



Project Report

BMI & Daily Calorie Intake Analyzer

Submitted by: Ansh Jain

Registration Number: 25BCE10890

Course: B.Tech CSE – Core

Subject: Introduction To Problem Solving

Course Code: CSE1021

Introduction

Health awareness among people has significantly increased in recent years due to the rise of obesity, eating disorders, lifestyle diseases, and sedentary living. Understanding one's physical health requires basic knowledge of **BMI (Body Mass Index)** and **daily calorie intake**, which are the most widely used indicators of nutritional and physical fitness. Many individuals, especially students living away from home, struggle to track their eating habits or understand whether they are consuming more or less than required.

This Python-based project, **BMI & Daily Calorie Intake Analyzer**, allows users to enter their weight and height to calculate BMI and determine their corresponding health category. Additionally, users can input the food items consumed throughout the day and calculate the total calorie intake based on a pre-stored food database. The program provides suggestions based on the calorie limit, helping users make better dietary choices.

The system is simple, fully based on **basic Python concepts**, and does not use external libraries, databases, or file handling. The goal is to encourage beginners to understand the fundamentals of logic building, user input handling, conditional branching, loops, function usage, and dictionary-based lookups.

Problem Statement

A large number of people lack awareness about whether they are eating the right amount or maintaining a healthy body weight. While multiple mobile apps exist, they require registration, complex interfaces, or internet connectivity, making them unsuitable for beginner-level learners.

The challenge is to design a **simple, offline Python console application** that can:

- Calculate BMI accurately and identify the category of an individual.
- Maintain a calorie count based on food consumed throughout the day.
- Provide suggestions based on calorie limits.
- Contain a database of common everyday food items for reference.

This project aims to solve the lack of accessibility and complexity by providing an easy-to-use, interactive program.

Functional Requirements

Functional requirements describe what the system should be able to do. For this project, they include:

1. The system must allow the user to input weight and height.
2. The system must calculate BMI using the BMI formula.
3. The system must determine BMI category (Underweight, Normal, Overweight, Obese).
4. The system must display a large list of food items with calorie values.
5. The system must allow repeated entry of food consumed.
6. The system must calculate the total calories consumed.
7. The system must provide a suggestion message based on the total calories and predefined limit.
8. The system should allow the user to stop adding food when the user types "**done**".

Non-Functional Requirements

These describe constraints regarding quality and performance of the system:

1. **Usability**: Interface should be easy to use and understandable for beginners. The user interacts through simple text-based menus.
2. **Performance**: The program must process inputs instantly without delays.
3. **Reliability**: Should handle invalid inputs (e.g., non-numeric values), using error handling.
4. **Scalability**: The dictionary of food items can be expanded with more foods in the future.
5. **Portability**: Works on any system that supports Python (Windows/Mac/Linux).
6. **Maintainability**: Code is modular using functions so updates can be made easily.

System Architecture

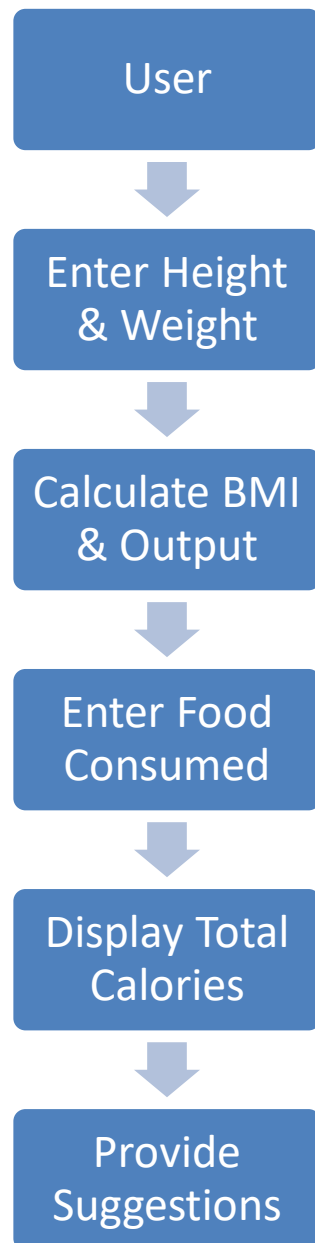
The system architecture follows a straightforward **menu-driven modular structure**. The user interacts with a main menu that navigates to different functional modules. The control flow is sequential and loops until the user chooses to exit.

