VIT Bhopal University

Introduction to Problem Solving and Programming

2025-2026

Project
Heart Rate Analyzing Tool

Submitted By
Siddharth Meena
(25BCE11269)

Problem Statement:

Program a heart rate analysis tool on   the basis of readings

Introduction

This project represents the development of a simple Heart-Rate Analysis Tool using the Python programming language. The essence of this project lies in making it easier for a user to understand their heart-rate readings in simple and comprehensible terms. While it is possible for many people to measure their heart rate by using a watch, a fitness band, or even by counting their pulse, they often have no idea if those readings are normal, high, or low. This project tries to solve that problem with a small program doing the basic analysis for the user.

This program enables the user to input a few heart-rate readings along with the time of each reading. It then calculates the minimum, maximum, and average heart rate after taking the readings and checks whether the average value lies in a safe range. It prints out a simple text report and also displays a line graph of heart rate versus time using matplotlib. This project is a good example in applying programming to a real-life situation, which involves taking user input, handling errors, working with lists, performing basic calculations, and creating a simple graph.

Functional Requirements(FR):

FR1 – Input of Heart Rate Readings

The system shall allow the user to input heart rate readings and the time of each reading. The system shall verify that the heart rate value is a positive number and will ask for input again if the value is invalid.

FR2 – Storage of Readings

The system shall store the times and heart-rate values entered by the user in structure data structures (lists) for later analysis.

FR3 – Calculation of Statistics

This tool will calculate the minimum ,maximum and average heart rate from readings entered by the user.

FR4 - Safety Range Check

Tool will Check whether the average Lies in normal range or not. Will Give the signals as per average.

FR5 - Report Generation and Graphical Representation

The system generates and displays a simple line graph of heart rate versus time based on the entered readings.

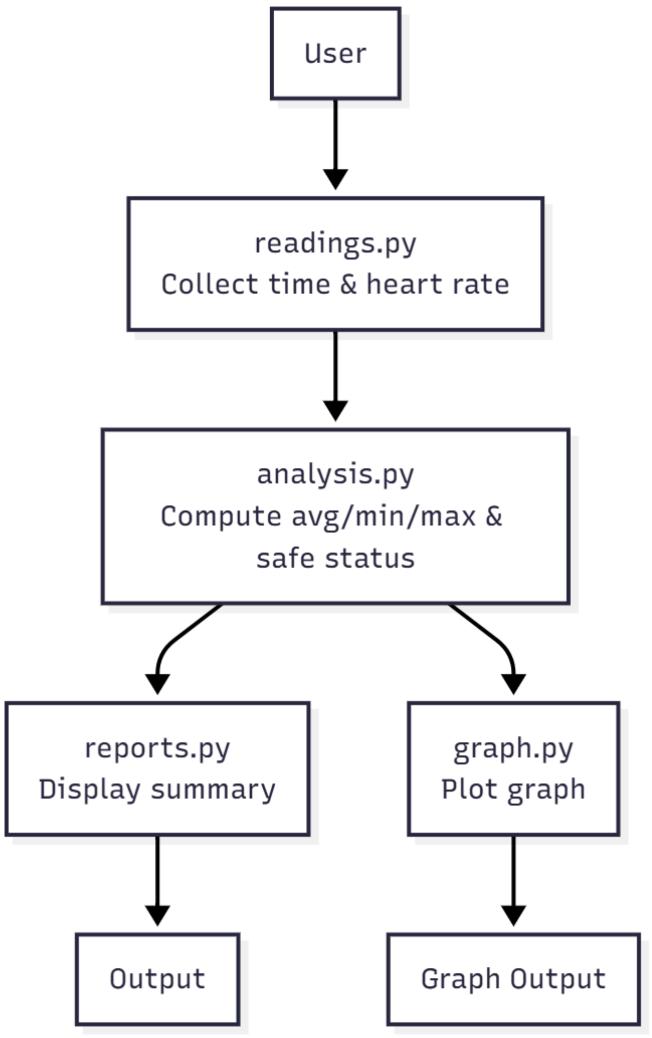
Non Functional Requirement:

1. The tool should be easy to use and understandable for new user
2. The tool should have proper error handling like only positive heart rate.
3. The code should be modular and readable, so that it can be modify in the future for betterment
4. The Tool should be portable and should run on various OS.

