

Matt Bryant - T1A3

Terminal App – Calorie Calculator

Application Overview

The main section of the app featuring the input parameters

This section was simple to display, it relied on basic input for the user, displaying errors, and showing each input when moving on to the next section.

A welcome message and basic instructions were put at the beginning with some character lines as separations

```
Hello and welcome to your calorie calculator!
-----
To help you understand how many calories you should be eating, We will gather some basic information and determine your goals.
-----
Please enter your name: Matt
Please enter your age: 29
Please enter your gender: male
Please enter your weight in kilograms: 86
Please enter your height in centimeters: 171
Please enter your activity level (sedentary, lightly active, moderately active, very active, extra active): lightly active
What are your current goals? (lose weight, build muscle, maintain): lose weight
-----
```

The resulting output

```
-----  
Thanks for that Matt, Your basal metabolic rate (BMR) is: 1788.75  
Your Basal Metabolic Rate is how much energy your body burns per day by itself  
-----  
To maintain your current weight, you should be consuming 2460 calories per day.  
-----  
Your goal is to Lose Weight - 1960 calories calories per day.  
-----
```

After information was taken, user BMR was displayed, followed by maintenance calories, and then depending on the goal the newly calculated calories

Meal breakdown by calories

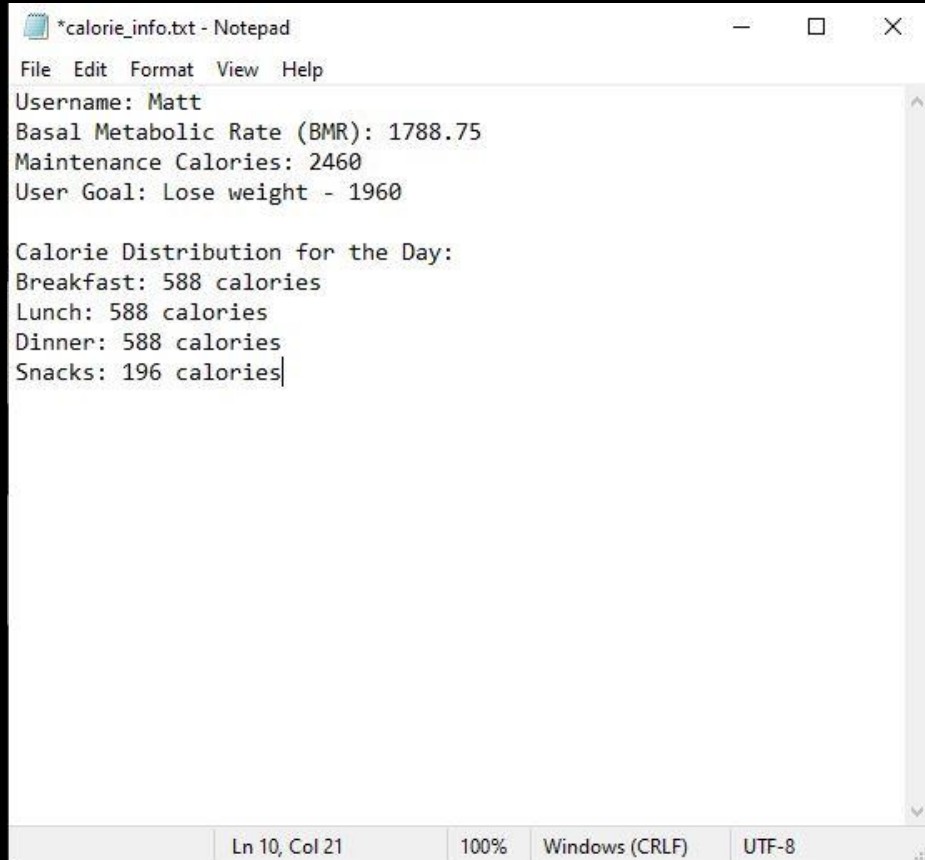
```
-----  
Calorie Distribution for the Day:  
Breakfast: 588 calories  
Lunch: 588 calories  
Dinner: 588 calories  
Snacks: 196 calories  
-----
```

Followed by this section was the meal breakdown dividing the adjusted calories in ratios and set in average meals per day

Text output message

```
-----  
A text file named 'calorie_info' has been saved in the directory 'src' :-)  
-----
```

Finally, a message to display that a text file had been saved with all the info



A screenshot of a Notepad window titled "*calorie_info.txt - Notepad". The window contains the following text:

```
File Edit Format View Help
Username: Matt
Basal Metabolic Rate (BMR): 1788.75
Maintenance Calories: 2460
User Goal: Lose weight - 1960

Calorie Distribution for the Day:
Breakfast: 588 calories
Lunch: 588 calories
Dinner: 588 calories
Snacks: 196 calories|
```

The status bar at the bottom indicates "Ln 10, Col 21", "100%", "Windows (CRLF)", and "UTF-8".

