Python Password Generator

A

CASE STUDY REPORT

for

Skill Lab – Python Programming

Submitted by

Prasad Pritam Jayakar 20240022009
Yash Anil Vardham 20240022025

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION

- **Introduction: -**

In today’s digital age, where the majority of our interactions take place online, ensuring strong security protocols has become crucial. One fundamental aspect of securing digital platforms is the use of strong and complex passwords. Simple or predictable passwords are highly susceptible to brute-force attacks and unauthorized access. Hence, tools that automatically generate strong, random passwords have become invaluable.
This case study explores a Python-based solution for generating random and customizable passwords. The code allows users to specify password length, as well as the inclusion of uppercase letters, numbers, and special characters. This provides flexibility to meet a variety of security requirements and personal preferences.

- **Literature Survey: -**

Password generation is not a new concept. Over the years, various algorithms and methods have been proposed for creating secure passwords. Some of the main techniques include:

- Random Selection Methods: Passwords are created by randomly selecting characters from predefined sets (e.g., letters, numbers, symbols). The strength of the password depends on the length and variety of characters used.

- Password Strength and Entropy: Password strength is generally measured using entropy, which is the level of unpredictability. A good password generator must balance complexity (i.e., hard to guess) with usability (i.e., easy to remember). The National Institute of Standards and Technology (NIST) guidelines emphasize using longer passphrases with a mix of character types for maximum security.

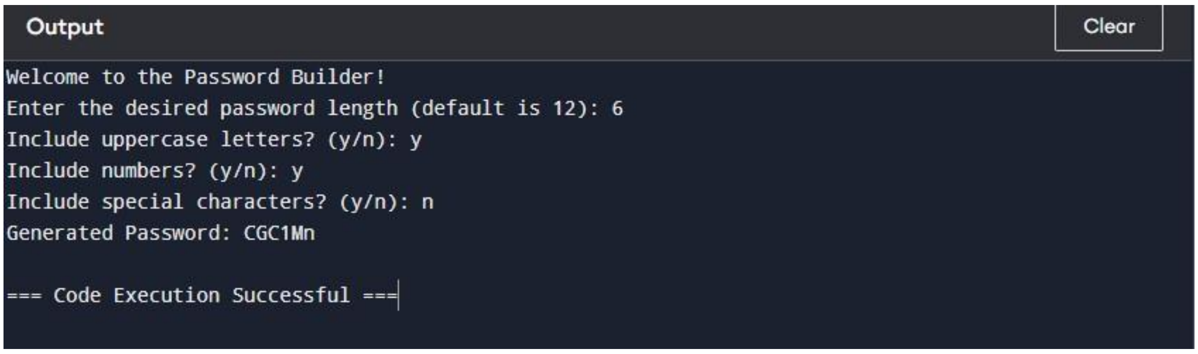
- Password Management Systems: There are dedicated tools like LastPass, KeePass, and 1Password that offer automatic generation and storage of secure passwords. These tools often include customizable parameters, similar to our Python implementation, to cater to different needs.

The Python random library plays a key role in generating pseudorandom numbers, which are essential in producing unpredictable passwords. In addition, the string module provides easy access to predefined character sets like uppercase letters, lowercase letters, digits, and symbols.

