Password Generator

CODE :

```
import string

import random

import tkinter as tk

from tkinter import messagebox, scrolledtext


def generate_random_passwords(character_pool, length,
num_passwords):

    passwords = set()  # Use a set to avoid duplicates

    while len(passwords) < num_passwords:

        password = ''.join(random.choice(character_pool) for _ in
range(length))

        passwords.add(password)

    return list(passwords)


def generate_all_passwords(length=12, use_uppercase=True,
use_lowercase=True, use_digits=True, use_special_chars=True):

    # Create a pool of characters based on user preferences

    character_pool = ''


    if use_uppercase:

        character_pool += string.ascii_uppercase
    if use_lowercase:
        character_pool += string.ascii_lowercase
    if use_digits:
        character_pool += string.digits
    if use_special_chars:
        character_pool += string.punctuation

    # Ensure the character pool is not empty
    if not character_pool:
        raise ValueError("At least one character type must be selected.")

    return character_pool

def save_passwords_to_file(passwords, filename='passwords.txt'):
    with open(filename, 'w', encoding='utf-8') as file:
        for password in passwords:
            file.write(password + '\n')

def generate_passwords():
    try:
        length = int(length_entry.get())
        num_passwords = int(num_passwords_entry.get())  # Get the
number of passwords to generate
        use_uppercase = uppercase_var.get()
        use_lowercase = lowercase_var.get()
        use_digits = digits_var.get()
        use_special_chars = special_chars_var.get()

        # Generate character pool
        character_pool = generate_all_passwords(length,
use_uppercase, use_lowercase, use_digits, use_special_chars)

        # Generate random passwords
        passwords = generate_random_passwords(character_pool,
length, num_passwords)

        # Save passwords to a file
        save_passwords_to_file(passwords)
        output_text.delete(1.0, tk.END)  # Clear previous output
        output_text.insert(tk.END, f"{len(passwords)} random passwords
of length {length} have been saved to 'passwords.txt'.\n")
    except Exception as e:
        messagebox.showerror("Error", str(e))

# Create the main window
root = tk.Tk()
root.title("Password Generator")
root.geometry("800x600")  # Set initial size
root.configure(bg="#f0f0f0")
```

```
# Create input fields
tk.Label(root, text="Password Length:", bg="#f0f0f0", font=("Arial", 12)).grid(row=0, column=0, padx=10, pady=10, sticky='w')
length_entry = tk.Entry(root, width=10, font=("Arial", 12))
length_entry.grid(row=0, column=1, padx=10, pady=10, sticky='ew')

tk.Label(root, text="Number of Passwords:", bg="#f0f0f0", font=("Arial", 12)).grid(row=1, column=0, padx=10, pady=10, sticky='w')
num_passwords_entry = tk.Entry(root, width=10, font=("Arial", 12))
num_passwords_entry.grid(row=1, column=1, padx=10, pady=10, sticky='ew')

uppercase_var = tk.BooleanVar(value=True)
tk.Checkbutton(root, text="Include Uppercase Letters", variable=uppercase_var, bg="#f0f0f0", font=("Arial", 12)).grid(row=2, columnspan=2, padx=10, pady=5, sticky='w')

lowercase_var = tk.BooleanVar(value=True)
tk.Checkbutton(root, text="Include Lowercase Letters", variable=lowercase_var, bg="#f0f0f0", font=("Arial", 12)).grid(row=3, columnspan=2, padx=10, pady=5, sticky='w')

digits_var = tk.BooleanVar(value=True)
```

```
tk.Checkbutton(root, text="Include Digits", variable=digits_var,
bg="#f0f0f0", font=("Arial", 12)).grid(row=4, columnspan=2,
padx=10, pady=5, sticky='w')


special_chars_var = tk.BooleanVar(value=True)

tk.Checkbutton(root, text="Include Special Characters",
variable=special_chars_var, bg="#f0f0f0", font=("Arial",
12)).grid(row=5, columnspan=2, padx=10, pady=5, sticky='w')


# Create buttons frame

button_frame = tk.Frame(root, bg="#f0f0f0")

button_frame.grid(row=6, columnspan=2, pady=20)


# Generate button

generate_button = tk.Button(button_frame, text="Generate
Passwords", command=generate_passwords, bg="#4CAF50",
fg="white", font=("Arial", 12), width=20)

generate_button.grid(row=0, column=0, padx=10)


# Output text area

output_text = scrolledtext.ScrolledText(root, width=80, height=15,
bg="#ffffff", fg="#000000", font=("Arial", 12))

output_text.grid(row=7, columnspan=2, padx=10, pady=10)


for i in range(2):

    root.grid_columnconfigure(i, weight=1)
```

```
# Start the GUI event loop
root.mainloop()
```

