This project report covers the Number Guessing Game.

1. General Info.
The project title is Number Guessing Game with Performance Analysis.
Alok Kumar Sharma prepared this report.
The date is November 23, 2025.
The institution is VIT Bhopal.

2. Introduction.
This Python based number guessing game challenges players to find a secret number within a limited number of attempts. It includes basic elements like input validation, loops, and conditional statements, along with some intelligent guessing strategies. The game also provides feedback on the player's performance.

3. Problem Statement.
The task involves creating a simple console based game in Python that selects a random number and allows users to guess it while enforcing a limit on attempts. The system must validate inputs for correctness, offer hints such as whether the guess is too high or too low, and evaluate the efficiency of the guessing process. It should calculate the probability of success and determine the minimum additional guesses required using an optimal strategy.

4. Functional Requirements.
The program selects a random number between 1 and 100.
It accepts integer guesses from the user and ensures they are valid.
The game prevents users from repeating the same guess more than once.
After every guess, it indicates if the input is higher or lower than the secret number.
Users get only 7 guesses in total.
The system computes the odds of winning based on the session's activity.
It determines the fewest extra guesses needed through a strategic method.
At the end, it displays all results and exits the program cleanly.

5. Non functional Requirements.
The console interface remains straightforward for all users to navigate easily.
It handles invalid inputs gracefully without causing the program to crash.
Responses occur quickly, within one second for each operation.
The game runs on any environment with Python 3 or higher versions.
It relies solely on standard Python libraries without requiring additional installations.
Clear error messages appear whenever issues arise during play.

6. System Architecture.
The architecture divides tasks into distinct components, with the main module overseeing the entire process, including input validation, hint provision, and final calculations. Built in Python modules support the functionality, such as random for generating numbers, sys for handling inputs, time for introducing delays, and math for performing computations.

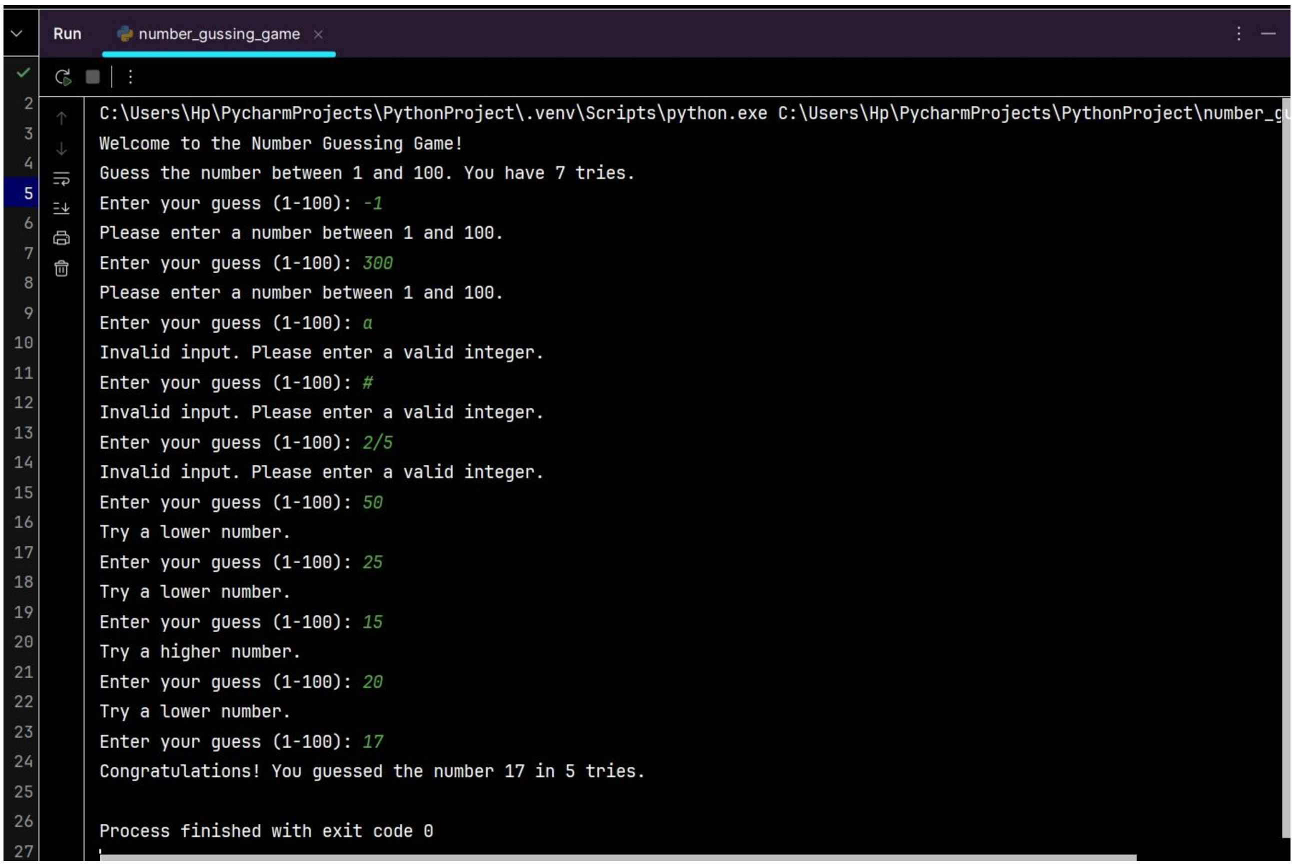
7. Design Diagrams.
In the Use Case Diagram, the player initiates the game, submits guesses, and receives hints in return. The system validates the input, generates the secret number, monitors attempts, and conducts the performance analysis.
The Workflow Diagram begins with starting the game. It then selects the secret number. A loop handles guess collection,

