VIT
BHOPAL

NAME:	AKSHAT SINGH BAGHEL
PROJECT TOPIC:	PASSWORD GENERATOR
REG. NUMBER:	25MIP10055

INTRODUCTION:

In today’s digital age, password security is a critical component of cybersecurity. Weak or predictable passwords often lead to unauthorized access, data breaches, and identity theft. This project, titled “Strong Password Generator”, addresses this issue by providing a simple yet powerful Python-based solution that generates secure passwords tailored to user preferences.

The program allows users to specify the number of letters, numbers, and symbols in their password, giving them full control over its complexity. The character sets include both uppercase and lowercase letters, digits from 0-9, and a broad set of commonly used special characters, ensuring compliance with most password policies.

Designed using modular programming techniques, the code separates the user input process, password generation logic, and constants configuration. Input validation ensures that users cannot accidentally enter invalid or insecure values (like negative numbers or empty inputs). Internally, the code leverages the random module’s choices() method for selection and shuffle() for randomness, helping to eliminate predictable patterns.

By offering a fully customizable password creation process and ensuring robust randomness in output, this project serves as a practical tool for individuals, developers, or organizations aiming to strengthen digital authentication practices. It can be integrated into sign-up forms, used as a command-line tool, or extended into a GUI-based application for enhanced user experience.

PROBLEM STATEMENT

In an increasingly digital world, individuals and organizations are required to create and manage multiple passwords across various platforms. However, most users tend to create weak, repetitive, or predictable passwords due to difficulty in remembering complex ones. This behavior exposes systems to security threats such as brute-force attacks, unauthorized access, and data breaches.

The core problem lies in the lack of a simple, customizable, and secure method to generate strong passwords that balance complexity with user-defined structure.

This project aims to solve the problem by developing a Python-based password generator that:

- Accepts user input for the number of letters, numbers, and symbols desired in the password.
- Randomly selects characters from predefined secure character sets.
- Validates user inputs to avoid errors or security loopholes.
- Returns a strong, randomized password resistant to guesswork and common attack patterns.

This solution addresses the gap between usability and security by empowering users to generate secure, non-repetitive passwords in a flexible and efficient manner.

FUNCTIONAL REQUIREMENTS

1. Prompt user for number of letters, symbols, and numbers.

2. Validate inputs to ensure they are non-negative integers.

3. Randomly generate specified number of letters, digits, and symbols.

4. Shuffle all characters for randomness.

5. Combine characters into one password string.

6. Display the final password to the user.

7. Handle invalid or empty inputs gracefully.

NON-FUNCTIONAL REQUIREMENTS

1. Usability – Simple and user-friendly input prompts.

2. Reliability – Handles invalid inputs without errors.

3. Performance – Fast password generation.

4. Security – Strong, randomized passwords.

5. Maintainability – Clean, modular code for easy updates.

6. Portability – Runs on any system with Python.

SYSTEM ARCHITECTURE

1. Input Layer
   - Accepts user inputs for number of letters, symbols, and numbers.
   - Validates input using get_integer_input() function.

