**Artificial Intelligent (Lab)**

**Task # 01**

****

**Submitted To: Sir Rasikh Ali**

**Submitted By: Shumaila Maryam**

**Roll no: SU92-BSDSM-F24-062**

**Submitted On: 8-August-2025**

**Section: BSDS-3A**

**Department of Software Engineering, Superior University, Lahore**

**Hangman Game**

The Hangman game is a word guessing game where the player tries to figure out a hidden word by guessing one letter at a time.

**How and why my code work?**

1. import random

The program begins by choosing a random category, such as Animals or Fruits. From that category, it selects a random word for the player to guess. This random selection ensures the game is different every time, keeping it fun and challenging.

1. HANGMAN\_PICS = [''']

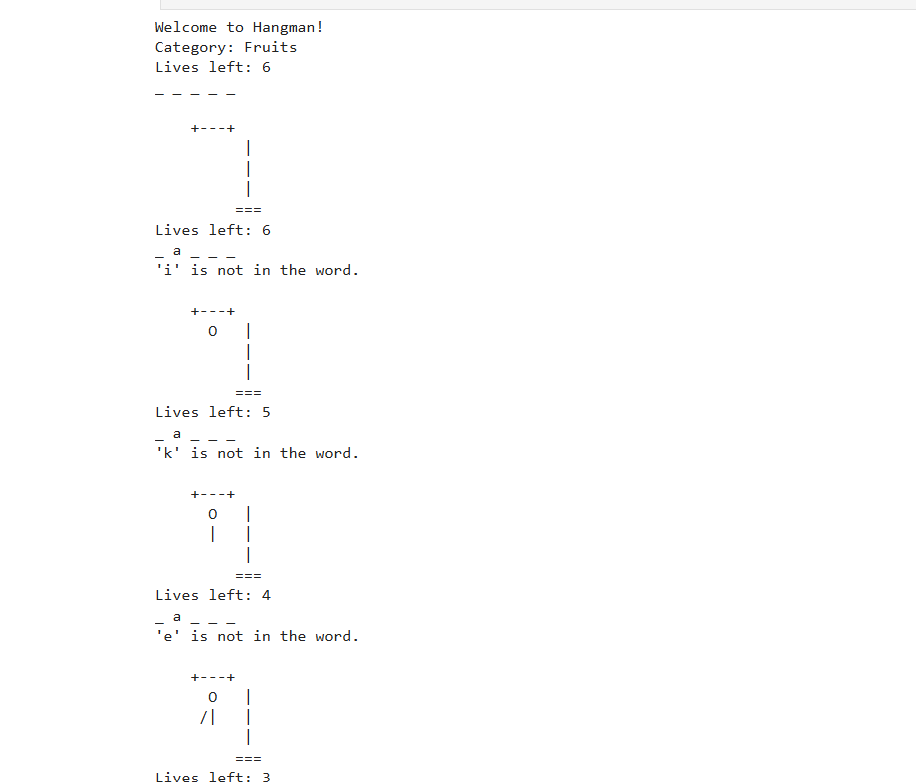
This is a list (an ordered collection) named HANGMAN\_PICS. Each item in this list is a multi-line string that visually represents a stage of the Hangman drawing. The drawings are ordered from the least complete (when the player has many lives) to the most complete (when the player has zero lives).

1. At the start, the game prepares blank spaces for the chosen word, showing one blank for each letter. This helps the player see how many letters they need to guess, giving a visual clue about the word’s length. The player also begins with a set number of lives, I set six, which represents how many incorrect guesses they can make before losing.
2. During the game, the player guesses one letter at a time. The program checks whether the letter has been guessed before. If it has, the player is informed, preventing repeated guesses and making the game fair. If the letter is in the word, the program reveals all occurrences of that letter in the word. If the letter is not in the word, the program reduces the player’s lives by one and updates the hangman drawing to reflect progress toward losing. This visual representation motivates the player to be careful with their guesses.
3. The game continues in a loop, repeatedly asking the player for letters, updating the displayed word, tracking guessed letters, and showing the hangman. This loop ensures the game is interactive and dynamic, reacting to each guess and keeping the player engaged.
4. The game ends in one of two ways: if the player successfully fills in all the blanks, they win. If the player runs out of lives before guessing the word, they lose, and the program reveals the correct word.

In Summary,

* **Random selection of categories and words to keep the game unpredictable.**
* **Tracking guessed letters to avoid repeated input.**
* **Updating the displayed word whenever a correct guess is made.**
* **Reducing lives and showing hangman drawings to visually indicate wrong guesses.**
* **Looping until the game ends, ensuring continuous interaction**.

