* **EMP ID: T1929**

**Set-3**

* **EMP NAME: SRAVYA SONTYANA**

**1. Code Generation & Debugging**

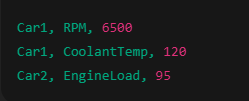
**Python**

**Scenario:**  
AI (ChatGPT/Copilot) generated a multi-file project:

* **diagnostic.py** → Defines a Diagnostic class (id, type, value). Types = "RPM", "EngineLoad", "CoolantTemp".
* **car.py** → Holds diagnostics, computes performance score.
* **garage\_monitor.py** → Manages multiple cars, checks for driver abuse & engine issues.
* **main.py** → Loads diagnostic data from CSV and prints car status.

**Program Features:**

* Load car diagnostics from CSV. Example:



* Compute performance score:
* A black background with white text

  AI-generated content may be incorrect.
* Print alerts:
  + score < 40 → **“Severe Engine Stress”**
  + Missing diagnostic → **“Sensor Failure Detected”**

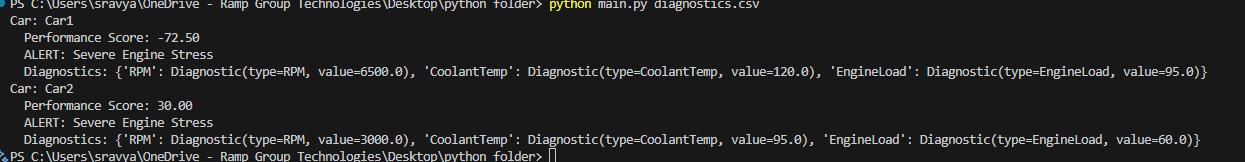
**Prompt:**Act as an expert Python software developer specializing in automotive systems. Generate a clean, modular, multi-file console-based Python project with the following structure:  
· diagnostic.py → Defines a Diagnostic class (id, type, value). Types = "RPM", "EngineLoad", "CoolantTemp".

· car.py → Holds diagnostics, computes performance score.

· garage\_monitor.py → Manages multiple cars, checks for driver abuse & engine issues.

· main.py → Loads diagnostic data from CSV and prints car status.  
Requirements:  
->CSV Input:  
\*Format: (car\_id,type,value), example:  
-> Car1,RPM,6500  
-> Car1,CoolantTemp,120  
-> Car2,EngineLoad,95  
  
\*Handle duplicate diagnostics for a car by keeping the latest value for a given type.  
\*Validate input: reject malformed lines (e.g., missing fields, invalid types, non-numeric values).

->compute the Performance Score:  
*Formula: score = 100 - (rpm/100 + engineLoad*0.5 + (coolantTemp-90)\*2)  
\*If any diagnostic (RPM, EngineLoad, CoolantTemp) is missing, mark the score as invalid.  
Print alerts:  
score < 40 → “Severe Engine Stress”  
Missing diagnostic → “Sensor Failure Detected”  
**output**

  
  
Car: Car1

Performance Score: -72.50

ALERT: Severe Engine Stress

Diagnostics: {'RPM': Diagnostic(type=RPM, value=6500.0), 'CoolantTemp': Diagnostic(type=CoolantTemp, value=120.0), 'EngineLoad': Diagnostic(type=EngineLoad, value=95.0)}

Car: Car2

Performance Score: 30.00

ALERT: Severe Engine Stress

Diagnostics: {'RPM': Diagnostic(type=RPM, value=3000.0), 'CoolantTemp': Diagnostic(type=CoolantTemp, value=95.0), 'EngineLoad': Diagnostic(type=EngineLoad, value=60.0)}

**2. Code Review & Refactoring**

Messy AI-generated code (all in **main.py**):

