# Build an Auto Password Generator with Python

By Nneoma Uche 3 min See views

**Table of Contents**



## Introduction

This is a step-by-step tutorial for building a simple password generator with Python. This project is suitable for an entry-level Python portfolio.

A password generator project in Python highlights core programming concepts like:

- String Manipulation
- Control Flow
- Input/Output Handling
- Randomisation and Character Handling

## Limitations and Exclusions

- This project does not include frontend development, GUI development or CI/CD deployment.
- No persistence or integration: this program can not save generated passwords or be integrated with password management tools yet. End-users are required to record their passwords manually.

## Inside the Development Process

### Import the relevant modules and define the pool of possible characters

Two built-in Python modules come in handy: “string” and “random”. The string module helps define the pool of characters from which the password is generated, while the random module derives a random sequence from the predefined character set.

Still in this step, initialise a variable to hold a list of all the possible characters that can make up a password. The list combines letters (including all upper-case and lower-case alphabets), digits, and some special characters.

```
import string
import random
possible_char = list(string.ascii_letters + string.digits + "!@=-$*&%"
```

If I were to print the `possible_char` variable, a list would appear in the terminal like so:

```
print(possible_char)
```

```
# Ouput: ['a', 'b', 'c', ... 'Z'...1, 2, 3...0...etc]
```

`string.punctuation` functions similarly to `string.ascii_letters and string.digits`

However, that constant in the string module contains virtually all punctuation characters and may riddle the generated passkey with more symbols than letters/digits. Selecting a small pool of special characters (in a string: `“!@=-$*&%"` works well in this case.

### Define a function that only executes if the end-user wants an automatic password

```
def create_password():
```

```
auto = input("Would you like an automatic password? )
```

```
if auto.lower() == "yes":
    create_password()
```

```
elif auto.lower() == "no":
    print("Program ended")
    quit()
```

```
else:
    print("Invalid input. Enter 'Yes/No'")
    quit()
```

The above code will likely raise an error due to improper indentation. But notice that the `create_password()` function is only called or triggered when the user opts for an automatic password.

### Shuffle the `possible_char` list and confirm the password length

Use the ‘Enter’ key to create space between the function itself and the function call within the ‘auto’ block. Shuffle the `possible_char` list to make the password structure unpredictable, then prompt the user for their preferred password length.

random.shuffle(name_of_list_variable) reorganises a mutable sequence like a list, in place.

```
def create_password():
    random.shuffle(possible_char)
    password_length = int(input("Preferred length (8 - 15 characters is ideal): ")
```

`int(input(...))` directly converts the user’s input to an integer. This is necessary for generating a specific number of items.

### Initialise an empty list to store the random password characters

```
password = []
```

### Create a for loop that iterates `password_length` times

Say the user inputs 9 as their preferred password length, the for loop iterates over `possible_char` 9 times, each time selecting and appending a random character from the character set to the empty `password` list

Use random.choice(sequence_variable) to pick a random element from the list of possible characters.

```
for p in range(password_length):
    char = random.choice(possible_char)
    password.append(char)
    random.shuffle(password) #The second shuffle is optional. But it offers an added layer of randomness
```

### Convert the `password` list into a string and print

`passwordString = "".join(password)`

`print(passwordString)`

- The ‘join’ method concatenates the elements of an iterable into a single string
- The empty string (“”) is a delimiter, meaning that the list elements are joined without spacing or symbols between them
- `print(passwordString)` outputs the completed password to the console/terminal

## Complete Code Reference with Screenshots

```python
import string
import random

possible_char = list(string.ascii_letters + string.digits + "!@=-$&*%")

def create_password():
    random.shuffle(possible_char)
    password_length = int(input("Preferred length (8-15 characters is ideal): "))

    password = []
    for p in range(password_length):
        ch = random.choice(possible_char)
        password.append(ch)
```

1st Slide

```python
    random.shuffle(password)
    passwordString = "".join(password)

    print(passwordString)


auto = input("Would you like an automatic password? ")
if auto.lower() == "yes":
    create_password()
elif auto.lower() == "no":
    print("Program ended")
    quit()
else:
    print("Invalid input. Please input 'Yes' or 'No'")
    quit()
```

2nd slide

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\DELL\.vscode> & c:/Users/DELL/.vscode/env/S
Would you like an automatic password? yes
Preferred length (8-15 characters is ideal): 11
8h2RYcZ1zY&
PS C:\Users\DELL\.vscode>
```

# Output in VS Code Terminal when `password_length == 11`