SecurePass : Advanced Password Generator

Project Overview

The Secure Password Manager application is a desktop-based tool that helps users generate strong passwords, store them securely, and manage credentials efficiently. It focuses on encryption, password policies, and user-friendly interaction. This project ensures secure handling of sensitive information while providing an intuitive GUI.

Key Features:

- Developed using Python, Tkinter, SQLite, and Cryptography library.
- Generates random, strong passwords with customizable length.
- Stores encrypted passwords in a local database securely.

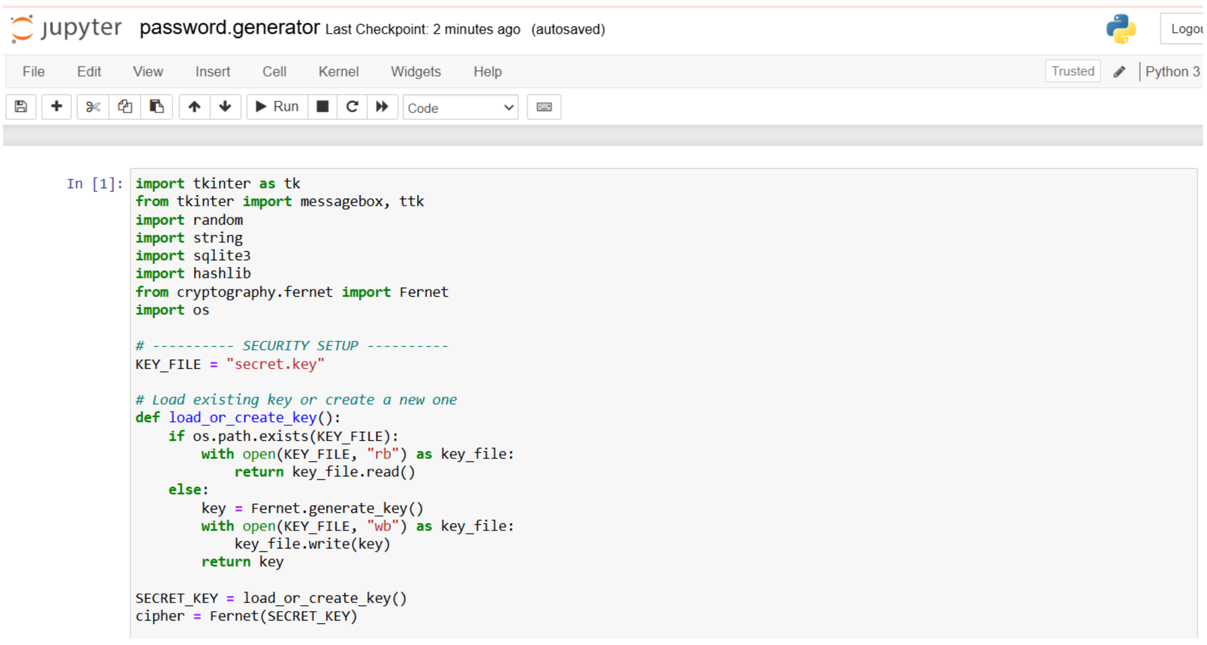
Security Setup

The application uses a secret key to encrypt stored passwords and SHA-256 hashing for secure login verification. A key file is created automatically if not present, ensuring encryption can be used safely without manual setup.

Key Features:

- Generates or loads a Fernet key for encryption.
- Encrypts passwords before saving to the database.
- Ensures no sensitive data is stored in plain text.

Code:


jupyter password.generator Last Checkpoint: 2 minutes ago (autosaved)
Logou
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [1]:
```
import tkinter as tk
from tkinter import messagebox, ttk
import random
import string
import sqlite3
import hashlib
from cryptography.fernet import Fernet
import os

# ---------- SECURITY SETUP ----------
KEY_FILE = "secret.key"

# Load existing key or create a new one
def load_or_create_key():
    if os.path.exists(KEY_FILE):
        with open(KEY_FILE, "rb") as key_file:
            return key_file.read()
    else:
        key = Fernet.generate_key()
        with open(KEY_FILE, "wb") as key_file:
            key_file.write(key)
        return key

SECRET_KEY = load_or_create_key()
cipher = Fernet(SECRET_KEY)
```

Figure : Security Key Setup

Login and Authentication

A login screen ensures that only authorized users can access the password generator. Passwords are hashed, input is masked, and verification is done securely without storing plaintext credentials.

Key Features:

- Hardcoded username with hashed password (SHA-256).
- Masked password input for privacy.
- Prevents unauthorized access to password management features.

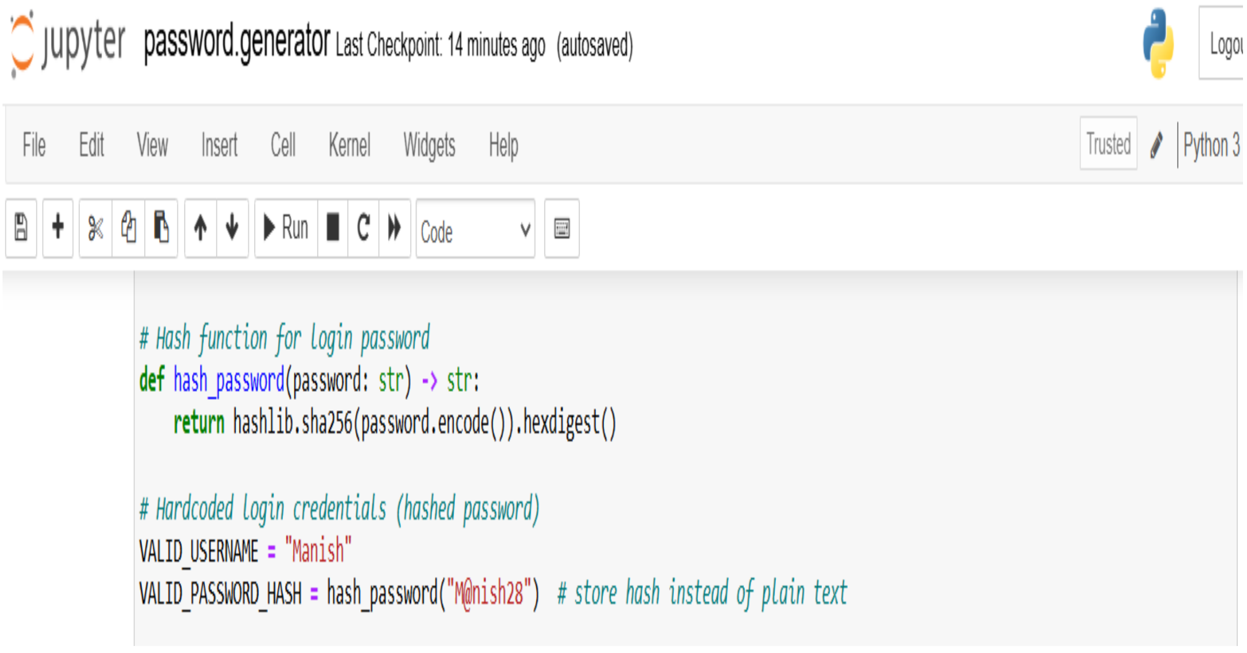
Password Generator

Users can generate passwords of custom length including uppercase, lowercase, digits, and special characters. Randomization ensures each password is unique and strong.

Key Features:

- Fully customizable password length.
- Uses random.sample for unpredictable passwords.
- Supports letters, numbers, and symbols for strong security.