Code Explanation :

1. **Imports**: This section imports necessary libraries:

Code :
```
import string
import random
import tkinter as tk
from tkinter import messagebox, scrolledtext
```

- **string**: Provides access to string constants (like uppercase letters, lowercase letters, digits, and punctuation).
- **random**: Used for generating random selections.
- **tkinter**: The main library for creating the GUI.
- **messagebox** and **scrolledtext**: Specific components from **tkinter** for displaying messages and creating scrollable text areas.

2. **Function Definition**: **generate_random_passwords** takes three parameters:

Code :
```
def generate_random_passwords(character_pool, length, num_passwords):
    passwords = set()  # Use a set to avoid duplicates
    while len(passwords) < num_passwords:
            password = ''.join(random.choice(character_pool) for _ in range(length))
            passwords.add(password)
        return list(passwords)
```

- **character_pool**: A string containing all possible characters to use in the passwords.
- **length**: The desired length of each password.
- **num_passwords**: The number of unique passwords to generate.

❖ **Set for Uniqueness**: A set named **passwords** is created to store unique passwords (sets automatically handle duplicates).

❖ **While Loop**: Continues generating passwords until the desired number is reached.

- **Password Generation**: Inside the loop, a password is created by randomly selecting characters from **character_pool** for the specified **length**.
- **Add to Set**: The generated password is added to the **passwords** set.

❖ **Return Statement**: Finally, the function returns a list of unique passwords.

3. **Function Definition**: **generate_all_passwords** creates a character pool based on user preferences.

Code : def generate_all_passwords(length=12, use_uppercase=True, use_lowercase=True, use_digits=True, use_special_chars=True):

```
        character_pool = ''
```

- ❖ **Parameters**:
  - **length**: Default is 12 (not used in this function but can be useful for future modifications).
  - **use_uppercase**, **use_lowercase**, **use_digits**, **use_special_chars**: Boolean flags indicating whether to include these character types in the pool.
- ❖ **Character Pool Initialization**: An empty string **character_pool** is initialized to build the pool of characters.

4. **Conditional Statements**: Each **if** statement checks the corresponding flag. If **True**, it appends the relevant character set (uppercase letters, lowercase letters, digits, or special characters) to **character_pool**.

Code : if use_uppercase:

```
        character_pool += string.ascii_uppercase
    if use_lowercase:
        character_pool += string.ascii_lowercase
    if use_digits:
        character_pool += string.digits
    if use_special_chars:
        character_pool += string.punctuation
```

5. **Validation**: Checks if **character_pool** is empty. If it is, a **ValueError** is raised, indicating that at least one character type must be selected.

Code :
```
if not character_pool:
            raise ValueError("At least one character type must be selected.")
```

6. **Return Statement**: The function returns ththe constructed ‘character_pool’.

Code :
```
return character_pool
```

7. **Function Definition**: **save_passwords_to_file** takes two parameters:

Code :
```
def save_passwords_to_file(passwords, filename='passwords.txt'):
            with open(filename, 'w', encoding='utf-8') as file:
                  for password in passwords:
                        file.write(password + '\n')
```

- **passwords**: A list of passwords to save.
- **filename**: The name of the file to save the passwords to (default is 'passwords.txt').