Design Decisions & Rationale

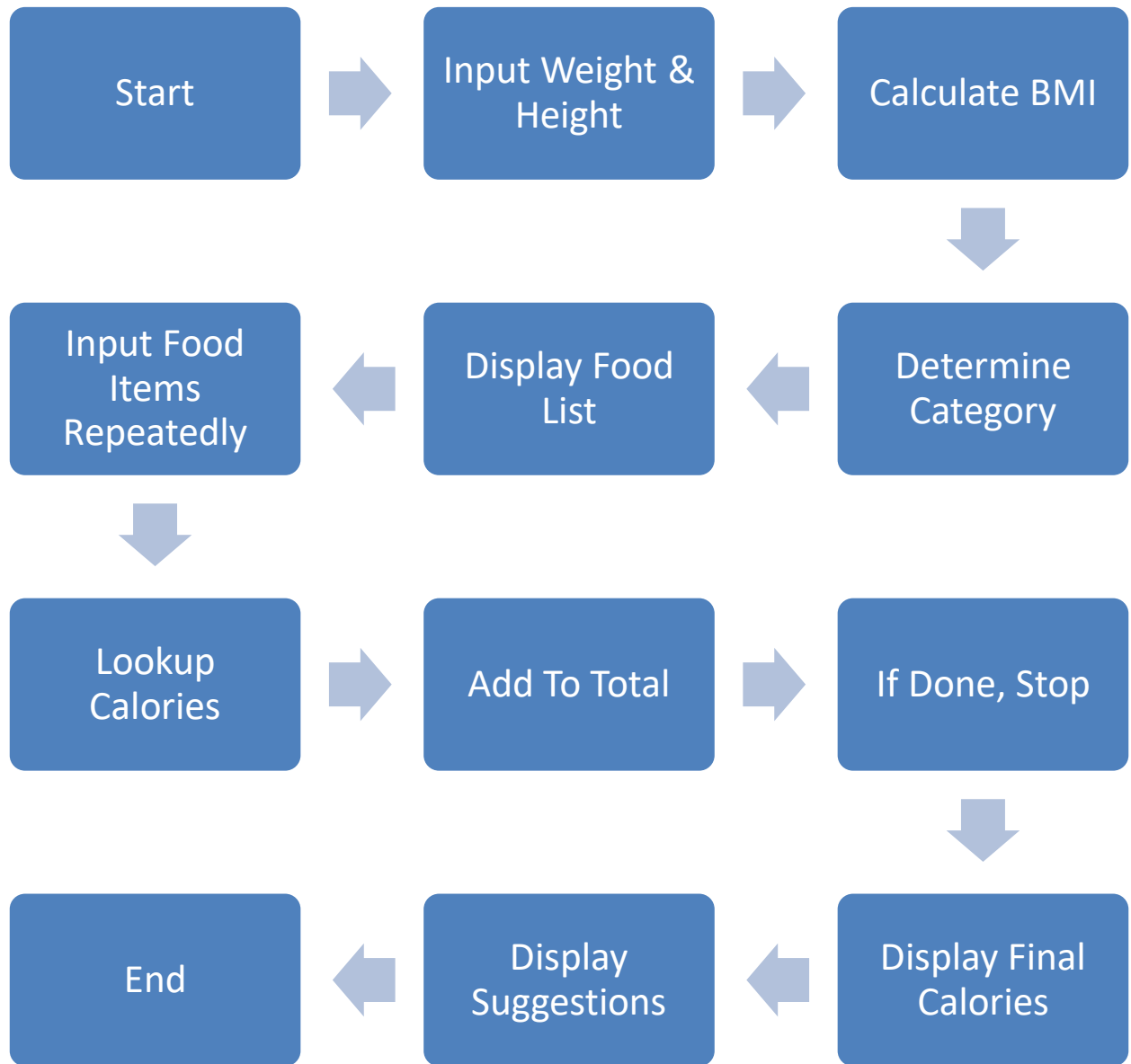
- **Functions were used** to improve code readability and modularity.
- **Dictionary-based food database** allows fast lookups and easy expansion.
- The program uses **while loops** to repeatedly accept food entries.
- **Try/Except error handling** ensures user does not break the program by entering incorrect values.
- Fixed calorie limit (2000 calories) acts as a reference benchmark for average requirements.

