Python Project 06: Password Generator

1. Project Overview

A **Password Generator** creates strong random passwords based on user-selected options such as password length, uppercase letters, lowercase letters, numbers, and symbols.

The project has two main parts:

- **Frontend:** Tkinter graphical interface
- **Backend:** Password generation and validation logic

Python's `secrets` module is used instead of `random` because it is designed for generating security-sensitive random values such as passwords and authentication tokens.

Tkinter provides Python's standard interface for building desktop graphical applications.

---

2. Features

- Custom password length
- Uppercase letters
- Lowercase letters
- Numbers
- Special symbols
- Secure random generation
- Generate button
- Copy password button
- Show/Hide password
- Input validation

- Clean dark interface
- No external packages required

---

3. Technologies

- Python 3
- Tkinter
- secrets
- string
- Object Oriented Programming

---

4. Folder Structure

```
password_generator/
□□□ main.py
□□□ ui.py
□□□ generator.py
□□□ README.md
```

File Responsibilities

File	Responsibility
main.py	Starts the application
ui.py	Handles the graphical interface

generator.py	Handles password generation
README.md	Project documentation

---

5. How the Project Works

The application follows this flow:

```
User selects password options
        ↓
Frontend collects settings
        ↓
PasswordGenerator
        ↓
Backend validates settings
        ↓
secrets.choice()
        ↓
Password generated
        ↓
Frontend displays password
```

The frontend does not contain the password-generation algorithm. It simply collects user input and sends it to the backend.

This separation makes the project easier to understand and maintain.

---

6. Complete Backend Code

generator.py

```
import secrets
import string


class PasswordGenerator:
    def generate(
        self,
        length,
        use_uppercase=True,
        use_lowercase=True,
        use_numbers=True,
        use_symbols=True
    ):
        if length < 4:
            raise ValueError(
                "Password length must be at least 4."
            )

        selected_sets = []

        if use_uppercase:
            selected_sets.append(string.ascii_uppercase)

        if use_lowercase:
            selected_sets.append(string.ascii_lowercase)

        if use_numbers:
            selected_sets.append(string.digits)

        if use_symbols:
```

```
        selected_sets.append(string.punctuation)

    if not selected_sets:
        raise ValueError(
            "Select at least one character type."
        )

    if length < len(selected_sets):
        raise ValueError(
            "Password length is too short for the selected options."
        )

    # Make sure every selected character category
    # appears at least once.
    password_characters = [
        secrets.choice(characters)
        for characters in selected_sets
    ]

    # Combine all selected character sets.
    all_characters = "".join(selected_sets)

    # Fill the remaining password positions.
    remaining_length = length - len(password_characters)

    password_characters.extend(
        secrets.choice(all_characters)
        for _ in range(remaining_length)
    )

    # Securely shuffle the generated characters.
    for index in range(len(password_characters) - 1, 0, -1):
        random_index = secrets.randbelow(index + 1)
```

```
            password_characters[index], password_characters[random_index] = (
                password_characters[random_index],
                password_characters[index]
            )

        return "".join(password_characters)
```

Backend Responsibilities

The backend:

1. Checks the password length.
2. Checks which character types are selected.
3. Makes sure at least one character type is selected.
4. Adds at least one character from every selected category.
5. Fills the remaining positions.
6. Securely shuffles the result.
7. Returns the final password.

`secrets.choice()` selects a random item from a sequence, while `secrets.randbelow()` provides secure random numbers suitable for this type of use.

---

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import messagebox

from generator import PasswordGenerator
```

```
class PasswordGeneratorUI:

    BG = "#0F172A"
    CARD = "#1E293B"
    INPUT = "#334155"
    TEXT = "#F8FAFC"
    MUTED = "#94A3B8"
    ACCENT = "#38BDF8"
    SUCCESS = "#22C55E"

    def __init__(self, root):
        self.root = root

        self.root.title("Password Generator")
        self.root.geometry("520x620")
        self.root.resizable(False, False)
        self.root.configure(bg=self.BG)

        self.password_visible = True

        self.length_var = tk.IntVar(value=16)

        self.upper_var = tk.BooleanVar(value=True)
        self.lower_var = tk.BooleanVar(value=True)
        self.number_var = tk.BooleanVar(value=True)
        self.symbol_var = tk.BooleanVar(value=True)

        self.build_ui()

    def build_ui(self):

        container = tk.Frame(
            self.root,
            bg=self.BG,
```

```
            padx=35,
            pady=30
        )

        container.pack(
            fill="both",
            expand=True
        )

