Student Name: Altolane Jackson (AJ)
Project Title: Password Strength Checker CLI Tool
Language: Python
Project Type: Terminal-based Application

Comprehensive Report: Python Password Strength Checker Project

Objective

To create a Python-based CLI application that evaluates password strength and generates secure passwords. This project must meet specific programming criteria related to variable usage, control structures, data types, code modularity, documentation, and list manipulation.

What is a CLI?

A **Command Line Interface (CLI)** is a way to interact with a computer program by typing commands into a text-based console or terminal. Instead of clicking buttons or icons like in a graphical interface, you enter specific instructions as text. CLIs are powerful tools favored by developers and cybersecurity professionals because they offer precise control, faster workflows, and easy automation — all through simple text commands.

Project Features Overview

1. Interactive CLI menu to check password strength or generate a password.

2. Custom strength evaluation function using logical rules.
3. Random strong password generator based on mixed character sets.
4. Suggestions for improvement when passwords are weak.
5. Loop-based structure to allow repeated use until the user exits.

Requirement Breakdown and Fulfillment with Definitions

1. Descriptive Variable Names

Definition: Variables should have names that clearly describe their content or purpose. This improves code readability and maintainability.

In Project:

- user_password — the password input by the user.
- strength_score — numerical strength rating.
- feedback_messages — list storing suggestions for weak passwords.
- has_lowercase, has_uppercase, has_digit, has_symbol — booleans used in evaluation.
- generated_password — holds the output of the password generator.

2. Three Distinct Data Types

Definition: Use at least three types of data in your program, such as strings, integers, and booleans, to demonstrate understanding of Python's data handling.

In Project:

- String (str): used for password input, character pools.

- Integer (int): used for strength scoring.
- Boolean (bool): used to determine presence of character types.

3. Decision-Making Structure

Definition: Use conditional logic (if, elif, else) to allow the program to make decisions based on user input or data.

In Project:

- Password evaluation uses decision structures to assess length and character complexity.
- Menu selection handled by evaluating user_choice against expected values.

4. Looping Structure

Definition: Use a loop (for, while) to repeat actions until a condition is met.

In Project:

- while True: loop lets the user repeatedly choose menu options.
- for tip in suggestions: loop is used to iterate over feedback list and display tips.

5. Custom Functions

Definition: Create reusable blocks of code that perform a specific task and can be called multiple times.

In Project:

- evaluate_password_strength(user_password) calculates score and returns suggestions.
- generate_strong_password(length=16) creates a random strong password.

6. Use of a List (Sequence)

Definition: Use Python's list type to hold a collection of data and iterate over it as needed.

In Project:

- feedback_messages is a list that stores improvement tips.
- Iterated with a for loop to display suggestions to the user.

7. Documentation ***

Definition: Use comments and docstrings to explain the purpose of your code and how it works.

In Project:

- Every function includes a clear """docstring""".
- Inline comments explain decision logic and flow.

Conclusion

This CLI project not only serves as a functional password evaluation tool but demonstrates strong foundational programming practices. It integrates clear logic, multiple data types, modular functions, and appropriate use of structures. It is fully interactive and meets all educational criteria required for successful completion.

Files Used:

- Password_strength_checker.py

***Documentation

Effective documentation is a fundamental practice in software development that improves code readability, maintainability, and collaboration. In this project, clear documentation is incorporated through function docstrings and inline comments, ensuring that both the purpose and inner workings of the code are well-explained.

Function Docstrings

Each function in the project includes a descriptive docstring at the beginning. For example, the `evaluate_password_strength` function contains the following:

python

Copy code

```
"""

Evaluates the strength of a given password based on length, character types, and uniqueness.

Returns a tuple: (strength_score, feedback_messages).

"""
```

This docstring clearly states:

- **What** the function does (evaluates password strength)
- **How** it evaluates (based on length, character types, uniqueness)
- **What it returns** (a strength score and improvement suggestions)

Similarly, the `generate_strong_password` function includes:

python

Copy code

```
"""

Generates a secure password using lowercase, uppercase, digits, and symbols.

"""
```

This clearly communicates the function's role in creating secure, randomized passwords.