Code:


jupyter password.generator Last Checkpoint: 14 minutes ago (autosaved)
Logou
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```python
# Hash function for login password
def hash_password(password: str) -> str:
    return hashlib.sha256(password.encode()).hexdigest()

# Hardcoded login credentials (hashed password)
VALID_USERNAME = "Manish"
VALID_PASSWORD_HASH = hash_password("M@nish28")  # store hash instead of plain text
```

Figure: Password Generator Setup

Password Strength Meter

A real-time strength meter evaluates the complexity of the password. Strength is shown as Weak, Medium, or Strong based on length, inclusion of letters, numbers, and special characters.

Key Features:

- Dynamic progress bar updates with each key press or generated password.
- Provides instant visual feedback for password improvement.
- Educates users on creating secure passwords.

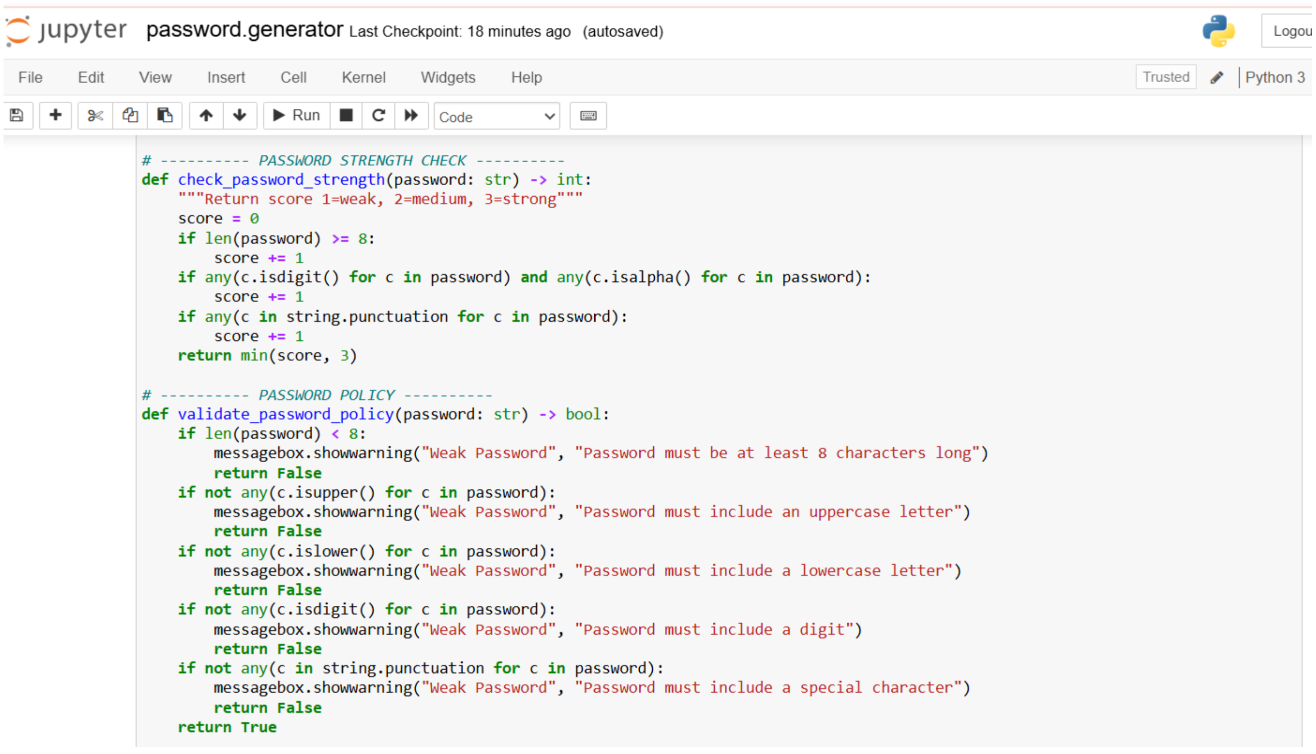
Password Policy Enforcement

Before saving, passwords are validated against a strict security policy. Violations trigger warnings guiding the user to strengthen the password.

Key Features:

- Minimum 8 characters, must include uppercase, lowercase, digit, and special character.
- Prevents saving weak or insecure passwords.
- Ensures all stored passwords follow best security practices.

Code:



```
# ---------- PASSWORD STRENGTH CHECK ----------
def check_password_strength(password: str) -> int:
    """Return score 1=weak, 2=medium, 3=strong"""
    score = 0
    if len(password) >= 8:
        score += 1
    if any(c.isdigit() for c in password) and any(c.isalpha() for c in password):
        score += 1
    if any(c in string.punctuation for c in password):
        score += 1
    return min(score, 3)