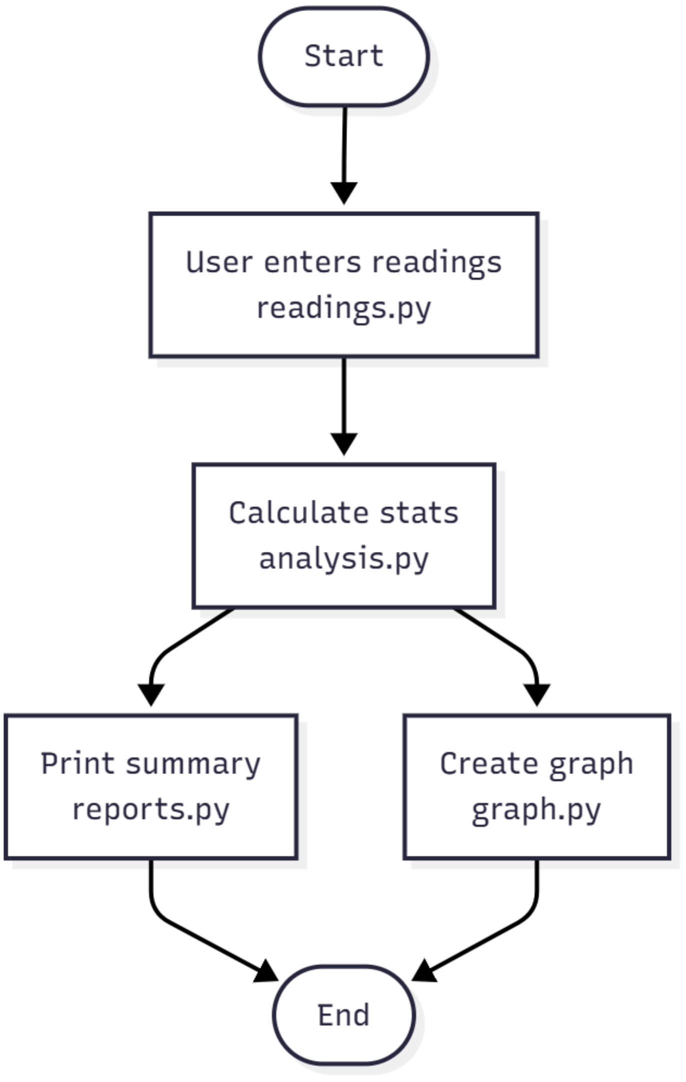
System Architecture

The system follows a modular structure in following way:

1. readings.py takes user input.
2. analysis.py processes heart-rate values and computes summary statistics.
3. reports.py formats and displays analysis results.
4. graph.py generates a graphical visualization of the readings.
5. main.py acts as the organized that connects all modules and manages the workflow.

