```
import tkinter as tk
from tkinter import messagebox
import re
```

– imports module

– initializes the core dependencies for a graphical password strength assessment tool built with Tkinter

```
import tkinter as tk
```

– ' tkinter as tk ' : Core Tkinter library for creating the GUI interface

```
from tkinter import messagebox
```

– ' tkinter.messagebox ' : provides standard dialog boxes for user feedback

```
import re
```

– regular expressions module for analyzing password patterns

==========

```
# Common passwords and dictionary words lists
COMMON_PASSWORDS = {"password", "123456", "qwerty", "admin", "letmein", "welcome"}
DICT_WORDS = {"apple", "computer", "dragon", "monkey"}
```

– defines static sets of common passwords and dictionary words used to detect weak, predictable passwords vulnerable to dictionary attacks

```
COMMON_PASSWORDS = {"password", "123456", "qwerty", "admin", "letmein", "welcome"}
```

– set of 6 frequently used weak passwords
– quick lookup for immediately rejected passwords

```
DICT_WORDS = {"apple", "computer", "dragon", "monkey"}
```

– set of 4 common dictionary words
– penalizes passwords containing everyday words

==========

```python
def evaluate_password(pw):
    """
    Evaluates password strength based on 7 criteria.
    Returns dict with met criteria count and details.
    """
    criteria_met = 0
    details = []

    # 1. Min. length 12
    if len(pw) >= 12:
        criteria_met += 1
        details.append("/ Min length (12+)")
    else:
        details.append("X Min length (<12)")

    # 2. Uppercase
    if re.search(r'[A-Z]', pw):
        criteria_met += 1
        details.append("/ Uppercase letter")
    else:
        details.append("X Uppercase letter")

    # 3. Lowercase
    if re.search(r'[a-z]', pw):
        criteria_met += 1
        details.append("/ Lowercase letter")
    else:
        details.append("X Lowercase letter")
```

```
9   def evaluate_password(pw):
38      # 4. Numbers
39      if re.search(r'\d', pw):
40          criteria_met += 1
41          details.append("/ Number")
42      else:
43          details.append("X Number")
44
45      # 5. Special chars
46      if re.search(r'[!@#$%^&*()_+\-=\[\]{};\':"\\|,.<>/?]', pw):
47          criteria_met += 1
48          details.append("/ Special character")
49      else:
50          details.append("X Special character")
51
52      # 6. Common passwords
53      if pw.lower() in COMMON_PASSWORDS:
54          details.insert(0, "X Common password!")
55          return {"met": 0, "details": details} # Auto weak
56      else:
57          criteria_met += 1
58          details.append("/ Not common password")
59
60      # 7. Dictionary words
61      has_dict_word = any(word in pw.lower() for word in DICT_WORDS)
62      if not has_dict_word:
63          criteria_met += 1
64          details.append("/ No dictionary word")
65      else:
66          details.append("X Contains dictionary word")
67
68      return {"met": criteria_met, "details": details}
```

– core evaluation function

```
9   def evaluate_password(pw):
```

– comprehensive password strength analysis using 7 security criteria
– ' pw ' (str) : user-provided password to evaluate

```
68      return {"met": criteria_met, "details": details}
```

– ‘ “met” ‘ (int) : number of criteria passed
– ‘ “details” (list) : human-readable status for each criterion

```
    # 1. Min. length 12
    if len(pw) >= 12:
        criteria_met += 1
        details.append("/ Min length (12+)")
    else:
        details.append("X Min length (<12)")

    # 2. Uppercase
    if re.search(r'[A-Z]', pw):
        criteria_met += 1
        details.append("/ Uppercase letter")
    else:
        details.append("X Uppercase letter")

    # 3. Lowercase
    if re.search(r'[a-z]', pw):
        criteria_met += 1
        details.append("/ Lowercase letter")
    else:
        details.append("X Lowercase letter")

    # 4. Numbers
    if re.search(r'\d', pw):
        criteria_met += 1
        details.append("/ Number")
    else:
        details.append("X Number")

    # 5. Special chars
    if re.search(r'[!@#$%^&*()_+\-=\[\]{};\':"\\|,.<>/?]', pw):
        criteria_met += 1
        details.append("/ Special character")
    else:
        details.append("X Special character")

    # 6. Common passwords
    if pw.lower() in COMMON_PASSWORDS:
        details.insert(0, "X Common password!")
        return {"met": 0, "details": details} # Auto weak
    else:
        criteria_met += 1
        details.append("/ Not common password")
```

```
60      # 7. Dictionary words
61      has_dict_word = any(word in pw.lower() for word in DICT_WORDS)
62      if not has_dict_word:
63          criteria_met += 1
64          details.append("/ No dictionary word")
65      else:
66          details.append("X Contains dictionary word")
67
68      return {"met": criteria_met, "details": details}
```

– evaluation criteria (7 in total)
  – length : >=12 characters
  – uppercase : contains A to Z
  – lowercase : contains a to z
  – numbers : contains 0 to 9
  – special : contains all special characters
  – common : not in 'COMMON_PASSWORDS' set
  – dictionary : no words from 'DICT_WORDS' set

