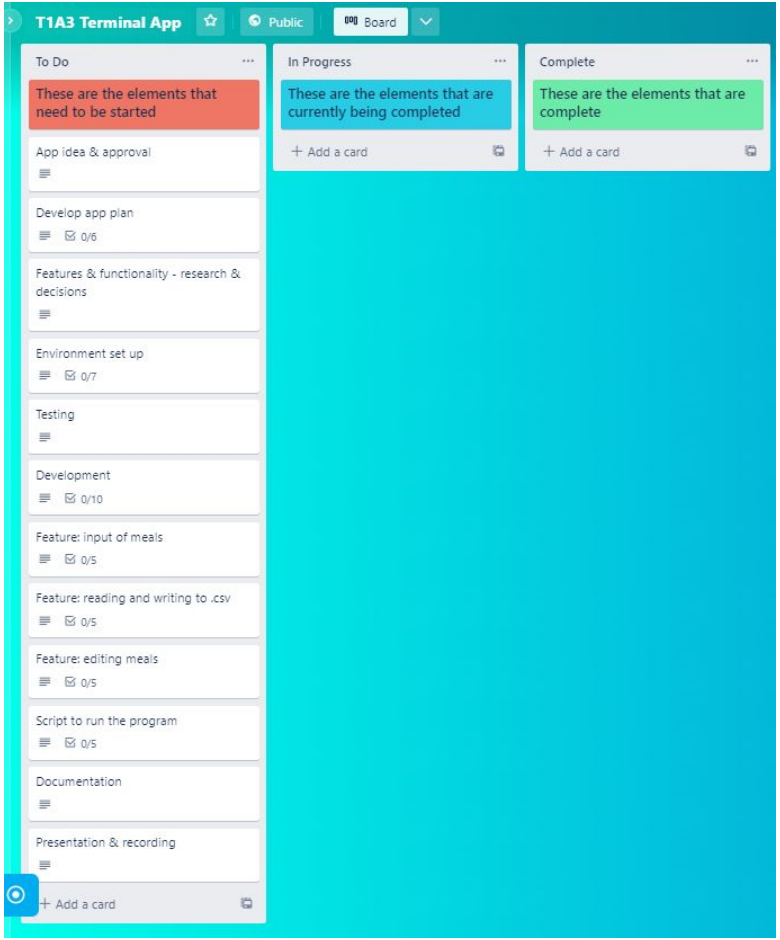
```
def additional_selections():
    from termcolor import colored
    selection_input = input("Enter your selection: ")
    if selection_input == "1":
        edit_diary()
    if selection_input == "2":
        edit_yesterday_diary()
    elif selection_input == "3":
        view_diary()
    elif selection_input == "4":
        print(colored("Thanks for using the food diary. Come back tomorrow to input and track your meals.", 'blue'))
        quit()
    else:
        print(colored("Your selection isn't recognised. Please try again.", 'red'))
        additional_options()
        additional_selections()
```

PROJECT PLAN

The project plan was tracked using Trello.
View the board here: https://trello.com/b/TwGduS2x/t1a3-terminal-app.

The project was broken down into 5 stages:
Stage 1: Initial Planning where a full outline of the tasks and subtasks required to complete the project were listed in Trello in order of what was to be actioned and completed first. This ensured no steps were missed and provided a useful guideline for the order of operations.

Stage 2: Planning Progress including the initial approval of the app idea by CA educators and the development of a full app plan including target users and app features. Then, decisions on key app features and functionality were made, and the research of Python packages and useful functions commenced.

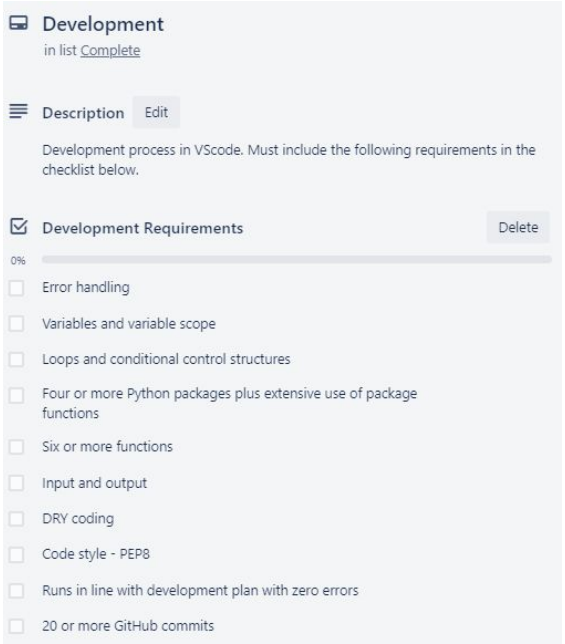


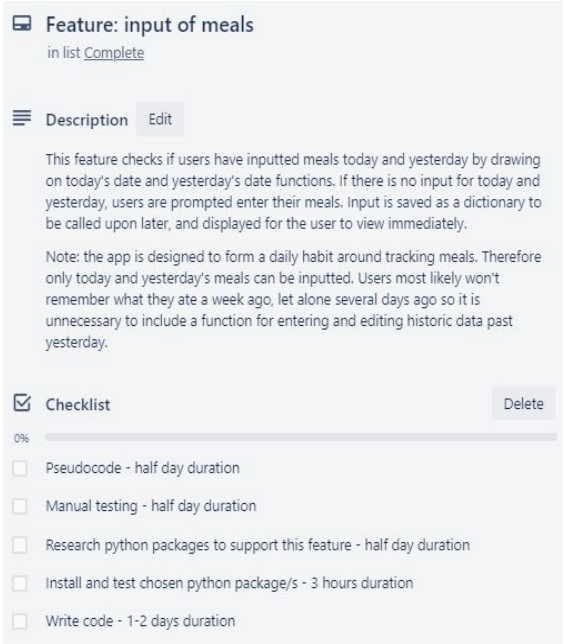
PROJECT PLAN

Stage 3: Testing & Development where manual tests were conducted on two key features (inputting meals and saving to .csv) and then developed in Visual Studio Code. The manual tests were both unit tests which were designed to test several scenarios to ensure the functions worked as intended, and enhanced the overall user experience.
A development plan board containing a checklist of all the requirements of the assignment helped to guide the features and functionality of the app, as well as other important best practices (e.g. PEP8 style convention, regular GitHub commits).
Key feature plan boards were also created as these features are critical to the app running as expected, without errors. Again, a checklist was used including approximate durations to guide the development process.

Stage 4: Scripting & Documentation where a bash script was written to facilitate execution of the program. In addition, documentation (README.md) was written containing all information necessary for users to run the program plus additional helpful information about the program and its functions and features.

Stage 5: Completion & Submission where I packaged up all files and documentation into a .zip file as per the assessment requirements, and submitted via Canvas.





REFLECTION

Much like the portfolio website this project was particularly challenging as I've never coded in Python before, let alone developed a fully functioning app.

The most difficult parts of the project were creating logical code structures and functions that were Pythonic and DRY. There are so many ways to write Python code that works, but it was ensuring that code was efficient and handled errors that took up the majority of project time.

One of the most enjoyable parts of the project was creating the project plan and documentation. I enjoy the planning stages of projects and the documentation allowed me to reflect on what I had created, and even helped to identify some missing pieces and rectify this.

This app is very simple, and I would like to further develop additional features to make it more valuable and user-friendly.