# ---------- PASSWORD POLICY ----------
def validate_password_policy(password: str) -> bool:
    if len(password) < 8:
        messagebox.showwarning("Weak Password", "Password must be at least 8 characters long")
        return False
    if not any(c.isupper() for c in password):
        messagebox.showwarning("Weak Password", "Password must include an uppercase letter")
        return False
    if not any(c.islower() for c in password):
        messagebox.showwarning("Weak Password", "Password must include a lowercase letter")
        return False
    if not any(c.isdigit() for c in password):
        messagebox.showwarning("Weak Password", "Password must include a digit")
        return False
    if not any(c in string.punctuation for c in password):
        messagebox.showwarning("Weak Password", "Password must include a special character")
        return False
    return True
```

Figure : Password Strength Meter

Main Password Generator Application

The password generator application is designed to create, evaluate, and securely store passwords. The GUI is implemented using Tkinter, with fields for username, password length, and generated password. Users can generate strong passwords, check their strength, save them securely, and view saved records.

Key Features:

- Tkinter window titled "Password Generator" with size 600x500 and orange background.
- Labels and entry fields for username, password length, and generated password.
- Dynamic password strength meter using ttk.Progressbar.
- SQLite database integration with passwords table to store encrypted passwords.

Password Strength Meter

A live strength meter evaluates passwords based on length, characters, digits, and symbols. It gives immediate visual feedback to the user.

Key Features:

- Weak, Medium, Strong scoring (1–3) displayed via a progress bar.
- Strength label changes color based on score: red (weak), orange (medium), green (strong).
- Automatically updates as the password is typed or generated.

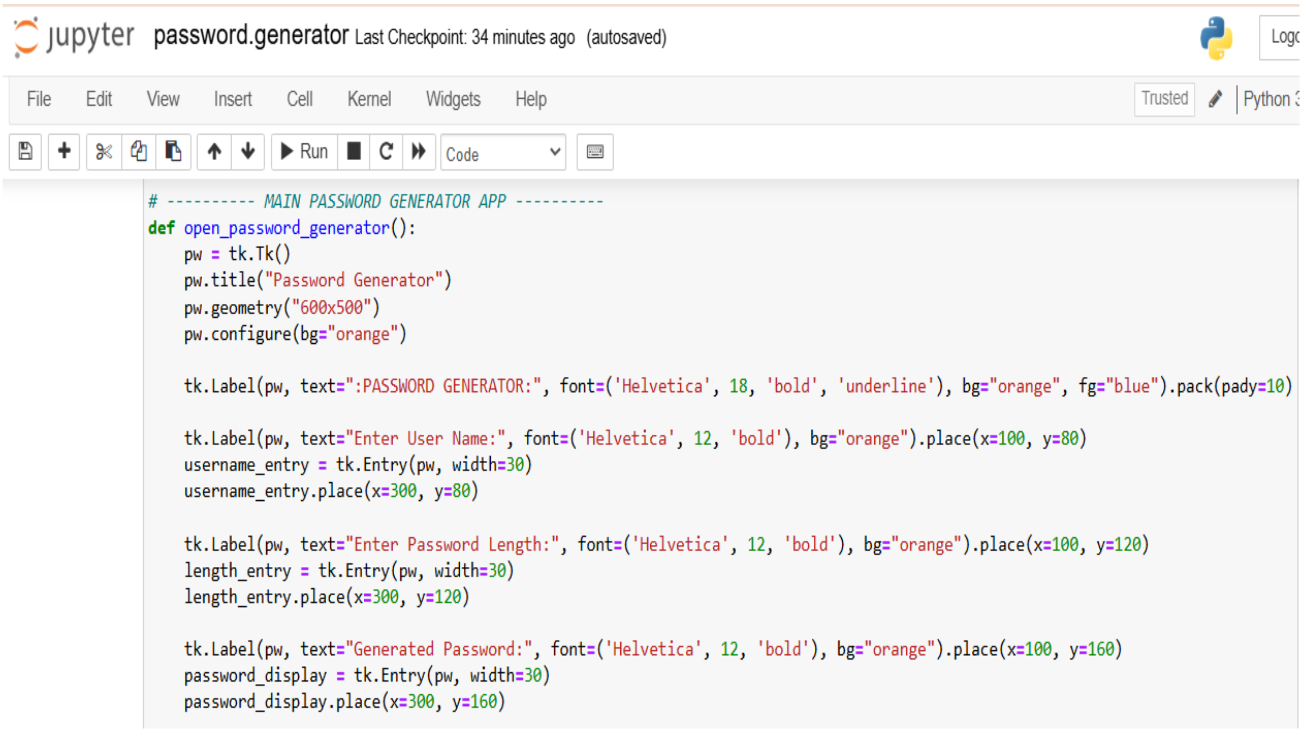
Generate Password Function

The generate_password() function allows users to create secure, random passwords.

Key Features:

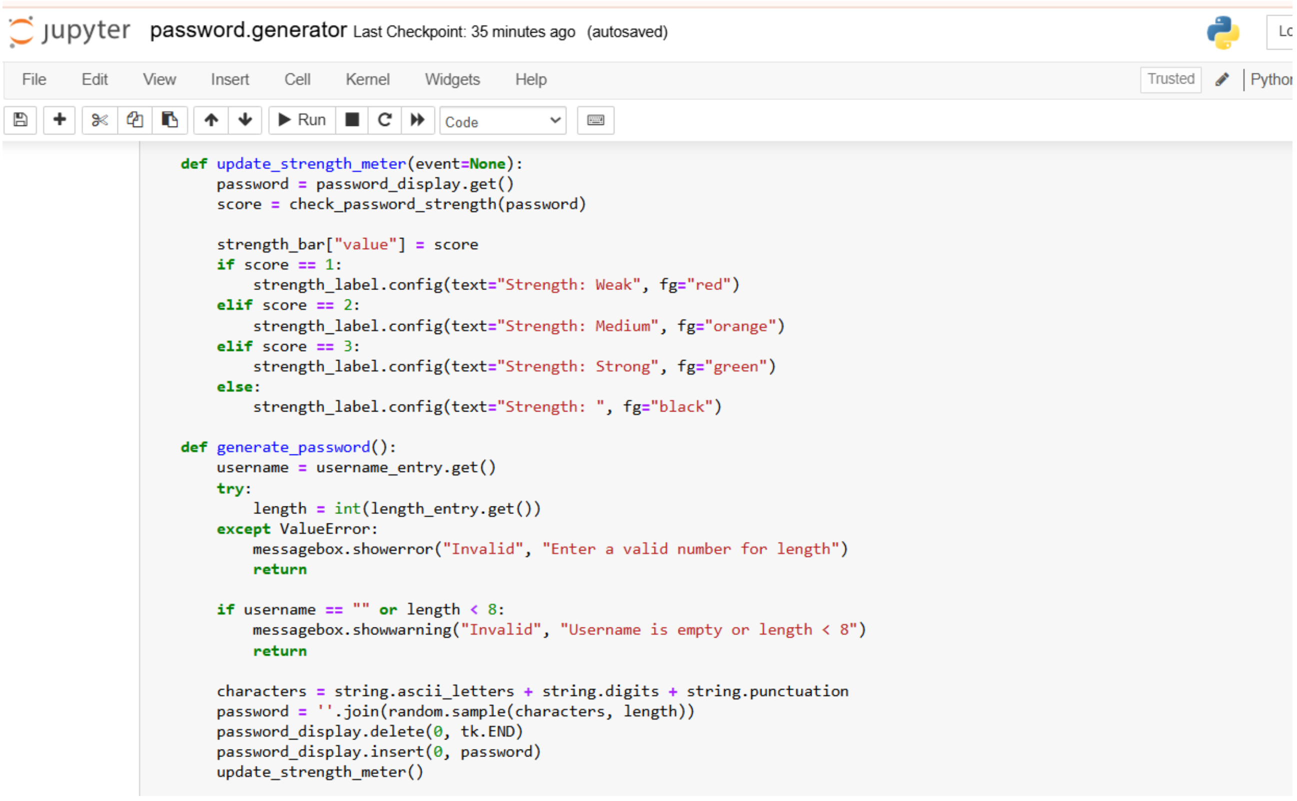
- Accepts username and length input.
- Generates password using uppercase, lowercase, digits, and special characters.
- Inserts generated password into the display field.
- Updates strength meter in real-time.

Code:


jupyter password.generator Last Checkpoint: 34 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
# ---------- MAIN PASSWORD GENERATOR APP ----------
def open_password_generator():
    pw = tk.Tk()
    pw.title("Password Generator")
    pw.geometry("600x500")
    pw.configure(bg="orange")

    tk.Label(pw, text=":PASSWORD GENERATOR:", font=('Helvetica', 18, 'bold', 'underline'), bg="orange", fg="blue").pack(pady=10)

    tk.Label(pw, text="Enter User Name:", font=('Helvetica', 12, 'bold'), bg="orange").place(x=100, y=80)
    username_entry = tk.Entry(pw, width=30)
    username_entry.place(x=300, y=80)

    tk.Label(pw, text="Enter Password Length:", font=('Helvetica', 12, 'bold'), bg="orange").place(x=100, y=120)
    length_entry = tk.Entry(pw, width=30)
    length_entry.place(x=300, y=120)

    tk.Label(pw, text="Generated Password:", font=('Helvetica', 12, 'bold'), bg="orange").place(x=100, y=160)
    password_display = tk.Entry(pw, width=30)
    password_display.place(x=300, y=160)
```

Figure : Tkinter Password Generator application


jupyter password.generator Last Checkpoint: 35 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code

```
    def update_strength_meter(event=None):
        password = password_display.get()
        score = check_password_strength(password)

        strength_bar["value"] = score
        if score == 1:
            strength_label.config(text="Strength: Weak", fg="red")
        elif score == 2:
            strength_label.config(text="Strength: Medium", fg="orange")
        elif score == 3:
            strength_label.config(text="Strength: Strong", fg="green")
        else:
            strength_label.config(text="Strength: ", fg="black")

    def generate_password():
        username = username_entry.get()
        try:
            length = int(length_entry.get())
        except ValueError:
            messagebox.showerror("Invalid", "Enter a valid number for length")
            return

        if username == "" or length < 8:
            messagebox.showwarning("Invalid", "Username is empty or length < 8")
            return

        characters = string.ascii_letters + string.digits + string.punctuation
        password = ''.join(random.sample(characters, length))
        password_display.delete(0, tk.END)
        password_display.insert(0, password)
        update_strength_meter()
```

Figure : Password strength meter and generate password function

Accept and Save Password Function

The accept_password() function validates passwords against a security policy, encrypts them, and saves them in SQLite.

Key Features:

- Checks password policy: minimum 8 characters, uppercase, lowercase, digits, and special characters.
- Encrypts password using Fernet symmetric encryption.
- Saves username and encrypted password to SQLite database.
- Prevents duplicate usernames using UNIQUE constraint.
- Provides user feedback via message boxes.

Reset Fields and Clear Database

The reset_fields_and_data() function clears all input fields and deletes all saved records from the database.