Inline Comments

Inline comments are used throughout the code to explain specific logic steps. For example:

python

Copy code

```
strength_score = 0  # Initialize strength score

feedback_messages = []  # List to hold suggestions for improvement


# Booleans to track character variety

has_lowercase = any(char.islower() for char in user_password)  # Check
for lowercase

has_uppercase = any(char.isupper() for char in user_password)  # Check
for uppercase
```

These comments guide readers through each logic step and clarify the reasoning behind variable use and condition checks.

Why Documentation Matters

- **Improves Readability:** Makes the code easier to understand at a glance.
- **Facilitates Maintenance:** Allows future updates or debugging without relearning the logic.
- **Supports Collaboration:** Enables others to contribute or review the code effectively.

- **Encourages Best Practices:** Shows professionalism and thoughtful structure.

Conclusion:
This project demonstrates strong documentation practices through meaningful comments and detailed function descriptions, resulting in code that is maintainable, understandable, and professional.

Presentation Notes: Password Strength Checker (Python CLI Project)

Student Name: AJ
Presentation Style: Speaker Notes with Expandable Talking Points
Project Type: Python Terminal-Based App

Slide 1: Introduction

Speaker Note:

- "Today I'm presenting my Python project — a Password Strength Checker."
- "It helps users evaluate their password security and generate strong passwords."
- "It runs in the terminal and follows key software development principles."

Slide 2: Why This Project?

Speaker Note:

- "Strong passwords are a crucial part of cybersecurity."
- "This tool helps educate users about what makes a password strong."
- "It also shows how simple logic and Python structures can create useful tools."

Slide 3: Features Overview

Speaker Note:

- "There's an interactive menu: check a password, generate one, or exit."
- "If a password is weak, the app gives improvement suggestions."
- "It uses a custom scoring system based on length and character variety."

Slide 4: Programming Requirements Met

Speaker Note:

- "This project was designed to meet specific class programming goals."
- "Let me break those down one by one."

Slide 5: Descriptive Variable Names

Speaker Note:

- "I used clear names like `user_password`, `strength_score`, and `feedback_messages`."
- "This improves readability and makes the code easier to debug or expand."

Slide 6: Use of Three Data Types

Speaker Note:

- "Strings were used for input and generation."
- "Integers calculated strength scores."
- "Booleans checked if the password had certain character types."

- "Using multiple data types shows comfort with Python's versatility."

Slide 7: Decision Structures

Speaker Note:

- "The app makes decisions using `if`, `elif`, and `else` conditions."
- "For example, it checks if the password includes lowercase letters and gives feedback accordingly."

Slide 8: Looping Structure

Speaker Note:

- "I used a `while True:` loop so the program keeps running until the user decides to exit."
- "I also looped over a list of feedback tips to display each suggestion one by one."

Slide 9: Custom Functions

Speaker Note:

- "I created two functions: one for checking password strength and one for generating strong passwords."
- "Functions help keep the code modular and reusable."

Slide 10: List Usage

Speaker Note:

- "I used a list called `feedback_messages` to hold suggestions for improving weak passwords."

- "Then I used a loop to print each message out."

Slide 11: Code Documentation

Speaker Note:

- "Each function has a docstring explaining what it does."
- "I also added comments next to key lines of logic."
- "This helps future developers (or myself) understand the purpose of the code."

🔹

🔸 Clear docstrings for each function

- Inline comments explain logic
- Improves readability and collaboration
- Shows professional coding habits

🎤 **Speaker Notes:**

> “Every function in my script includes a short explanation — a docstring — that outlines what it does, what input it expects, and what it returns.”
>
> “I also use inline comments to walk through my logic — like explaining how I check for lowercase letters or why I initialized certain variables.”
>
> “Good documentation is one of the best habits in coding. It helps others — or your future self — understand what’s happening quickly, even if you haven’t looked at the code in a while.”
>
> “This is especially important in cybersecurity, where clarity and traceability can impact audits and response time.”
>
> “It also shows that I’ve thought through the project carefully and professionally.”