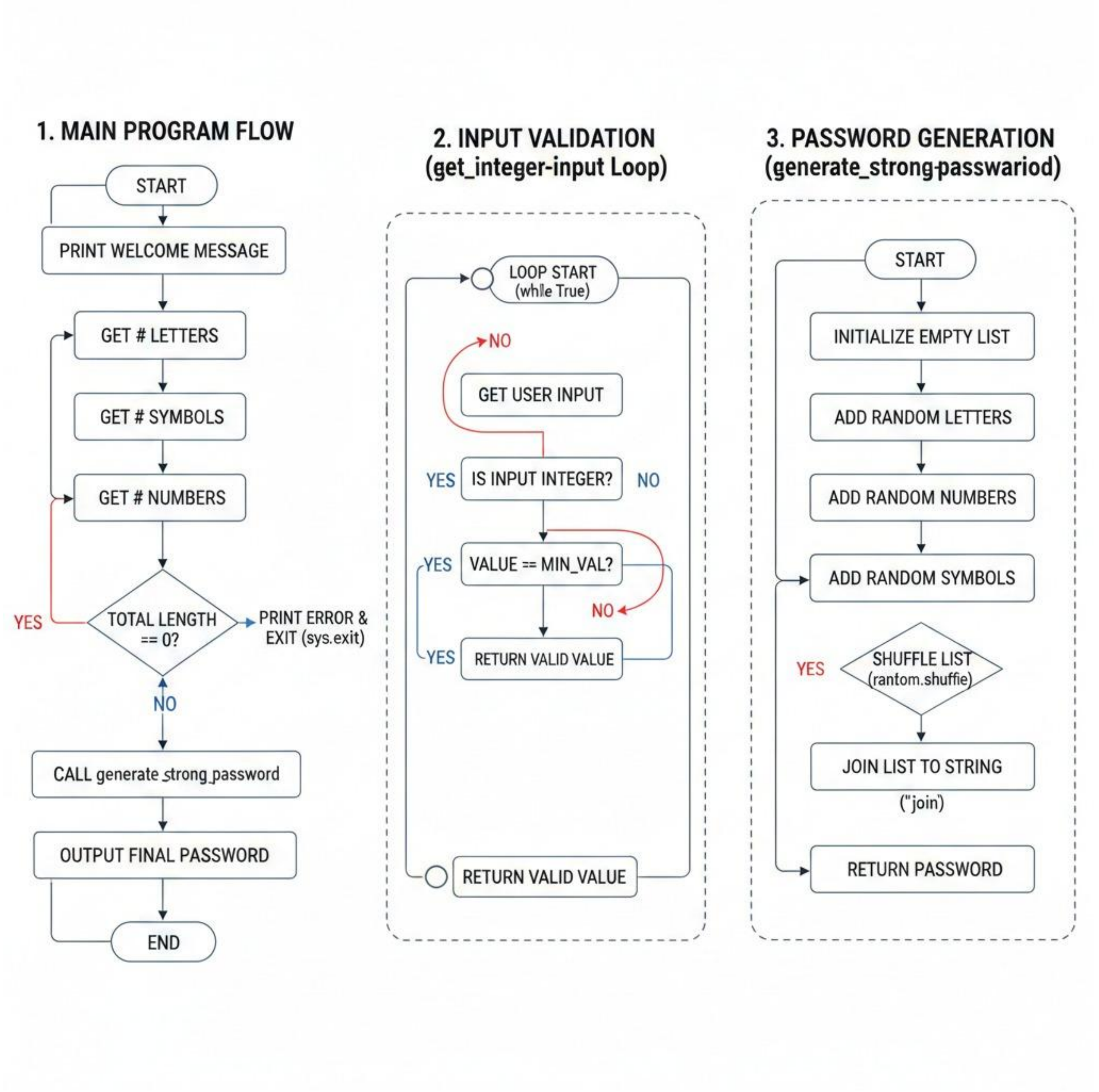
2. Processing Layer (Core Logic)
   - Uses generate_strong_password() to:
     - Randomly select characters from predefined sets.
     - Combine and shuffle to ensure randomness.

3. Output Layer
   - Returns and displays the final generated password to the user.

4. Modules Used

- random: For selecting and shuffling characters.
- Standard I/O: For user interaction.

DESIGN DIAGRAMS



DESIGN DECISIONS & RATIONALE

1. Modular Functions

   - Code is split into functions (get_integer_input, generate_strong_password) for better readability and reuse.

2. Character Set Constants

   - Defined globally for easy updates and maintenance of password rules.

3. Input Validation

   - Used robust input handling with error messages to ensure correct data entry.

4. Random Module

   - random.choices and random.shuffle provide strong randomness in password creation.

5. Shuffling After Combining

   - Ensures that characters from different types (letters, numbers, symbols) are well mixed, increasing password strength.

6. Error Handling

   - Displays error if user chooses zero characters, improving usability.

IMPLEMENTATION DETAILS

1. The program starts by importing the random module to help with random selections.

2. It defines separate lists of characters to organize lowercase letters, uppercase letters, digits, and special symbols clearly.

