import tkinter as tk

from tkinter import ttk, messagebox

import re

import secrets

import string

import math

# Function to calculate entropy

def calculate\_entropy(password):

if not password:

return 0

lowercase = string.ascii\_lowercase

uppercase = string.ascii\_uppercase

digits = string.digits

special = "!@#$%^&\*"

charset\_size = 0

if any(c in lowercase for c in password):

charset\_size += len(lowercase)

if any(c in uppercase for c in password):

charset\_size += len(uppercase)

if any(c in digits for c in password):

charset\_size += len(digits)

if any(c in special for c in password):

charset\_size += len(special)

password\_length = len(password)

entropy = password\_length \* math.log2(charset\_size) if charset\_size > 0 else 0

return entropy

# Function to estimate time to crack based on entropy (assuming 1 billion guesses/second)

def estimate\_time\_to\_crack(entropy):

if entropy <= 0:

return "Instant (0 seconds)"

# Number of possible combinations = 2^entropy

total\_combinations = 2 \*\* entropy

# Assume an attacker can make 1 billion (10^9) guesses per second

guesses\_per\_second = 1\_000\_000\_000

# Calculate time in seconds

time\_seconds = total\_combinations / guesses\_per\_second

# Convert to human-readable format

if time\_seconds < 1:

return "Instant (<1 second)"

elif time\_seconds < 60:

return f"{time\_seconds:.2f} seconds"

elif time\_seconds < 3600:

minutes = time\_seconds / 60

return f"{minutes:.2f} minutes"

elif time\_seconds < 86400:

hours = time\_seconds / 3600

return f"{hours:.2f} hours"

elif time\_seconds < 31536000: # Less than a year

days = time\_seconds / 86400

return f"{days:.2f} days"

elif time\_seconds < 3153600000: # Less than 100 years

years = time\_seconds / 31536000

return f"{years:.2f} years"

else:

centuries = time\_seconds / 3153600000

return f"{centuries:.2f} centuries"

# Password Strength Checker

def check\_password\_strength(password):

if not password:

return "", "Evaluate your password to check strength.", 0, 0, "Instant (0 seconds)"

strength = 0

suggestions = []

# Define character sets

lowercase = string.ascii\_lowercase

uppercase = string.ascii\_uppercase

digits = string.digits

special = "!@#$%^&\*"

# Check if any character type is missing

has\_lower = any(c in lowercase for c in password)

has\_upper = any(c in uppercase for c in password)

has\_digit = any(c in digits for c in password)

has\_special = any(c in special for c in password)

# If any character type is missing, classify as "Weak" immediately

if not (has\_lower and has\_upper and has\_digit and has\_special):

return "Weak", "Password must include lowercase letters, uppercase letters, numbers, and special characters (!@#$%^&\*).", 0, 0, "Instant (0 seconds)"

# Check for continuous repetition of characters (e.g., "aaa", "111", "@@@")

for i in range(len(password) - 1):

if password[i] == password[i + 1]:

# Count the length of the continuous repetition starting at this position

repetition\_length = 1

while i + repetition\_length < len(password) and password[i] == password[i + repetition\_length]:

repetition\_length += 1

# If there are 3 or more consecutive identical characters, classify as "Weak"

if repetition\_length >= 3:

return "Weak", "Avoid continuous repetition of characters for stronger security.", 0, 0, "Instant (0 seconds)"

# Check for low variety or repetitive patterns

# Count the frequency of each character

char\_count = {}

for char in password:

char\_count[char] = char\_count.get(char, 0) + 1

# Find the most frequent character and its count

max\_count = max(char\_count.values(), default=0)

if max\_count > 0: # Avoid division by zero

repetition\_ratio = max\_count / len(password) # Ratio of most frequent character to total length

# If more than 50% of the password consists of the same character, classify as "Weak"

if repetition\_ratio > 0.5:

return "Weak", "Consider more varieties. Avoid repetitive characters for stronger security.", 0, 0, "Instant (0 seconds)"

# Length check

password\_length = len(password)

if password\_length <= 7:

strength = 0

suggestions.append("Use at least 8 characters.")

elif 8 <= password\_length <= 11:

strength = 1

suggestions.append("Use at least 12 characters for a strong password.")

elif 12 <= password\_length <= 16:

strength = 2

else:

strength = 3

# Character type checks (already ensured above, but kept for consistency)

if has\_lower:

strength += 1

if has\_upper:

strength += 1

if has\_digit:

strength += 1

if has\_special:

strength += 1

entropy = calculate\_entropy(password)

# Estimate time to crack

time\_to\_crack = estimate\_time\_to\_crack(entropy)