**rpm = [6500, 3000]**

**load = [95, 40]**

**temp = [120, 85]**

**for i in range(2):**

**score = 100 - (rpm[i] / 100 + load[i] \* 0.5 + (temp[i] - 90) \* 2)**

**print(f"Car {i} score={score}")**

**if score < 40:**

**print("ALERT")**

**Issues Identified:**

* No OOP (just lists)
* Hardcoding of values
* Magic numbers (100, 0.5, 90) scattered
* No encapsulation or constructors
* No CSV input or validation
* No exception handling

**Refactor into OOP design:**

* **class Diagnostic: …**
* **class Car: …**
* **class GarageMonitor: …**
* Use encapsulation, constructors, constants for thresholds.
* Add CSV parsing using Python’s csv module.
* Implement error handling for **empty/malformed CSV**.

**Prompt:**

The current code is **procedural and messy**, lacking structure and best practices.  
 Act as a Python software developer performing code review and refactoring on messy AI-generated code and **analyze, review, and refactor** it into a clean, maintainable, **object-oriented design**.

#file:reviewcode.py

and also inlcude and refine the code using the below requirements

Refactor into OOP design:

class Diagnostic: …

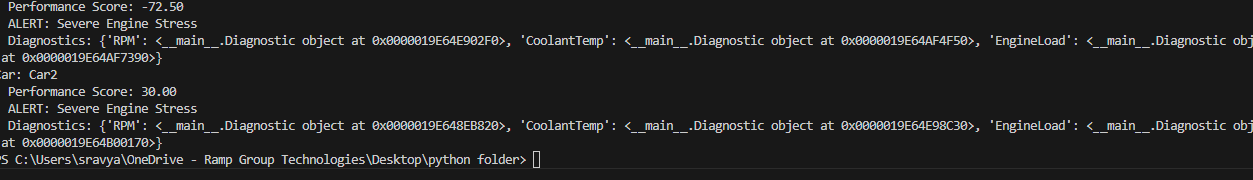
class Car: …

class GarageMonitor: …

Use encapsulation, constructors, constants for thresholds.

Add CSV parsing using Python’s csv module.

Implement error handling for empty/malformed CSV.

Output:  
  
Car: Car1

Performance Score: -72.50

ALERT: Severe Engine Stress

Diagnostics: {'RPM': <\_\_main\_\_.Diagnostic object at 0x0000019E64E902F0>, 'CoolantTemp': <\_\_main\_\_.Diagnostic object at 0x0000019E64AF4F50>, 'EngineLoad': <\_\_main\_\_.Diagnostic object at 0x0000019E64AF7390>}

Car: Car2

Performance Score: 30.00

ALERT: Severe Engine Stress

Diagnostics: {'RPM': <\_\_main\_\_.Diagnostic object at 0x0000019E648EB820>, 'CoolantTemp': <\_\_main\_\_.Diagnostic object at 0x0000019E64E98C30>, 'EngineLoad': <\_\_main\_\_.Diagnostic object at 0x0000019E64B00170>}

**3. Unit & Integration Testing**

Use **unittest** (built-in) or **pytest**.

**Tests to implement:**

* Car with rpm=6500, load=95, temp=120 → score < 40 → **Severe Engine Stress**.
* Car missing CoolantTemp → **Sensor Failure Detected**.
* Garage with cars [score=70, score=30] → average = 50.
* Boundary: exactly score=40 → no alert.
* Empty CSV → must raise **Exception**.

Prompt:  
Act as a Python software developer specializing in testing and quality assurance. Write unit tests and integration tests for the Automotive Diagnostics project using unittest (built-in)

Testing Requirements

Unit Tests (Car & Diagnostics)

Car with rpm=6500, load=95, temp=120 → score < 40 → Severe Engine Stress.

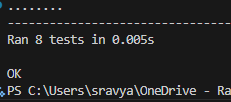
Car missing CoolantTemp → Sensor Failure Detected.

Garage with cars [score=70, score=30] → average = 50.

Boundary: exactly score=40 → no alert.

Empty CSV → must raise Exception.

