Introduction
This project is a simple command-line “Disease Finder” written in Python that demonstrates how to build a basic rule-based system for predicting a likely disease from user-entered symptoms. It uses a small, hard-coded dictionary of diseases and their associated symptoms, then compares those with what the user types to find which disease has the most overlapping symptoms. The program is intended purely for learning Python concepts such as dictionaries, functions, input handling, set operations, and basic decision logic, not for real medical diagnosis.

Knowledge base: diseases and symptoms
At the top of the file, a dictionary named disease_symptoms defines a tiny knowledge base:

Keys are disease names like "Common Cold", "Flu", "Migraine", "Food Poisoning", "Dengue", and "COVID-19".

Values are lists of symptom strings for each disease, for example:

"Common Cold": ["cough", "sore throat", "runny nose", "sneezing", "mild fever", "headache"]

"Flu": ["high fever", "chills", "body ache", "fatigue", "cough", "headache"]

This structure lets the rest of the program loop over known diseases and compare them against the user’s symptoms.

Helper function: cleaning symptoms
python
```
def clean_symptom(symptom: str) -> str:
    """Normalize symptom text."""
    return symptom.strip().lower()
```
strip() removes leading and trailing whitespace from a symptom string.

lower() converts the text to lowercase so that “Fever”, “fever”, and “ FEVER ” are treated the same.

Normalization like this avoids mismatches caused by extra spaces or capitalization.

Getting user symptoms from input
python
```
def get_user_symptoms():
    print("Enter your symptoms separated by commas.")
    print("Example: fever, cough, headache")
    raw = input("Symptoms: ")
    # split and clean
    return [clean_symptom(s) for s in raw.split(",") if s.strip()]
```
Step-by-step:

Prints instructions telling the user how to enter symptoms (comma-separated).

input("Symptoms: ") reads one line from the terminal as a raw string.

raw.split(",") breaks the input into a list using commas as separators.

The list comprehension:

if s.strip() filters out empty items (e.g., an extra comma).

clean_symptom(s) applies the cleaning function to each non-empty piece.

The function returns a clean list like ["fever", "cough", "headache"].

Scoring diseases against user symptoms
python
```
def score_diseases(user_symptoms):
    """
    For each disease, count how many symptoms match the user symptoms.
    Returns a dict: {disease: score}
    """
    scores = {}
    for disease, sym_list in disease_symptoms.items():
        normalized = [clean_symptom(s) for s in sym_list]
        # intersection size = score
        match_count = len(set(user_symptoms) & set(normalized))
        scores[disease] = match_count
    return scores
```
What this does:

Takes the cleaned user_symptoms list as input.

Creates an empty dictionary scores to hold results.

Loops over each disease and its symptom list in disease_symptoms.

For each disease:

Normalizes that disease's symptoms with clean_symptom (defensive, in case data changes).

Converts both user_symptoms and normalized to sets and uses & to get the intersection.

len(set(user_symptoms) & set(normalized)) is the number of common symptoms.

Stores that count in scores[disease].

Returns a dictionary like:

{"Common Cold": 2, "Flu": 3, "Migraine": 0, "Food Poisoning": 1, ...}

This is the core "matching" logic: more overlapping symptoms → higher score.

Predicting the most likely disease
python
```
def predict_disease(user_symptoms, min_matches=1):
    scores = score_diseases(user_symptoms)

    # find disease with maximum score
    best_disease = None
    best_score = 0
    for disease, score in scores.items():
        if score > best_score:
            best_score = score
            best_disease = disease

    # if no disease has enough matches, return None
```

```
    if best_score < min_matches:
        return None, scores
    return best_disease, scores
```
Logic:

Calls score_diseases to compute match scores for all diseases.

Initializes best_disease to None and best_score to 0.

Iterates over each (disease, score) pair:

If the current score is greater than best_score, updates both best_score and best_disease.

After the loop:

If best_score is less than min_matches, it means even the best disease doesn’t have enough matching symptoms.

In that case, returns (None, scores) to signal “no likely disease found.”

Otherwise, returns (best_disease, scores).

min_matches is a simple threshold to avoid suggesting a disease when there is almost no overlap.

Main function: tying everything together
python
```
def main():
    print("=== Simple Disease Finder (Educational Only) ===")
    user_symptoms = get_user_symptoms()

    if not user_symptoms:
        print("No symptoms entered. Exiting.")
        return

    disease, scores = predict_disease(user_symptoms, min_matches=1)

    print("\nYour symptoms:", ", ".join(user_symptoms))

    if disease is None:
        print("No likely disease found in the small knowledge base.")
    else:
        print(f"\nMost likely disease (from this small dataset): {disease}")
        print("Match score:", scores[disease])

    print("\nOther disease match scores:")
    for d, s in scores.items():
        print(f"  {d}: {s}")

    print("\nNote: This is a simple educational project and NOT a real medical tool.")
```
Flow:

Prints a title line to explain what the program is.

Calls get_user_symptoms() to read and clean the user’s input.

If the user provided no valid symptoms, prints a message and returns early.

Calls predict_disease with the user symptoms and a minimum of 1 match.

Prints the user’s symptoms for confirmation.

If disease is None, tells the user that no likely disease was found in this tiny dataset.

Otherwise, prints:

The “most likely” disease from this small knowledge base.

Its match score (how many symptoms matched).

Prints “Other disease match scores:” and loops through the scores dictionary to show the score for every disease.

Ends with a clear disclaimer that this is only an educational project and not a real medical tool.

Entry point check
python
```
if __name__ == "__main__":
    main()
```
This ensures main() runs only when the file is executed directly (python disease_finder.py).

If the file is imported as a module from another script, main() will not run automatically.

Conclusion
This disease_finder.py program demonstrates a basic rule-based disease prediction system using simple Python constructs. It stores a handful of diseases and their symptoms in a dictionary, normalizes user input, counts overlapping symptoms using set intersections, and selects the disease with the highest match score, printing full scores for transparency. While it is far too simple and limited to be used in real healthcare, it is a good starting point for understanding how symptom-based prediction works and for practicing Python concepts; from here, you can expand the knowledge base, add better text processing, adjust thresholds, or even wrap the logic in a Flask or GUI application.