CHAPTER 1

INTRODUCTION

A random password generator is a software program or hardware device that takes input from a random or pseudo-random number generator and automatically generates a password. Random passwords can be generated manually, using simple sources of randomness such as dice or coins, or they can be generated using a computer**.**

While there are many examples of "random" password generator programs available on the Internet, generating randomness can be tricky, and many programs do not generate random characters in a way that ensures strong security. A common recommendation is to use open source security tools where possible, since they allow independent checks on the quality of the methods used. Simply generating a password at random does not ensure the password is a strong password, because it is possible, although highly unlikely, to generate an easily guessed or cracked password. In fact, there is no need at all for a password to have been produced by a perfectly random process: it just needs to be sufficiently difficult to guess.

A password generator can be part of a password manager. When a password policy enforces complex rules, it can be easier to use a password generator based on that set of rules than to manually create passwords.

Long strings of random characters are difficult for most people to memorize. Mnemonic hashes, which reversibly convert random strings into more memorable passwords, can substantially improve the ease of memorization. As the hash can be processed by a computer to recover the original 60-bit string, it has at least as much information content as the original string.[1] Similar techniques are used in memory sport.

CHAPTER 2

SYSTEM REQUIREMENTS SPECIFICATION

- **Hardware Requirements:**

  Operating system: MS Windows XP or Ubuntu

  RAM: 512 MB

  Hard disk: 5GB

- **Software Requirements:**

  Python 3

  Jupitor notebook

CHAPTER 3

SYSTEM DESIGN

1. Start

2. Store all the characters as a list. Use the sting module of Python to store them all.

3. Ask the user to enter the length of the password.

4. Shuffle the characters using random.shuffle method

5. Initialize an empty list to store the password.

6. Write a loop that iterates length times - Pick a random character from all the characters using random.choice method - Append the random character to the password

7. Shuffle the resultant password list to make it random.

8. Convert the password list to string using the join method.

9. Print the password.

3.1 Flow Diagram:

CHAPTER 4

IMPLEMENTATION

During the implementation process, developers must write enough comments inside the code so that if anybody starts working on the code later, he/she can understand what has already been written. Writing good comments is very important as all other documents, no matter how good they are, will be lost eventually. Ten years after the initial work, you may find only that information which is present inside the code in the form of comments. Development tools also play an important role in this phase of the project. Good development tools save a lot of time for the developers, as well as saving money in terms of improved productivity. The most important development tools for time saving are editors and debuggers. A good editor helps a developer to write code quickly. A good debugger helps make the written code operational in a short period. Before starting the coding process, you should spend some time choosing good development tools.

```
import string
import random

## characters to generate password from
alphabets = list(string.ascii_letters)
digits = list(string.digits)
special_characters = list("!@#$%^&*()")
characters = list(string.ascii_letters + string.digits + "!@#$%^&*()")

def generate_random_password():
    ## length of password from the user
    length = int(input("Enter password length: "))

## number of character types
    alphabets_count = int(input("Enter alphabets count in password: "))
    digits_count = int(input("Enter digits count in password: "))
    special_characters_count = int(input("Enter special characters count in password: "))

    characters_count = alphabets_count + digits_count + special_characters_count


    ## check the total length with characters sum count
    ## print not valid if the sum is greater than length
    if characters_count > length:
        print("Characters total count is greater than the password length")
        return


    ## initializing the password
    password = []
```

```
## picking random alphabets
        for i in range(alphabets_count):
                password.append(random.choice(alphabets))


        ## picking random digits
        for i in range(digits_count):
                password.append(random.choice(digits))


## picking random alphabets
        for i in range(special_characters_count):
                password.append(random.choice(special_characters))


 ## if the total characters count is less than the password length
## add random characters to make it equal to the length
if characters_count < length:
        random.shuffle(characters)


for i in range(length - characters_count):9
         password.append(random.choice(characters))


         ## shuffling the resultant password
         random.shuffle(password)



         ## converting the list to string
         ## printing the list
        print("".join(password))


## invoking the function
generate_random_password()
```

CHAPTER 5

RESULTS

Testing is probably the most important phase for long-term support as well as for the reputation of the company. If you don't control the quality of the software, it will not be able to compete with other products on the market. If software crashes at the customer site, your customer loses productivity as well money and you lose credibility. Sometimes these losses are huge. Unhappy customers will not buy your other products and will not refer other customers to you. You can avoid this situation by doing extensive testing.

Example test case:

Sample Input:

Enter password length:
Enter alphabets count in password:
Enter digits count in password:
Enter special characters count in password:

Expected output:

A randomly generated password with the number of respective characters.

Supplied Input:

Enter password length: 10
Enter alphabets count in password: 3
Enter digits count in password: 2
Enter special characters count in password: 3

Obtained output:

V2(&#XlQq1

If you see the password generated this time, it has the minimum number of characters that the user wants. And the program has included 2 more random characters to make the password length equal to user input.

Hurray! We have a complete strong password generator.

CHAPTER 6

CONCLUSION

With the help of this project one can easily generate a random password anytime he/she wants. This project is very useful for those who are always in a perplexed state about what password to use for a tight security purpose. As these passwords are randomly generated, no one can ever guess them and therefore it provides a tight security be it for lockers or any software etc. which requires protection from outsiders. We can also add many other features to improvise the code as per our requirement and make sure that the resultant password is strong enough.

REFERENCES

1. https://www.w3schools.com.
2. https://www.mysql.com.