Slide 12: Demo & Walkthrough (Optional)

Speaker Note:

- "Let me show how it works. I'll enter a weak password and walk through the feedback."
- "Then I'll generate a strong password and show how that's evaluated."

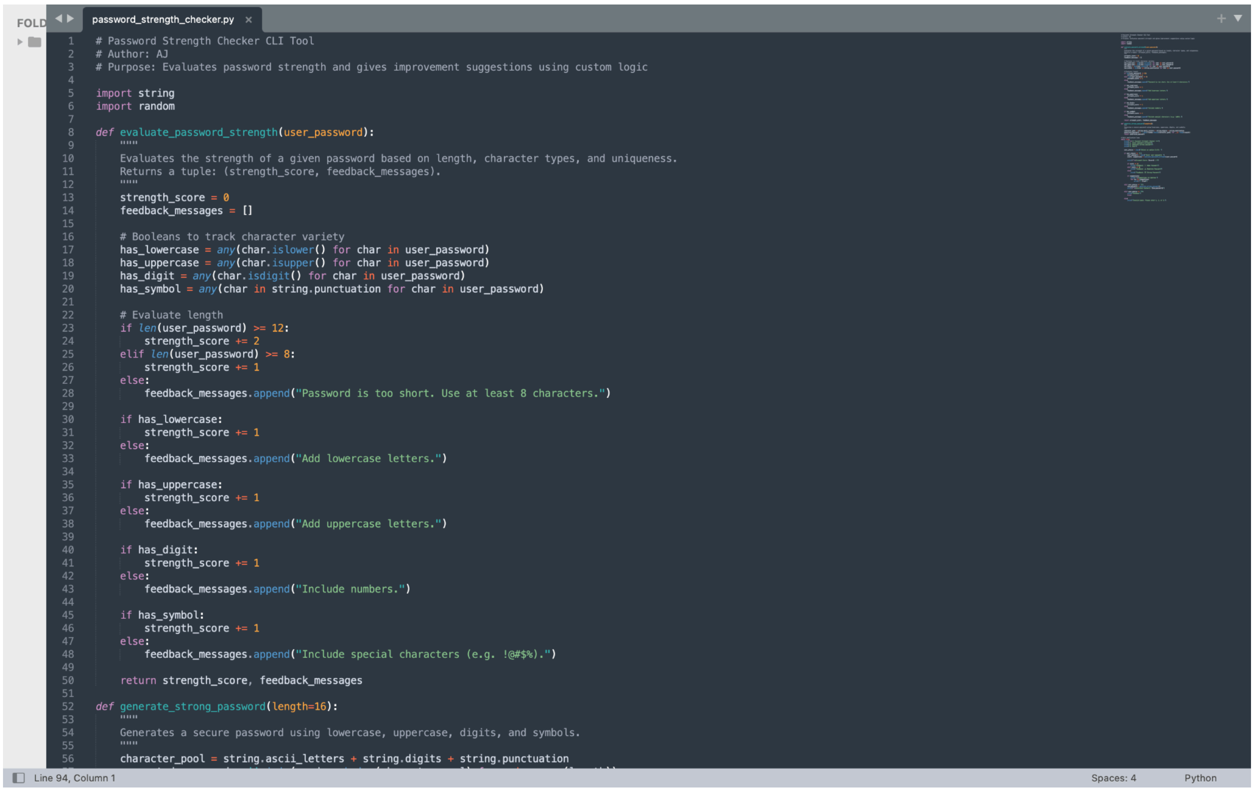
Slide 13: Conclusion

Speaker Note:

- "This project demonstrates basic but powerful programming concepts."
- "It solves a real-world problem and follows clean coding principles."
- "Thank you — I'm happy to answer any questions."

```
Student@DAEDMAC09 desktop % python3 password_strength_checker.py


=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 1
Enter your password: AJ123456

Strength Score: 3 / 6
Feedback: ⚠️Moderate Password

Suggestions to improve:
- Add lowercase letters.
- Include special characters (e.g. !@#$%).

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 1
Enter your password: Aj12345!@#

Strength Score: 5 / 6
Feedback: ✅ Strong Password

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 1
Enter your password: AJwnvvownv234567!@#$

Strength Score: 6 / 6



Select an option (1-3): 1
Enter your password: AJwnvvownv234567!@#$

Strength Score: 6 / 6
Feedback: ✅ Strong Password

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 2

Generated Password: c8@O40;.2Y&\}M`%

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 2

Generated Password: JZGKB1tcfrw&,V*i

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 2

Generated Password: VHn)EoED*8nC)S|{

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1-3): 
```