Key Features:

- Clears username, password, and length fields.
- Resets strength meter to 0.
- Deletes all records from the passwords table.
- Shows confirmation message to user.

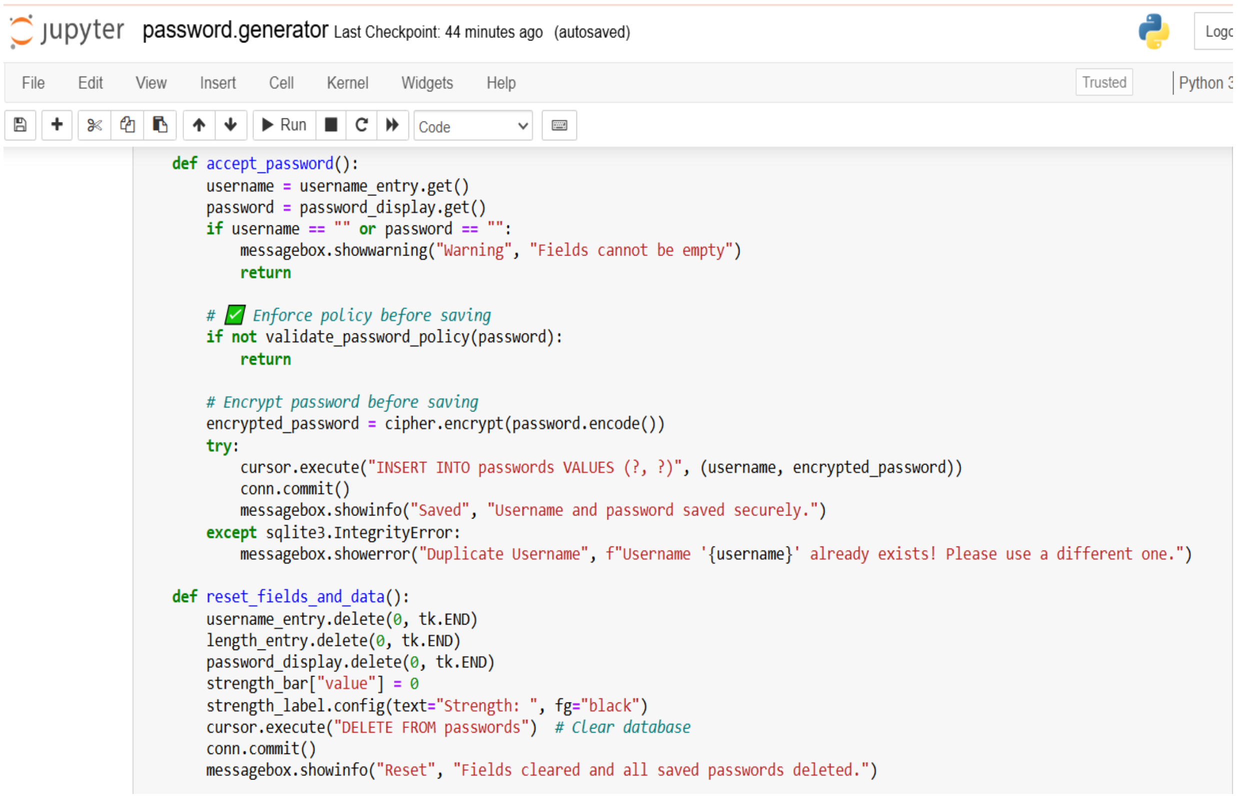
View Saved Passwords

The view_saved_passwords() function displays all stored usernames and decrypted passwords in a new window.

Key Features:

- Fetches all records from SQLite.
- Opens a new Tkinter window with a light yellow background.
- Decrypts passwords and displays username-password pairs.
- Handles decryption errors gracefully if the key has changed.

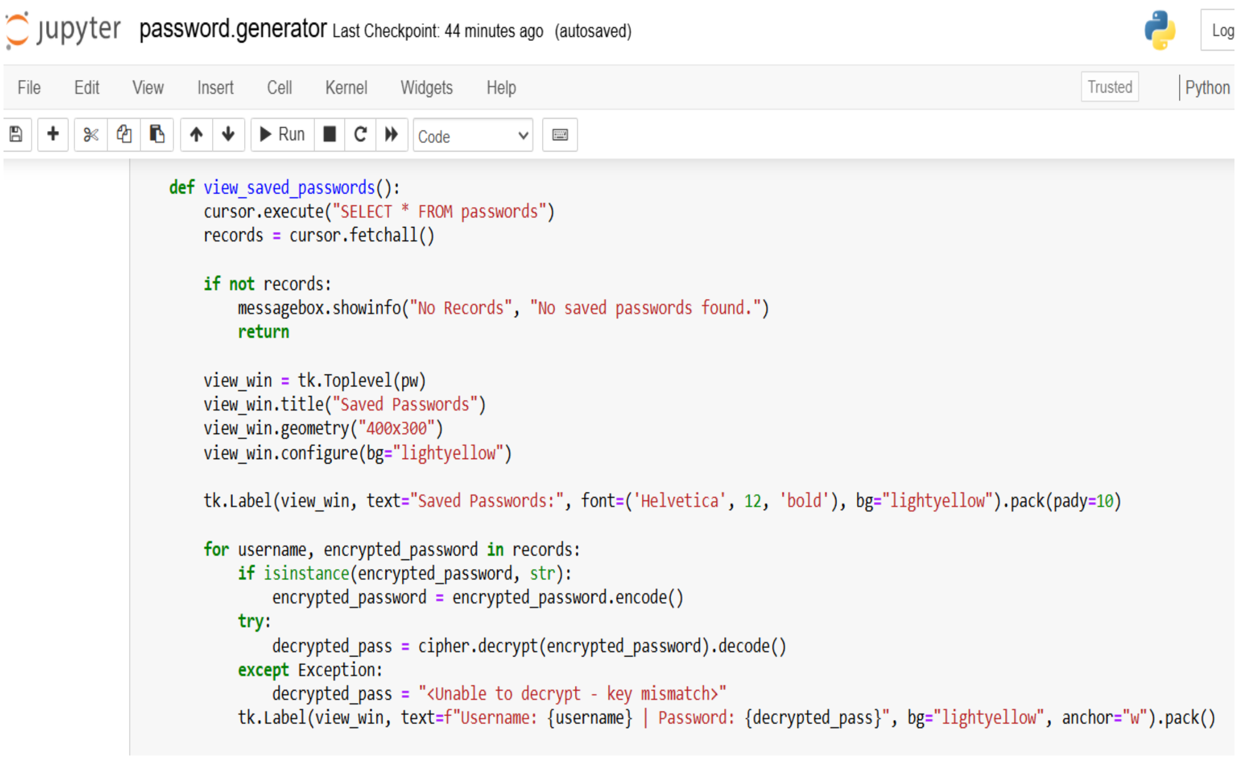
Code:



jupyter password.generator Last Checkpoint: 44 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Code

```python
def accept_password():
    username = username_entry.get()
    password = password_display.get()
    if username == "" or password == "":
        messagebox.showwarning("Warning", "Fields cannot be empty")
        return

    # ✅ Enforce policy before saving
    if not validate_password_policy(password):
        return

    # Encrypt password before saving
    encrypted_password = cipher.encrypt(password.encode())
    try:
        cursor.execute("INSERT INTO passwords VALUES (?, ?)", (username, encrypted_password))
        conn.commit()
        messagebox.showinfo("Saved", "Username and password saved securely.")
    except sqlite3.IntegrityError:
        messagebox.showerror("Duplicate Username", f"Username '{username}' already exists! Please use a different one.")

def reset_fields_and_data():
    username_entry.delete(0, tk.END)
    length_entry.delete(0, tk.END)
    password_display.delete(0, tk.END)
    strength_bar["value"] = 0
    strength_label.config(text="Strength: ", fg="black")
    cursor.execute("DELETE FROM passwords")  # Clear database
    conn.commit()
    messagebox.showinfo("Reset", "Fields cleared and all saved passwords deleted.")
```