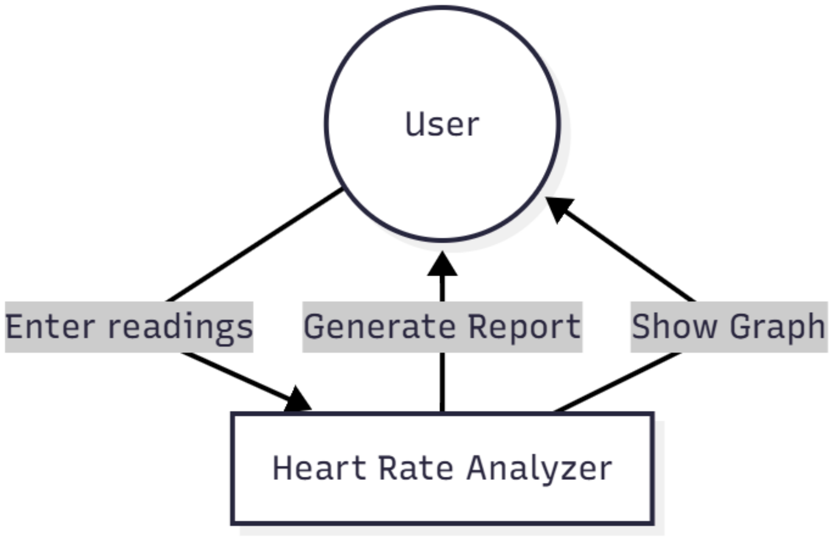


Use Case Diagram



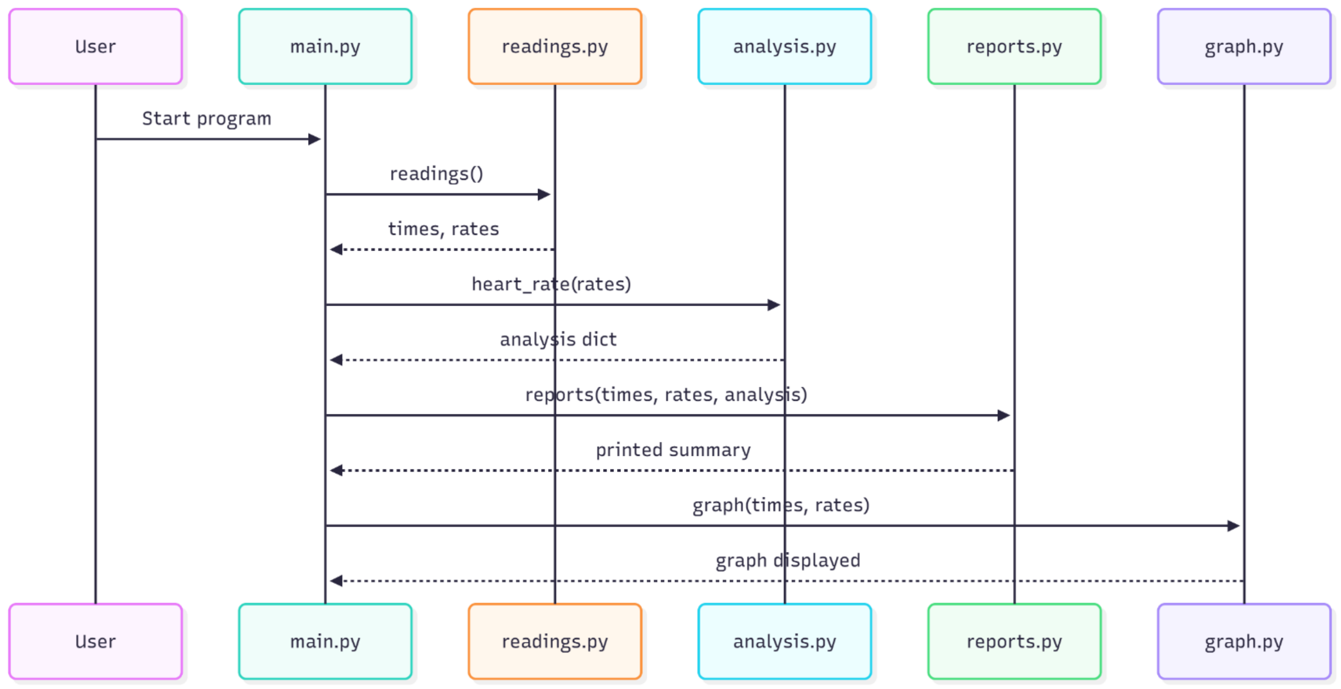
Website Used:Mermaidchart.com

Work Flow Diagram



Website Used:Mermaidchart.com

Sequence Diagram



Website Used:Mermaidchart.com

Design Decisions & Rationale:

1. Modular structure chosen to meet project requirements and ensure clean separation of concerns.
2. Manual input chosen simplify user interaction.
