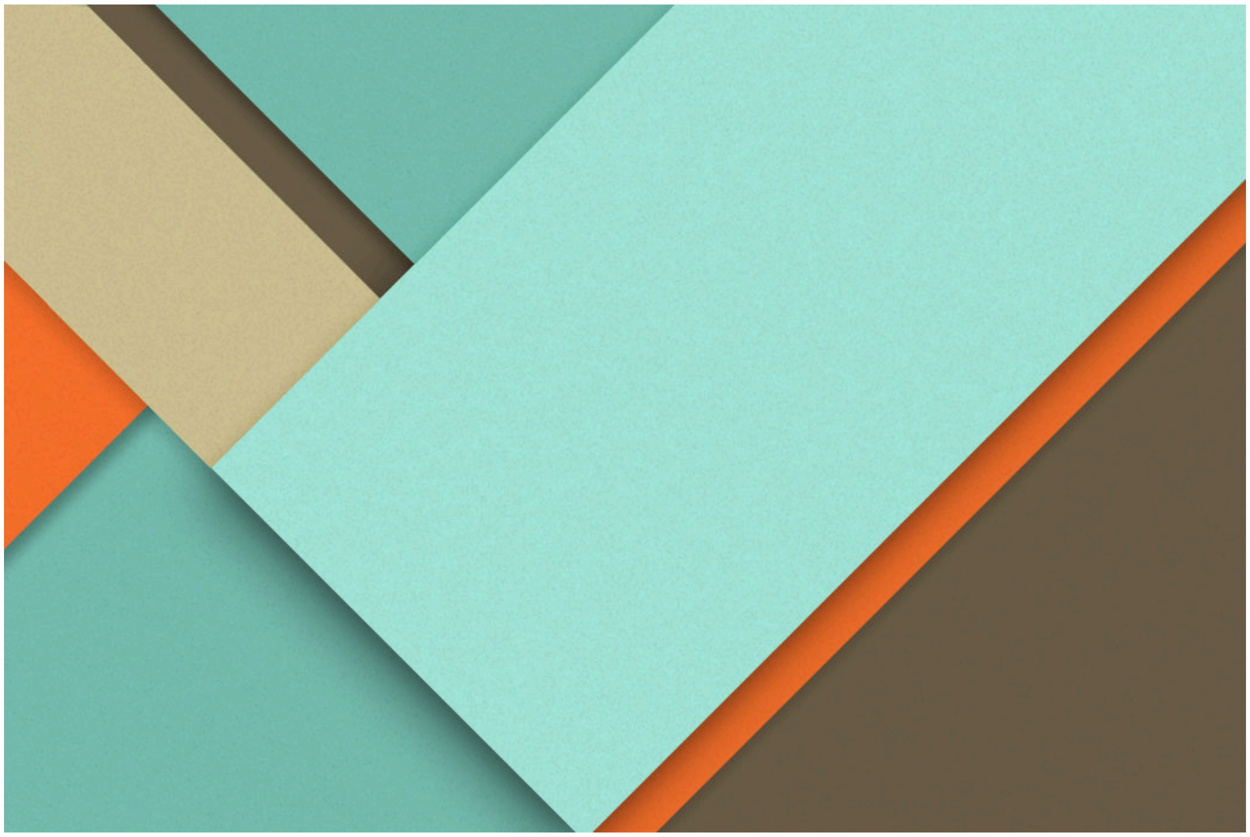


DISEASE PREDICTION SYSTEM

VITyarthi Project

Course: Python Essential

—

Avinash Singh Tomar
[25BHI10055]
VIT BHOPAL UNIVERSITY

1. Introduction

The **Disease Prediction System** is a small Python project that helps a user guess what common illness they might have based on the symptoms they select. It runs in the command line and shows how modular programming works in a simple and easy way.
This project is only for learning — not for real medical use.

2. Problem Statement

People often want a quick idea of what common illness their symptoms may be related to. They may not know how symptoms connect to different diseases.
This project solves that basic need by giving:

- A simple list of common symptoms
- A way to select symptoms easily
- A system that checks which diseases match those symptoms
- Clear results that help users understand the output

The main goal is to create a clean, easy-to-use, and modular Python program that shows how symptom-matching can be done in a basic form.