The final outputted text file with entered user data

Code Overview

Initial input fields taking strings and using while loops to prevent invalid input

```
# take user name
def get_name():
    while True:
        user_name = input(Fore.RED + "Please enter your name: ").strip()
        if user_name and user_name.isalpha():
            return user_name
        else:
            print("Please enter a valid name.")

# take user age
def get_age():
    while True:
        try:
            user_age = int(input("Please enter your age: "))
            if 0 <= user_age <= 100:
                return user_age
            print("Invalid age. Please enter a valid age (0-100).")
        except ValueError:
            print("Invalid input. Please enter a numeric age.")
```

The code structure begun with using separate functions to take each input, originally I had used global variables outside of these functions but after realizing it was bad practice, re-factored to use function returns instead.

A while loops was used to prevent invalid input and an .isalpha() method to make sure only characters were entered

An integer input followed the same structure but used the while loop to ensure user enters an age between 0 and 100 as reasonable variance.

A try and except block was used to prevent invalid input type

Code Overview

Taking the user gender input was done using a tuple of acceptable inputs and a .lower() method to ensure no errors from capital letters. A while loop was also used to prevent invalid entry.

User weight and height followed the same structure as age with different limitations, and a Try and Except block to handle errors

Gender input using a list to contain correct answers

```
# take user gender
def get_gender():
    genders = ["male", "female", "m", "f"]
    while True:
        user_gender = input("Please enter your gender: ").strip().lower()
        if user_gender in genders:
            return user_gender
        print("Please enter either 'Male' or 'Female'.")

# take user weight
def get_weight():
    while True:
        try:
            user_weight = float(
                input("Please enter your weight in kilograms: "))
            if 0 < user_weight <= 130:
                return user_weight
            print("Invalid weight. Please enter a valid weight (0-130 kg).")
        except ValueError:
            print("Invalid input. Please enter a numeric weight.")

# take user height
def get_height():
    while True:
        try:
            user_height = float(
                input("Please enter your height in centimeters: "))
            if 0 < user_height <= 250:
                return user_height
            print("Invalid height. Please enter a valid height (0-250 cm).")
        except ValueError:
            print("Invalid input. Please enter a numeric height.")
```

Input fields that require integer/float inputs with Try & Except blocks to prevent input error

Calculating BMR based on the previous user inputs then taking activity levels

```
# calculate bmr for either male or female
def calculate_bmr(user_age, user_gender, user_weight, user_height):
    if user_gender == "male":
        male_bmr = (10 * user_weight) + \
            (6.25 * user_height) - (5 * user_age) + 5
        return male_bmr
    elif user_gender == "female":
        female_bmr = (10 * user_weight) + \
            (6.25 * user_height) - (5 * user_age) - 161
        return female_bmr
    else:
        raise ValueError("Invalid gender.")

# take user activity level
def get_activity_level():
    activity_levels = ["sedentary", "lightly active",
        "moderately active", "very active", "extra active"]
    while True:
        user_activity_level = input(
            "Please enter your activity level (sedentary, "
            "lightly active, moderately active, "
            "very active, extra active): ").strip().lower()
        if user_activity_level in activity_levels:
            return user_activity_level
        print("Please enter a valid activity level.")
```

To calculate the BMR I made a function that takes the previous input parameters and combines them using a mathematical formula found online (Mifflin-St Jeor.), I used if and elif statements to control whether a male or female BMR was required, and an else statement to prevent error.

The next function took the users activity level by storing values in a list and checking if input matched any values in the list, if not it would continue to ask for input.

Calculating maintenance calories using a dictionary with multiplier values

The next function calculated the maintenance calories by taking a value associated with the activity level, and multiplying the BMR by that values. Values were stored in a dictionary. An if statement was used to check correct entry and an else statement to raise a value error and take input again.

Next function determined the user goals as a means of calculating a new calorie amount. Values were stored in a list, and if input matched list an if statement performed the associated calculation. If and else statements were used to ensure valid input was captured

```
# calculate user maintenance calories
def calculate_maintenance_calories(bmr, user_activity_level):
    activity_multipliers = {
        "sedentary": 1.2,
        "lightly active": 1.375,
        "moderately active": 1.55,
        "very active": 1.725,
        "extra active": 1.9
    }
    if user_activity_level in activity_multipliers:
        multiplier = activity_multipliers[user_activity_level]
        maintenance_calories = round(bmr * multiplier)
        return maintenance_calories
    else:
        raise ValueError("Invalid activity level.")

# take user goals and calculate new caloric intake
def get_goals(maintenance_calories):
    while True:
        user_goal = input(
            "What are your current goals? (lose weight, "
            "build muscle, maintain): ").strip().lower()
        if user_goal in ["lose weight", "build muscle", "maintain"]:
            if user_goal == "lose weight":
                return "Lose Weight - {} calories".format(maintenance_calories - 500)
            elif user_goal == "build muscle":
                return "Build Muscle - {} calories".format(maintenance_calories + 500)
            else:
                return "Maintain - {} calories".format(maintenance_calories)
        print("Please enter a valid goal (lose weight, build muscle, maintain).")
```