        # Title

        tk.Label(
            container,
            text="Password Generator",
            font=("Segoe UI", 24, "bold"),
            bg=self.BG,
            fg=self.TEXT
        ).pack(pady=(0, 5))

        tk.Label(
            container,
            text="Create a strong random password",
            font=("Segoe UI", 11),
            bg=self.BG,
            fg=self.MUTED
        ).pack(pady=(0, 25))

        # Password display

        password_card = tk.Frame(
            container,
            bg=self.CARD,
            padx=15,
            pady=15
```

```
        )

        password_card.pack(
            fill="x",
            pady=(0, 25)
        )

        self.password_var = tk.StringVar(
            value="Your password will appear here"
        )

        self.password_entry = tk.Entry(
            password_card,
            textvariable=self.password_var,
            font=("Consolas", 14),
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat",
            justify="center"
        )

        self.password_entry.pack(
            side="left",
            fill="x",
            expand=True,
            ipady=12
        )

        self.visibility_button = tk.Button(
            password_card,
            text="Hide",
            command=self.toggle_visibility,
            bg=self.CARD,
```

```
            fg=self.ACCENT,
            activebackground=self.CARD,
            activeforeground=self.TEXT,
            relief="flat",
            borderwidth=0,
            font=("Segoe UI", 10, "bold"),
            cursor="hand2"
        )

        self.visibility_button.pack(
            side="right",
            padx=(10, 0)
        )

        # Password length

        tk.Label(
            container,
            text="Password Length",
            font=("Segoe UI", 11, "bold"),
            bg=self.BG,
            fg=self.TEXT
        ).pack(anchor="w")

        length_frame = tk.Frame(
            container,
            bg=self.BG
        )

        length_frame.pack(
            fill="x",
            pady=(8, 20)
        )
```

```
        self.length_scale = tk.Scale(
            length_frame,
            from_=4,
            to=64,
            orient="horizontal",
            variable=self.length_var,
            bg=self.BG,
            fg=self.TEXT,
            troughcolor=self.INPUT,
            highlightthickness=0,
            activebackground=self.ACCENT,
            length=400
        )

        self.length_scale.pack(side="left")

        self.length_label = tk.Label(
            length_frame,
            text="16",
            font=("Segoe UI", 12, "bold"),
            bg=self.BG,
            fg=self.ACCENT,
            width=3
        )

        self.length_label.pack(side="right")

        self.length_var.trace_add(
            "write",
            self.update_length_label
        )

        # Character options
```

```
        tk.Label(
            container,
            text="Character Types",
            font=("Segoe UI", 11, "bold"),
            bg=self.BG,
            fg=self.TEXT
        ).pack(
            anchor="w",
            pady=(0, 8)
        )

        options = [
            ("Uppercase Letters  A-Z", self.upper_var),
            ("Lowercase Letters  a-z", self.lower_var),
            ("Numbers  0-9", self.number_var),
            ("Symbols  !@#$%", self.symbol_var)
        ]

        for text, variable in options:

            tk.Checkbutton(
                container,
                text=text,
                variable=variable,
                bg=self.BG,
                fg=self.TEXT,
                activebackground=self.BG,
                activeforeground=self.TEXT,
                selectcolor=self.INPUT,
                font=("Segoe UI", 10),
                anchor="w"
            ).pack(
                fill="x",
                pady=3
```

```
        )

        # Generate button

        tk.Button(
            container,
            text="Generate Password",
            command=self.generate_password,
            bg=self.ACCENT,
            fg="#0F172A",
            activebackground="#7DD3FC",
            activeforeground="#0F172A",
            font=("Segoe UI", 12, "bold"),
            relief="flat",
            borderwidth=0,
            cursor="hand2"
        ).pack(
            fill="x",
            pady=(25, 10),
            ipady=10
        )

        # Copy button

        tk.Button(
            container,
            text="Copy Password",
            command=self.copy_password,
            bg=self.INPUT,
            fg=self.TEXT,
            activebackground="#475569",
            activeforeground=self.TEXT,
            font=("Segoe UI", 11, "bold"),
            relief="flat",
```

```
            borderwidth=0,
            cursor="hand2"
        ).pack(
            fill="x",
            ipady=9
        )

        # Status

        self.status_label = tk.Label(
            container,
            text="",
            font=("Segoe UI", 10),
            bg=self.BG,
            fg=self.MUTED
        )

        self.status_label.pack(
            pady=(15, 0)
        )

    def update_length_label(self, *args):

        self.length_label.config(
            text=str(self.length_var.get())
        )

    def generate_password(self):

        try:

            generator = PasswordGenerator()

            password = generator.generate(
```

```
                length=self.length_var.get(),
                use_uppercase=self.upper_var.get(),
                use_lowercase=self.lower_var.get(),
                use_numbers=self.number_var.get(),
                use_symbols=self.symbol_var.get()
            )

            self.password_var.set(password)

            self.status_label.config(
                text="Password generated successfully.",
                fg=self.SUCCESS
            )

        except ValueError as error:

            messagebox.showerror(
                "Invalid Settings",
                str(error)
            )

    def copy_password(self):

        password = self.password_var.get()

        if (
            not password
            or password == "Your password will appear here"
        ):
            messagebox.showwarning(
                "No Password",
                "Generate a password first."
            )
            return
```