# Determine strength level (only if all character types are present, no continuous repetition, and no high overall repetition)

if password\_length <= 7:

return "Weak", "\n".join(suggestions), 20, entropy, time\_to\_crack

elif 8 <= password\_length <= 11:

return "Medium", "\n".join(suggestions), 50, entropy, time\_to\_crack

elif 12 <= password\_length <= 16:

if strength >= 5:

return "Strong", "✅ Secure Password", 80, entropy, time\_to\_crack

else:

return "Medium", "⚠ Consider adding more variety.\n" + "\n".join(suggestions), 60, entropy, time\_to\_crack

else:

if strength >= 5 and entropy >= 120:

return "Very Strong", "✅ Extremely Secure Password", 100, entropy, time\_to\_crack

elif strength >= 5:

return "Strong", "✅ Secure Password", 80, entropy, time\_to\_crack

else:

return "Medium", "⚠ Consider adding more variety.\n" + "\n".join(suggestions), 60, entropy, time\_to\_crack

# Password Generator

def generate\_strong\_password():

# Define the character sets

lowercase = string.ascii\_lowercase

uppercase = string.ascii\_uppercase

digits = string.digits

special = "!@#$%^&\*"

# Initialize the password with one character from each category

password = [

secrets.choice(lowercase), # One lowercase letter

secrets.choice(uppercase), # One uppercase letter

secrets.choice(digits), # One digit

secrets.choice(special) # One special character

]

# Combine all character sets for the remaining characters

all\_chars = lowercase + uppercase + digits + special

charset\_size = len(all\_chars) # Total unique characters (26 + 26 + 10 + 8 = 70)

# Calculate the remaining length (20 characters to reach 24 total)

remaining\_length = 24 - len(password) # 24 - 4 = 20

# Generate the remaining characters, ensuring high entropy and randomness

for \_ in range(remaining\_length):

password.append(secrets.choice(all\_chars))

# Use secrets.SystemRandom().shuffle() for cryptographically secure randomization

secrets.SystemRandom().shuffle(password)

# Join the characters into a string

password = ''.join(password)

# Verify entropy (should be > 120 bits with 24 chars and all char sets)

entropy = calculate\_entropy(password)

if entropy < 120:

# If entropy is unexpectedly low (unlikely), regenerate

return generate\_strong\_password()

# Double-check that the password includes at least one of each character type

has\_lower = any(c in lowercase for c in password)

has\_upper = any(c in uppercase for c in password)

has\_digit = any(c in digits for c in password)

has\_special = any(c in special for c in password)

if not (has\_lower and has\_upper and has\_digit and has\_special):

# If any character type is missing (extremely unlikely due to initial inclusion), regenerate

return generate\_strong\_password()

return password

# Check Password Action

def on\_check():

password = password\_entry.get()

if not password:

if not password:

messagebox.showerror("Error", "Please enter a password.")

return

strength\_label.config(text="Strength: -", font=("Arial", 18), fg="white")

suggestion\_label.config(text="Evaluate your password to check strength.", font=("Arial", 18))

progress\_bar['value'] = 0

entropy\_label.config(text="Entropy: -", font=("Arial", 18))

time\_to\_crack\_label.config(text="Time to Crack: Instant (0 seconds)", font=("Arial", 18))

return

strength, advice, progress, entropy, time\_to\_crack = check\_password\_strength(password)

strength\_label.config(text=f"Strength: {strength}", fg=("green" if strength == "Very Strong" else "blue" if strength == "Strong" else "orange" if strength == "Medium" else "red"))

suggestion\_label.config(text=advice, font=("Arial", 18))

progress\_bar['value'] = progress

entropy\_label.config(text=f"Entropy: {entropy:.2f} bits", font=("Arial", 18))

time\_to\_crack\_label.config(text=f"Time to Crack: {time\_to\_crack}", font=("Arial", 18))

# Add entropy and time to crack explanation

entropy\_summary = (

"In summary, the higher the entropy (in bits), the stronger and more secure the data is.\n"

"Aim for entropy levels above 60 bits for strong security and above 120 bits for very strong security.\n"

"Time to Crack estimates how long a brute-force attack would take, assuming 1 billion guesses/second."

)

messagebox.showinfo("Password Strength", f"Strength: {strength}\nEntropy: {entropy:.2f} bits\nTime to Crack: {time\_to\_crack}\n\nSuggestions:\n{advice}\n\n{entropy\_summary}")

# Generate Strong Password Action

def on\_generate():

strong\_password = generate\_strong\_password()

password\_entry.delete(0, tk.END)

password\_entry.insert(0, strong\_password)

update\_strength()

# Copy Password to Clipboard

def on\_copy():

password = password\_entry.get()

if not password:

messagebox.showerror("Error", "No password to copy.")

return

root.clipboard\_clear()

root.clipboard\_append(password)

root.update()

messagebox.showinfo("Copied", "Password copied to clipboard!")