Design Diagrams

Use Case Diagram



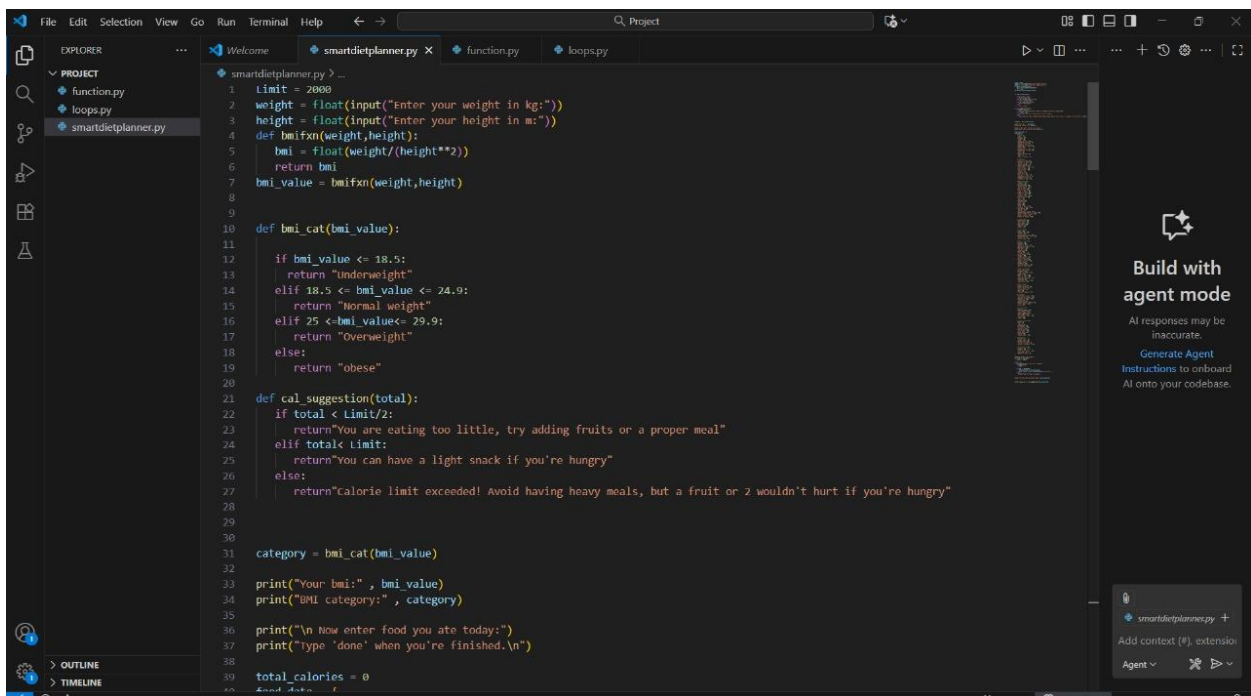
Workflow Diagram



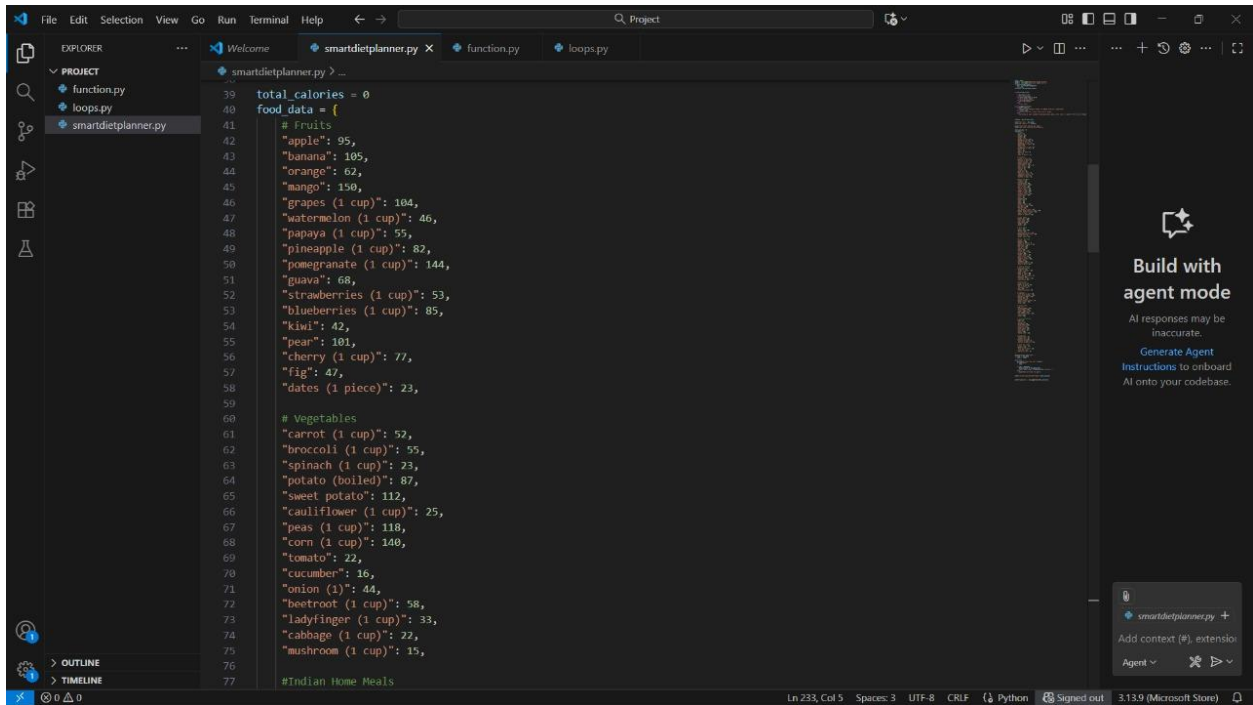
Implementation

The program is implemented entirely using basic Python concepts:

- Input & Output operations
- Mathematical formula calculations
- Functions for modular programming
- Conditional statements (if-elif-else)
- Looping structure (while True)
- Error handling using try-except
- Dictionary data structure for calorie storage
- No file handling, JSON, database, GUI or external modules

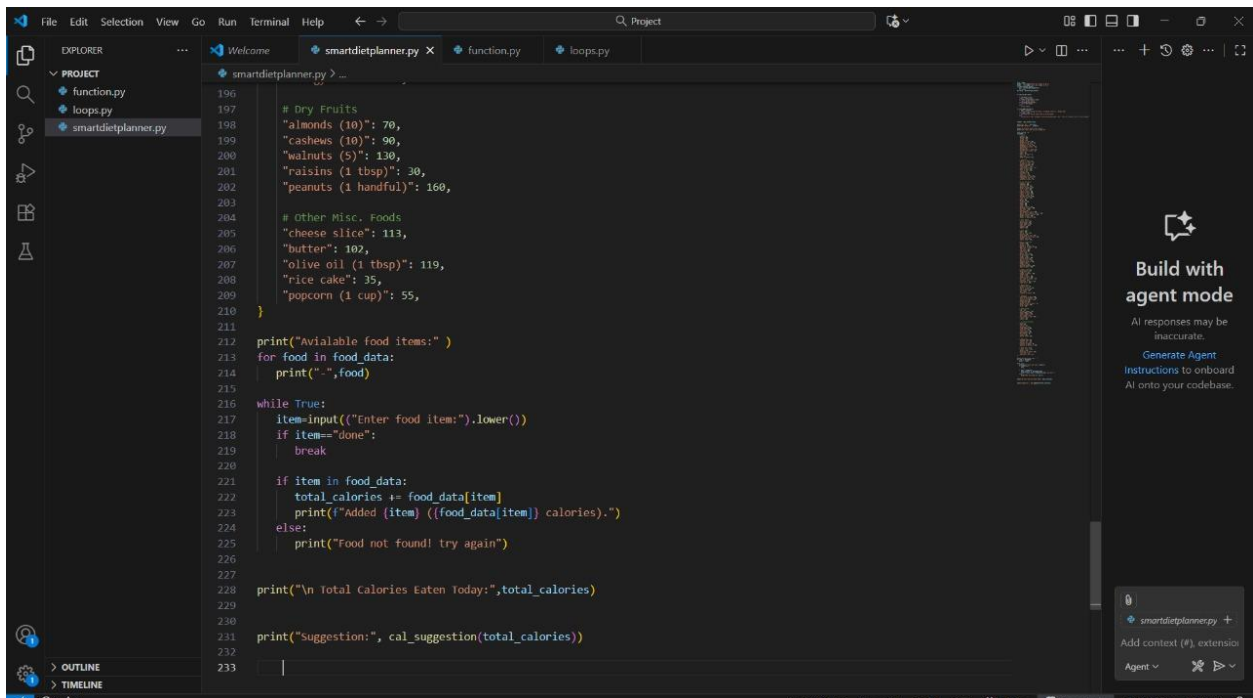


```
1 limit = 2000
2 weight = float(input("Enter your weight in kg:"))
3 height = float(input("Enter your height in m:"))
4 def bmfzn(weight,height):
5     bmi = float(weight/(height**2))
6     return bmi
7 bmi_value = bmfzn(weight,height)
8
9
10 def bmi_cat(bmi_value):
11
12     if bmi_value <= 18.5:
13         return "Underweight"
14     elif 18.5 <= bmi_value <= 24.9:
15         return "Normal weight"
16     elif 25 <=bmi_value<= 29.9:
17         return "Overweight"
18     else:
19         return "obese"
20
21 def cal_suggestion(total):
22     if total < limit/2:
23         return"You are eating too little, try adding fruits or a proper meal"
24     elif total< limit:
25         return"You can have a light snack if you're hungry"
26     else:
27         return"Calorie limit exceeded! Avoid having heavy meals, but a fruit or 2 wouldn't hurt if you're hungry"
28
29
30
31 category = bmi_cat(bmi_value)
32
33 print("Your bmi:" , bmi_value)
34 print("BMI category:" , category)
35
36 print("\n Now enter food you ate today:")
37 print("type 'done' when you're finished.\n")
38
39 total_calories = 0
40 food_dict = {}
```



