



# HANGMAN GAME IN PYTHON

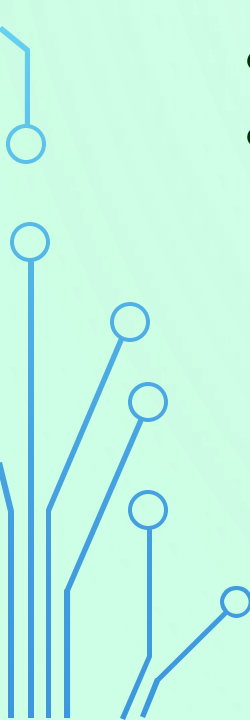
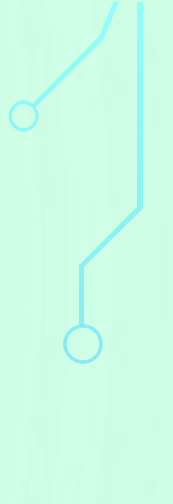
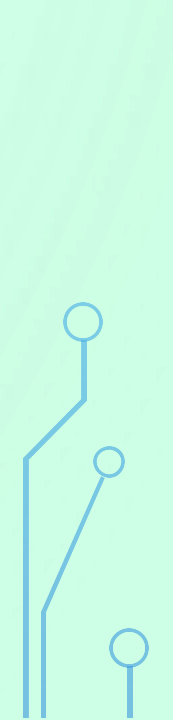
*A Simple Console-Based Word Guessing Game*

## **PROJECT OVERVIEW:**

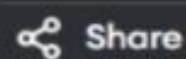
- *A classic Hangman game implemented in Python.*
- *Text-based, using basic programming concepts.*
- *Player guesses one letter at a time to find the hidden word.*
- *Limited to 6 wrong attempts.*



# TOOLS & TECHNOLOGIES:

- *Language: Python*
  - *Concepts Used:*
    - *random module*
    - *while loops*
    - *if-else statements*
    - *Strings & Lists*
    - *Console I/O*
- 
- 
- 

main.py



Share

Run

```
1  import random
2
3  # Predefined list of words
4  word_list = ["apple", "house", "robot", "pizza", "snake"]
5
6  # Randomly select a word
7  secret_word = random.choice(word_list)
8  guessed_letters = []
9  incorrect_guesses = 0
10 max_attempts = 6
11
12 # Create a display version of the word with underscores
13 display_word = ["_"] * len(secret_word)
14
15 print("Welcome to Hangman!")
16 print("Guess the word, one letter at a time.")
17 print(f"You have {max_attempts} incorrect guesses allowed.\n")
18
19 while incorrect_guesses < max_attempts and "_" in display_word:
20     print("Word: " + " ".join(display_word))
21     guess = input("Enter a letter: ").lower()
22
23     if len(guess) != 1 or not guess.isalpha():
24         print("Please enter a single alphabet letter.\n")
```



```
23-     if len(guess) != 1 or not guess.isalpha():
24-         print("Please enter a single alphabet letter.\n")
25-         continue
26-
27-     if guess in guessed_letters:
28-         print("You already guessed that letter. Try another.\n")
29-         continue
30-
31-     guessed_letters.append(guess)
32-
33-     if guess in secret_word:
34-         print("correct guess!\n")
35-         for idx, letter in enumerate(secret_word):
36-             if letter == guess:
37-                 display_word[idx] = guess
38-     else:
39-         incorrect_guesses += 1
40-         print(f"wrong guess! Attempts left: {max_attempts - incorrect_guesses}\n")
41-
42- # End of game result
43- if "_" not in display_word:
44-     print("Congratulations! You guessed the word:", secret_word)
45- else:
46-     print("You ran out of attempts. The word was:", secret_word)
```

main.py



Run

Output

Clear

```
23- if len(guess) != 1 or not guess.isalpha():
24-     print("Please enter a single alphabet letter.\n")
25-     continue
26-
27- if guess in guessed_letters:
28-     print("You already guessed that letter. Try another.\n")
29-     continue
30-
31- guessed_letters.append(guess)
32-
33- if guess in secret_word:
34-     print("correct guess!\n")
35-     for idx, letter in enumerate(secret_word):
36-         if letter == guess:
37-             display_word[idx] = guess
38- else:
39-     incorrect_guesses += 1
40-     print(f"wrong guess! Attempts left: {max_attempts - incorrect_guesses}\n")
41-
42- # End of game result
43- if "_" not in display_word:
44-     print("Congratulations! You guessed the word:", secret_word)
45- else:
46-     print("You ran out of attempts. The word was:", secret_word)
```