❖ **File Handling**: Opens the specified file in write mode ('**w**').
❖ **Loop Through Passwords**: Iterates through each password in the **passwords** list and writes it to the file, followed by a newline character.

8. **Function Definition**: **generate_passwords** is the main function that orchestrates the password generation process.

Code :
```
def generate_passwords():
        try:
                length = int(length_entry.get())
                num_passwords = int(num_passwords_entry.get
                use_uppercase = uppercase_var.get()
                use_lowercase = lowercase_var.get()
                use_digits = digits_var.get()
                use_special_chars = special_chars_var.get()
```

❖ **Try Block**: Used to handle exceptions that may occur during execution.
❖ **Input Retrieval**:

- **length**: Gets the desired password length from the **length_entry** input field.
- **num_passwords**: Gets the number of passwords to generate from the **num_passwords_entry** input field.
- The boolean variables (**uppercase_var**, **lowercase_var**, **digits_var**, **special_chars_var**) are retrieved to determine which character types to include.

9. **Character Pool Generation**: Calls **generate_all_passwords** to create the character pool based on user preferences.

Code : character_pool = generate_all_passwords(length, use_uppercase, use_lowercase, use_digits, use_special_chars)

10. **Password Generation**: Calls **generate_random_passwords** to generate the specified number of passwords using the created character pool.

Code : passwords = generate_random_passwords(character_pool, length, num_passwords)

11. **Save Passwords**: Calls **save_passwords_to_file** to save the generated passwords to a file.

Code : save_passwords_to_file(passwords)

        output_text.delete(1.0, tk.END) # Clear previous output

        output_text.insert(tk.END, f"{len(passwords)} random passwords of length {length} have been saved to 'passwords.txt'.\n"

- ❖ **Output Text Area**: Clears any previous output in the **output_text** area and inserts a message indicating how many passwords were generated and saved.

12. **Exception Handling**: If any error occurs during the execution of the **try** block, a message box is displayed with the error message.

Code : except Exception as e:

        messagebox.showerror("Error", str(e))

13. **Main Window Creation**: Initializes the main application window using **tkinter**.

Code : root = tk.Tk()

```
root.title("Password Generator")

root.geometry("800x600")  # Set initial size

root.configure(bg="#f0f0f0")  # Set background color
```

- ❖ **Title and Size**: Sets the title of the window and its initial size.
- ❖ **Background Color**: Configures the background color of the window.

14. **Label and Entry for Password Length**: Creates a label and an entry field for the user to input the desired password length. The label is placed in the first column, and the entry field is placed in the second column of the grid layout.

Code : tk.Label(root, text="Password Length:", bg="#f0f0f0", font=("Arial", 12)).grid(row=0, column=0, padx=10, pady=10, sticky='w')

```
length_entry = tk.Entry(root, width=10, font=("Arial", 12))

length_entry.grid(row=0, column=1, padx=10, pady=10,
sticky='ew')
```

15. **Label and Entry for Number of Passwords**: Similar to the previous section, this creates a label and entry field for the user to input the number of passwords to generate.

Code : tk.Label(root, text="Number of Passwords:", bg="#f0f0f0", font=("Arial", 12)).grid(row=1, column=0, padx=10, pady=10, sticky='w')

num_passwords_entry = tk.Entry(root, width=10, font=("Arial", 12))

num_passwords_entry.grid(row=1, column=1, padx=10, pady=10, sticky='ew')

16. **Checkbox for Uppercase Letters**: Creates a checkbox that allows the user to select whether to include uppercase letters in the generated passwords. The state of the checkbox is stored in **uppercase_var**.