3. A function get_integer_input is used to safely take user inputs and ensure they enter valid, non-negative numbers.

4. The password generation logic uses the random.choices() method, which picks multiple random characters at once for efficiency.

5. The chosen characters from different categories are combined into one list before shuffling.

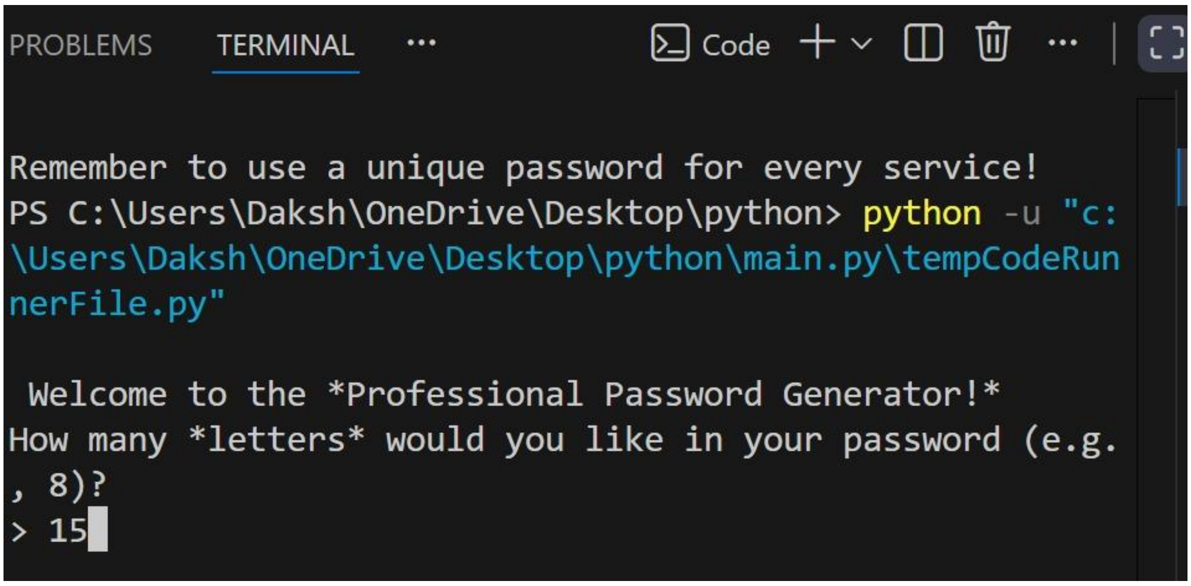
6. Shuffling is done using random.shuffle() to ensure the password characters are mixed well, increasing password strength.

7. The program checks if the total length of the password is zero, and if so, it prevents generating an empty password by returning an error message.