3. Functional Requirements

The program must be able to:

- Show all available symptoms in a clean, sorted list
- Allow the user to enter symptom numbers separated by commas
- Check if the user’s input is valid
- Compare the selected symptoms with the database
- Show only diseases that have at least one matching symptom
- Display the disease name and its description
- Allow the user to run the check again without restarting the program

4. Non-Functional Requirements

- The program should be easy to use, with clear instructions and simple input.
- The code should be modular, split into three separate Python files (data, logic, and interface).
- The system should handle wrong or invalid input safely without crashing.
- The matching logic should be correct and reliable, giving proper results based on the symptoms.

- The program should run smoothly on any normal Python installation without extra libraries.
- The output should be clear and readable, even for beginners.

5. System Architecture

The project uses **three modules**:

1. **mod1.py (Data Module):**
   Stores the **`disease_database`** dictionary with symptoms and descriptions.

2. **mod2.py (Logic Module):**
   Has functions to collect all unique symptoms and to match symptoms with diseases.

3. **main.py (Interface Module):**
   Shows the symptoms, takes user input, calls the logic functions, and displays the final results.

Flow:
`main.py` → get symptoms → user selects symptoms → convert numbers → send to logic → get matching diseases → show results

6. Workflow Diagram

User Input (Symptoms)

↓

Input Validation (mod3)

↓

Symptom Processing (mod2)

↓

Database Query (mod1)

↓

Pattern Matching (mod2)

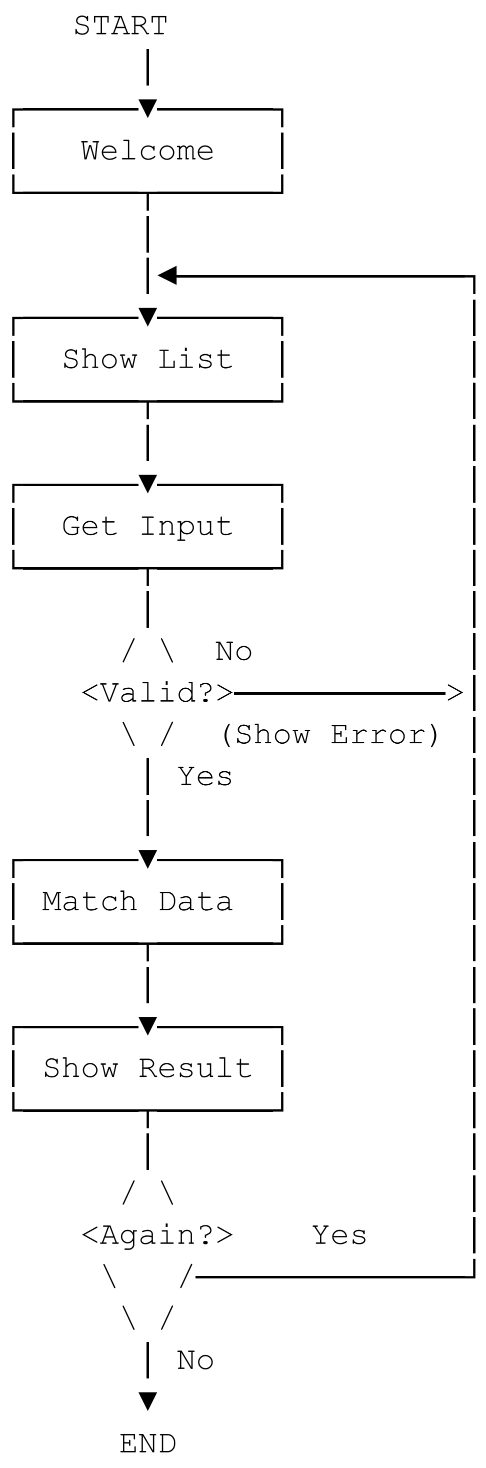
↓

Result Formatting (mod3)

↓

Display to User

Program flowchart:



7. Design Decisions & Rationale

1. Data Storage (Design)

Use a Python dictionary to store diseases and their symptoms.

Rationale:

- Easy to read and update.
- Good for a small and fixed dataset.
- Helps beginners understand key-value data structures.

2. Modular Code Structure (Design)

Split the project into three files: mod1.py, mod2.py, and main.py

Rationale:

- Keeps the code clean and organized.
- Each file has one clear job.
- Makes updates and debugging easier.