Code : uppercase_var = tk.BooleanVar(value=True)

tk.Checkbutton(root, text="Include Uppercase Letters", variable=uppercase_var, bg="#f0f0f0", font=("Arial", 12)).grid(row=2, columnspan=2, padx=10, pady=5, sticky='w')

17. **Checkbox for Lowercase Letters**: Similar to the uppercase checkbox, this allows the user to select whether to include lowercase letters.

Code : lowercase_var = tk.BooleanVar(value=True)

tk.Checkbutton(root, text="Include Lowercase Letters", variable=lowercase_var, bg="#f0f0f0", font=("Arial", 12)).grid(row=3, columnspan=2, padx=10, pady=5, sticky='w')

18. **Checkbox for Digits**: Allows the user to select whether to include digits in the generated passwords.

Code : digits_var = tk.BooleanVar(value=True)

```
tk.Checkbutton(root, text="Include Digits",
variable=digits_var, bg="#f0f0f0", font=("Arial", 12)).grid(row=4,
columnspan=2, padx=10, pady=5, sticky='w')
```

19. **Checkbox for Special Characters**: Allows the user to select whether to include special characters in the generated passwords.

Code : special_chars_var = tk.BooleanVar(value=True)

```
tk.Checkbutton(root, text="Include Special Characters",
variable=special_chars_var, bg="#f0f0f0", font=("Arial",
12)).grid(row=5, columnspan=2, padx=10, pady=5, sticky='w')
```

20. **Button Frame**: Creates a frame to hold the buttons, which helps in organizing the layout.

Code : button_frame = tk.Frame(root, bg="#f0f0f0")

```
button_frame.grid(row=6, columnspan=2, pady=20)
```

21. **Generate Button**: Creates a button that, when clicked, calls the **generate_passwords** function. The button is styled with a green background and white text.

Code : generate_button = tk.Button(button_frame, text="Generate Passwords", command=generate_passwords, bg="#4CAF50", fg="white", font=("Arial", 12), width=20)

```
generate_button.grid(row=0, column=0, padx=10)
```

22. **Output Text Area**: Creates a scrollable text area to display messages, such as the number of passwords generated. It is styled with a white background and black text.

Code : output_text = scrolledtext.ScrolledText(root, width=80, height=15, bg="#ffffff", fg="#000000", font=("Arial", 12))

output_text.grid(row=7, columnspan=2, padx=10, pady=10)

23. **Column Configuration**: Configures the grid layout to ensure that the columns are centered and evenly spaced.

Code : for i in range(2):

root.grid_columnconfigure(i, weight=1)

24. **Main Loop**: Starts the GUI event loop, which waits for user interactions and keeps the application running.

Code : root.mainloop()

Steps to Use the Password Generator Program

1. **Launch the Application**:
   - Run the Python script containing the password generator code. This will open the GUI window for the application.

2. **Set Password Length**:
   - In the field labeled "Password Length:", enter the desired length for the passwords you want to generate. This should be a positive integer.

3. **Set Number of Passwords**:
   - In the field labeled "Number of Passwords:", enter the number of unique passwords you wish to generate. Again, this should be a positive integer.

4. **Select Character Types**:
   - Below the input fields, you will see several checkboxes:
     - **Include Uppercase Letters**: Check this box if you want the generated passwords to include uppercase letters (A-Z).