The screenshot shows the VS Code editor with the file `smartdietplanner.py` open. The code defines three dictionaries: `food_data` (fruits), `veg_data` (vegetables), and `indian_home_meals`. The `food_data` dictionary lists various fruits and their calorie counts. The `veg_data` dictionary lists various vegetables and their calorie counts. The `indian_home_meals` dictionary lists various Indian home meals and their calorie counts.

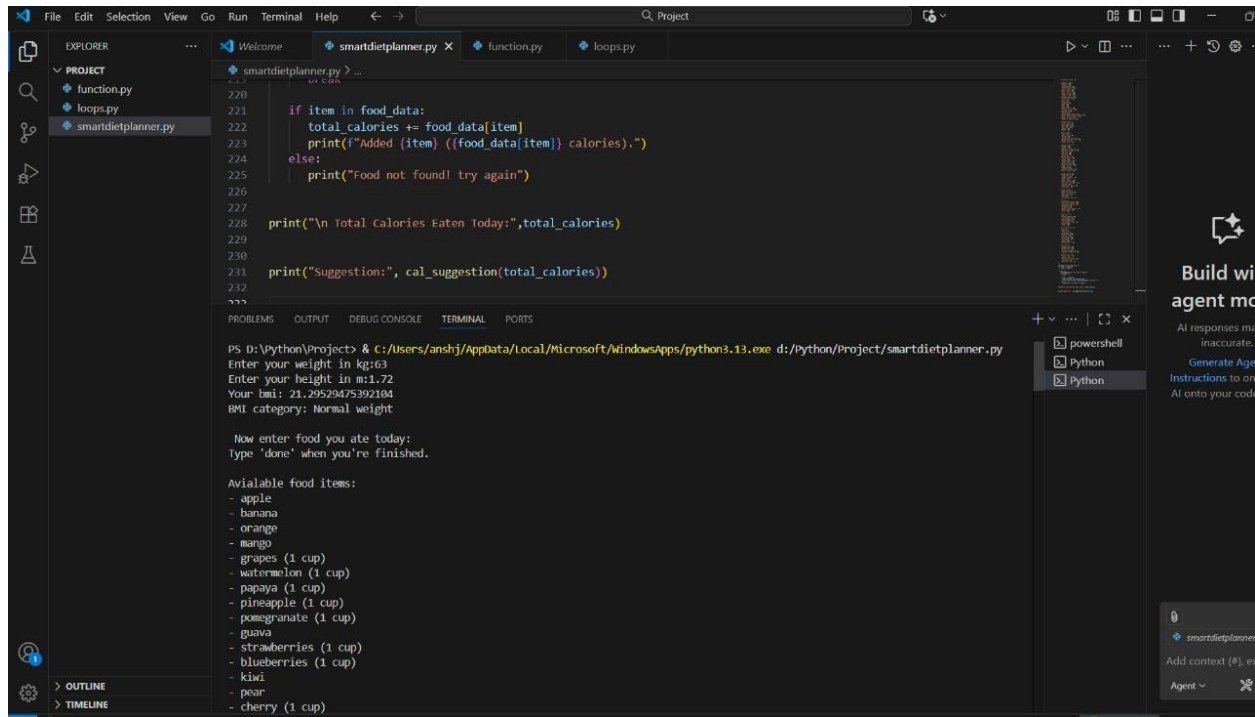
```
39 total_calories = 0
40 food_data = {
41     # Fruits
42     "apple": 95,
43     "banana": 105,
44     "orange": 62,
45     "mango": 150,
46     "grapes (1 cup)": 104,
47     "watermelon (1 cup)": 46,
48     "papaya (1 cup)": 55,
49     "pineapple (1 cup)": 82,
50     "pomegranate (1 cup)": 144,
51     "guava": 68,
52     "strawberries (1 cup)": 53,
53     "blueberries (1 cup)": 85,
54     "kiwi": 42,
55     "pear": 101,
56     "cherry (1 cup)": 77,
57     "fig": 47,
58     "dates (1 piece)": 23,
59
60     # Vegetables
61     "carrot (1 cup)": 52,
62     "broccoli (1 cup)": 55,
63     "spinach (1 cup)": 23,
64     "potato (boiled)": 87,
65     "sweet potato": 112,
66     "cauliflower (1 cup)": 25,
67     "peas (1 cup)": 118,
68     "corn (1 cup)": 140,
69     "tomato": 22,
70     "cucumber": 16,
71     "onion (1)": 44,
72     "beetroot (1 cup)": 58,
73     "ladyfinger (1 cup)": 33,
74     "cabbage (1 cup)": 22,
75     "mushroom (1 cup)": 15,
76
77     # Indian Home Meals
```



The screenshot shows the VS Code editor with the file `smartdietplanner.py` open. The code continues from the previous screenshot, defining `dry_fruit_data` and `other_misc_foods` dictionaries. It then implements a loop that prompts the user to enter food items, checks if they are in the data dictionaries, and updates the `total_calories` variable. Finally, it prints the total calories and a suggestion.

```
196
197
198     # Dry Fruits
199     "almonds (10)": 70,
200     "cashews (10)": 90,
201     "walnuts (5)": 130,
202     "raisins (1 tbsp)": 30,
203     "peanuts (1 handful)": 160,
204
205     # Other Misc. Foods
206     "cheese slice": 113,
207     "butter": 102,
208     "olive oil (1 tbsp)": 119,
209     "rice cake": 35,
210     "popcorn (1 cup)": 55,
211
212 print("Available food items: ")
213 for food in food_data:
214     print("-", food)
215
216 while True:
217     item = input(("Enter food item:").lower())
218     if item == "done":
219         break
220
221     if item in food_data:
222         total_calories += food_data[item]
223         print(f"Added {item} ({food_data[item]} calories).")
224     else:
225         print("Food not found! try again")
226
227 print("\n Total Calories Eaten today:", total_calories)
228
229
230 print("suggestion:", cal_suggestion(total_calories))
231
232
233
```