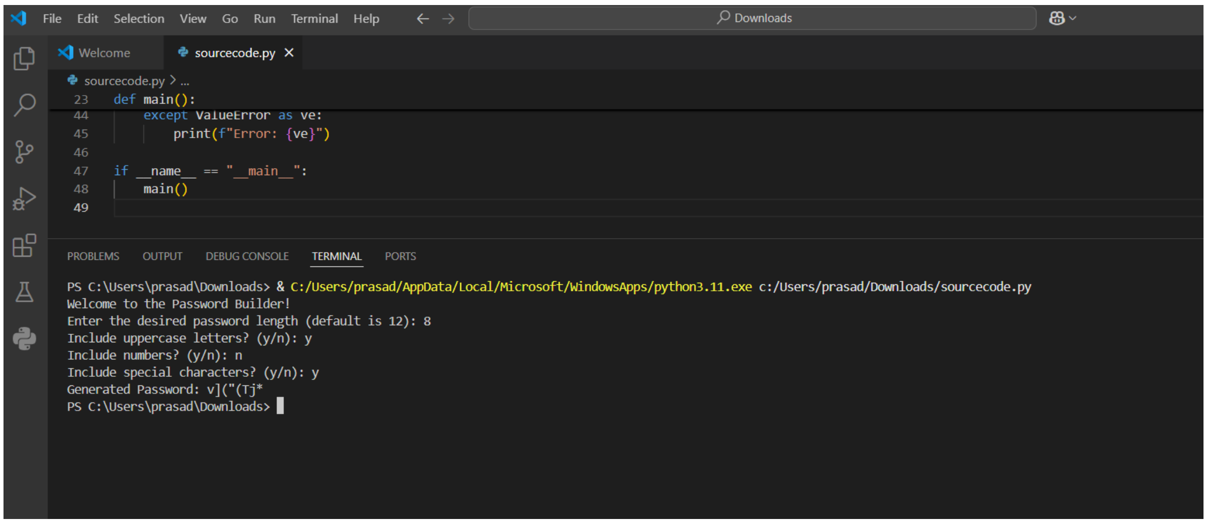
- **Code: -**

```
import random
import string

def generate_password(length=12, use_uppercase=True, use_numbers=True,
use_symbols=True):
   """Generate a random password with specified length and character types."""
   # Base character sets     lowercase
= string.ascii_lowercase      uppercase =
string.ascii_uppercase if use_uppercase else ''
numbers = string.digits if use_numbers else ''     symbols =
string.punctuation if use_symbols else ''

   # Combine all selected character sets
   all_characters = lowercase + uppercase + numbers + symbols

   # Ensure at least one character type is selected     if
len(all_characters) == 0:          raise ValueError("At least one
character type must be selected.")

   # Generate password by randomly picking characters from the selected set
password = ''.join(random.choice(all_characters) for _ in range(length))

   return password

def main():      print("Welcome to the
Password Builder!")

   # User input for password specifications     length = int(input("Enter the desired
password length (default is 12): ") or 12)     use_uppercase = input("Include
uppercase letters? (y/n): ").strip().lower() == 'y'     use_numbers = input("Include
numbers? (y/n): ").strip().lower() == 'y'     use_symbols = input("Include special
characters? (y/n): ").strip().lower() == 'y'

   # Generate and display the password
   password = generate_password(length, use_uppercase, use_numbers, use_symbols)
print(f"Generated Password: {password}")

if __name__ == "__main__":
main()
```

- **Output Screenshot: -**



```
Welcome to the Password Builder!
Enter the desired password length (default is 12): 6
Include uppercase letters? (y/n): y
Include numbers? (y/n): y
Include special characters? (y/n): n
Generated Password: CGC1Mn

=== Code Execution Successful ===
```


File Edit Selection View Go Run Terminal Help
Downloads
Welcome
sourcecode.py
sourcecode.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
def main():
    except ValueError as ve:
        print(f"Error: {ve}")

if __name__ == "__main__":
    main()
```

```
PS C:\Users\prasad\Downloads> & C:/Users/prasad/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/prasad/Downloads/sourcecode.py
Welcome to the Password Builder!
Enter the desired password length (default is 12): 8
Include uppercase letters? (y/n): y
Include numbers? (y/n): n
Include special characters? (y/n): y
Generated Password: v]("(Tj*
PS C:\Users\prasad\Downloads>
```

Code Explain: -

Imports:

The random module is used to select random characters.

The string module provides predefined character sets like lowercase (ascii_lowercase), uppercase (ascii_uppercase), digits, and punctuation symbols.

- Function**: generate_password**:

Parameters:

length: The length of the password (default is 12 characters).

use_uppercase, use_numbers, use_symbols: Boolean flags that allow inclusion/exclusion of uppercase letters, numbers, and symbols, respectively.

Character sets (lowercase, uppercase, numbers, symbols) are dynamically included or excluded based on the user's input.

The selected character sets are combined into all_characters.

If no character types are selected, an error is raised (ValueError).

The password is generated by selecting random characters from all_characters for the specified length.

- Function**: main**:

This is the user interface of the program.

The user is asked to input the desired password length and whether to include uppercase letters, numbers, and special symbols.

Based on the user's choices, the generate_password () function is called to create the password.

The generated password is printed for the user.

- Running **the Program**:

  The program is executed by calling main () when the script is run directly (if __name__ == "__main__":).

- **Bibliography: -**

[1] Python Documentation
Python Software Foundation. "Random — Generate pseudo-random numbers." *Python 3.10.4 Documentation*. https://docs.python.org/3/library/random.html

[2] Password Best Practices
NIST (National Institute of Standards and Technology). "Digital Identity Guidelines: Authentication and Lifecycle Management." NIST Special Publication 800-63B. https://pages.nist.gov/800-63-3/sp800-63b.html

[3] Password Strength and Entropy
Dell'Amico, Matteo, Pietro Michiardi, and Yves Roudier. "Password Strength: An Empirical Analysis." *Proceedings of IEEE INFOCOM 2010*. https://doi.org/10.1109/INFCOM.2010.5461951

[4] Random Character Generation
Fernandez, Jose M., et al. "A Study of Password Generation Techniques." *Information Security Journal: A Global Perspective* (2016). https://doi.org/10.1080/19393555.2016.1152931

[5] Password Management and Security Tools
Schneier, Bruce. "Secrets and Lies: Digital Security in a Networked World." Wiley Publishing, 2000. https://www.schneier.com/books/secrets-lies/