Project E
Mini Project Spec: Hangman (Single-Player CLI Game)

1) Theme

A classic word-guessing game: the computer picks a **random word**, the player guesses **one letter at a time**, and the word is gradually revealed. The player has a limited number of **wrong attempts**.

2) Learning goals (concepts)

- **Loops**: keep the game running until win/lose
- **Decisions**: correct vs wrong guess, repeated guess checks
- **Lists / strings**: store guessed letters, build the revealed word
- **Functions**: modular game logic (pick word, update display, validate input)

3) Game rules

1. The system chooses **one random word** from a predefined word list.
2. The player sees the word as blanks (example: `_ _ _ _`).
3. Player enters **one letter** per turn.
4. If the letter is in the word:
    - reveal it in all correct positions
5. If not:
    - reduce remaining attempts by 1
6. Player wins if all letters are revealed.
7. Player loses if attempts reach 0.

4) Inputs / Outputs

Input (user)

- A single letter guess (example: `a`)
- Commands (optional):
    - `exit` to quit
    - `hint` (extension)

Output (system)

- Current word progress: `_ a _ _ a _`
- Wrong attempts left: `4`
- Used letters: `a, e, s`
- Messages:
    - “Correct!”
    - “Wrong guess!”
    - “Already guessed that letter.”

5) Data structures (recommended)

- `words` (list of strings)
    - example: `["python", "school", "kashmir", "laptop", "logic"]`
- `secret_word` (string)
- `guessed_letters` (set or list) → letters guessed so far
- `wrong_letters` (set or list) → incorrect guesses
- `attempts_left` (int)

Display building approach

- `display = []`
- for each character in secret_word:
    - if char in guessed_letters → show char
    - else → show `_`

6) CLI flow

1. Print welcome + rules + attempts allowed (example: 6)
2. Pick random word from list
3. Start loop:
    - show current display
    - show attempts left + used letters
    - ask user for a guess
    - validate input
    - update game state
    - check win/lose
4. Print final result:
    - Win: “You won! The word was: _____”
    - Lose: “You lost. The word was: _____”
5. Ask if user wants to play again (optional)

7) Validation rules (must-have)

- Accept only **one alphabet character** (a–z)

- Convert input to lowercase
- If user enters more than one character → show error and retry
- If user enters non-letter (like `3` or `@`) → show error and retry
- If letter was already guessed → do not reduce attempts, just warn

8) Plain-English pseudocode

```
1. Create a list of words.
2. Randomly choose one word as the secret word.
3. Set attempts left to a fixed number (like 6).
4. Create an empty collection for guessed letters.
5. While attempts left is greater than 0:
    ○ Show the word with blanks for letters not guessed.
    ○ Ask the user to enter one letter.
    ○ If the input is invalid, ask again.
    ○ If the letter was already guessed, tell the user and continue.
    ○ Add the letter to guessed letters.
    ○ If the letter is in the secret word, tell the user “Correct”.
    ○ Otherwise, reduce attempts left by 1 and tell the user “Wrong”.
    ○ If all letters of the secret word are revealed, end the loop and declare win.
6. If attempts left becomes 0, declare loss and show the secret word.
```

9) Functional requirements (minimum)

- Random word selection
- Correct revealing of letters in all positions
- Wrong attempt counting
- Repeated guess handling
- Win/loss detection
- Clean console output each turn

10) Suggested functions (beginner-friendly)

- `pick_word(words) -> string`
- `get_display_word(secret_word, guessed_letters) -> string`
- `is_valid_guess(text) -> bool`
- `is_word_revealed(secret_word, guessed_letters) -> bool`
- `play_game()` (main loop)

11) Acceptance criteria (Definition of Done)

- Game chooses a random word from a list
- Correct guesses reveal letters properly
- Wrong guesses reduce attempts
- Repeated guesses do not reduce attempts
- Game ends with correct win/lose message

12) Test cases

1. Secret word: banana, guess: a → display _ a _ a _ a
2. Guess wrong letter z → attempts reduce by 1
3. Guess same letter twice → show “Already guessed”, attempts unchanged
4. Guess non-letter 1 → invalid input message, attempts unchanged
5. Win condition: all letters guessed → “You won”
6. Lose condition: attempts hit 0 → “You lost”, show word

13) Optional enhancements

- **Difficulty levels**
  - Easy: 8 attempts
  - Medium: 6 attempts
  - Hard: 4 attempts
- **Hints**
  - One hint per game reveals a random unguessed letter
- **Word categories**
  - “Animals / Places / Tech”
- **Scoring**
  - score = attempts_left × word_length
- **ASCII hangman drawing**
  - show stage-by-stage hangman based on wrong guesses