MiA robotics

AI TEAM TRAINING'26

TASK 1

Race Theme

Welcome to the engineering pit of the world-famous *M.I.A. F1 Racing Team*! You're the newest artificial intelligence engineers on the team, and your skills are needed to give our driver the winning edge. Today, you'll be working on three critical components of our new car, the *HMS-26*.

Task 1.1: The Steering Wheel Digital Gear Indicator

The Scenario:

Every F1 car's steering wheel is a complex piece of technology. One of the most critical displays for the driver is the gear indicator. A clear, instant readout of the current gear is essential for optimizing performance during corners and straights. Your first task as a junior race engineer is to program the logic for this digital display.

Your Objective:

Write a Python script that takes a gear number (an integer from 0 to 8, where 0 represents Neutral) and prints a visual representation of it onto the console using a simulated 7-segment display.

Technical Requirements:

1. Create a function, display_gear(gear_number).
2. Inside the function, use a 2D list *(5*4)* (a list of lists containing characters like '#' and ' ') to represent the display grid.
3. Based on the gear_number input, modify the grid to form the shape of that number.
4. The function should then print the grid to the console in a readable format.
5. Your main script should prompt the user to "Enter Gear (0-8):", call your function, and show the result.

Example: **display_gear(3):**

Output:

```
####
   #
####
   #
####
```

Bonus Section (for Experienced Drivers):

- **Input Validation:** Modify your script to handle invalid inputs. What should happen if the user enters 9 or a letter like 'R'?
- **Gear Shift Animation:** Create a function animate_shift(from_gear, to_gear) that first displays from_gear, pauses for a fraction of a second, clears the screen, and then displays to_gear to simulate a real gear shift.

Task 1.2: Secure Race Radio Transmission Protocol

The Scenario:

You aced the gear display! Now for a bigger challenge. The race strategist needs to send a sequence of commands to the driver, like "Push now", "Box, box", or "Check temperatures". These commands are sent as a single packet of data over the radio.

We need a robust way to **encode** a list of command strings into a *single string* for transmission, and then **decode** it back to the original list of commands on the driver's end, flawlessly every time.

Your Objective:

Implement a Codec class with two methods:

1. **encode(self, list_of_commands):** Takes a list of strings and returns a single, encoded string.
2. **decode(self, encoded_string):** Takes the single, encoded string and returns the original list of strings.

The decode of an encode must always result in the original list.

Technical Requirements:

1. Implement the encode and decode
2. Your encoding method must work for any possible string, including empty strings or strings containing numbers and special characters.
3. Your script should demonstrate that your code works by:
    - Creating a list of sample F1 commands.
    - Encoding the list and printing the single string.
    - Decoding that string and printing the resulting list to match the original.

Example

Input: `["Push","Box,box","Push","Overtake"]`

Output: `["Push","Box,box","Push","Overtake"]`

Task 1.3: The Final Race – Verstappen vs Mostafa

About

In the roaring circuits of Silverstone, Max Verstappen and Hassan Mostafa face off in a thrilling Formula 1 showdown. Both drivers push their machines to the limit, using clever tactics, defensive maneuvers, and sheer speed to outpace the other. Tire conditions and fuel levels become critical as the laps go by. Who will cross the finish line first?

Requirements

Design a **Python** program to simulate a head-to-head race between Max Verstappen and Hassan Mostafa using Object-Oriented Programming (OOP) principles.

Maximize the use of OOP principles as possible (Abstraction, Inheritance, Polymorphism and Encapsulation).

Race Mechanics:

- The race alternates between Max and Hassan, each taking a turn per round.
- On each turn:
  - The driver selects a move
  - The opponent may respond with an offensive move or a defensive maneuver.
  - Print after each turn:
    - Who took the move?
    - What tactic was used?
    - Remaining tire health and fuel of both racers.
- The race ends when: A driver's tire health drops to zero or below.
- The racer is not allowed to use defensive mechanism if the fuel will be less than 0
- Print the winner and final race stats.

Attribute	Description	Initial Value
Tire health	Represents how worn the tires are (0–100).	100
Fuel	Represents remaining fuel in the car (0–500).	500

Verstappen's Offensive Moves

Move Name	Fuel Cost	Tire Impact on opponent	Uses	Description
DRS Boost	45	12	∞	Drag Reduction System, is a mechanism that allows drivers to temporarily increase their straight-line speed
Red Bull Surge	80	20	∞	Aggressive acceleration, high tire wear.
Precision Turn	30	8	∞	Tactical turn to gain time with minimal fuel.

Verstappen's Defensive Tactics

Defense Name	Fuel Cost	Damage Reduction	Uses	Description
Brake Late	25	30% of opponent's damage	∞	Uses ultra-late braking to reduce attack impact. Common but risky.
ERS Deployment	40	50% of opponent's damage	3	Deploys electric recovery system defensively to absorb incoming pressure and recover next turn.

Mostafa's Offensive Moves

Move Name	Fuel Cost	Tire Impact on opponent	Uses	Description
Turbo Start	50	10	∞	Early burst of speed.
Mercedes Charge	90	22	∞	Full-throttle attack.
Corner Mastery	25	7	∞	Skilled turning for efficiency.

Mostafa's Defensive Tactics

Defense Name	Fuel Cost	Damage Reduction	Uses	Description
Slipstream Cut	20	40% of opponent's damage	∞	Cuts into the airflow behind the leading car to reduce their advantage and limit damage.
Agressive Block	35	100% (blocks the entire move)	2	Swerves defensively to completely block a single incoming move. Can only be used once due to risk.

Bonus

In a futuristic F1 setup, Verstappen and Mostafa now rely on voice commands to trigger their racing maneuvers. As part of your simulation, integrate a Speech-to-Text (STT) system using the Groq API to allow players to control the drivers via spoken instructions.

Requirements

1. **Use a Speech-to-Text (STT) model** via **Groq API** or any other method to transcribe voice commands into text.
2. Based on the transcribed command, trigger the corresponding move or defense for a driver.
3. If the transcription doesn't match any known move/defense, prompt the user to try again.
4. You can implement this for just one driver (e.g., Hassan)

Check Groq documentation **Here**

Submission Guidelines

- You will submit one zip file including your proposed solution to each one of the tasks in its own environment (directory/python files).
- Name each file with its task number.

- For illustration questions, you could either write a md file, or any readable text format with the task number.
- You can submit your code multiple times, only the last submission will be taken into consideration.
- The Task's deadline is 3/8 11:59 PM.
- It's highly encouraged to do the bonus tasks
- Including functions, comments, and making your code clean, well-documented and readable will give bonus points.
- We are here to help!!

It is highly discouraged to cheat, the goal of the process is learning, so keep an eye for that.