validation, and feedback provision. The game concludes with the analysis report.
The Sequence Diagram shows the player entering a guess. The system validates it. It compares the guess to the secret number. Feedback follows immediately. The system tracks all attempts. The game ends, and the analysis displays.
The Class Component Diagram features key functions for input validation, core game logic, and performance calculations. It depends on standard Python modules for support.
The ER Diagram does not apply here. No long term data storage occurs in this project.

8. Design Decisions and Rationale.
A console interface seemed best because it simplifies development and keeps the experience direct. The 7 guess limit aligns closely with the steps in a binary search for numbers in this range. The range from 1 to 100 provides a balance of challenge without overwhelming most players. Robust input validation ensures smooth gameplay and user friendliness. Including probability calculations highlights real world aspects of guessing scenarios. Using only built in modules eliminates dependencies on external packages or setup issues.

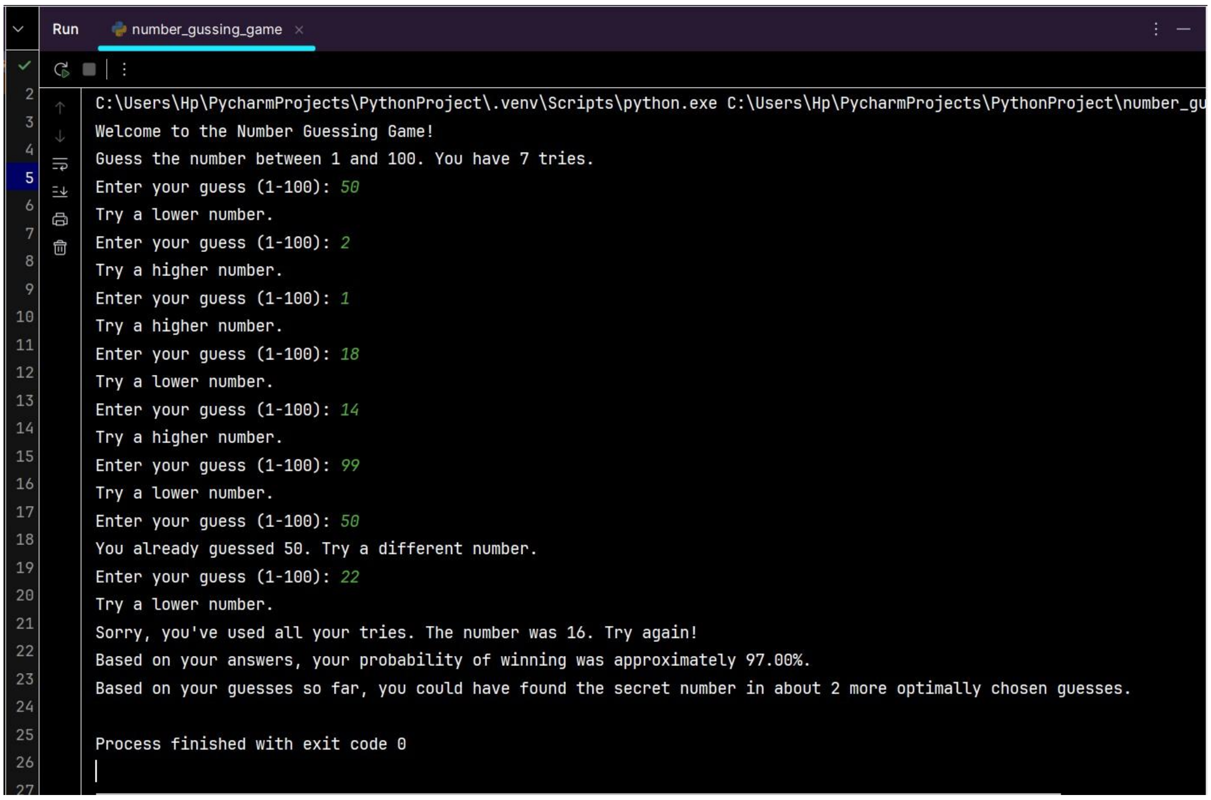
9. Implementation Details.
The get valid guess function verifies that the input is an integer within the allowed range and has not been used before. The analyze guesses function estimates probability by assessing how many possible numbers the guesses eliminated. The min additional tries function applies a base 2 logarithm to calculate the extra steps required, similar to binary search efficiency. The main loop gathers all guesses, delivers hints after each one, and finalizes the session.