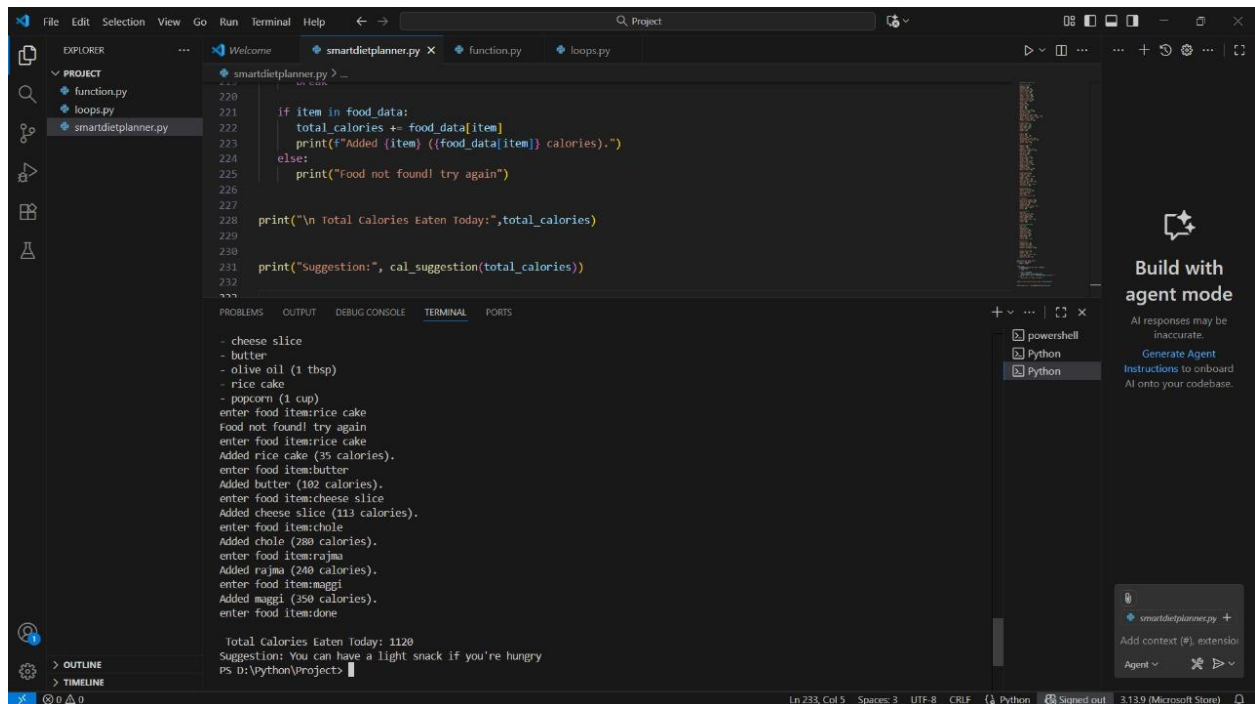
Result



```
PS D:\Python\Project> & C:/Users/anshj/AppData/Local/Microsoft/WindowsApps/python3.13.exe d:/Python/Project/smartdietplanner.py
Enter your weight in kg:63
Enter your height in m:1.72
Your bmi: 21.29529475392104
BMI category: Normal weight

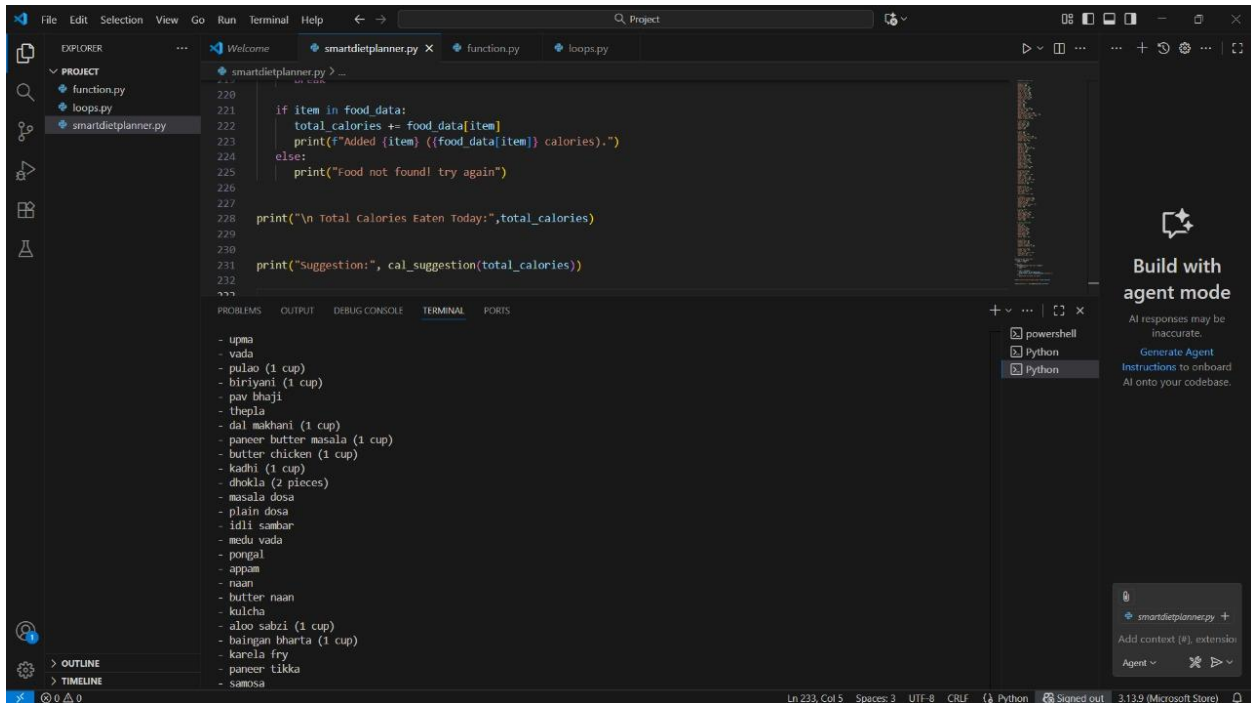
Now enter food you ate today:
Type 'done' when you're finished.

Available food items:
- apple
- banana
- orange
- mango
- grapes (1 cup)
- watermelon (1 cup)
- papaya (1 cup)
- pineapple (1 cup)
- pomegranate (1 cup)
- guava
- strawberries (1 cup)
- blueberries (1 cup)
- kiwi
- pear
- cherry (1 cup)
```



```
- cheese slice
- butter
- olive oil (1 tbsp)
- rice cake
- popcorn (1 cup)
enter food item:rice cake
Food not found! try again
enter food item:rice cake
Added rice cake (35 calories).
enter food item:butter
Added butter (102 calories).
enter food item:cheese slice
Added cheese slice (113 calories).
enter food item:chole
Added chole (280 calories).
enter food item:rajma
Added rajma (240 calories).
enter food item:maggi
Added maggi (350 calories).
enter food item:done

Total Calories Eaten Today: 1120
Suggestion: You can have a light snack if you're hungry
PS D:\python\Project>
```