3. Matplotlib chosen for plotting due to reliability and ease of use.
4. Safe range (60–100 bpm) set using general heart-rate guidelines.

Implementation Details:

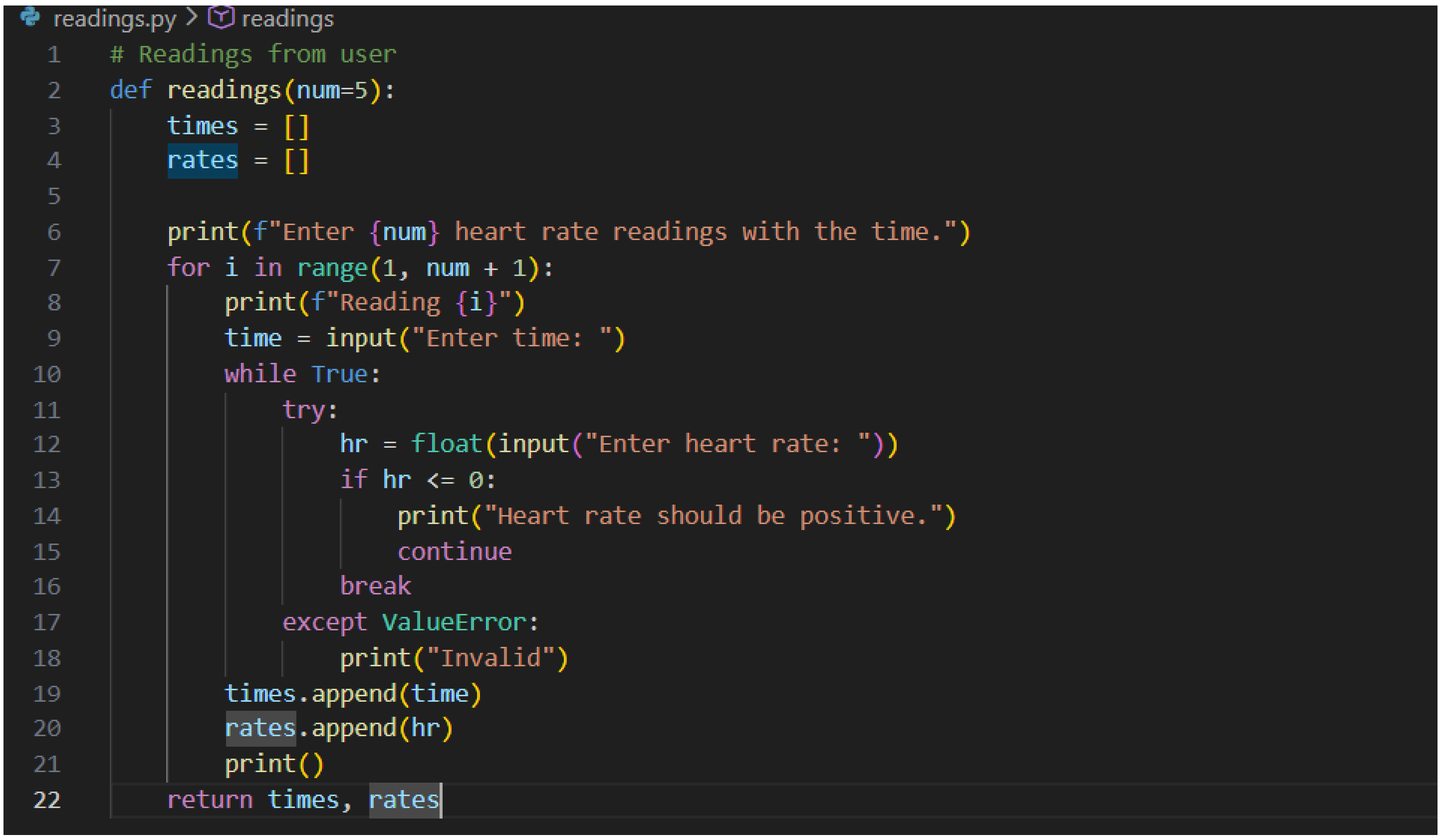
1. readings.py – handles user input.
2. analysis.py – calculates stats.
3. reports.py – prints analysis summary.
4. graph.py – visualizes heart rate vs time.
5. main.py – organize all files.

