



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

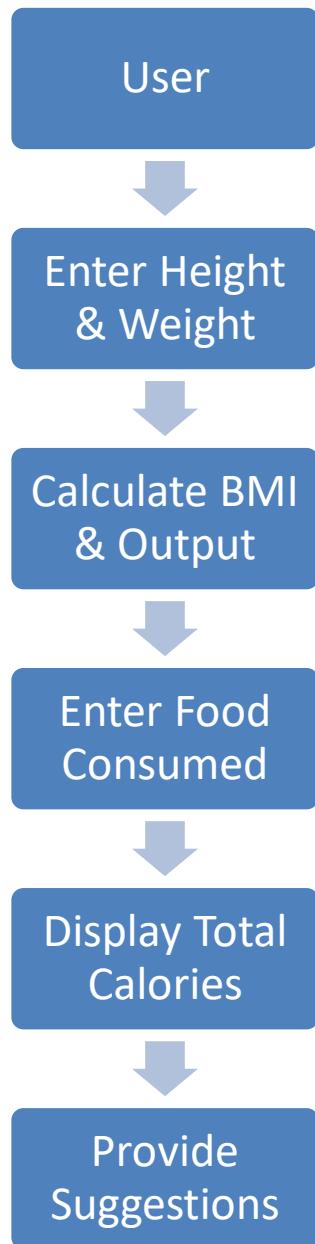
The system architecture follows a straightforward **menu-driven modular structure**. The user interacts with a main menu that navigates to different functional modules. The control flow is sequential and loops until the user chooses to exit.

Design Decisions & Rationale

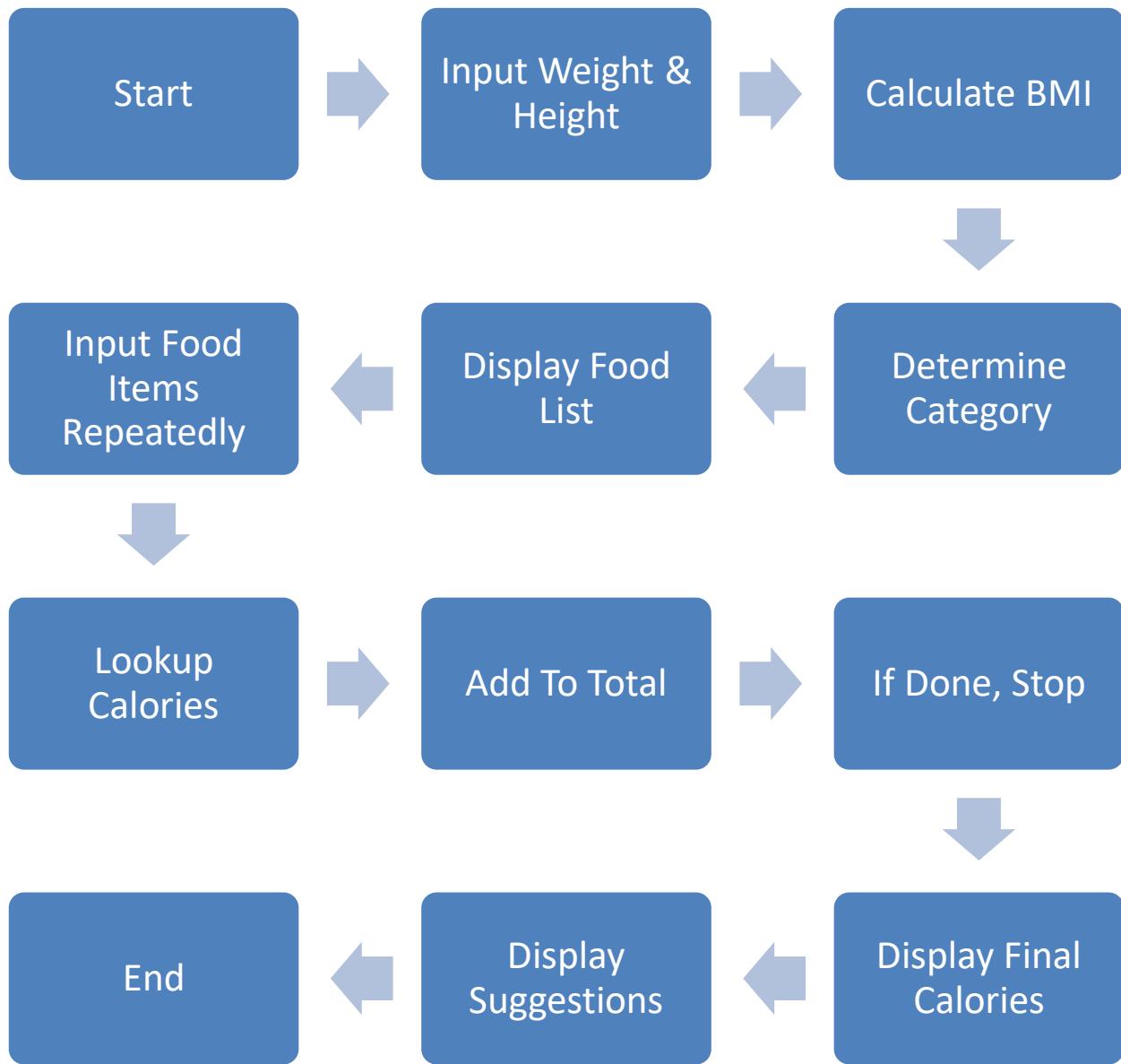
- **Functions were used** to improve code readability and modularity.
- **Dictionary-based food database** allows fast lookups and easy expansion.
- The program uses **while loops** to repeatedly accept food entries.
- **Try/Except error handling** ensures user does not break the program by entering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