Testing Approach

Testing included:

- **Functional Testing:** Checked accuracy of BMI for various combinations of height/weight.
- **Input Validation Testing:** Entering alphabets or special characters where numbers are required.
- **Boundary Testing:** Very low BMI, high calorie intake, empty input handling.
- **Usability Testing:** Verifying user flow was understandable and intuitive.

Challenges Faced

While developing the application, several challenges were encountered:

1. Indentation Errors

Python is indentation-sensitive, and even a single misplaced space caused program breakdown. Fixing indentation taught careful alignment and code formatting.

2. Handling Invalid User Input

Users entering text instead of numbers originally caused crashes. Implementing **try-except ValueError** resolved this.

3. Managing Repetitive Input

Creating a continuous loop to allow multiple food entries while still giving option to exit required thoughtful logic.

4. Large Dictionary Management

Adding and formatting hundreds of food items manually required patience and debugging due to typos.

5. Logic Design

Creating calorie suggestions based on numeric ranges required planning threshold values.

These challenges strengthened understanding of debugging and structured program flow.

Learnings & Key Takeaways

This project helped in strengthening core Python programming skills. Understanding function-based modular design and better error handling significantly improved programming confidence. The importance of testing and debugging became clear, as even minor mistakes like indentation or wrong list indexing caused failure. Overall, it improved analytical thinking and problem-solving abilities.

Future Enhancements

In the future, the program can be extended with:

- Graphical User Interface (Tkinter or Pygame)
- Database storage of daily history
- Automatic calorie recommendation depending on age, gender, and activity level
- Plot graph of calories vs days
- Voice-based input support using speech recognition
- Barcode scanner for packaged food items

References

- Python Official Documentation (docs.python.org)
- Basic programming course materials
- Classroom lecture notes from Fundamentals of AI & ML
- GeeksforGeeks Tutorials