**ABSTRACT**

Hangman is a classic word-guessing game where the player tries to uncover a hidden word by guessing one letter at a time. This version is a text-based Python implementation, designed for the command line or terminal. The game randomly selects a word from a predefined list, and the player has a limited number of attempts to guess all the letters correctly. This project presents a **text-based Hangman game** developed using Python. The objective of the game is for the player to guess a randomly selected word by suggesting letters within a limited number of incorrect attempts. The game is implemented using core Python features such as functions, loops, conditionals, and basic data structures like lists and sets. It emphasizes user interaction through the command line and includes input validation, progress tracking, and end-game outcomes (win or lose). This project serves as an educational tool to reinforce programming fundamentals and can be further enhanced with additional features like external word files, difficulty levels, or graphical representations.

**INTRODUCTION**

The Hangman Game is a classic word-guessing game designed to test a player’s vocabulary and logical thinking. This project implements a text-based version of Hangman using the Python programming language.

In the game, the computer randomly selects a hidden word from a predefined list, and the player attempts to guess the word one letter at a time. Each incorrect guess brings the player closer to losing by incrementing the hangman stage, with a limited number of wrong attempts allowed .The game ends either when the player successfully guesses all the letters in the word or exhausts all allowed incorrect guesses.

This project serves as a simple, fun, and interactive way to demonstrate the use of loops, conditionals, string manipulation, and basic input/output in Python. It is an excellent choice for beginner programmers who want to build their first complete command-line game.

**1.SYSTEM REQUIREMENTS**

**Software Requirements**

* Python: Version 3.6

Operating System

* + - Windows
    - MacOS
    - Linux/Unix

You can use any of the following to edit the code:

* Visual Studio Code
* PyCharm
* Sublime Text
* Notepad

**2.SYSTEM STUDY**

**2.1 Problem Statement**

Traditional word-guessing games like Hangman are widely known but are often only available in paper format or as complex graphical applications. Beginners looking to learn programming need a simpler, command-line version to understand fundamental concepts while still enjoying an interactive experience.

**2.2 Existing System**

In many existing Hangman games:

* The gameplay is either graphical (requires GUI libraries) or web-based (requires web development knowledge).
* These versions are not always beginner-friendly from a coding or learning perspective.
* Installation may require external libraries or frameworks.

**2.3 Proposed System**

This project presents a text-based Hangman game that:

* Runs entirely in a terminal or command line.
* Requires only basic Python (no external dependencies).
* Provides interactive gameplay with user-friendly messages.
* Allows players to guess letters and visually track their progress via simple text output.

**2.4 Objectives**

* Implement core gameplay mechanics of Hangman in Python.
* Ensure the game is intuitive and easy to play.
* Apply basic programming constructs: loops, conditionals, functions, string operations, and error handling.
* Create a foundation that can be extended into a GUI or multiplayer version in the future.

**3.SYSTEM DEIGN AND DEVELOPMENT**

**3.1 File design**

Hangman.py

* Main executable script.
* Contains the game logic, input/output handling, and main game loop.

README.md

* Project overview, instructions, features, sample output, and licensing.

**3.2 Input Design**

User Inputs:

* Letter Guess: The player inputs a single alphabetical character.

Validation Rules:

* Input must be a single letter (A-Z or a-z).
* Input is case-insensitive.
* Repeated guesses are identified and not processed again.

Input Method:

* Taken via Python's input() function in the terminal
* **3.3Output Design**

Outputs Displayed to User:

* Masked word with underscores for unguessed letters.
* Letters guessed so far.
* Remaining number of incorrect guesses.
* Messages indicating:
  + Correct or incorrect guess
  + Duplicate guess
  + Win or loss outcome

Output Format:

* Simple, readable, and clear text messages printed using print() statements.

Example:

yaml

CopyEdit

Word: \_ y t \_ o n

Guessed Letters: a e i o u y

Guess a letter:

**3.4 Database Design**

Current Version:

* No database is used.