```
        self.root.clipboard_clear()

        self.root.clipboard_append(password)

        self.root.update()

        self.status_label.config(
            text="Password copied to clipboard.",
            fg=self.SUCCESS
        )

    def toggle_visibility(self):

        if self.password_visible:

            self.password_entry.config(
                show="•"
            )

            self.visibility_button.config(
                text="Show"
            )

            self.password_visible = False

        else:

            self.password_entry.config(
                show=""
            )

            self.visibility_button.config(
                text="Hide"
```

```
        )
        self.password_visible = True
```

---

8. Main Application Code

main.py

```
import tkinter as tk
from ui import PasswordGeneratorUI


def main():
    root = tk.Tk()
    PasswordGeneratorUI(root)
    root.mainloop()


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

---

9. How Frontend and Backend Connect

The connection happens inside `ui.py`.

When the user clicks **Generate Password**, this method runs:

```
def generate_password(self):

    generator = PasswordGenerator()

    password = generator.generate(
        length=self.length_var.get(),
        use_uppercase=self.upper_var.get(),
        use_lowercase=self.lower_var.get(),
        use_numbers=self.number_var.get(),
        use_symbols=self.symbol_var.get()
    )

    self.password_var.set(password)
```

The process is:

```
Generate Password button
        ↓
generate_password()
        ↓
PasswordGenerator()
        ↓
generator.generate()
        ↓
Backend creates password
        ↓
password returned
        ↓
password_var.set()
        ↓
Password displayed
```

This gives the project a simple separation:

Frontend

User Input

Backend

Password Logic

Result

Frontend

---

10. How to Run

Open the terminal inside the project folder:

```
cd password_generator
```

Then run:

```
python main.py
```

No external packages are required.

Tkinter is included with standard Python installations on many platforms and provides the GUI functionality used by this project.

---

11. Basic Testing

Test 1: Default Password

Run the application and click:

```
Generate Password
```

Expected result:

```
A random 16-character password
```

Test 2: Password Length

Move the length slider to:

```
24
```

Generate a password.

Expected result:

```
Password contains 24 characters.
```

Test 3: Character Options

Enable:

```
Uppercase
Lowercase
Numbers
Symbols
```

Generate a password.

Expected result:

`Password contains characters from the selected categories.`

Test 4: Copy

Generate a password and click:

`Copy Password`

Expected result:

`Password copied to clipboard.`

Test 5: Show/Hide

Click:

`Hide`

Expected result:

`Password characters become hidden.`

Click again:

`Show`

Expected result:

`Password becomes visible again.`

---

12. Small Improvements

The project can later be extended with:

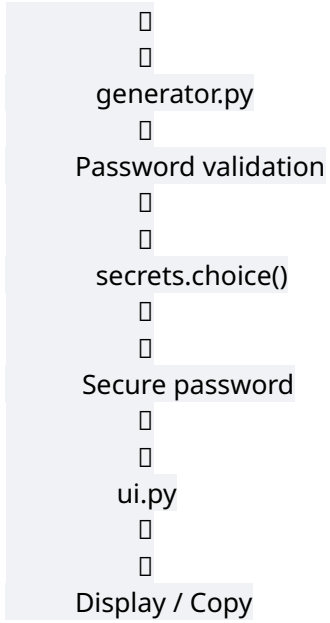
- Password strength indicator
- Password history
- Save generated passwords
- Exclude similar characters such as `O`, `0`, `l`, and `1`
- Passphrase generation
- Custom symbol selection
- Dark/light themes
- Password expiration timer
- Multiple password generation
- Export passwords to a file

For real applications, generated passwords should be handled carefully and should not be unnecessarily stored in plain text. Python's documentation specifically recommends secure password handling rather than storing recoverable passwords.

---

13. Project Architecture

```
        main.py
          
          
        ui.py
          
    User selects options
```



The main concept of this project is:

User Input → Validation → Business Logic → Result → UI

This is the same separation pattern used in larger applications, where the interface and business logic should not be unnecessarily mixed.

Key Takeaways

- Use Tkinter to build a Python desktop interface.
- Keep password-generation logic separate from the UI.
- Validate user-selected options before generating a password.
- Use `secrets` for security-sensitive random generation.

- Keep the project modular with separate files.
- Connect frontend and backend through clear methods and return values.

Visit haas.dev for more resources and guides.