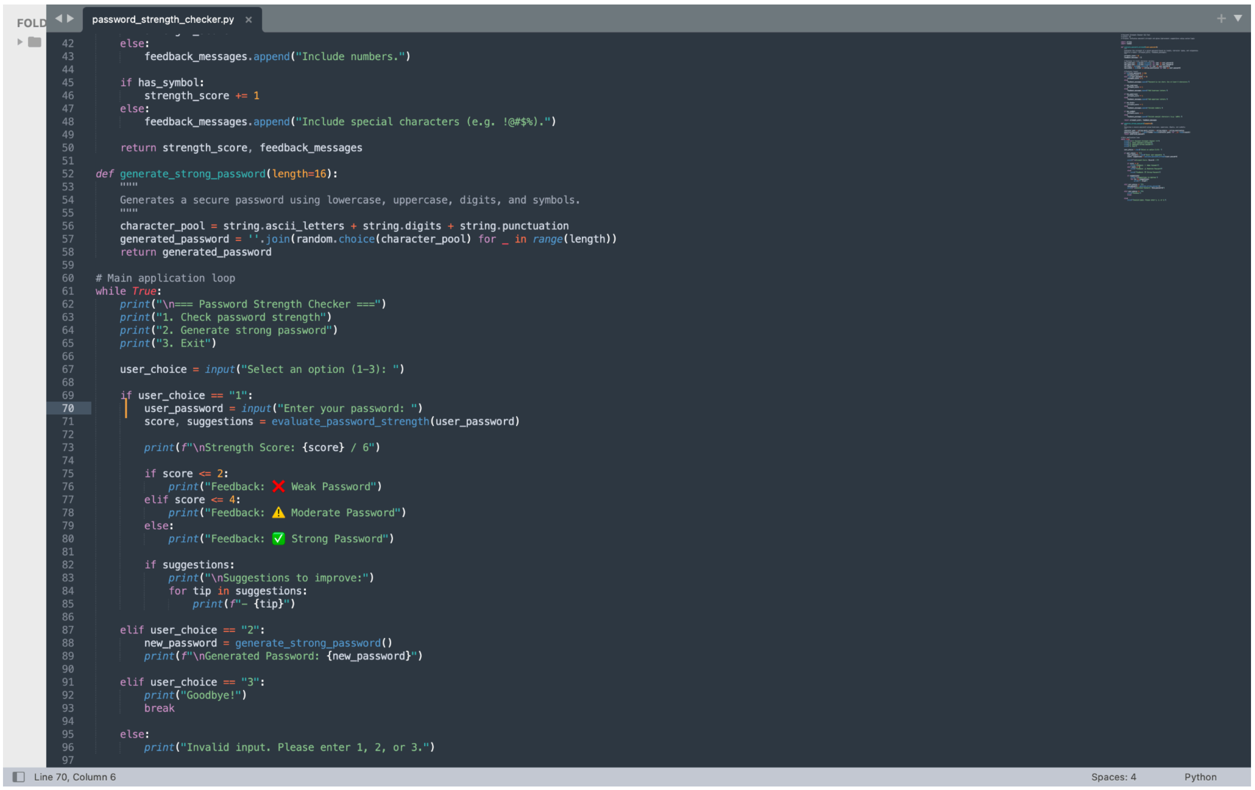
```
3. Exit
Select an option (1–3): 2

Generated Password: F2oa$.h}h/,T2ReR

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1–3): 3
Goodbye!
Student@DAEDMAC09 desktop % python3 password_strength_checker.py


=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1–3): 1
Enter your password: fycug

Strength Score: 1 / 6
Feedback: ❌ Weak Password

Suggestions to improve:
- Password is too short. Use at least 8 characters.
- Add uppercase letters.
- Include numbers.
- Include special characters (e.g. !@#$%).

=== Password Strength Checker ===
1. Check password strength
2. Generate strong password
3. Exit
Select an option (1–3): 
```