8. Finally, all characters are joined together into a string to form the final password that is displayed to the user.

SCREENSHOTS / RESULTS

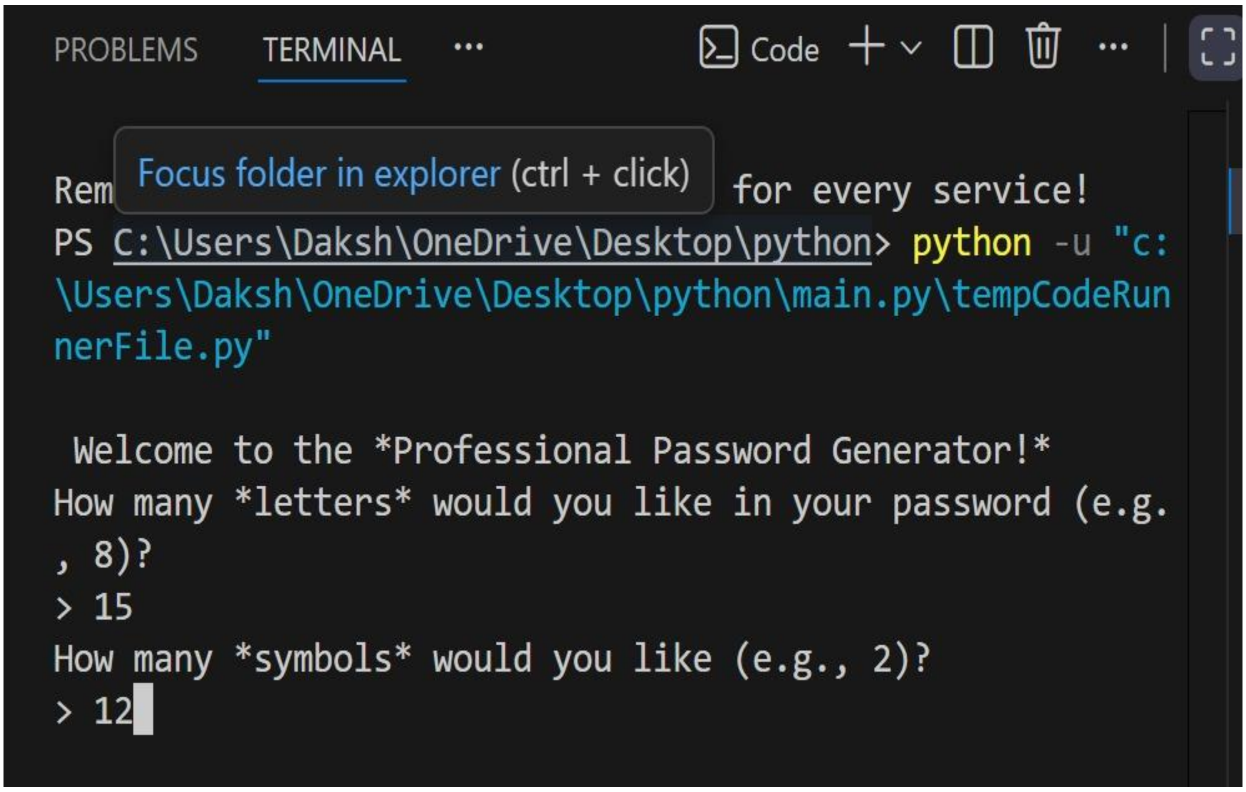
1) Screenshot-1
   Asks the user to add the number of letters they want in their password



```
Remember to use a unique password for every service!
PS C:\Users\Daksh\OneDrive\Desktop\python> python -u "c:
\Users\Daksh\OneDrive\Desktop\python\main.py\tempCodeRun
nerFile.py"

 Welcome to the *Professional Password Generator!*
How many *letters* would you like in your password (e.g.
, 8)?
> 15
```

