VITYARTHI PROJECT

FOR INTRODUCTION TO PROBLEM SOLVING AND PROGRAMMING

NAME: ARYAN AWASTHI

REG NO: 25BCY20178

SUBJECT : INTRODUCTION TO PROBLEM SOLVING AND PROGRAMMING

SUBJECT CODE: CSE1021

PROJECT NAME: JAUNDICE INPECTION

SUBMITTED ON: 22 NOV. 2025

2. INTRODUCTION

The Jaundice Risk Predictor is a Python-based command-line application designed to assess the likelihood of jaundice based on user-reported symptoms. The system interacts with the user, collects responses for common jaundice symptoms, calculates risk levels, and provides immediate feedback. Additionally, the system generates a health report and visualizes the diagnosis using a pie chart.

This project demonstrates how simple logic, user input handling, file management, and data visualization can be combined to build a functional health support tool.

---

3. Problem Statement

Early identification of jaundice symptoms is crucial for preventing complications and seeking timely medical advice. However, many people may overlook or misjudge symptoms.

There is a need for a simple, accessible tool that can quickly analyze symptoms and give a preliminary risk assessment.

This project aims to build a lightweight CLI-based solution for symptom evaluation and risk prediction.

---

4. Functional Requirements

1. The system must accept the user's name.

2. The system must display a list of symptoms and allow the user to answer using y or n.

3. The system must calculate a risk score based on the number of symptoms.

4. The system must classify the risk level into five categories.

5. The system must display the score, selected symptoms, and risk status.

6. The system must save the report in a text file (report.txt).

7. The system must generate a pie chart visualization using matplotlib.

8. The system should handle invalid inputs.

---

5. Non-Functional Requirements

1. Usability: The system should be easy to use with simple CLI instructions.

2. Performance: Results should be displayed immediately after input.

3. Reliability: The program should handle incorrect entries gracefully.

4. Portability: Runs on any system with Python installed.

5. Maintainability: Code should be modular and easy to update (OOP structure).

6. Scalability: Future symptoms or modules can be easily added.

---

6. System Architecture

The system architecture is based on a modular OOP structure:

main.py: Entry point of the application.

detector.py: Core logic for symptom checking, scoring, visualization, and file handling.

External Library: matplotlib for data visualization.

Text storage: report.txt for saving results.

---

7. Design Diagrams

(You will draw these manually or digitally and add them as images.)

7.1 Use Case Diagram

Actors:

User

Use Cases:

Enter name

Select symptoms

View result

Save report

View pie chart

7.2 Workflow Diagram

Steps:

1. Start application

2. Enter name

3. Answer symptom questions

4. Calculate risk

5. Show result

6. Save report

7. Display pie chart

8. End

1.

7.3 Sequence Diagram

Components:

User

JaundiceDetector class

File system

Matplotlib

Sequence:

1. User inputs name ⟶ System stores

2. User answers symptoms ⟶ System updates score

3. System calculates risk

4. System saves report

5. System generates chart

7.4 Class/Component Diagram

Classes:

JaundiceDetector

Attributes: symptoms, score, selected, risk_levels

Methods: get_user_name, collect_symptoms, calculate_risk, save_report, show_pie_chart, run

7.5 ER Diagram

Since the project uses simple text file storage, ER Diagram is optional / not applicable

(You can still add a basic diagram showing User ⟶ Report File).

---

8. Design Decisions & Rationale

1. CLI Interface chosen because it is lightweight and easy to implement.

2. Python OOP structure used to keep code modular and maintainable.

3. matplotlib selected for visualization due to its wide documentation and simplicity.

4. Text file storage used instead of database because this project does not require complex storage.

5. Symptom-based scoring system was chosen as it is simple yet effective for preliminary health assessment.

---

9. Implementation Details

The project is implemented in Python 3.

The main logic resides in detector.py inside the JaundiceDetector class.

The application starts execution through main.py.

The user interacts through terminal inputs.

A score variable increments for every “yes” response.

Based on the score, the risk status is selected from predefined levels.

Final results are printed on-screen, saved to a file, and displayed graphically using a pie chart.

---

10. Screenshots / Results

(You will add these yourself after running the program.)

Suggested screenshots:

CLI symptom input

Final report output

Saved report.txt file

Generated pie chart

---

11. Testing Approach

Testing was carried out using:

1. Input Validation Testing

Provided valid (y/n) and invalid inputs to ensure system handles errors.

2. Functional Testing

Verified each symptom increases score correctly.

Verified risk classification for multiple combinations.

3. File Handling Testing

Ensured report.txt generates correctly in append mode.

4. Visualization Testing

Verified pie chart displays correct percentage distribution.

5. Edge Case Testing

Score = 0

Score = maximum symptoms

No symptoms selected

---

12. Challenges Faced

Handling invalid inputs smoothly inside loops.

Managing pie chart display for environments without GUI backend.

Ensuring file creation permissions work on all operating systems.

Keeping the program beginner-friendly while maintaining code structure.

---

13. Learnings & Key Takeaways

Learned how to apply OOP concepts in real-world mini projects.

Gained experience in user input validation and error handling.

Understood how to generate files programmatically in Python.

Enhanced skills in data visualization using matplotlib.

Realized the importance of modular programming for scalability.

---

14. Future Enhancements

Add GUI using Tkinter or PyQt.

Integrate a database for storing multiple user reports.

Add more symptoms and detailed medical suggestions.

Provide PDF report generation.

Add machine learning-based prediction models.

Convert CLI project into a full web application.

---

15. References

1. Python Documentation – https://docs.python.org

2. Matplotlib Documentation – https://matplotlib.org

3. Health information on jaundice symptoms – General medical references

4. Class-based Python code examples – Various online resources