3. Input Method (Design)

Allow users to choose symptoms by entering numbers (1, 2, 3, ...).

Rationale:

- Faster than typing long symptom names.
- Prevents spelling mistakes.
- Works smoothly in a command-line program.

4. Index Conversion (Design)

Convert user input from 1-based numbering to Python’s 0-based list indexing.

Rationale:

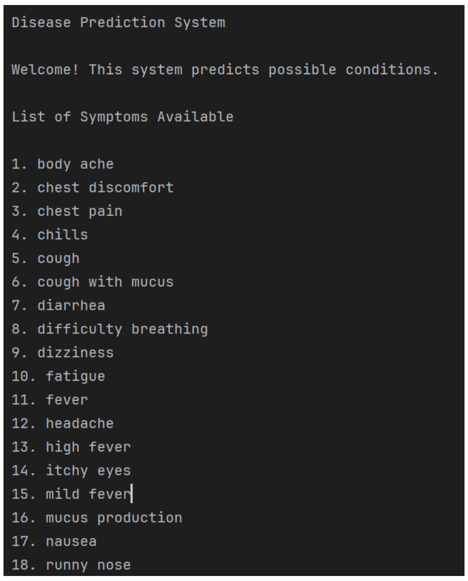
- Python lists start at index 0.
- Prevents index errors.
- Ensures correct matching of symptoms.

8. Implementation Details

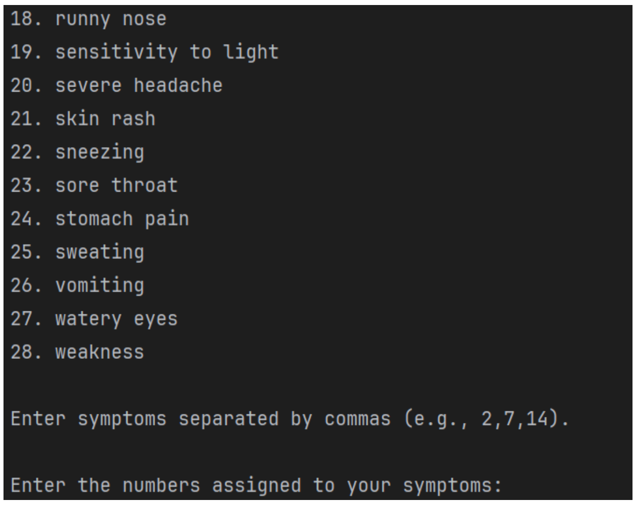
- **Symptom Collection:** Loops through all diseases, collects symptoms, removes duplicates, and sorts them.
- **Prediction:** Checks each disease and counts how many symptoms match the user's input.
- **Results:** Only diseases with at least one match are shown.
  Descriptions are fetched safely using `.get()`.
- Everything uses simple Python lists, loops, and dictionaries.

9. Screenshots

1. **Welcome message and lists of available symptoms**

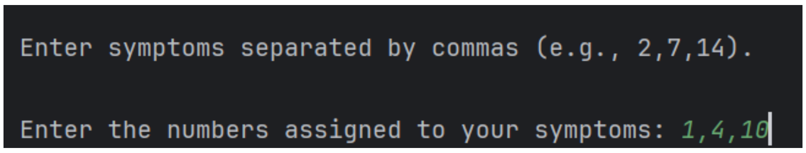


```
Disease Prediction System

Welcome! This system predicts possible conditions.

List of Symptoms Available

1. body ache
2. chest discomfort
3. chest pain
4. chills
5. cough
6. cough with mucus
7. diarrhea
8. difficulty breathing
9. dizziness
10. fatigue
11. fever
12. headache
13. high fever
14. itchy eyes
15. mild fever
16. mucus production
17. nausea
18. runny nose
```



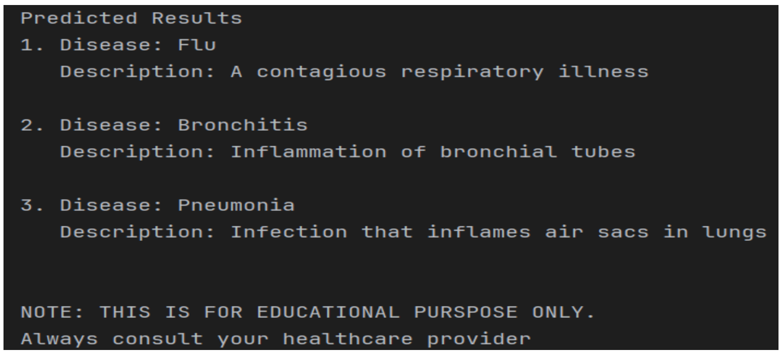
```
18. runny nose
19. sensitivity to light
20. severe headache
21. skin rash
22. sneezing
23. sore throat
24. stomach pain
25. sweating
26. vomiting
27. watery eyes
28. weakness

Enter symptoms separated by commas (e.g., 2,7,14).

Enter the numbers assigned to your symptoms:
```