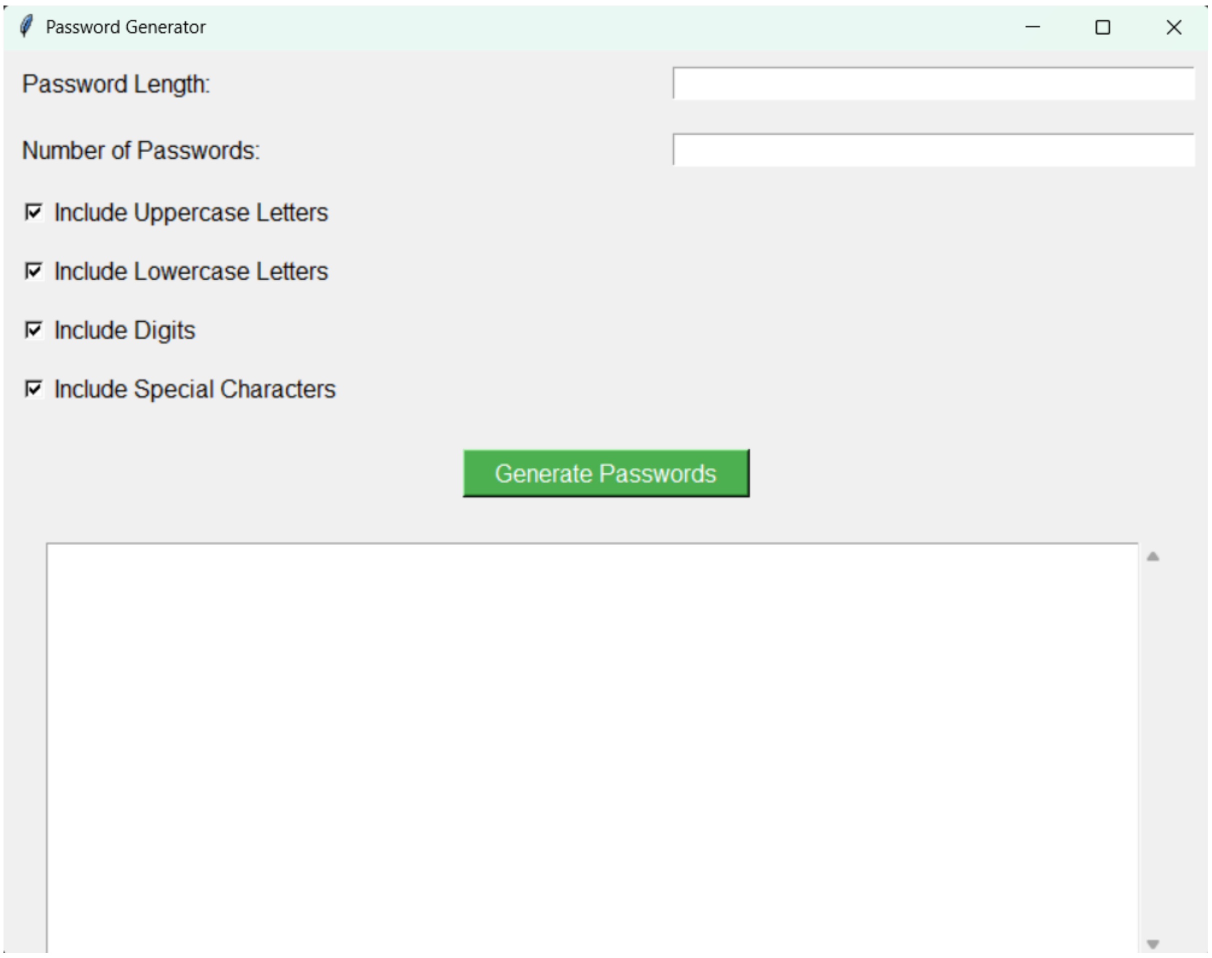
        - **Include Lowercase Letters**: Check this box if you want the generated passwords to include lowercase letters (a-z).
        - **Include Digits**: Check this box if you want the generated passwords to include digits (0-9).
        - **Include Special Characters**: Check this box if you want the generated passwords to include special characters (e.g., !, @, #, $, etc.).
    - You can select any combination of these options based on your password requirements.

5. **Generate Passwords**:
    - Click the **"Generate Passwords"** button. This will trigger the password generation process based on the criteria you have set.
    - The program will generate the specified number of passwords using the selected character types and length.