password_strength_checker.py

```python
# Password Strength Checker CLI Tool
# Author: AJ
# Purpose: Evaluates password strength and gives improvement suggestions using custom logic

import string
import random

def evaluate_password_strength(user_password):
    """
    Evaluates the strength of a given password based on length, character types, and uniqueness.
    Returns a tuple: (strength_score, feedback_messages).
    """
    strength_score = 0
    feedback_messages = []

    # Booleans to track character variety
    has_lowercase = any(char.islower() for char in user_password)
    has_uppercase = any(char.isupper() for char in user_password)
    has_digit = any(char.isdigit() for char in user_password)
    has_symbol = any(char in string.punctuation for char in user_password)

    # Evaluate length
    if len(user_password) >= 12:
        strength_score += 2
    elif len(user_password) >= 8:
        strength_score += 1
    else:
        feedback_messages.append("Password is too short. Use at least 8 characters.")

    if has_lowercase:
        strength_score += 1
    else:
        feedback_messages.append("Add lowercase letters.")

    if has_uppercase:
        strength_score += 1
    else:
        feedback_messages.append("Add uppercase letters.")

    if has_digit:
        strength_score += 1
    else:
        feedback_messages.append("Include numbers.")

    if has_symbol:
        strength_score += 1
    else:
        feedback_messages.append("Include special characters (e.g. !@#$%).")

    return strength_score, feedback_messages

def generate_strong_password(length=16):
    """
    Generates a secure password using lowercase, uppercase, digits, and symbols.
    """
    character_pool = string.ascii_letters + string.digits + string.punctuation
```