Dependencies: Python, matplotlib.

Execution: python main.py

Screenshots Of Program:

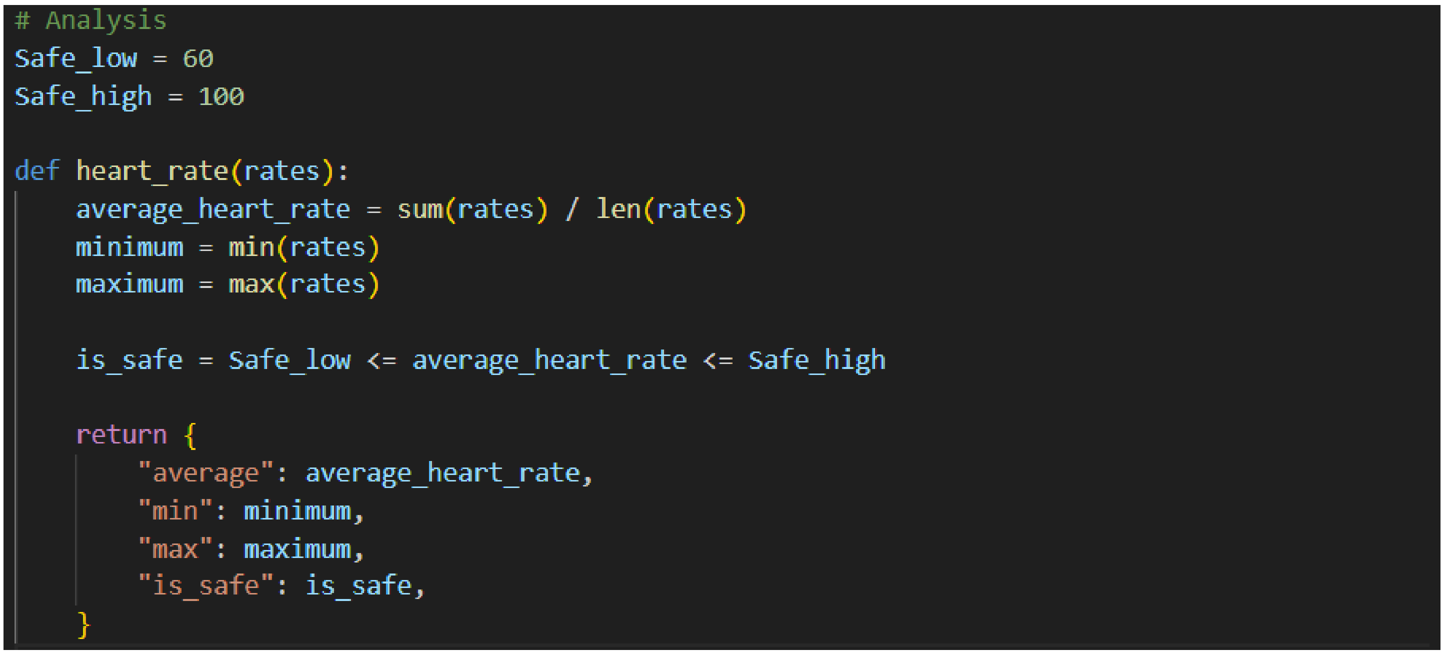
1.Readings Input Source Code



readings.py > readings

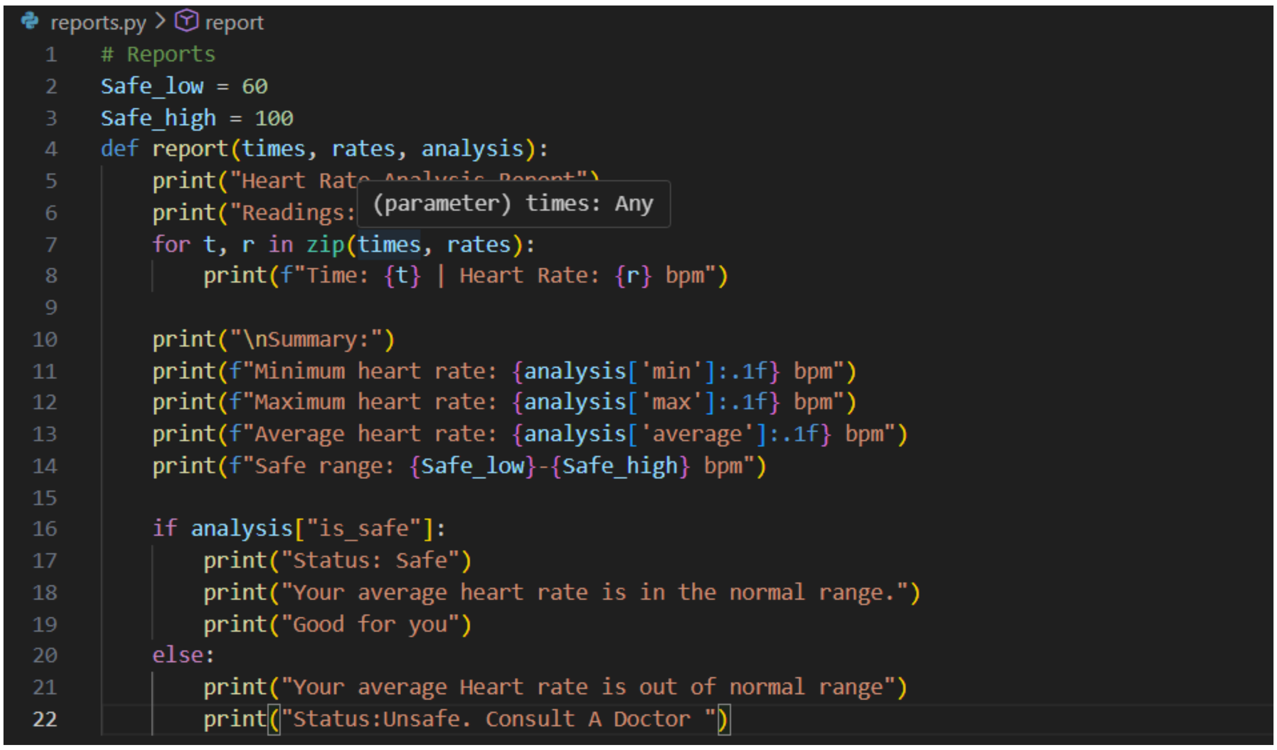
```python
# Readings from user
def readings(num=5):
    times = []
    rates = []

    print(f"Enter {num} heart rate readings with the time.")
    for i in range(1, num + 1):
        print(f"Reading {i}")
        time = input("Enter time: ")
        while True:
            try:
                hr = float(input("Enter heart rate: "))
                if hr <= 0:
                    print("Heart rate should be positive.")
                    continue
                break
            except ValueError:
                print("Invalid")
        times.append(time)
        rates.append(hr)
        print()
    return times, rates
```

2.Analysis Of Data Source Code



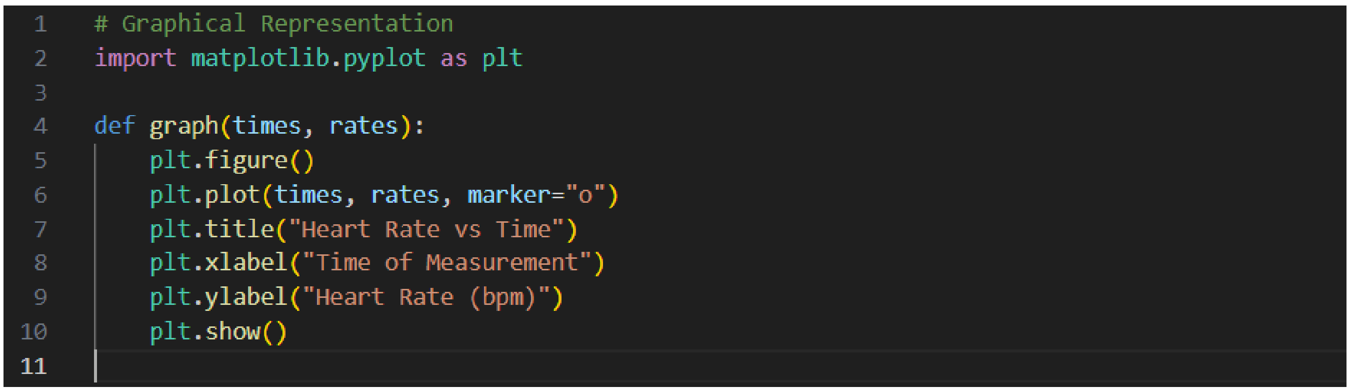
```python
# Analysis
Safe_low = 60
Safe_high = 100

def heart_rate(rates):
    average_heart_rate = sum(rates) / len(rates)
    minimum = min(rates)
    maximum = max(rates)

    is_safe = Safe_low <= average_heart_rate <= Safe_high

    return {
        "average": average_heart_rate,
        "min": minimum,
        "max": maximum,
        "is_safe": is_safe,
    }
```