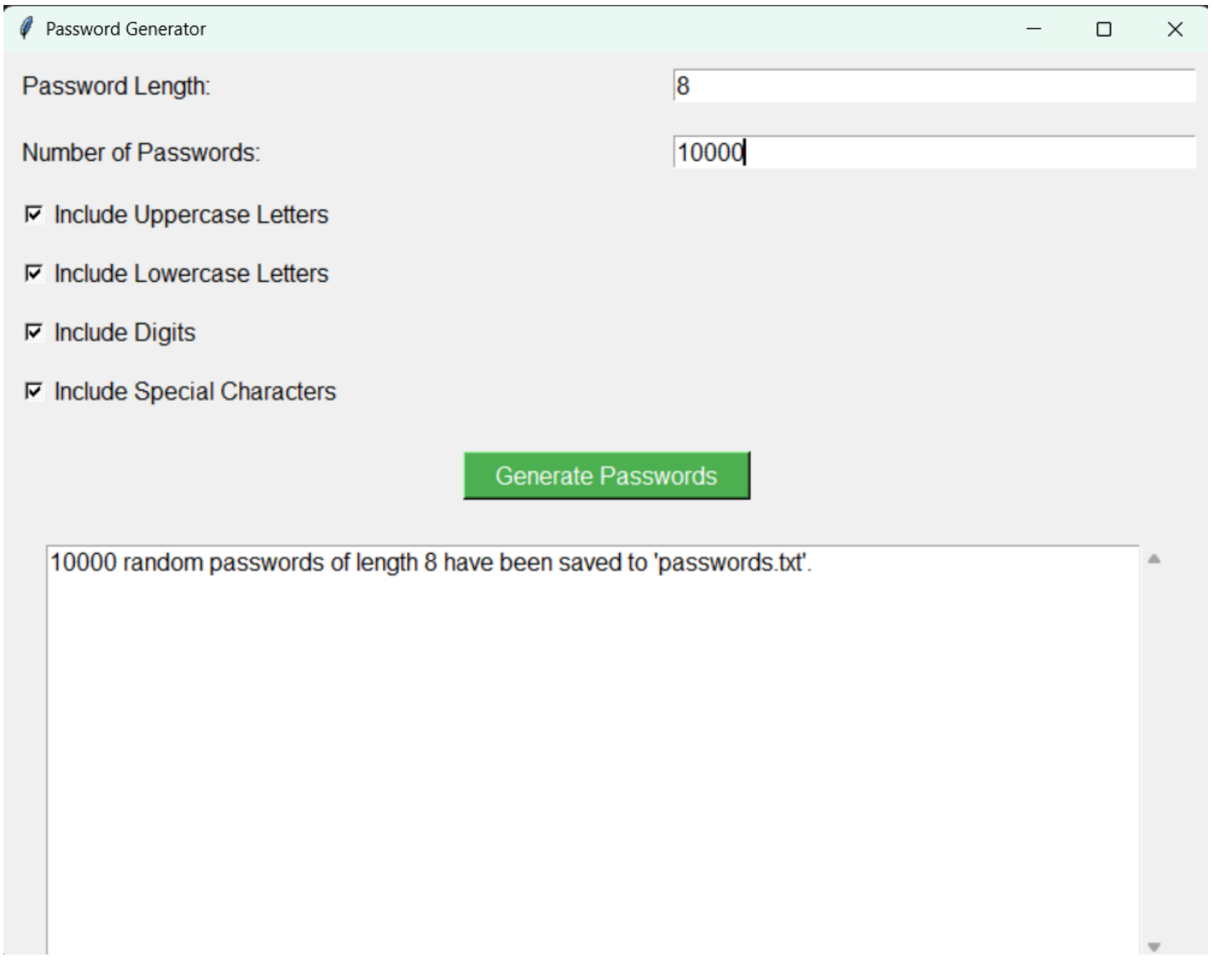
6. **View Results**:
    - After the passwords are generated, a message will appear in the output text area indicating how many passwords were generated and saved to a file named **passwords.txt**.