Word: \_ \_ \_ \_ \_

Enter a letter: a

wrong guess! Attempts left: 5

Word: \_ \_ \_ \_ \_

Enter a letter: r

correct guess!

Word: r \_ \_ \_ \_

Enter a letter: o

correct guess!

Word: r o \_ o \_

Enter a letter: b

correct guess!

Word: r o b o \_

Enter a letter: t

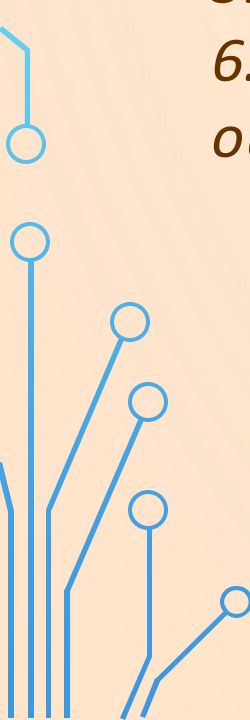
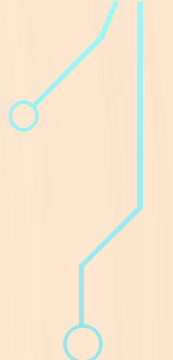
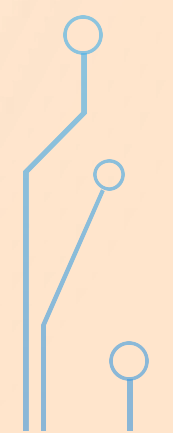
correct guess!

Congratulations! You guessed the word: robot

=== Code Execution Successful ===



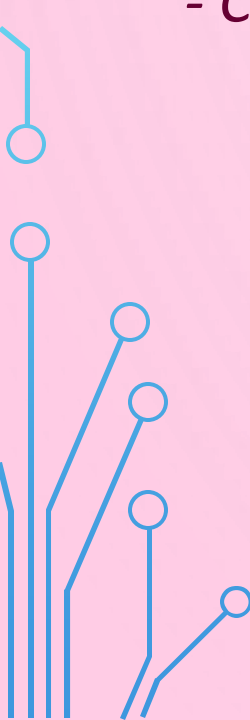
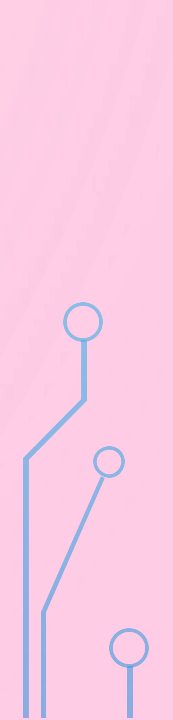
## GAME FLOW:

1. Choose a random word from a predefined list.
  2. Display underscores for each letter in the word.
  3. User guesses one letter at a time.
  4. Reveal letters if correct.
  5. Reduce attempts if incorrect.
  6. Game ends when the word is guessed or attempts run out.
- 
- 
- 





## **CHALLENGES FACED:**

- *Validating user input.*
  - *Avoiding repeated guesses.*
  - *Keeping track of correct vs incorrect letters.*
  - *Clear and simple UI in the console.*
- 
- 



## **CONCLUSION:**

- *A fun and educational project.*
- *Reinforced core Python concepts.*
- *Easy to extend (add hints, difficulty levels, etc.)*
- *Great for beginners learning loops and logic.*

The background is a solid light blue. In the four corners, there are decorative elements resembling circuit board traces or neural network connections. These consist of thin, light blue lines that branch out and terminate in small circles. The lines are more densely packed in the bottom-left and top-right corners, while the top-left and bottom-right corners have fewer, more sparse lines.

***THANK YOU***