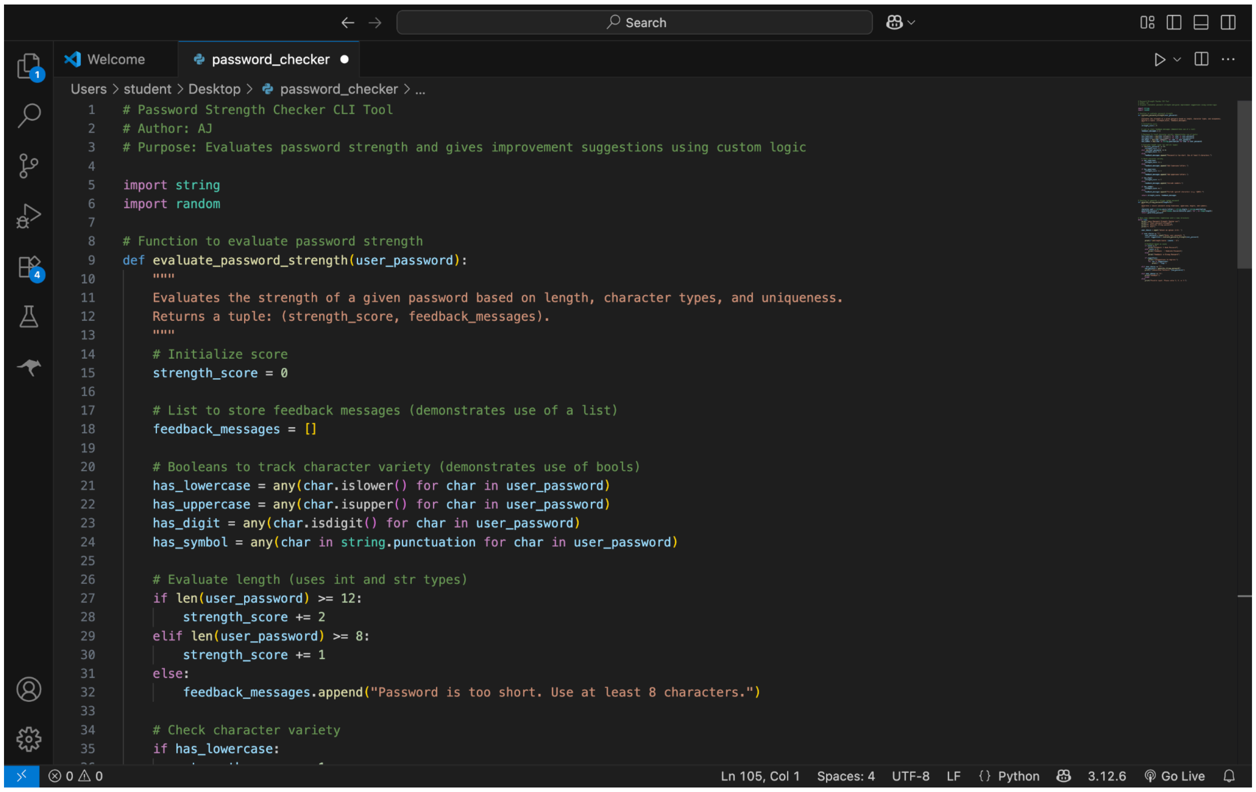
Line 94, Column 1   Spaces: 4   Python



password_strength_checker.py

```python
    else:
        feedback_messages.append("Include numbers.")

    if has_symbol:
        strength_score += 1
    else:
        feedback_messages.append("Include special characters (e.g. !@#$%).")

    return strength_score, feedback_messages

def generate_strong_password(length=16):
    """
    Generates a secure password using lowercase, uppercase, digits, and symbols.
    """
    character_pool = string.ascii_letters + string.digits + string.punctuation
    generated_password = ''.join(random.choice(character_pool) for _ in range(length))
    return generated_password

# Main application loop
while True:
    print("\n=== Password Strength Checker ===")
    print("1. Check password strength")
    print("2. Generate strong password")
    print("3. Exit")

    user_choice = input("Select an option (1-3): ")

    if user_choice == "1":
        user_password = input("Enter your password: ")
        score, suggestions = evaluate_password_strength(user_password)

        print(f"\nStrength Score: {score} / 6")

        if score <= 2:
            print("Feedback: ❌ Weak Password")
        elif score <= 4:
            print("Feedback: ⚠️ Moderate Password")
        else:
            print("Feedback: ✅ Strong Password")

        if suggestions:
            print("\nSuggestions to improve:")
            for tip in suggestions:
                print(f"- {tip}")

    elif user_choice == "2":
        new_password = generate_strong_password()
        print(f"\nGenerated Password: {new_password}")

    elif user_choice == "3":
        print("Goodbye!")
        break

    else:
        print("Invalid input. Please enter 1, 2, or 3.")
```