2. **Enter your symptoms in numerical form from the list. (For eg: 1,4,10 [i.e:body ache,chills,fatigue])**



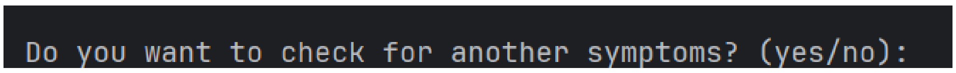
```
Enter symptoms separated by commas (e.g., 2,7,14).

Enter the numbers assigned to your symptoms: 1,4,10
```

3. **The system gives, predicted outputs:**



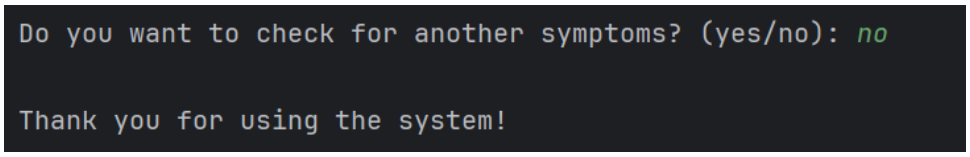
```
Predicted Results
1. Disease: Flu
   Description: A contagious respiratory illness

2. Disease: Bronchitis
   Description: Inflammation of bronchial tubes

3. Disease: Pneumonia
   Description: Infection that inflames air sacs in lungs


NOTE: THIS IS FOR EDUCATIONAL PURSPOSE ONLY.
Always consult your healthcare provider
```

4. **Now , it will ask you if you want to know disease for any other symptom,(yes/no):**



```
Do you want to check for another symptoms? (yes/no):
```

5. **If you type yes, Then the it will again give you lists of symptoms otherwise it will display 'thankyou message':**



```
Do you want to check for another symptoms? (yes/no): no

Thank you for using the system!
```

10. Testing Approach

1. Valid Input Test

- Entered correct symptom numbers like 2, 5, 8.
- The program correctly matched symptoms with diseases.
- The predicted diseases and their descriptions were displayed properly.

2. Multiple Symptoms Test

- Entered several symptoms that appear in more than one disease (e.g., cough, fever, fatigue).

- The program showed all diseases that shared those symptoms.
- The result list was accurate and complete.

3. Out-of-Range Input Test

- Entered values outside the available range (e.g., 50, 200).
- The program ignored those numbers and continued running without crashing.
- Only valid symptoms were processed, showing correct results.

11. Challenges Faced

1. **Indentation Issue:**
   The prediction list was placed outside the loop, causing only the last disease to be saved.
   Fix: Moved the append statement inside the loop.

2. **Input Conversion:**
   Mapping user numbers to list indices needed careful checks to avoid errors.

12. What I Learned

While building this project, I learned how helpful modular programming is. Splitting the code into three files made everything easier to follow. I also got better at working with dictionaries and loops, especially when collecting and sorting symptoms.

I understood how important good input handling is because even small mistakes can break the program. I also improved at debugging and thinking from the user's point of view. Overall, this project helped me feel more confident in writing clean and organised Python code

13. Future Improvements

- Rank diseases by number of matched symptoms
- Add severity levels
- Store disease in a JSON or CSV file (This would make updates easier and will keep the data separate from the code)

14. References

1. **World Health Organization (WHO)**
   https://www.who.int/
   Used for: Verified disease symptom associations
2. **Centers for Disease Control and Prevention (CDC)**
   https://www.cdc.gov/
   Used for: Disease descriptions and symptom lists
3. **Python for Beginners - YouTube Tutorials**

   Used for: Understanding modular programming concepts

4. **W3Schools Python Tutorial**

   https://www.w3schools.com/python/

   Used for: Syntax reference, function definitions, module imports