Taking user goals and using BMR and maintenance calories to calculate new total

Function contain all write to file info, controlling the output to text file

The final function was probably the most complex (at my level of experience) and consisted of writing the info to file, beginning with a 'with open as file' statement that would open and write a text file with each consecutive formatted line inside it. I called on the format method to arrange each file.write accordingly and then referenced the needed variable to show the correct info.

The Calorie Distribution file.write came last as it was a feature that look me a while to troubleshoot, calling on each respective variable.

```
# write info to text file
def write_to_file(user_name, user_bmr, maintenance_calories, user_goal, meal_calories):
    with open("calorie_info.txt", "w") as file:
        file.write("User Name: {}\n".format(user_name))
        file.write("Basal Metabolic Rate (BMR): {}\n".format(user_bmr))
        file.write("Maintenance Calories: {}\n".format(maintenance_calories))
        file.write("User Goal: {}\n".format(user_goal))

        file.write("\nCalorie Distribution for the Day:\n")
        for meal, calories in meal_calories.items():
            file.write(f"{meal.capitalize()}: {calories} calories\n")
```

Main function containing all the print functions and displaying messages

```
# main function
def main():
    user_name = get_name()
    user_age = get_age()
    user_gender = get_gender()
    user_weight = get_weight()
    user_height = get_height()
    user_bmr = calculate_bmr(user_age, user_gender, user_weight, user_height)
    user_activity_level = get_activity_level()
    maintenance_calories = calculate_maintenance_calories(
        user_bmr, user_activity_level)
    user_goal = get_goals(maintenance_calories)

    print("-----")
    print(f"Thanks for that {user_name}, Your basal metabolic rate (BMR) is: {user_bmr}")
    print("Your Basal Metabolic Rate is how much energy your body burns per day by itself")
    print("-----")
    print(
        f"To maintain your current weight,"
        f"you should be consuming {maintenance_calories} calories per day.")
    print("-----")
    print(f"Your goal is to {user_goal} calories per day.")
    print("-----")
    meal_calories = distribute_calories(
        int(user_goal.split(" - ")[1].split(" ")[0]))

    print("\nCalorie Distribution for the Day:")
    for meal, calories in meal_calories.items():
        print(f"{meal.capitalize()}: {calories} calories")

    print()
    print("-----")
    print("A text file named 'calorie_info' has been saved")
    print("in the directory above 'src' :-)")
    print("-----")

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

I created a main function to act as starting point for execution of the function, called by the statement `if __name__=="__main__":`. This ensured that the variables from each function were able to be used outside their scope and be passed into the latter functions that required calculations. After each function was passed, all the print messages were displayed as a summary to the program. I chose to use empty print statements to separate lines which I'm not sure if that is classified as bad practice, but the intention was for aesthetic purposes.

I also included the meal division of calories in this section, using a for loop to print each category.

Finally, a message to describe where the text file would be located was shown.

Favorite Parts

I found this project incredibly challenging particularly because Python hasn't quite clicked for me just yet, I understand the logic and the syntax but when it comes to actually writing code I struggle to come up with the most efficient way of achieving the goal let alone whether I can achieve it at all.

Having said that, I enjoyed breaking down and understanding the purpose of using functions to organize the code, using PEP8 was useful in keeping things organized and satisfyingly clean. I learned a very valuable lesson in refactoring and not using global variables. As a summary:

- Learning to organize code with functions
- Refactoring to make a program function the same under cleaner visual code
- The satisfaction in making a program that runs as it's supposed to
- Learning to organize code and actually see real-world application of a Python program as a whole

Challenges

The challenges probably outweighed the favorite parts in this project, I struggled with learning Python and so this brought plenty of obstacles. I think the main challenge came from needing to perform a task in the most efficient way possible. E.g originally I'd created functions with global variables because that made the most sense in my head, down the track I realized it wasn't good practice and had to learn a different way of doing those tasks.

- The learning curve behind a main() function was steep and took quite some time
 - Using lists and multipliers to perform calculations
 - All of the write to file functions were hard to wrap my head around
 - Dividing the meal categories based on the ratio multipliers I'd set
- Keeping the code as clean as possible was hard, I suspect there are still lots of sections of code that could be cleaned up