Line 70, Column 6 Spaces: 4 Python

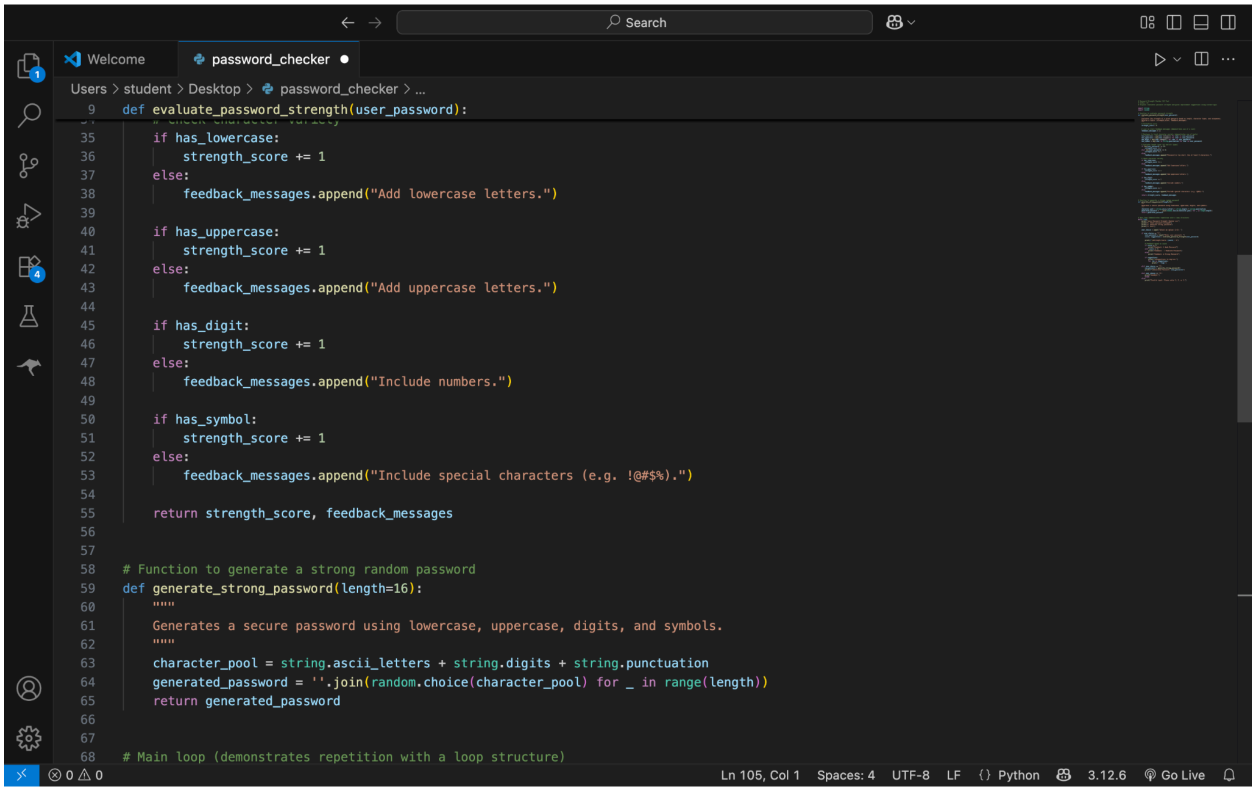


Welcome password_checker

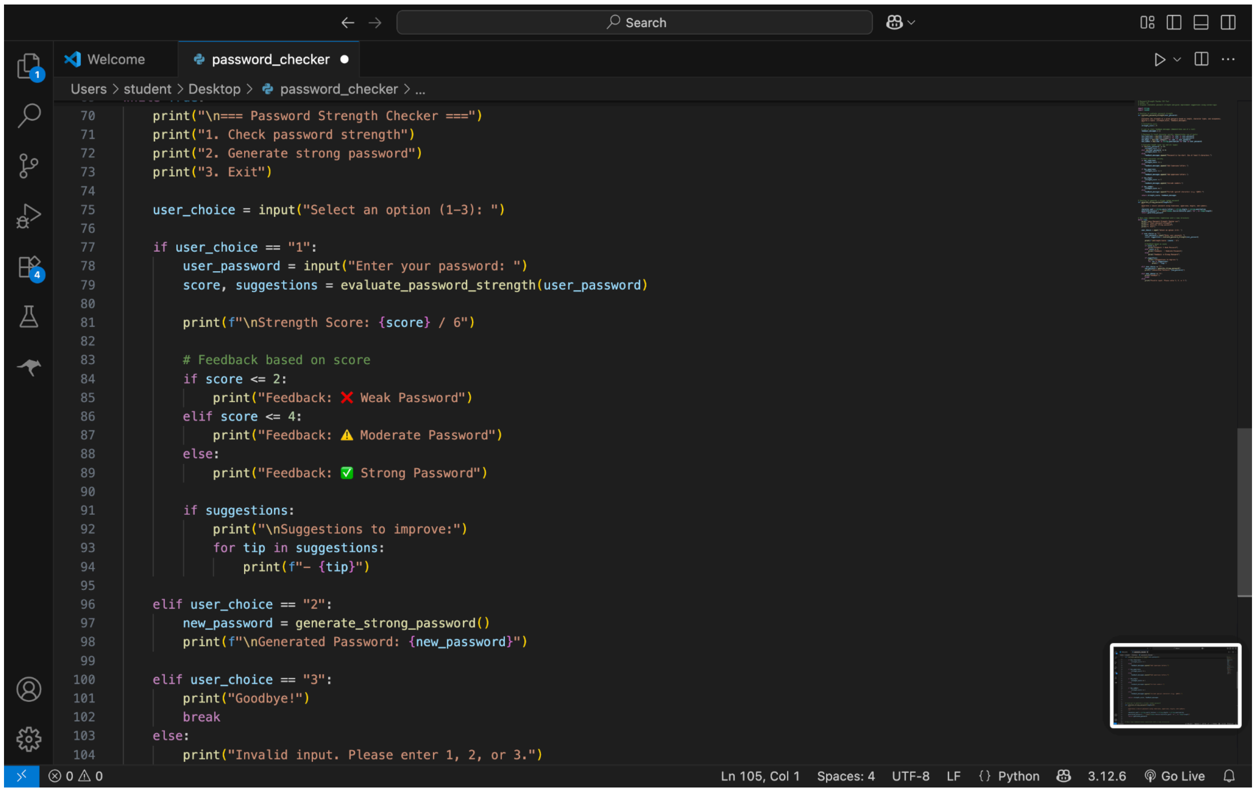
Users > student > Desktop > password_checker > ...

```python
# Password Strength Checker CLI Tool
# Author: AJ
# Purpose: Evaluates password strength and gives improvement suggestions using custom logic

import string
import random

# Function to evaluate password strength
def evaluate_password_strength(user_password):
    """
    Evaluates the strength of a given password based on length, character types, and uniqueness.
    Returns a tuple: (strength_score, feedback_messages).
    """
    # Initialize score
    strength_score = 0

    # List to store feedback messages (demonstrates use of a list)
    feedback_messages = []

    # Booleans to track character variety (demonstrates use of bools)
    has_lowercase = any(char.islower() for char in user_password)
    has_uppercase = any(char.isupper() for char in user_password)
    has_digit = any(char.isdigit() for char in user_password)
    has_symbol = any(char in string.punctuation for char in user_password)

    # Evaluate length (uses int and str types)
    if len(user_password) >= 12:
        strength_score += 2
    elif len(user_password) >= 8:
        strength_score += 1
    else:
        feedback_messages.append("Password is too short. Use at least 8 characters.")

    # Check character variety
    if has_lowercase:
```