Future Enhancement Possibilities:

* Store player scores or statistics.
* Maintain a list of words in a database table.
* Use a simple SQLite database with the following possible schema:

sql

CopyEdit

CREATE TABLE Players (

player\_id INTEGER PRIMARY KEY,

name TEXT,

games\_played INTEGER,

games\_won INTEGER

);

CREATE TABLE Words (

word\_id INTEGER PRIMARY KEY,

word TEXT

);

**CODE DESIGN**

import random

def choose\_word():

"""Randomly selects a word from the list."""

word\_list = ['python', 'developer', 'hangman', 'programming', 'keyboard']

return random.choice(word\_list)

def display\_word(word, guessed\_letters):

"""Returns the word display with guessed letters revealed and others as underscores."""

return ' '.join([letter if letter in guessed\_letters else '\_' for letter in word])

def play\_hangman():

"""Main function to run the hangman game."""

word = choose\_word()

guessed\_letters = set()

incorrect\_guesses = 0

max\_incorrect = 6

print("🎮 Welcome to Hangman!")

print(f"You have {max\_incorrect} incorrect guesses allowed.")

while incorrect\_guesses < max\_incorrect:

print("\nWord: ", display\_word(word, guessed\_letters))

print("Guessed Letters: ", ' '.join(sorted(guessed\_letters)))

guess = input("Guess a letter: ").lower()

# Validate input

if not guess.isalpha() or len(guess) != 1:

print("⚠️ Please enter a single alphabetical character.")

continue

if guess in guessed\_letters:

print("❗ You already guessed that letter.")

continue

guessed\_letters.add(guess)

if guess in word:

print("✅ Good guess!")

if all(letter in guessed\_letters for letter in word):

print("\n🎉 Congratulations! You guessed the word:", word)

break

else:

incorrect\_guesses += 1

print(f"❌ Incorrect! You have {max\_incorrect - incorrect\_guesses} guesses left.")

else:

print("\n💀 Game Over! The word was:", word)

if \_\_name\_\_ == "\_\_main\_\_":

play\_hangman()

**3.5 System Development**

Development Approach:

* Incremental: Each feature was added step-by-step, starting with basic word display and progressing to input validation and win/loss logic.

Development Tools:

* Language: Python 3
* IDE/Text Editor: Any (e.g., VS Code, PyCharm, Notepad++)

Testing Methodology:

* Manual testing for edge cases:
  + Repeated letters
  + Non-alphabet characters
  + Capital/lowercase handling
  + Max incorrect guesses
* Each function tested independently before integration

Version Control (Optional):

* Git for tracking code changes

**4.TESTING AND IMPLEMENTATION**

**4.1 Testing**

**4.1.1 Test Objectives**

* Ensure the game runs without errors
* Validate user input handling
* Confirm correct and incorrect guesses are processed properly
* Verify win/loss logic works as expected

| **Type** | **Description** |
| --- | --- |
|  |  |
| **Unit Testing** | Each function (e.g., choose\_word(), display\_word()) was tested individually for correctness. |
| **Functional Testing** | The entire gameplay loop was tested to ensure expected behavior. |
| **Boundary Testing** | Edge cases like repeated inputs, invalid inputs, and case sensitivity were tested. |
| **Manual Testing** | Game was played multiple times manually to simulate real user behavior. |

| **Test Case ID** | **Description** | **Input** | **Expected Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC01 | Guessing a correct letter | 'p' | Reveals all occurrences of 'p' |  |
| TC02 | Guessing an incorrect letter | 'z' | Increases incorrect guess count |  |
| TC03 | Entering a non-alphabetical character | '1' | Prompts invalid input message |  |
| TC04 | Repeating the same letter | 'a' twice | Displays “already guessed” message |  |
| TC05 | Winning the game | Correct guesses only | Displays win message |  |
| TC06 | Losing the game | 6 wrong guesses | Displays game over message |  |

**4.2 Implementation**

**4.2.1 Implementation Steps**

1. **Code Setup**:
   * Developed in Python using any preferred code editor.
   * Stored in a single file hangman.py.
2. **Execution Environment**:
   * Terminal or command prompt (cross-platform)
   * Run with the command:
3. **User Instructions**:
   * The user is prompted to guess a letter.
   * The system checks the guess and updates the display accordingly.
   * The game continues until the word is guessed or all attempts are used.
4. **Error Handling**:
   * Input is validated to prevent crashes due to non-alphabet input or multiple characters.
   * Duplicate guesses are ignored and the user is notified.