# Toggle Password Visibility

def toggle\_password():

if password\_entry.cget("show") == "":

password\_entry.config(show="\*")

toggle\_button.config(text="👁 Show")

else:

password\_entry.config(show="")

toggle\_button.config(text="🙈 Hide")

# Real-time Strength Update

def update\_strength(event=None):

password = password\_entry.get()

if not password:

strength\_label.config(text="Strength: -", font=("Arial", 18), fg="white")

suggestion\_label.config(text="Evaluate your password to check strength.", font=("Arial", 18))

progress\_bar['value'] = 0

entropy\_label.config(text="Entropy: -", font=("Arial", 18))

time\_to\_crack\_label.config(text="Time to Crack: Instant (0 seconds)", font=("Arial", 18))

return

strength, advice, progress, entropy, time\_to\_crack = check\_password\_strength(password)

strength\_label.config(text=f"Strength: {strength}", fg=("green" if strength == "Very Strong" else "blue" if strength == "Strong" else "orange" if strength == "Medium" else "red"))

suggestion\_label.config(text=advice, font=("Arial", 18))

progress\_bar['value'] = progress

entropy\_label.config(text=f"Entropy: {entropy:.2f} bits", font=("Arial", 18))

time\_to\_crack\_label.config(text=f"Time to Crack: {time\_to\_crack}", font=("Arial", 18))

# GUI Setup

root = tk.Tk()

root.title("🔐 Password Strength Checker")

root.geometry("1280x720")

root.configure(bg="#4A148C") # Deep purple background for the root window

# Style configuration

style = ttk.Style()

style.configure("TButton", font=("Arial", 12, "bold"), padding=2, relief="flat", borderwidth=0, background="#2196F3", foreground="black") # Blue for Check Strength

style.map("TButton", background=[("active", "#1976D2")]) # Darker blue for active state

# Enhanced button style for smaller rounded corners with blending color for toggle

style.configure("Rounded.TButton", font=("Arial", 12, "bold"), padding=2, relief="flat", borderwidth=0, background="#2196F3", foreground="black", borderradius=30) # Blue for Show/Hide

style.map("Rounded.TButton", background=[("active", "#1976D2")]) # Darker blue for active state

# Configure Generate Strong Password button (green)

style.configure("GenerateStrong.TButton", font=("Arial", 12, "bold"), padding=2, relief="flat", borderwidth=0, background="#4CAF50", foreground="black")

style.map("GenerateStrong.TButton", background=[("active", "#388E3C")]) # Darker green for active state

# Configure Copy Password button (red)

style.configure("Copy.TButton", font=("Arial", 12, "bold"), padding=2, relief="flat", borderwidth=0, background="#F44336", foreground="black")

style.map("Copy.TButton", background=[("active", "#D32F2F")]) # Darker red for active state

# Configure progress bar style with blending color

style.configure("TProgressbar", background="#4A148C", troughcolor="#4A148C")

# UI Elements

tk.Label(root, text="How secure is My Password ?", font=("Arial", 20, "bold"), fg="white", bg="#4A148C").pack(pady=10)

# Password Entry and Toggle Button Frame

password\_frame = tk.Frame(root, bg="#4A148C")

password\_frame.pack(pady=10)

password\_entry = tk.Entry(password\_frame, width=30, font=("Arial", 12), show="\*", bg="white", fg="black", insertbackground="black", highlightthickness=0)

password\_entry.pack(side="left", padx=5)

toggle\_button = ttk.Button(password\_frame, text="👁 Show", command=toggle\_password, style="Rounded.TButton")

toggle\_button.pack(side="right")

# Bind key release to update strength

password\_entry.bind("<KeyRelease>", update\_strength)

# Button Frame

button\_frame = tk.Frame(root, bg="#4A148C")

button\_frame.pack(pady=10)

# Buttons with consistent width and alignment

buttons = [

("Check Strength", on\_check, "TButton"),

("Generate Strong Password", on\_generate, "GenerateStrong.TButton"),

("Copy Password", on\_copy, "Copy.TButton"),

]

# Set button width to match the longest button text

button\_width = max(len(text) for text, \_, \_ in buttons) + 2 # Add padding

for text, command, style\_name in buttons:

btn = ttk.Button(button\_frame, text=text, command=command, style=style\_name, width=button\_width)

btn.pack(pady=5, fill=tk.X, padx=20) # Use fill=tk.X to make buttons stretch horizontally

# Strength and Entropy Display

strength\_frame = tk.Frame(root, bg="#4A148C")

strength\_frame.pack(pady=10)

strength\_label = tk.Label(strength\_frame, text="Strength: -", font=("Arial", 18, "bold"), fg="white", bg="#4A148C")

strength\_label.pack()

progress\_bar = ttk.Progressbar(strength\_frame, length=300, mode="determinate", style="TProgressbar")

progress\_bar.pack(pady=5)

suggestion\_label = tk.Label(strength\_frame, text="Evaluate your password to check strength.", font=("Arial", 18), wraplength=380, justify="center", fg="red", bg="#4A148C")

suggestion\_label.pack(pady=5)

entropy\_label = tk.Label(strength\_frame, text="Entropy: -", font=("Arial", 18, "bold"), fg="white", bg="#4A148C")

entropy\_label.pack()

time\_to\_crack\_label = tk.Label(strength\_frame, text="Time to Crack: Instant (0 seconds)", font=("Arial", 18), fg="white", bg="#4A148C")

time\_to\_crack\_label.pack()

root.mainloop()