Ln 105, Col 1 Spaces: 4 UTF-8 LF {} Python 3.12.6 Go Live



```python
def evaluate_password_strength(user_password):
    if has_lowercase:
        strength_score += 1
    else:
        feedback_messages.append("Add lowercase letters.")

    if has_uppercase:
        strength_score += 1
    else:
        feedback_messages.append("Add uppercase letters.")

    if has_digit:
        strength_score += 1
    else:
        feedback_messages.append("Include numbers.")

    if has_symbol:
        strength_score += 1
    else:
        feedback_messages.append("Include special characters (e.g. !@#$%).")

    return strength_score, feedback_messages


# Function to generate a strong random password
def generate_strong_password(length=16):
    """
    Generates a secure password using lowercase, uppercase, digits, and symbols.
    """
    character_pool = string.ascii_letters + string.digits + string.punctuation
    generated_password = ''.join(random.choice(character_pool) for _ in range(length))
    return generated_password


# Main loop (demonstrates repetition with a loop structure)
```



```python
    print("\n=== Password Strength Checker ===")
    print("1. Check password strength")
    print("2. Generate strong password")
    print("3. Exit")

    user_choice = input("Select an option (1-3): ")

    if user_choice == "1":
        user_password = input("Enter your password: ")
        score, suggestions = evaluate_password_strength(user_password)

        print(f"\nStrength Score: {score} / 6")

        # Feedback based on score
        if score <= 2:
            print("Feedback: ❌ Weak Password")
        elif score <= 4:
            print("Feedback: ⚠️ Moderate Password")
        else:
            print("Feedback: ✅ Strong Password")

        if suggestions:
            print("\nSuggestions to improve:")
            for tip in suggestions:
                print(f"- {tip}")

    elif user_choice == "2":
        new_password = generate_strong_password()
        print(f"\nGenerated Password: {new_password}")

    elif user_choice == "3":
        print("Goodbye!")
        break
    else:
        print("Invalid input. Please enter 1, 2, or 3.")
```