Report on Random Password Generator in Python

Introduction

In this report, we will walk through a Python program that generates a random password based on user-specified length. The program uses a combination of alphabets (both uppercase and lowercase), numbers, and special characters to ensure the generated password is both secure and unpredictable. This password generator follows a 50-30-20 formula for distributing characters between alphabets, numbers, and special symbols. This report will explain the logic behind each step of the code and the reasoning for using specific functions.

Code Overview

The Python script starts by importing necessary modules and defining key variables. The **random** module is used for random selection of characters, while the **math** module helps in rounding operations during the calculation of password length components.

Python Code
import random import math

Character Sets and User Input

The program defines three different sets of characters:

- **alpha** for alphabetic characters (lowercase letters)
- **num** for numerical characters
- **special** for special characters commonly used in passwords like @,#,$,%,&, and *.

Python Code
alpha = "abcdefghijklmnopqrstuvwxyz" num = "0123456789" special = "@#$%&*"

The user is prompted to input the desired password length:

Python Code
pass_len = int(input("Enter Password Length : "))

Distribution of Character Types

The password generator follows a specific distribution of characters:

- **50% alphabets** (rounded down using //2 operation)
- **30% numeric characters** (calculated using math.ceil() for rounding up)
- **20% special characters**, which is the remainder after subtracting the number of alphabets and numeric characters.

Python Code
alpha_len = pass_len // 2 num_len = math.ceil(pass_len * 30 / 100) special_len = pass_len - (alpha_len + num_len)

Password Generation Function

A custom function **generate_pass()** is defined to generate random characters for each type (alphabets, numbers, special characters):

- The function iterates **length** times, where **length** represents how many characters of that type will be added to the password.
- The **random.randint()** function selects a random index within the provided array (alpha, num, or special) to pick a character.
- If the selected characters are alphabets, an additional random selection decides whether to convert the letter to uppercase.

Python Code

def generate_pass(length, array, is_alpha=False):
 	
for i in range(length):
 	
index = random.randint(0, len(array) - 1)
 	
character = array[index]
 	
if is_alpha:
 	
case = random.randint(0, 1)
 	
if case == 1:
 	
character = character.upper()
 	
password.append(character)

Alpha, Numeric, and Special Characters Addition

The program adds characters to the password by calling **generate_pass()** for alphabets, numbers, and special characters. The **is_alpha** argument is set to **True** for alphabetic characters to allow for random uppercasing.

Python Code
Generate alphabets generate_pass(alpha_len, alpha, True) # Generate numeric characters generate_pass(num_len, num) # Generate special characters generate_pass(special_len, special)

Password Shuffling and Final Output

After the password is generated, the list of characters is shuffled randomly to ensure the password's components are not predictable. The program uses the **random.shuffle()** function to rearrange the characters, and then the password list is converted into a single string.

Python Code
Shuffle the generated password list random.shuffle(password) # Convert List to String gen_password = "" for i in password: gen_password = gen_password + str(i) print(gen_password)

Working of the Code

The entire process can be broken down into the following steps:

1. The user specifies the length of the password.

2. The password length is divided into portions for alphabets, numbers, and special characters.
3. The function **generate_pass()** adds the required number of random characters from each category.
4. The list of characters is shuffled to avoid a predictable pattern (i.e., all letters first, followed by numbers, then special characters).
5. Finally, the shuffled password is converted from a list to a string and printed.

Example Execution

When the user runs the program, it might proceed like this:

Output Example
Enter Password Length: 10 Generated Password: 8D#a9S0$Qp

Conclusion

This password generator is a simple yet effective way to create strong, random passwords that follow industry-recommended practices. By using a mix of alphabets, numbers, and special characters, the generated password can be secure against common attacks like brute-force and dictionary-based attempts. The flexibility of specifying password length makes it adaptable to different security requirements.