These elements together create a fully functional, interactive Hangman game where the player can guess letters, see the results immediately, and either win by guessing the word or lose after too many wrong guesses. The combination of visual feedback, word tracking, and randomization makes the game engaging and fun to play.

**Output:**

**To-do List:**

This program is a simple To-Do List application that allows the user to add, view, and delete tasks. It is interactive, letting the user choose actions from a menu until they decide to quit.

**How and why my code work?**

1. The program keeps all tasks in a list, which is like a container that can hold multiple items. This list stores each task the user adds, making it easy to view or delete tasks later.
2. **Adding a Task**

* When the user wants to add a new task, the program:
* Asks the user to type the task.
* Adds the task to the list.
* Confirms to the user that the task has been added.

This allows the user to gradually build a list of tasks they want to complete.

1. **Listing Tasks**

When the user wants to see their tasks, the program:

* Checks if the list is empty. If it is, it informs the user that there are no tasks.
* If there are tasks, it **displays all tasks with numbers**, so the user knows the order and can easily identify each task.

Numbering tasks is important because it helps the user know which task to delete later.

1. **Deleting a Task**

When the user wants to delete a task, the program:

* Shows the current list of tasks with numbers.
* Asks the user to enter the number of the task to delete.
* Checks if the number is valid. If it is, the task is removed, and the program confirms it.
* If the number is invalid, the program informs the user and does not crash.

This ensures the program handles mistakes gracefully and avoids error.

1. **Main Menu Loop**

The program runs inside a **loop**, which repeatedly:

* Shows a menu of options: Add, Delete, List, or Quit.
* Accepts the user’s choice.
* Performs the selected action.

The loop continues until the user chooses to quit, allowing the user to perform multiple actions in one session.

1. **User Interaction and Validation**

The program is designed to interact with the user in a friendly way:

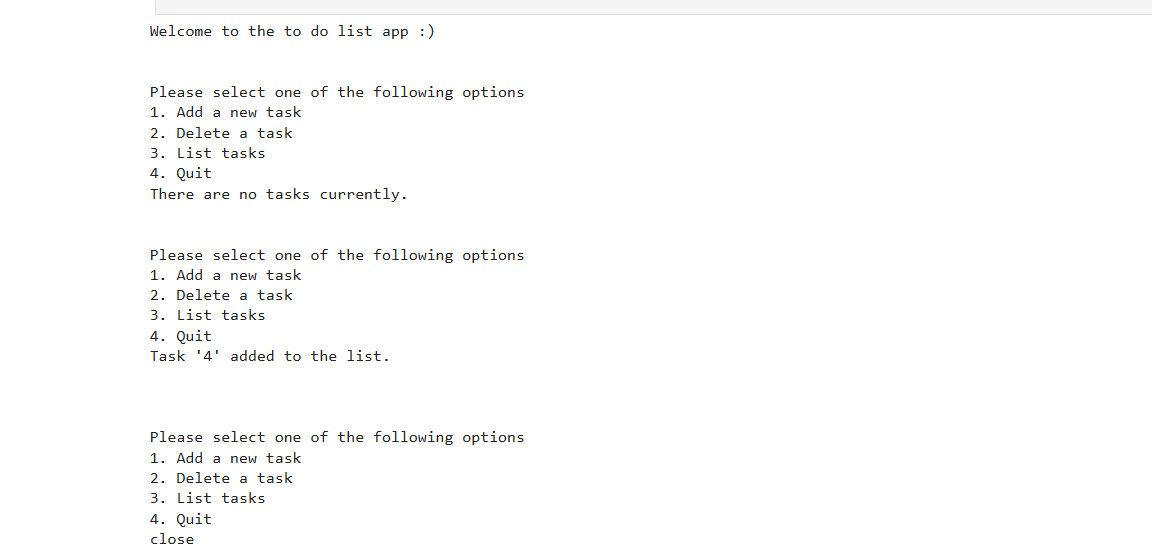
* It provides clear instructions.
* It confirms actions like adding or deleting tasks. It validates input, such as making sure the delete number is valid or alerting the user if they enter something wrong

**Code Explain:**

**This program works** by storing tasks in a list and using functions for adding, listing, and deleting tasks, the program keeps track of tasks and updates them dynamically based on user input. The menu loop allows continuous interaction.

This code makes the program easy to use, reliable, and prevents crashes due to invalid input. Functions separate the different actions, making the program organized and easy to maintain.

**Output:**

****