3.Reports Printing Source Code



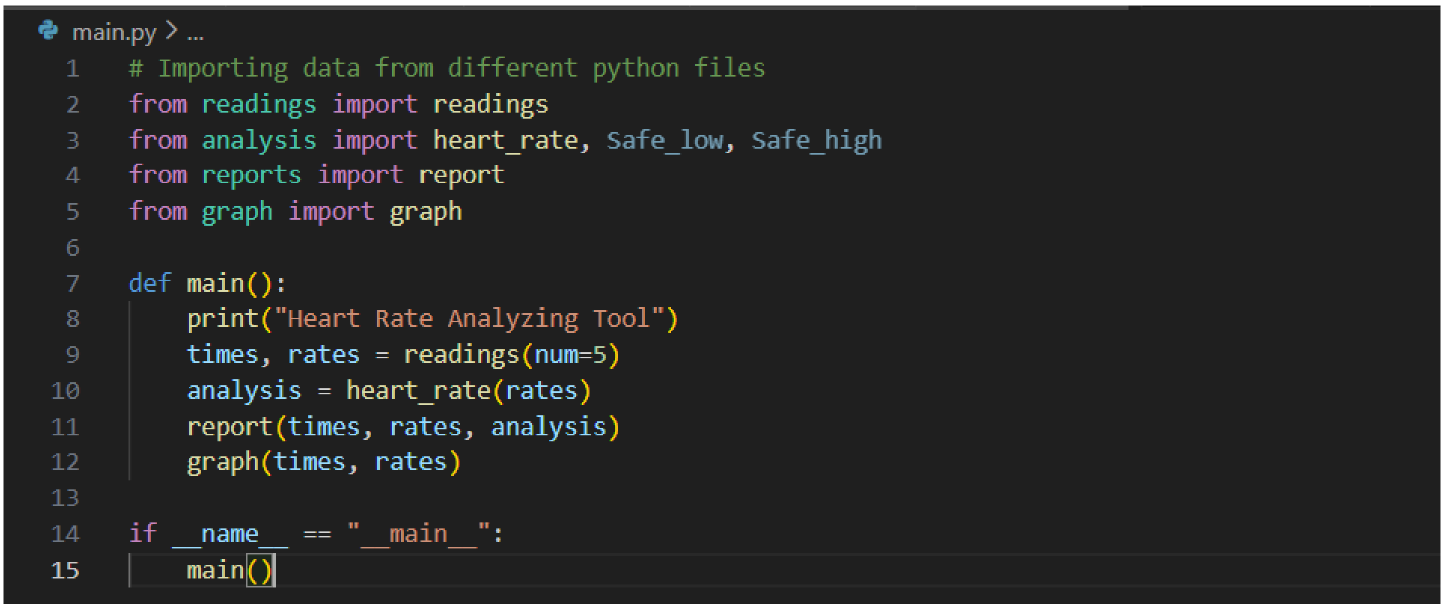
```python
# Reports
Safe_low = 60
Safe_high = 100
def report(times, rates, analysis):
    print("Heart Rate Analysis Report")
    print("Readings: (parameter) times: Any
    for t, r in zip(times, rates):
        print(f"Time: {t} | Heart Rate: {r} bpm")

    print("\nSummary:")
    print(f"Minimum heart rate: {analysis['min']:.1f} bpm")
    print(f"Maximum heart rate: {analysis['max']:.1f} bpm")
    print(f"Average heart rate: {analysis['average']:.1f} bpm")
    print(f"Safe range: {Safe_low}-{Safe_high} bpm")

    if analysis["is_safe"]:
        print("Status: Safe")
        print("Your average heart rate is in the normal range.")
        print("Good for you")
    else:
        print("Your average Heart rate is out of normal range")
        print("Status:Unsafe. Consult A Doctor ")
```

4.Graphical Representation:



