Task: Build a Command-Line Wordle Game in Python

1. Introduction

Welcome! Your task is to build a fully functional, text-based version of the popular word-guessing game, Wordle. This project will challenge you to apply fundamental Python concepts to create an interactive and engaging command-line application. The goal is to focus on clean, efficient, and well-structured code.

2. Understanding the Rules of Wordle

The objective of Wordle is to guess a secret, five-letter word in six attempts or fewer. After each guess, you receive feedback in the form of colors to indicate how close your guess was to the secret word.

- You have **6 attempts** to guess a **secret 5-letter word**.
- Each guess must be a **valid 5-letter word**.
- After each guess, give feedback for every letter:
    1. **Correct (Green):** Right letter, right position.
    2. **Present (Yellow):** Letter exists but in a different position.
    3. **Absent (Gray):** Letter is not in the word.

3. Project Requirements & Expectations

Your implementation should meet the following criteria:

Functional Requirements:

1. **Game Logic:** The game must correctly implement all the rules mentioned above.
2. **Command-Line Interface:** The entire game must run in the command line (terminal).
3. **Word List:** The game should choose a secret word at random from a predefined list of valid five-letter words. You will need to find a text file containing these words.
4. **Input Handling:**
    - The game should accept user input for guesses.
    - **Case-Insensitive:** User input should be treated as case-insensitive (e.g., "apple" and "APPLE" should be processed identically).
    - **Input Validation:** Your program must validate that the user's guess is exactly five

letters long and is a valid word from your word list. If not, it should prompt the user for a valid guess without penalizing them an attempt.

5. **Win/Loss Conditions:** The game must correctly identify when the player has won (guessed the word) or lost (run out of attempts) and display an appropriate message.

Technical & Code Quality Expectations:

1. **Modularity:** Use functions to break down the program into logical, reusable parts (e.g., a function to select the word, a function to check the guess, a function to display the game state, etc.).
2. **Readability:** Write clean, readable code with clear variable names and comments where necessary to explain complex logic.
3. **Optimization:** Think about efficiency. How can you store and access the word list efficiently? How can you optimize the logic for checking a guess? The code should run smoothly without noticeable delays.
4. **No Hardcoding:** The secret word must be chosen randomly from the list, not hardcoded into the program.

4. Recommended Python Resources

To complete this task, you will need a good grasp of Python fundamentals. Here are the key concepts and modules that will be most useful:

- **Core Python:** functions, loops, conditionals, string operations.
- **Data Structures:** lists/sets (word list & lookups), dictionaries (letter tracking).
- **File Handling:** read words from a `.txt` file (one per line).
- **Random Module:** use `random.choice()` to select a word.

5. Getting Started: A Suggested Workflow

1. **Find a Word List:** Your first step is to find a good source for five-letter English words. A simple text file with one word per line is ideal. You can find many such lists on GitHub.
2. **Set Up Your Environment:** Create a new Python file (e.g., wordle.py).
3. **Load the Words:** Write a function that opens and reads the word list file, cleans the data (e.g., removes newline characters), and stores the words in a suitable data structure (a list or a set is a great choice).
4. **Build the Game Loop:** Create the main structure of your game. This will be a loop that runs for a maximum of six turns.
5. **Implement Core Logic:**
    - Inside the loop, prompt the user for input.
    - Validate the input.
    - Write the crucial function that compares the guess to the secret word and

determines the feedback (correct, present, absent) for each letter.

6. **Display Feedback:** Show the results of the guess to the player.
7. **Check for Win/Loss:** After each guess, check if the player has won or if they are out of attempts.
8. **Refine and Comment:** Once it's working, go back through your code. Can you make it cleaner? More efficient? Add comments to explain what each part of your code does.

Good luck, and have fun building it!