Output:



Ran 8 tests in 0.004s

OK

PS C:\Users\sravya\OneDrive - Ramp Group Technologies\Desktop\python folder> python -m unittest test\_diagnostics.py

**4. Documentation**

Deliverables: **README.md**

Sections:

* **Introduction**
* **Explanation of classes/functions**
* **Sample Input/Output**

**Prompt:**

Act as a technical writer and Python software developer. Write clear and beginner-friendly documentation for the Automotive Diagnostics project in a README.md file.

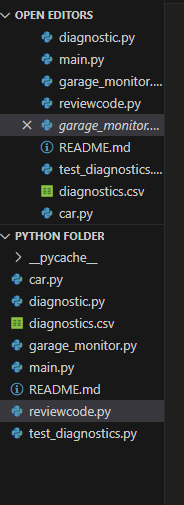
Create a README.md with the following sections:

Introduction

* Briefly explain what the Automotive Diagnostics project does.
* Mention its purpose (teaching modular Python/OOP with an automotive case study).
* Highlight key features (CSV parsing, performance scoring, alerts, error handling).

Explanation of Classes & Functions

* Diagnostic → stores diagnostic data (id, type, value).
* Car → holds diagnostics, computes performance score.
* GarageMonitor → manages multiple cars, checks driver abuse & engine issues.
* main.py → entry point: loads CSV, processes cars, prints tabular output.
* Mention important functions/methods and their roles



**5. Concurrency**

In Python, use:

* **threading module** → simulate real-time diagnostic updates (RPM, temperature, load).
* **Lock (from threading)** → to prevent race conditions when aggregating scores in GarageMonitor.
* Compare **single-thread vs multi-thread** execution performance.

**Prompt**

Act as a Python software developer specializing in concurrency and performance optimization. Extend the Automotive Diagnostics project to support real-time diagnostic updates using Python’s threading module.

* threading module → simulate real-time diagnostic updates (RPM, temperature, load).
* Lock (from threading) → to prevent race conditions when aggregating scores in GarageMonitor.
* Compare single-thread vs multi-thread execution performance.

Output:

unning single-threaded simulation...

Single-threaded time: 0.0012 seconds

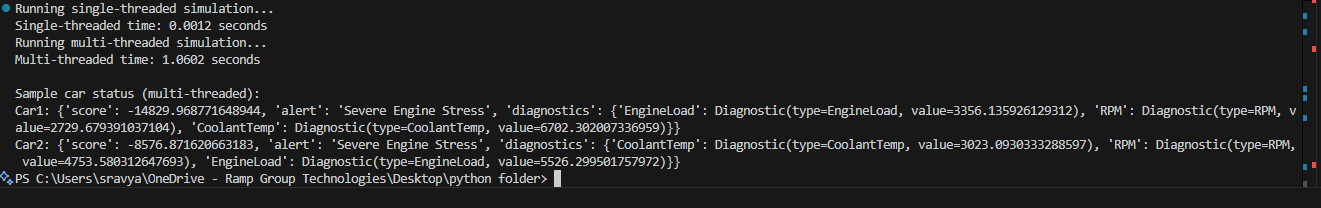
Running multi-threaded simulation...

Multi-threaded time: 1.0602 seconds

Sample car status (multi-threaded):

Car1: {'score': -14829.968771648944, 'alert': 'Severe Engine Stress', 'diagnostics': {'EngineLoad': Diagnostic(type=EngineLoad, value=3356.135926129312), 'RPM': Diagnostic(type=RPM, value=2729.679391037104), 'CoolantTemp': Diagnostic(type=CoolantTemp, value=6702.302007336959)}}

Car2: {'score': -8576.871620663183, 'alert': 'Severe Engine Stress', 'diagnostics': {'CoolantTemp': Diagnostic(type=CoolantTemp, value=3023.0930333288597), 'RPM': Diagnostic(type=RPM, value=4753.580312647693), 'EngineLoad': Diagnostic(type=EngineLoad, value=5526.299501757972)}}



**Java**