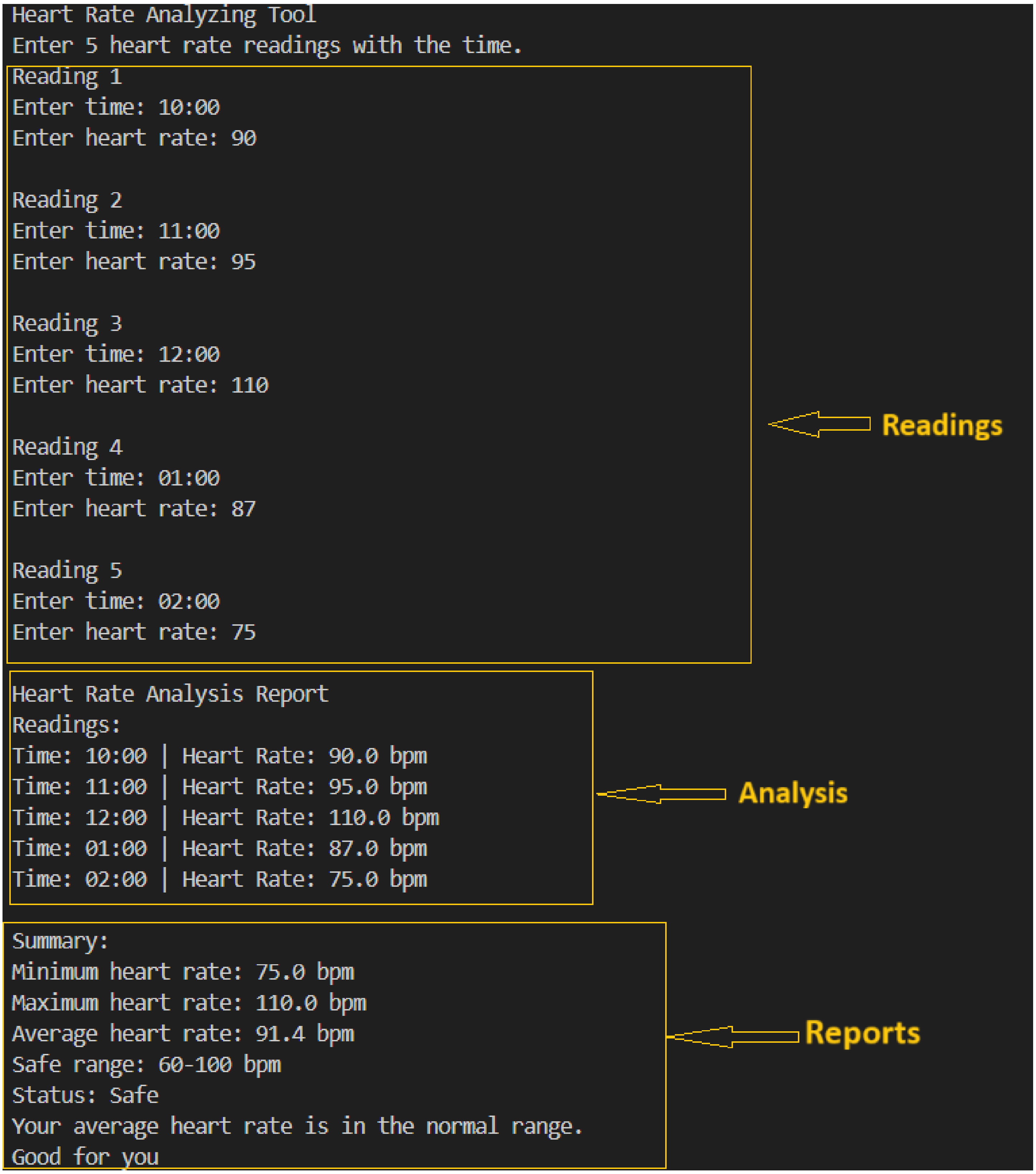
```python
# Graphical Representation
import matplotlib.pyplot as plt

def graph(times, rates):
    plt.figure()
    plt.plot(times, rates, marker="o")
    plt.title("Heart Rate vs Time")
    plt.xlabel("Time of Measurement")
    plt.ylabel("Heart Rate (bpm)")
    plt.show()

```

5.Main Organizer



```python
# Importing data from different python files
from readings import readings
from analysis import heart_rate, Safe_low, Safe_high
from reports import report
from graph import graph

def main():
    print("Heart Rate Analyzing Tool")
    times, rates = readings(num=5)
    analysis = heart_rate(rates)
    report(times, rates, analysis)
    graph(times, rates)

if __name__ == "__main__":
    main()
```