Figure : Storing user-entered passwords with encryption



jupyter password.generator Last Checkpoint: 44 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Code

```python
def view_saved_passwords():
    cursor.execute("SELECT * FROM passwords")
    records = cursor.fetchall()

    if not records:
        messagebox.showinfo("No Records", "No saved passwords found.")
        return

    view_win = tk.Toplevel(pw)
    view_win.title("Saved Passwords")
    view_win.geometry("400x300")
    view_win.configure(bg="lightyellow")

    tk.Label(view_win, text="Saved Passwords:", font=('Helvetica', 12, 'bold'), bg="lightyellow").pack(pady=10)

    for username, encrypted_password in records:
        if isinstance(encrypted_password, str):
            encrypted_password = encrypted_password.encode()
        try:
            decrypted_pass = cipher.decrypt(encrypted_password).decode()
        except Exception:
            decrypted_pass = "<Unable to decrypt - key mismatch>"
        tk.Label(view_win, text=f"Username: {username} | Password: {decrypted_pass}", bg="lightyellow", anchor="w").pack()
```

Figure : Retrieve and display encrypted passwords securely

Buttons and Interactivity

The application provides buttons to generate passwords, accept and save them, reset fields, and view saved records. Each button is linked to its respective function for seamless interaction.

Key Features:

- **Generate Password:** Creates a strong random password and updates the strength meter.
- **Accept:** Validates the password against policy, encrypts it, and saves it in the SQLite database.
- **Reset:** Clears all input fields, resets the strength meter, and deletes all saved passwords.
- **View Saved Passwords:** Opens a new window displaying all usernames and decrypted passwords.
- **Real-time Strength Update:** Password strength meter updates dynamically as the password is typed or generated.

This section ensures the GUI is interactive, user-friendly, and secure before login verification.

Login Screen

The login screen ensures that only authorized users can access the password generator.

Key Features:

- Tkinter window titled “Login” with size 600x400 and light blue background.
- Entry fields for username and password.
- Password hidden using show="*" for security.
- Verifies login credentials against hardcoded username and hashed password.
- Displays success or error messages based on login attempt.

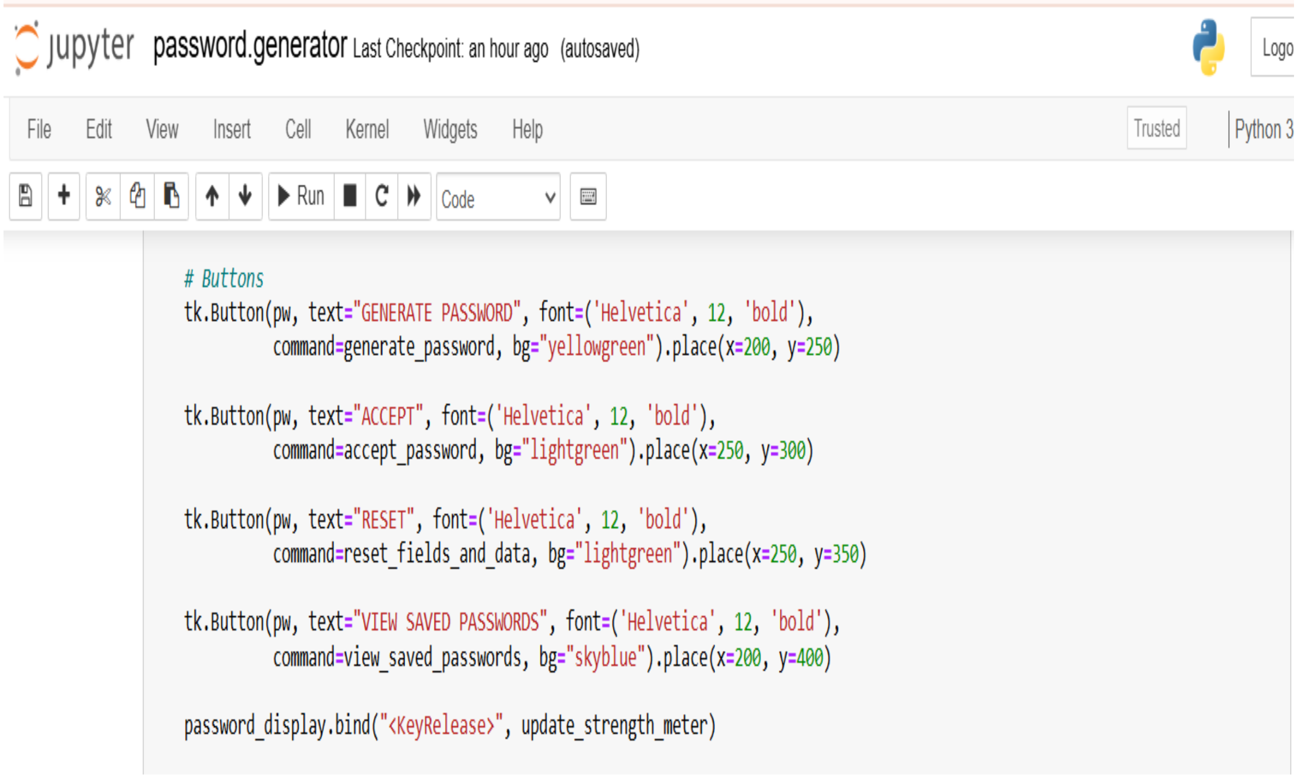
Run Application

The program begins by running the login screen. Upon successful login, the password generator GUI opens.

Key Features:

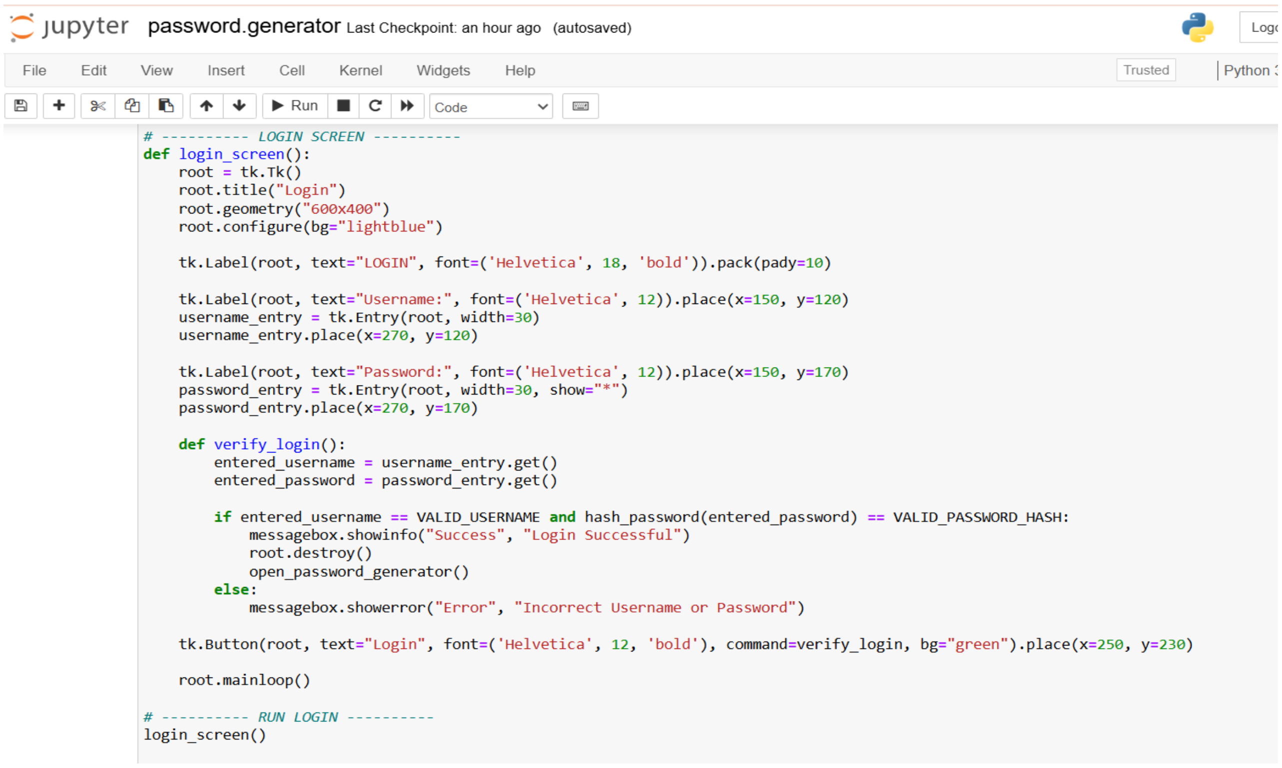
- Starts execution with login_screen().
- Ensures secure login before accessing password management functions.
- Integrates all GUI, database, encryption, and password functionality seamlessly.