10. Screenshots Results.
A typical run demonstrates the user entering guesses, the program's response to invalid inputs, the higher or lower hints provided, win or loss indications, and the performance summary displayed at the end.



```
C:\Users\Hp\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Hp\PycharmProjects\PythonProject\number_g
Welcome to the Number Guessing Game!
Guess the number between 1 and 100. You have 7 tries.
Enter your guess (1-100): -1
Please enter a number between 1 and 100.
Enter your guess (1-100): 300
Please enter a number between 1 and 100.
Enter your guess (1-100): a
Invalid input. Please enter a valid integer.
Enter your guess (1-100): #
Invalid input. Please enter a valid integer.
Enter your guess (1-100): 2/5
Invalid input. Please enter a valid integer.
Enter your guess (1-100): 50
Try a lower number.
Enter your guess (1-100): 25
Try a lower number.
Enter your guess (1-100): 15
Try a higher number.
Enter your guess (1-100): 20
Try a lower number.
Enter your guess (1-100): 17
Congratulations! You guessed the number 17 in 5 tries.

Process finished with exit code 0
```



```
C:\Users\Hp\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\Hp\PycharmProjects\PythonProject\number_gu
Welcome to the Number Guessing Game!
Guess the number between 1 and 100. You have 7 tries.
Enter your guess (1-100): 50
Try a lower number.
Enter your guess (1-100): 2
Try a higher number.
Enter your guess (1-100): 1
Try a higher number.
Enter your guess (1-100): 18
Try a lower number.
Enter your guess (1-100): 14
Try a higher number.
Enter your guess (1-100): 99
Try a lower number.
Enter your guess (1-100): 50
You already guessed 50. Try a different number.
Enter your guess (1-100): 22
Try a lower number.
Sorry, you've used all your tries. The number was 16. Try again!
Based on your answers, your probability of winning was approximately 97.00%.
Based on your guesses so far, you could have found the secret number in about 2 more optimally chosen guesses.

Process finished with exit code 0
```

11. Testing Approach.
Unit tests verified the input validation and analysis components independently for accuracy. Integration tests examined the complete game flow from initiation to conclusion. Edge cases received attention, including repeated guesses, boundary values like 1 or 100, and non numeric inputs. Manual playthroughs confirmed that hints and outcomes appeared correctly in every scenario.

12. Challenges Faced.
Handling various invalid inputs without disrupting the game's progression required careful planning and iteration. Developing an intuitive method to present probability and optimal extra guesses proved difficult to refine for clarity. Balancing the difficulty level ensured the game remained engaging rather than discouraging for players. Maintaining clean program execution involved proper termination sequences and brief pauses as needed.

13. Learnings and Key Takeaways.
Input validation and error management in code became much stronger skills through this work. Binary search concepts clearly demonstrate how to optimize guessing in such games. Breaking the code into focused modules, where each function handles a single responsibility, improves overall structure. Python's standard libraries proved invaluable for straightforward projects like this one.

14. Future Enhancements.
A graphical user interface could enhance appeal and interaction ease for players. Difficulty levels and multiplayer modes would add variety and competition. Persistent storage for statistics would allow users to monitor their improvement over sessions. Additional hints and a scoring system could incentivize effective strategies. Automated testing frameworks would streamline verification processes.

15. References.

Python Official Documentation at https colon slash slash docs dot python dot org slash 3 slash.
GeeksforGeeks tutorial on building a number guessing game.
Stack Overflow threads about algorithms for guessing games.
Books on algorithms covering binary search and basic probability.
Guides for Python's built in modules.