==========

```
70  def check_password():
71      """ Handles button click: evaluates and displays strength. """
72      pw = entry.get()
73      if not pw:
74          messagebox.showwarning("Input required", "Please enter a password before checking.")
75          return
76
77      result = evaluate_password(pw)
78
79      # Determine rating
80      met = result["met"]
81      if met < 4 or "X Common password!" in result["details"]:
82          rating = "Weak"
83      elif met < 7:
84          rating = "Moderate"
85      else:
86          rating = "Strong"
87
88      # Update display
89      details_text.delete(1.0, tk.END)
90      details_text.insert(tk.END, f"{rating} Password\n\n" + "\n".join(result["details"]))
91
```

– GUI event holder

```
70  def check_password():
```

– processes “Check Password” button click
– validates input, evaluates strength, updates display

```
72      pw = entry.get()
```

– get text from password entry field

```
73      if not pw:
```

– show warning dialog -> return

```
77      result = evaluate_password(pw)
```

– calls core analysis function

```
79      # Determine rating
80      met = result["met"]
81      if met < 4 or "X Common password!" in result["details"]:
82          rating = "Weak"
83      elif met < 7:
84          rating = "Moderate"
85      else:
86          rating = "Strong"
```

– rating classification

Criteria Met	Contains Common Password	Rating
<4	Yes	Weak
<4	No	Weak
4 to 6	No	Moderate
7	No	Strong

```
88      # Update display
89      details_text.delete(1.0, tk.END)
90      details_text.insert(tk.END, f"{rating} Password\n\n" + "\n".join(result["details"]))
```

– UI update

==========

```
92    # GUI setup
93    root = tk.Tk()
94    root.title("Password Strength Assessor")
95    root.geometry("400x500")
96    root.configure(bg="lightgray")
```

– main window setup

```
93    root = tk.Tk()
```

– create root window

```
94    root.title("Password Strength Assessor")
```

– window title bar
– clear app identification

```
95    root.geometry("400x500")
```

– fixed size
    – 400px width : fits entry field + criteria checklist comfortably
    – 500px height : accommodates password input + 7 criteria details + buttons
– compact layout for criteria list

```
96    root.configure(bg="lightgray")
```

– background
– professional, readable appearance
– high contrast for black text, clean corporate look

==========

```
98  # Title
99  tk.Label(root, text="Password Strength Checker", font=("Arial", 16, "bold"), bg="lightgray").pack(pady=10)
100
101 # Password Input
102 tk.Label(root, text="Enter Password:", font=("Arial", 12), bg="lightgray").pack()
103 entry = tk.Entry(root, font=("Arial", 12), width=30, show="*", justify="center")
104 entry.pack(pady=5)
105 entry.focus()
106
107 # Check button
108 tk.Button(root, text="Assess Strength", command=check_password, bg="lightblue", font=("Arial", 12), width=20).pack(pady=20)
109
110 # Results display
111 tk.Label(root, text="Results:", font=("Arial", 12, "bold"), bg="lightgray").pack(anchor="w", padx=20)
112 details_text = tk.Text(root, height=15, width=50, font=("Courier", 10), bg="white")
113 details_text.pack(pady=10, padx=20, fill=tk.BOTH, expand=True)
114
115 root.mainloop()
```

– complete widget layout and event loop

```
98  # Title
99  tk.Label(root, text="Password Strength Checker", font=("Arial", 16, "bold"), bg="lightgray").pack(pady=10)
```

– title label

```
101 # Password Input
102 tk.Label(root, text="Enter Password:", font=("Arial", 12), bg="lightgray").pack()
103 entry = tk.Entry(root, font=("Arial", 12), width=30, show="*", justify="center")
104 entry.pack(pady=5)
105 entry.focus()
```

– password input section

```
105 entry.focus()
```

– cursor ready in field on startup

```
103 entry = tk.Entry(root, font=("Arial", 12), width=30, show="*", justify="center")
```

– ‘ show=”*” ‘ : hides password characters

```
107 # Check button
108 tk.Button(root, text="Assess Strength", command=check_password, bg="lightblue", font=("Arial", 12), width=20).pack(pady=20)
```

– action button
– triggers ‘check_password()’ evaluation
– prominent light blue styling, centered

```
110   # Results display
111   tk.Label(root, text="Results:", font=("Arial", 12, "bold"), bg="lightgray").pack(anchor="w", padx=20)
112   details_text = tk.Text(root, height=15, width=50, font=("Courier", 10), bg="white")
113   details_text.pack(pady=10, padx=20, fill=tk.BOTH, expand=True)
```

– results section
– monospace preserves checklist alignment

```
113   details_text.pack(pady=10, padx=20, fill=tk.BOTH, expand=True)
```

– ' fill=tk.BOTH, expand=True ' : resizes to fill remaining space

```
115   root.mainloop()
```

– event loop
– starts Tkinter event processing

==========

RESULT:

Example password: Password_123