Code:


jupyter password.generator Last Checkpoint: an hour ago (autosaved)
Logo
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```python
# Buttons
tk.Button(pw, text="GENERATE PASSWORD", font=('Helvetica', 12, 'bold'),
          command=generate_password, bg="yellowgreen").place(x=200, y=250)

tk.Button(pw, text="ACCEPT", font=('Helvetica', 12, 'bold'),
          command=accept_password, bg="lightgreen").place(x=250, y=300)

tk.Button(pw, text="RESET", font=('Helvetica', 12, 'bold'),
          command=reset_fields_and_data, bg="lightgreen").place(x=250, y=350)

tk.Button(pw, text="VIEW SAVED PASSWORDS", font=('Helvetica', 12, 'bold'),
          command=view_saved_passwords, bg="skyblue").place(x=200, y=400)

password_display.bind("<KeyRelease>", update_strength_meter)
```

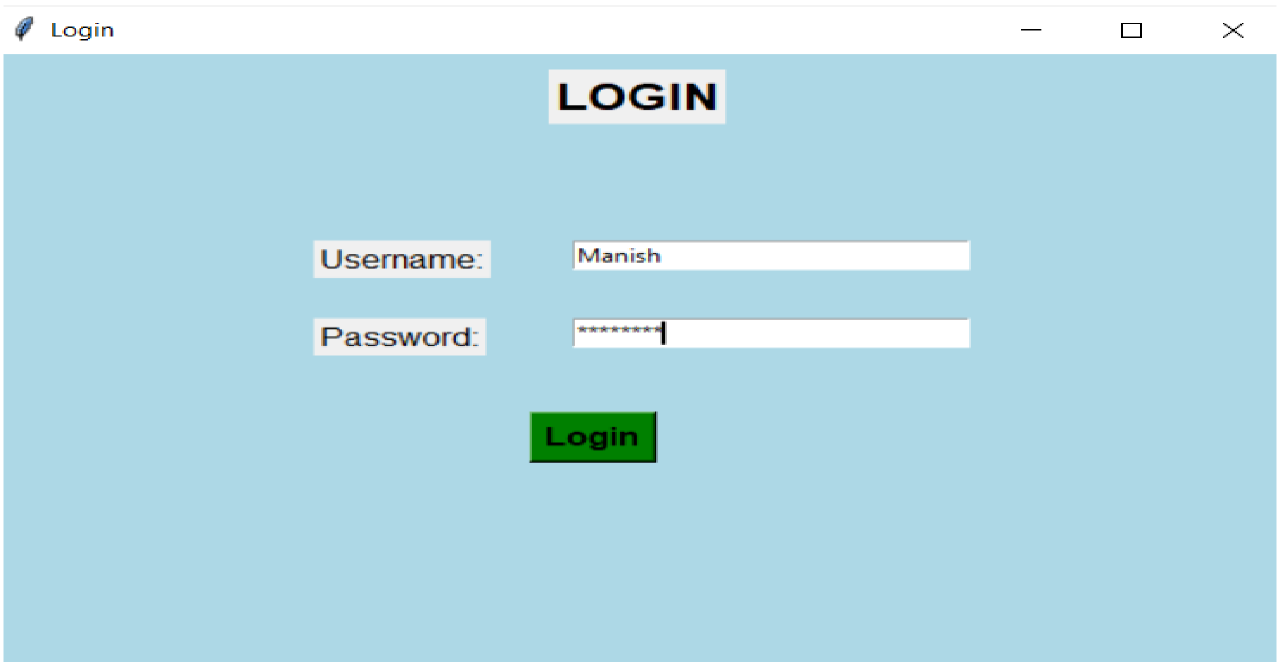
Figure : Tkinter GUI Button Configuration Code


jupyter password.generator Last Checkpoint: an hour ago (autosaved)
Logo
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```python
# ---------- LOGIN SCREEN ----------
def login_screen():
    root = tk.Tk()
    root.title("Login")
    root.geometry("600x400")
    root.configure(bg="lightblue")

    tk.Label(root, text="LOGIN", font=('Helvetica', 18, 'bold')).pack(pady=10)

    tk.Label(root, text="Username:", font=('Helvetica', 12)).place(x=150, y=120)
    username_entry = tk.Entry(root, width=30)
    username_entry.place(x=270, y=120)

    tk.Label(root, text="Password:", font=('Helvetica', 12)).place(x=150, y=170)
    password_entry = tk.Entry(root, width=30, show="*")
    password_entry.place(x=270, y=170)

    def verify_login():
        entered_username = username_entry.get()
        entered_password = password_entry.get()

        if entered_username == VALID_USERNAME and hash_password(entered_password) == VALID_PASSWORD_HASH:
            messagebox.showinfo("Success", "Login Successful")
            root.destroy()
            open_password_generator()
        else:
            messagebox.showerror("Error", "Incorrect Username or Password")

    tk.Button(root, text="Login", font=('Helvetica', 12, 'bold'), command=verify_login, bg="green").place(x=250, y=230)

    root.mainloop()

# ---------- RUN LOGIN ----------
login_screen()
```

Figure : Jupyter Notebook Login Screen Code

Login Screen:

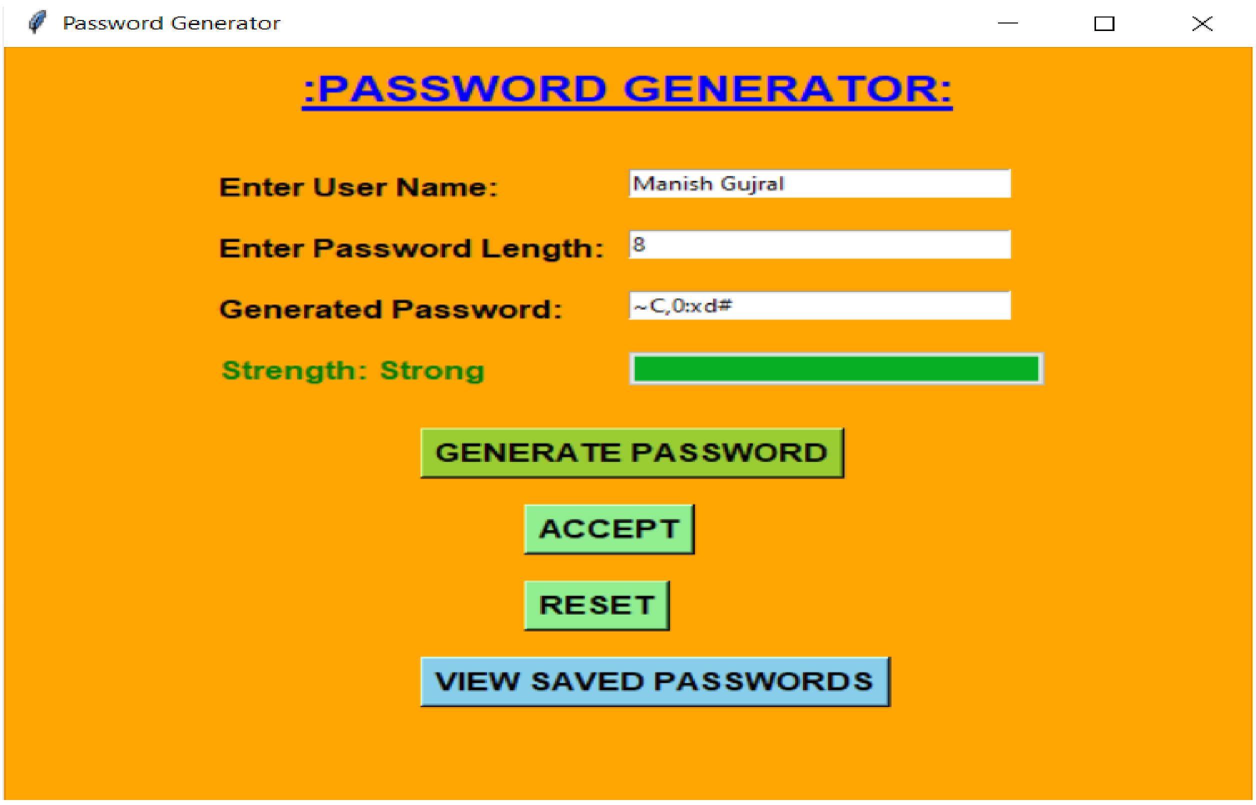


Username: Manish

Password: *********

Figure : Login Screen

Password Generator Dashboard:



Enter User Name: Manish Gujral

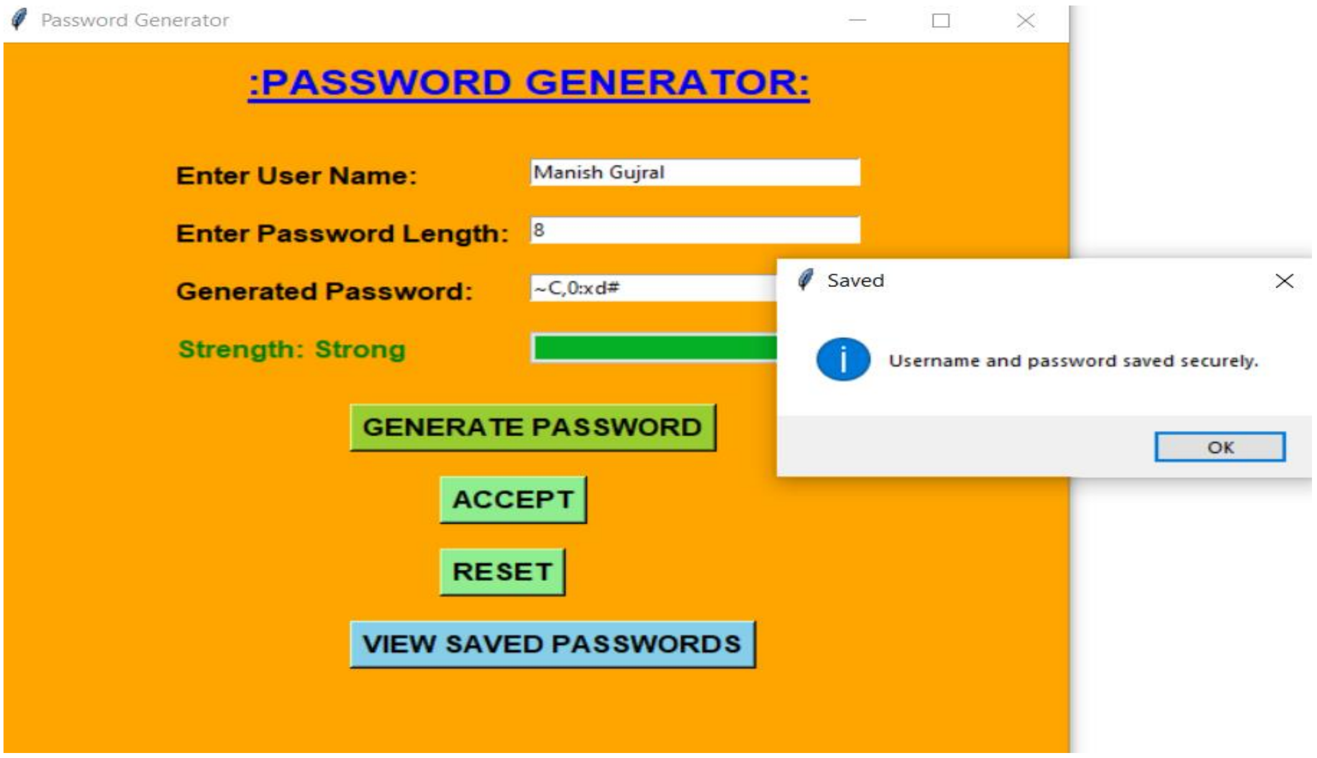
Enter Password Length: 8

Generated Password: ~C,0:xd#

Strength: Strong

Figure : Password Generator Dashboard

Accept function:


Password Generator

:PASSWORD GENERATOR:

Enter User Name: Manish Gujral

Enter Password Length: 8

Generated Password: ~C,0:xd#

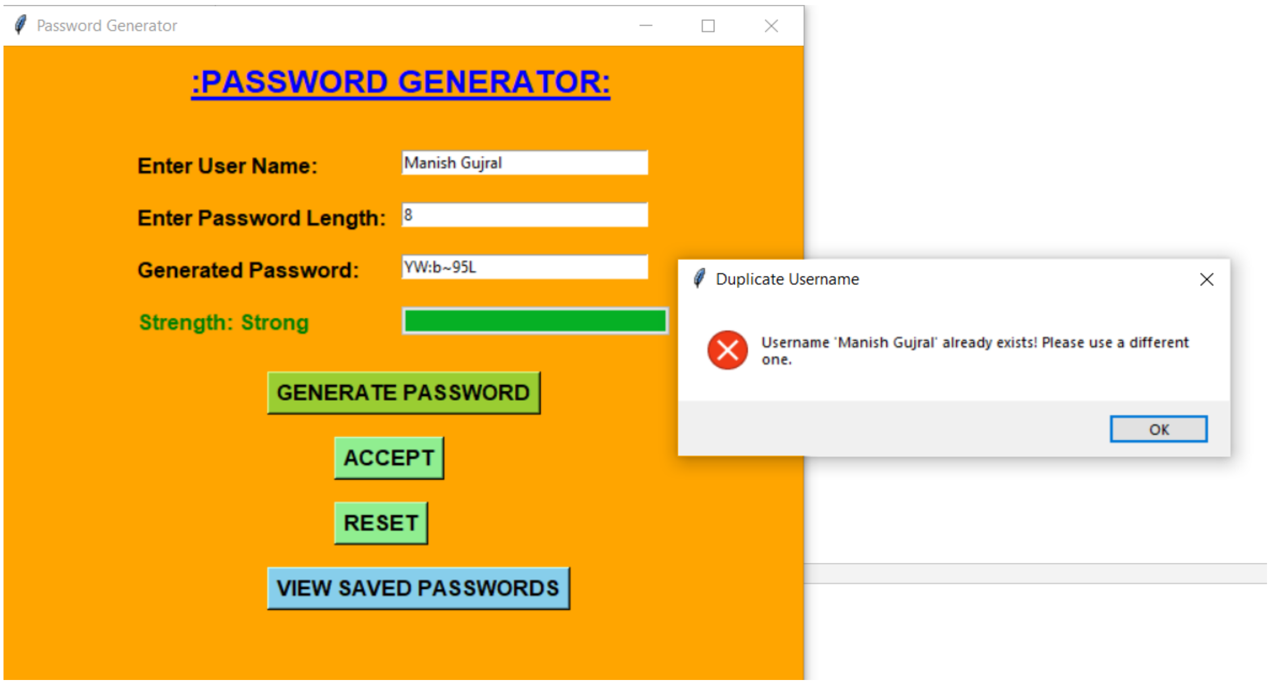
Strength: Strong

Saved

Username and password saved securely.