Screenshots :

1. This image shows password generator tool gui



2. User set password length to 8 and number of passwords will generate is 10000, generated passwords will save in passwords.txt file



Password Generator

Password Length: 8

Number of Passwords: 10000

☑ Include Uppercase Letters

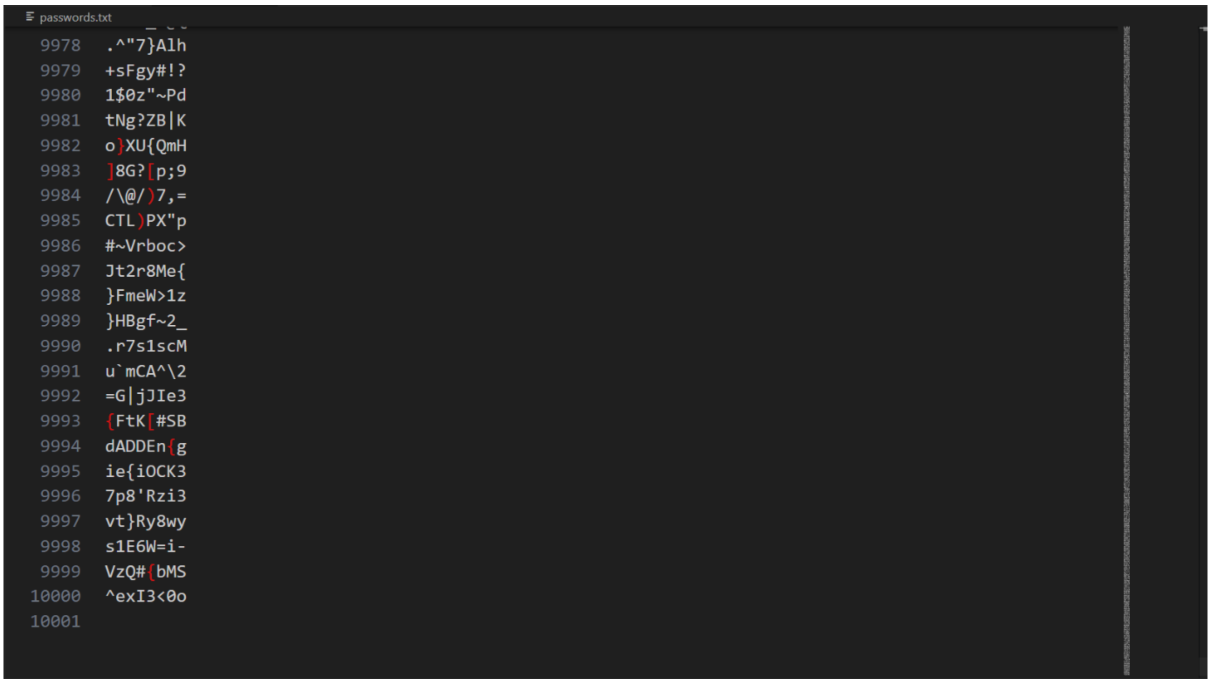
☑ Include Lowercase Letters

☑ Include Digits

☑ Include Special Characters

```
10000 random passwords of length 8 have been saved to 'passwords.txt'.
```

3. This image is of text file with 10000 passwords saved


passwords.txt

```
 9978   .^"7}Alh
 9979   +sFgy#!?
 9980   1$0z"~Pd
 9981   tNg?ZB|K
 9982   o}XU{QmH
 9983   ]8G?[p;9
 9984   /\@/)7,=
 9985   CTL)PX"p
 9986   #~Vrboc>
 9987   Jt2r8Me{
 9988   }FmeW>1z
 9989   }HBgf~2_
 9990   .r7s1scM
 9991   u`mCA^\2
 9992   =G|jJIe3
 9993   {FtK[#SB
 9994   dADDEn{g
 9995   ie{iOCK3
 9996   7p8'Rzi3
 9997   vt}Ry8wy
 9998   s1E6W=i-
 9999   VzQ#{bMS
10000   ^exI3<0o
10001
```