**AI – Lab Task 1**  
  
  
**Hangman Game**

At the start of the program, there is a list called Hangman. Each element of this **list** represents a stage of the hangman drawing. With every wrong guess, the next stage is displayed. This visually shows how close the player is to losing the game.

## **Messages**

The program also includes a list called **messages**. These are random messages that are displayed whenever the player makes a wrong guess. They make the game more interactive and fun.

## **Word Bank**

The list **words** contains possible words for the game, such as **Computer**, **Programming**, and **Python**. One of these is chosen randomly as the secret word for the player to guess.

## **The hangman() Function**

This is the **main function** that controls the flow of the game.

* It first selects a random word from the words list and converts it to lowercase.
* It then creates a dictionary mapping of letters and their positions using the g**uessing()** function.
* The game uses a while True loop to continuously ask the player to input a letter.
* If the input is invalid (not a single lowercase letter) or already guessed, it gives a warning.
* If the guessed letter is correct, it updates the display using the **replacing()** function.
* If the guessed letter is wrong, it shows the next hangman stage and a random insult message.
* The loop ends when the player either wins (all letters guessed) or loses (hangman fully drawn).

## **The guessing(word) Function**

This function creates a **dictionary** that stores each letter of the word and the positions where it appears.

* For example, if the word is "python", the dictionary looks like:  
  **{'p': [0], 'y': [1], 't': [2], 'h': [3], 'o': [4], 'n': [5]}.**
* This mapping helps the program know where to place each guessed letter in the display.

## **The** **replacing(dictionary, guessed, length\_of\_letters)** Function

This function is responsible for showing the current state of the guessed word.

* It starts with a list of underscores (\_) for each letter.
* Then, for each guessed letter, it replaces the underscores at the correct positions with the actual letter.
* For example, if the word is "python" and the user guessed "p" and "t", the display will be:  
  **['p', '\_', 't', '\_', '\_', '\_'].**
* The function returns the updated list, which helps track progress.

## Small Dry Run Example

Suppose the random word is "python".

1. The word mapping is:  
   **{'p': [0], 'y': [1], 't': [2], 'h': [3], 'o': [4], 'n': [5]}**
2. Player enters "p" → Correct! Display becomes: **['p', '\_', '\_', '\_', '\_', '\_']**
3. Player enters "z" → Wrong! Hangman stage 1 is shown with a random insult message.
4. Player enters "t" → Correct! Display becomes: **['p', '\_', 't', '\_', '\_', '\_']**
5. This continues until the player either guesses "python" completely or runs out of chances.