Figure : Accept Function

Same Username not allowed:


Password Generator

:PASSWORD GENERATOR:

Enter User Name: Manish Gujral

Enter Password Length: 8

Generated Password: YW:b~95L

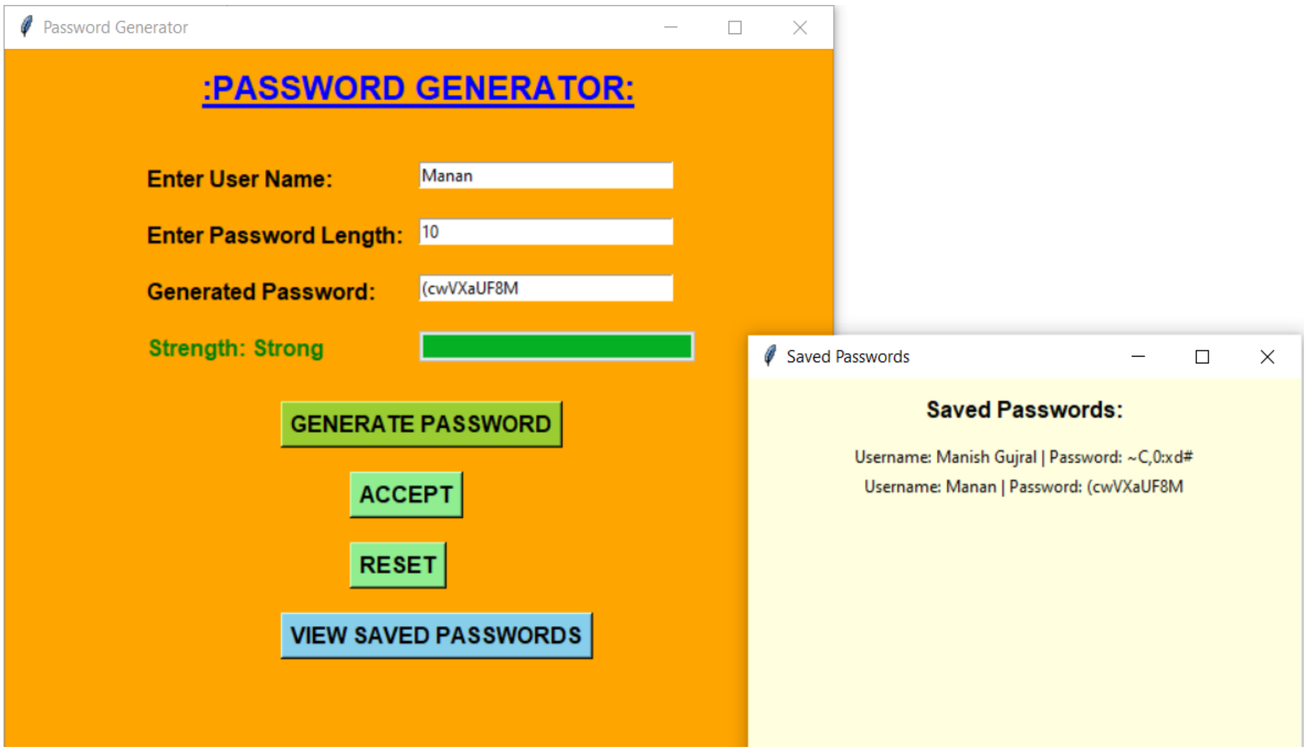
Strength: Strong

Duplicate Username

Username 'Manish Gujral' already exists! Please use a different one.

Figure : Same Username

View Saved Password Button:



Enter User Name: Manan

Enter Password Length: 10

Generated Password: (cwVXaUF8M

Strength: Strong

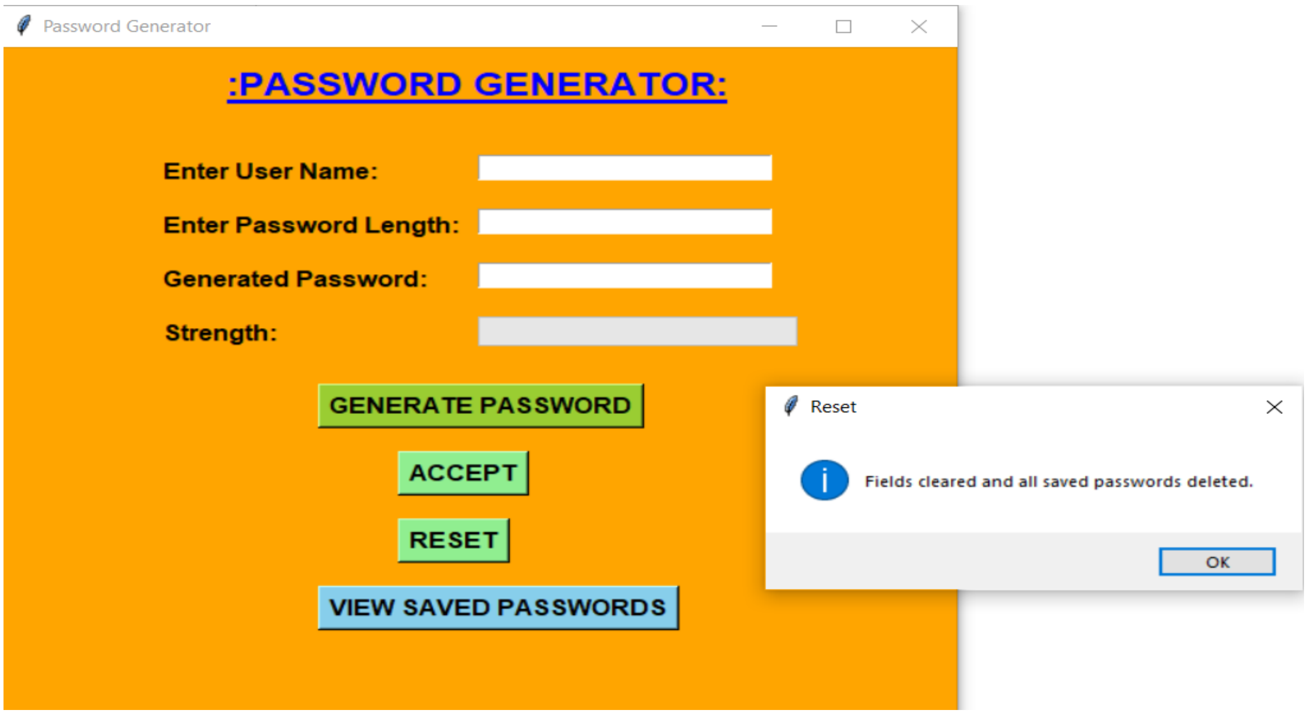
Saved Passwords:

Username: Manish Gujral | Password: ~C,0:xd#

Username: Manan | Password: (cwVXaUF8M

Figure : Saved Passwords

Reset Function:



Enter User Name:

Enter Password Length:

Generated Password:

Strength:

Fields cleared and all saved passwords deleted.

Figure : Reset Function