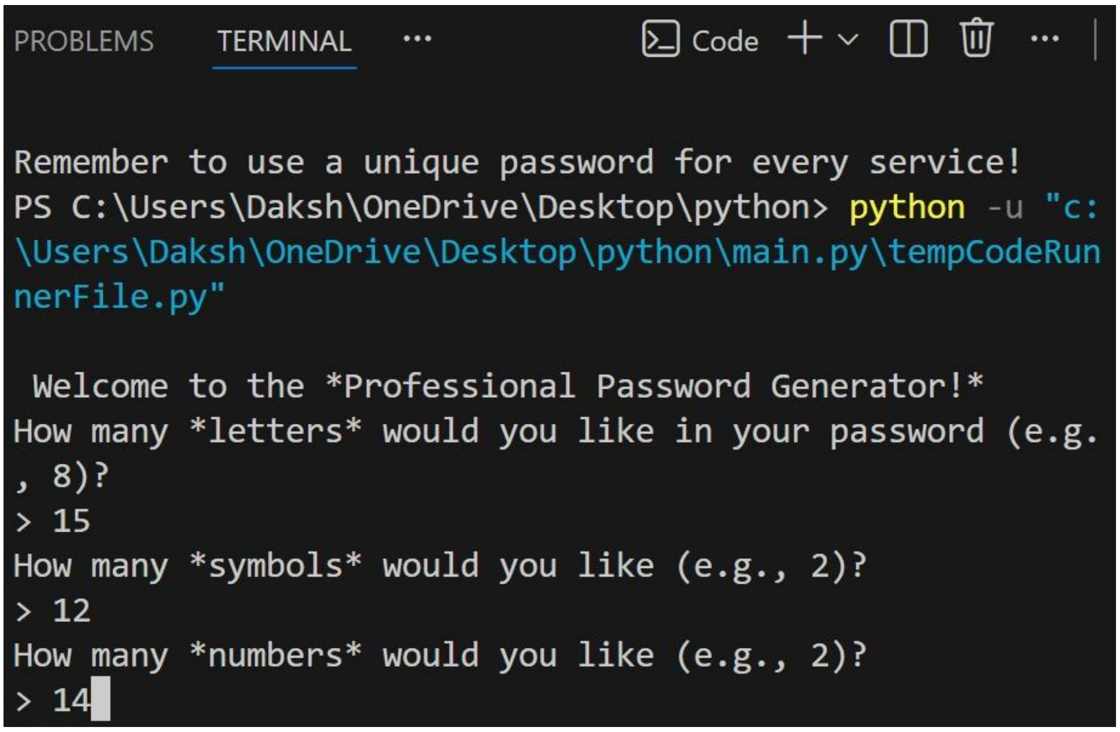
2) Screenshot-2
   Asks the user to add the number of symbols they want in their password.



```
Rem                                      for every service!
PS C:\Users\Daksh\OneDrive\Desktop\python> python -u "c:
\Users\Daksh\OneDrive\Desktop\python\main.py\tempCodeRun
nerFile.py"

 Welcome to the *Professional Password Generator!*
How many *letters* would you like in your password (e.g.
, 8)?
> 15
How many *symbols* would you like (e.g., 2)?
> 12
```

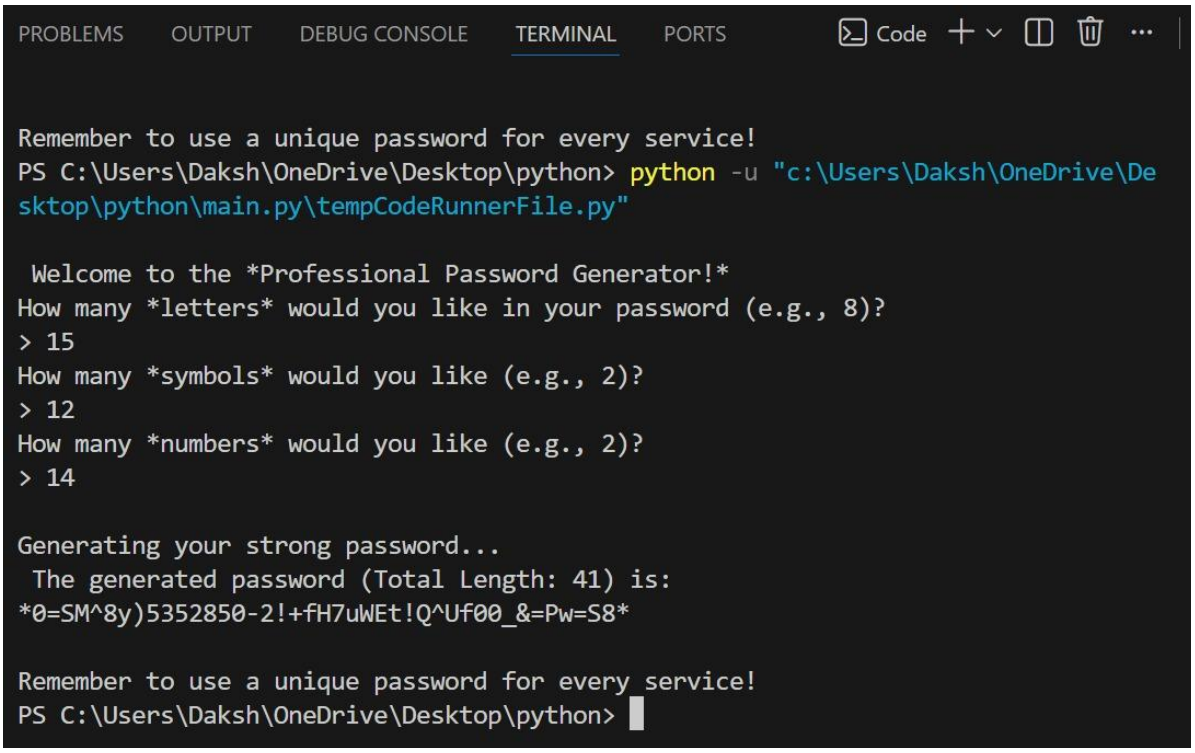
3) Screenshot-3
Asks the user to add the number of “numbers” they want in their password.



```
Remember to use a unique password for every service!
PS C:\Users\Daksh\OneDrive\Desktop\python> python -u "c:
\Users\Daksh\OneDrive\Desktop\python\main.py\tempCodeRun
nerFile.py"

 Welcome to the *Professional Password Generator!*
How many *letters* would you like in your password (e.g.
, 8)?
> 15
How many *symbols* would you like (e.g., 2)?
> 12
How many *numbers* would you like (e.g., 2)?
> 14
```