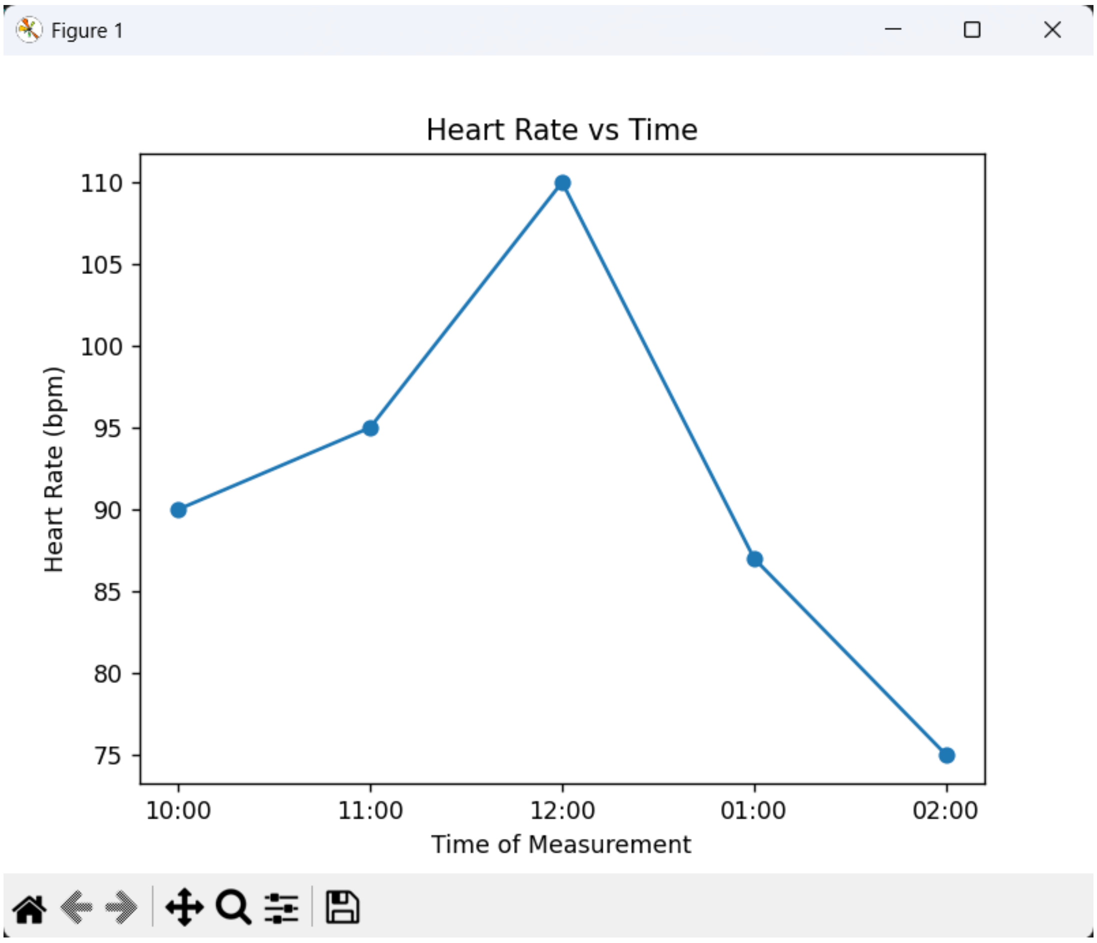
Command Line Terminal View:



```
Heart Rate Analyzing Tool
Enter 5 heart rate readings with the time.
Reading 1
Enter time: 10:00
Enter heart rate: 90

Reading 2
Enter time: 11:00
Enter heart rate: 95

Reading 3
Enter time: 12:00
Enter heart rate: 110

Reading 4
Enter time: 01:00
Enter heart rate: 87

Reading 5
Enter time: 02:00
Enter heart rate: 75

Heart Rate Analysis Report
Readings:
Time: 10:00 | Heart Rate: 90.0 bpm
Time: 11:00 | Heart Rate: 95.0 bpm
Time: 12:00 | Heart Rate: 110.0 bpm
Time: 01:00 | Heart Rate: 87.0 bpm
Time: 02:00 | Heart Rate: 75.0 bpm

Summary:
Minimum heart rate: 75.0 bpm
Maximum heart rate: 110.0 bpm
Average heart rate: 91.4 bpm
Safe range: 60-100 bpm
Status: Safe
Your average heart rate is in the normal range.
Good for you
```

Graphical Representation:

As per Data Entered On Command Line



Testing Approach

The project was tested with manual testing. Different heart-rate values, including normal, high, low, and random, were entered to check that the program calculates the average, minimum, maximum, and safe/unsafe status accurately. Invalid inputs, such as letters, zero, and negative numbers, were also provided to verify that the program displays the correct error messages. The plotted graph was visually inspected to ensure all readings are shown correctly on the charts.

Challenges Faced

1. It took me quite a bit of thinking to figure out how to split the program into five separate modules and still keep the imports working correctly.
2. The implementation of the invalid input (non-numeric and non-positive values) handling was quite challenging, as it required additional loops and validation checks.
3. I had to figure out how to make matplotlib display properly without interruption.
4. I was unfamiliar with Mermaid diagrams before, now I know how to use it.

Learnings & Key Takeaways

1. I Learned how to break down a program into different modules and how to separate the logic into nice, readable Python files that can be easily maintained.
2. I Got the knowledge to create basic system architecture and UML diagrams with the help of Mermaid.
3. I got the knowledge of how resting heart rates are measured and how data visualization is used to examine the trends.
4. Got the knowledge of how writing documents and drawing diagrams helps communication in a project.

Future Enhancements

There is a possibility of enhancing this project by adding the support of CSV or file-based input such that users can analyze a bulk of data. An elementary GUI with Tkinter of the tool. The allowable heart-rate interval can also be varied depending on the user's age or health condition. Moreover, functionalities like saving the chart automatically, exporting the report as a PDF, and keeping the previous readings for extended tracking can be utilized to the fullest and made more convenient by this application.

Reference:

- Python Documentation: https://docs.python.org
- Matplotlib Documentation: https://matplotlib.org
- Mermaid Diagram Documentation: https://mermaid.js.org
- Healthline: “Average Resting Heart Rate”
- Course Specification Document (VITyarthi Project Instructions)