Design Diagrams

Use Case Diagram



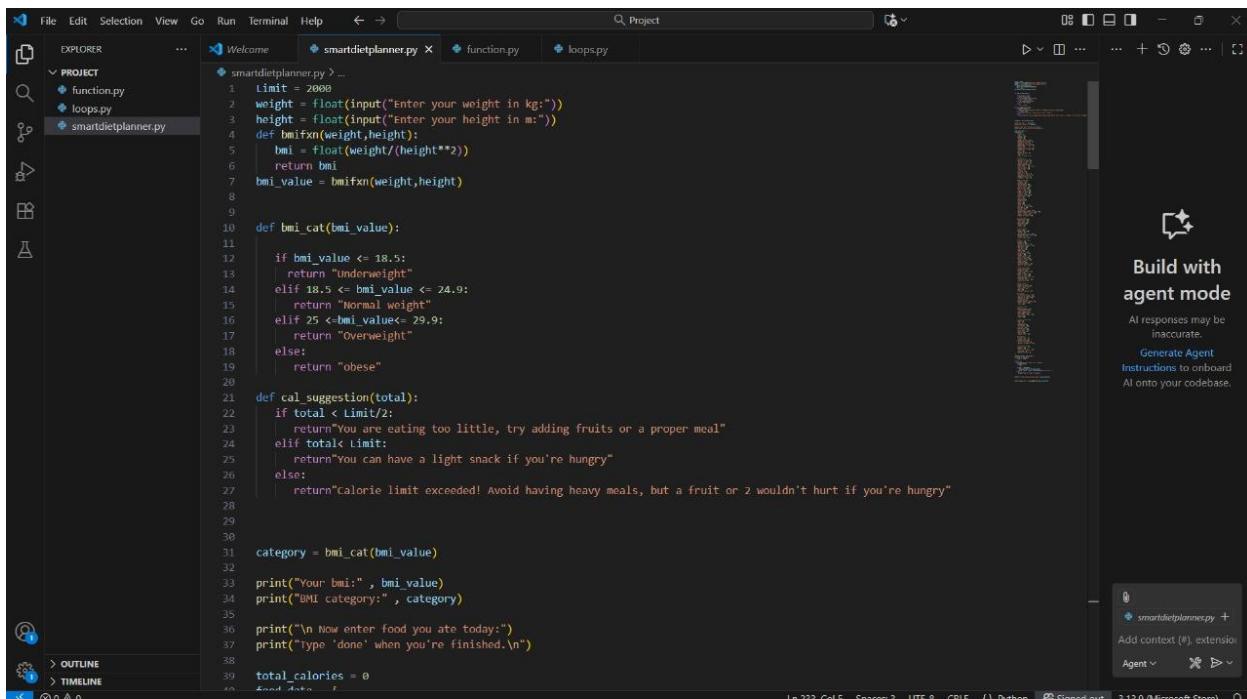
Workflow Diagram



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



A screenshot of the Microsoft Visual Studio Code interface. The left sidebar shows a project structure with files: smartdietplanner.py, function.py, and loops.py. The main editor area displays the `smartdietplanner.py` file. The code implements a BMI calculator and a calorie counter. It includes functions for calculating BMI, determining BMI categories, and providing meal suggestions based on total calorie intake. A sidebar on the right provides AI assistance, including a "Build with agent mode" button and a "Generate Agent" section.