4) Screenshot-4
Final Output of the code after the password is generated.



```
Remember to use a unique password for every service!
PS C:\Users\Daksh\OneDrive\Desktop\python> python -u "c:\Users\Daksh\OneDrive\Desktop\python\main.py\tempCodeRunnerFile.py"

 Welcome to the *Professional Password Generator!*
How many *letters* would you like in your password (e.g., 8)?
> 15
How many *symbols* would you like (e.g., 2)?
> 12
How many *numbers* would you like (e.g., 2)?
> 14

Generating your strong password...
 The generated password (Total Length: 41) is:
*0=SM^8y)5352850-2!+fH7uWEt!Q^Uf00_&=Pw=S8*

Remember to use a unique password for every service!
PS C:\Users\Daksh\OneDrive\Desktop\python>
```

TESTING APPROACH

1. Input Validation Testing
   - Test with valid inputs (e.g., positive integers) for letters, symbols, and numbers to ensure correct password length.
   - Test invalid inputs such as negative numbers, letters, special characters, and empty input to verify the input validation handles errors gracefully and prompts the user again.

2. Boundary Testing
   - Test edge cases like zero for all inputs to confirm the program exits with an appropriate error message.
   - Test minimum values (e.g., 1 letter, 0 symbols, 0 numbers) to ensure password generation works with smallest valid inputs.

3. Functionality Testing
   - Verify the password contains exactly the requested number of letters, symbols, and numbers.
   - Check the randomness by running multiple tests to ensure different passwords are generated each time.

4. Output Testing

- Confirm the output format is clear, showing the password and its length properly.
- Validate that the reminder message displays correctly.

5. Module Integration Testing
   - Test the interaction between the main script and the module (module1) to ensure functions like input handling and password generation work together seamlessly.

CHALLENGES FACED

- ➢ Handling invalid user inputs and ensuring robust input validation without program crashes.

- ➢ Ensuring randomness in password generation for better security.

- ➢ Managing modular code structure for easy maintenance and readability.

- ➢ Providing meaningful error messages for zero-length passwords.
- ➢ Balancing simplicity with functionality for a user-friendly experience.

LEARNINGS & KEY TAKEAWAYS

- Importance of input validation to avoid errors and improve user experience.

- How to use Python's random module to generate secure and random passwords.

- Benefits of modular programming by separating logic into reusable functions.

- The significance of clear error handling and user prompts.

- Understanding how to combine different character sets for strong password creation.

- Learning to shuffle and manipulate lists efficiently for randomness.

FUTURE ENHANCEMENTS

Future enhancements could include adding a user-friendly interface, password strength feedback, and options to customize password rules. Features like saving passwords securely or copying to clipboard would improve usability. Supporting passphrases and multiple languages can also make the tool more versatile.

REFERENCES

- ➢ Official Python documentation for the random module
- ➢ Online tutorials on password generation techniques
- ➢ Python coding style and best practices guides
- ➢ Resources on input validation and error handling in Python
- ➢ Examples from open-source projects related to password generators