**4.3 ✅ Results**

* The system passed all defined test cases.
* It behaves as expected under various conditions.
* No runtime errors encountered during multiple manual plays.
* User experience is smooth, with clear messages and feedback.

**CONCLUSION**

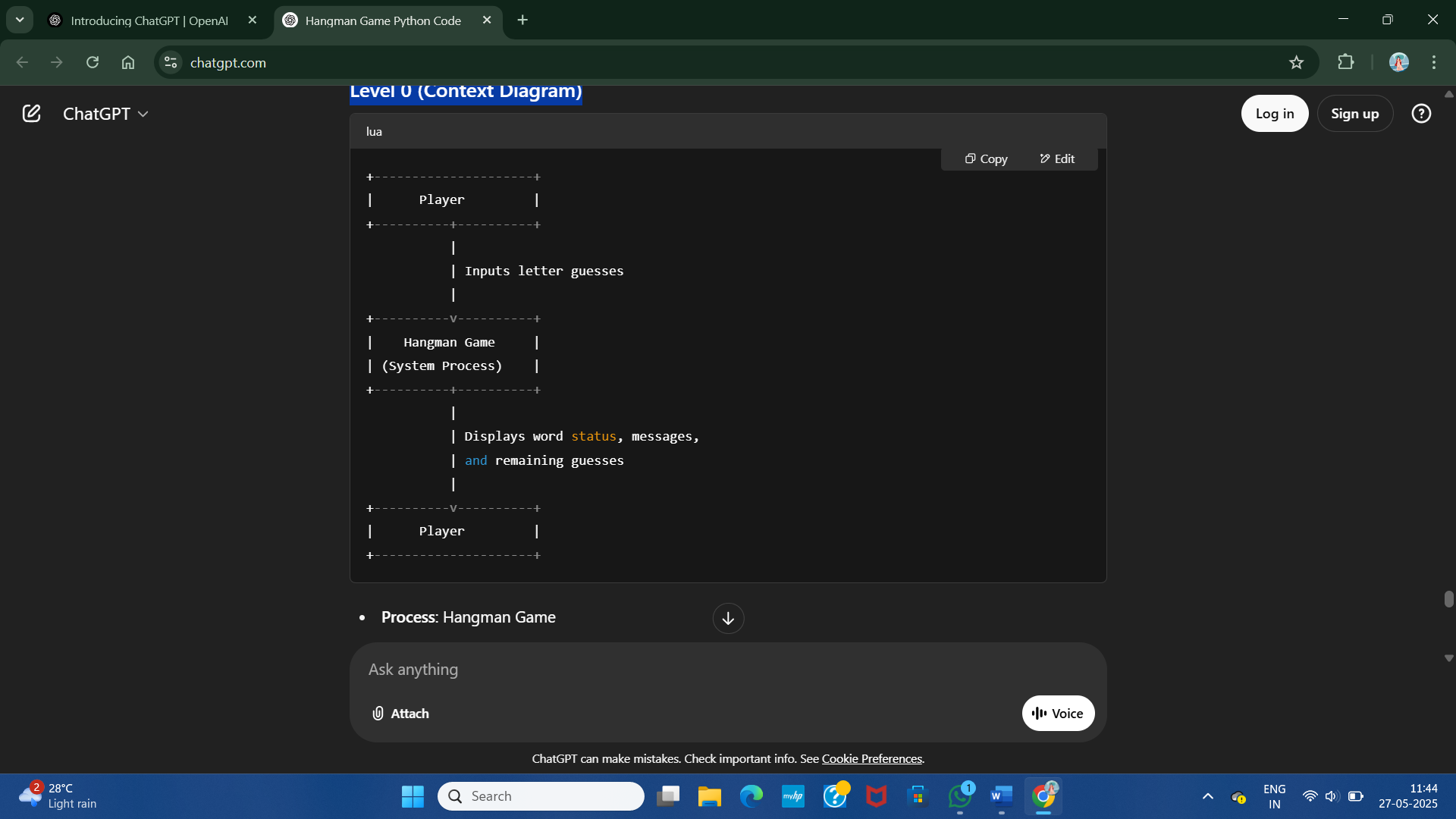
The Hangman Game project is a simple yet effective demonstration of basic Python programming concepts, including loops, conditionals, functions, sets, and user input handling. It provides an engaging way to practice coding logic while offering an interactive user experience. This project can serve as a foundation for more advanced enhancements such as graphical interfaces, dynamic word lists, or multiplayer functionality. Overall, it is a great starting point for beginners looking to build console-based games and improve their Python skills. Through the design and development of this project, the importance of structured coding, user interaction, and iterative testing was emphasized. The game offers a smooth and interactive user experience in a console environment and runs reliably across different platforms without any dependencies.