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

```
total_calories = 0
food_data = {
    # Fruits
    "apple": 95,
    "banana": 105,
    "orange": 62,
    "mango": 150,
    "grapes (1 cup)": 104,
    "watermelon (1 cup)": 46,
    "papaya (1 cup)": 55,
    "pineapple (1 cup)": 82,
    "pomegranate (1 cup)": 144,
    "guava": 68,
    "strawberries (1 cup)": 53,
    "blueberries (1 cup)": 85,
    "kiwi": 42,
    "pear": 101,
    "cherry (1 cup)": 77,
    "fig": 47,
    "dates (1 piece)": 23,
    # Vegetables
    "carrot (1 cup)": 52,
    "broccoli (1 cup)": 55,
    "spinach (1 cup)": 23,
    "potato (boiled)": 87,
    "sweet potato": 112,
    "cauliflower (1 cup)": 25,
    "peas (1 cup)": 118,
    "corn (1 cup)": 140,
    "tomato": 22,
    "cucumber": 16,
    "onion (1)": 44,
    "beetroot (1 cup)": 58,
    "ladyfinger (1 cup)": 33,
    "cabbage (1 cup)": 22,
    "mushroom (1 cup)": 15,
}
# Indian Home Meals
```

```
# Dry Fruits
"almonds (10)": 70,
"cashews (10)": 90,
"walnuts (5)": 130,
"raisins (1 tbsp)": 30,
"peanuts (1 handful)": 160,
# Other Misc. Foods
"cheese slice": 113,
"butter": 102,
"olive oil (1 tbsp)": 119,
"rice cake": 35,
"popcorn (1 cup)": 55,
}

print("Available food items: ")
for food in food_data:
    print("-" , food)

while True:
    item=input("Enter food item:".lower())
    if item=="done":
        break

    if item in food_data:
        total_calories += food_data[item]
        print(f"Added {item} ({food_data[item]} calories.)")
    else:
        print("Food not found! try again")

print("\n Total Calories Eaten Today:",total_calories)
print("Suggestion:", cal_suggestion(total_calories))
```

Result

The screenshot shows the Microsoft Visual Studio Code interface. The project 'smartdietplanner' is open, with files 'function.py', 'loops.py', and 'smartdietplanner.py' visible in the Explorer sidebar. The 'smartdietplanner.py' file is selected and its code is displayed in the main editor area:

```
if item in food_data:  
    total_calories += food_data[item]  
    print(f"Added {item} ({food_data[item]} calories).")  
else:  
    print("Food not found! try again")  
  
print("\n Total Calories Eaten Today:",total_calories)  
  
print("Suggestion:", cal_suggestion(total_calories))
```

The terminal window shows the execution of the script:

```
PS D:\Python\Projects> & 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: 21.29524075392104  
BMI category: Normal weight  
  
Now enter food you ate today:  
Type 'done' when you're finished.  
  
Avialable 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)
```

The status bar at the bottom indicates the script is running in Python 3.13.9.

This screenshot shows the same setup as the first one, but with a different set of food items entered into the terminal. The terminal output is as follows:

```
- 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:rajam  
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
```

The status bar at the bottom indicates the script is running in Python 3.13.9.

```
if item in food_data:  
    total_calories += food_data[item]  
    print(f"Added {item} ({food_data[item]} calories).")  
else:  
    print("Food not found! try again")  
  
print("\n Total Calories Eaten Today:",total_calories)  
  
print("Suggestion:", cal_suggestion(total_calories))
```

The sidebar lists the following Indian food items:

- upma
- vada
- pulao (1 cup)
- biriyani (1 cup)
- pav bhaji
- thepla
- dal makhani (1 cup)
- paneer butter masala (1 cup)
- butter chicken (1 cup)
- kadhi (1 cup)
- dhokla (2 pieces)
- masala dosa
- plain dosa
- idli sambar
- medu vada
- pongal
- appam
- naan
- butter naan
- kulcha
- aloo sabzi (1 cup)
- baingan bharta (1 cup)
- karela fry
- paneer tikka
- samosa

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