**BIBLIOGRAPHY**

1. **Python Software Foundation.** *Python 3 Documentation*.  
   <https://docs.python.org/3/>
2. **W3Schools.** *Python Tutorial*.  
   https://www.w3schools.com/python/
3. **GeeksforGeeks.** *Hangman Game in Python*.  
   https://www.geeksforgeeks.org/hangman-game-in-python/
4. **Real Python.** *Handling User Input in Python*.  
   https://realpython.com/python-input-output/
5. **Stack Overflow Community.** *Programming Q&A*.  
   <https://stackoverflow.com/>

**DATA FLOW DIAGRAM**

**Level 0 (Context Diagram)**



**Level 1 (Detailed DFD)**

+-------------------+ +---------------------+ +----------------+

| | | | | |

| Player +------->| Input Validation | | Word Selection |

| (Input Guess) | | (Check valid input) |<--------+ (Choose Word) |

| | | | | |

+-------------------+ +----------+----------+ +----------------+

|

v

+-----------+-----------+

| |

| Guess Processing |

| (Check guess in word,|

| update guessed set, |

| update counters) |

+-----------+-----------+

|

+-----------------------+-------------------------+

| | |

v v v

+-----------+-----------+ +----------+----------+ +---------+---------+

| Display current word | | Display messages | | Update incorrect |

| with guessed letters | | (Correct/Wrong/etc.) | | guess count |

+----------------------+ +---------------------+ +--------------------+

|

v

+--------+-------+

| Check win/loss |

| (End game?) |

+--------+-------+

|

v

+--------+-------+

| Output result |

|  |
| --- |
|  |

**SAMPLE INPUT**

import random

def choose\_word():

words = ['python', 'hangman', 'challenge', 'programming', 'developer']

return random.choice(words)

def display\_word(word, guessed\_letters):

return ' '.join(letter if letter in guessed\_letters else '\_' for letter in word)

def play\_hangman():

word = choose\_word()

guessed\_letters = set()

incorrect\_guesses = 0

max\_incorrect = 6

print("Welcome to Hangman!")

print(f"You have {max\_incorrect} incorrect guesses allowed.\n")

while incorrect\_guesses < max\_incorrect:

print(display\_word(word, guessed\_letters))

guess = input("Guess a letter: ").lower()

if len(guess) != 1 or not guess.isalpha():

print("Please enter a single alphabetical character.\n")

continue

if guess in guessed\_letters:

print("You already guessed that letter. Try again.\n")

continue

guessed\_letters.add(guess)

if guess in word:

print("Good guess!\n")

if all(letter in guessed\_letters for letter in word):

print(f"Congratulations! You guessed the word: {word}")

break

else:

incorrect\_guesses += 1

print(f"Wrong guess! You have {max\_incorrect - incorrect\_guesses} guesses left.\n")

else:

print(f"Game Over! The word was: {word}")

if \_\_name\_\_ == "\_\_main\_\_":

play\_hangman()

**SAMPLE OUTPUT**

