

Lab Assignment 8.1

1. (Password Strength Validator – Apply AI in Security Context)

Task:

Apply AI to generate at least 3 assert test cases for `is_strong_password(password)` and implement the validator function.

Requirements:

- Passwords must have at least 8 characters.
- Must include uppercase, lowercase, digit, and special character.
- Must not contain spaces.

Example Assert Test Cases:

- `assert is_strong_password("Abcd@123") == True`
- `assert is_strong_password("abcd123") == False`
- `assert is_strong_password("ABCD@1234") == True`

Expected Output #1:

- Password validation logic passing all AI-generated test cases.

The screenshot shows a code editor window with a dark theme. The file is named `ass_1_1.py`. The code defines a function `is_strong_password` that checks if a password is strong based on the requirements listed above. It uses several conditional statements to check for length, whitespace, uppercase, lowercase, digits, and special characters. Below the function definition are three assert statements testing the function with different passwords: "Abcd@123", "abcd123", and "ABCD@1234".

```
ass_1_1.py
def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not any(c.isupper() for c in password):
        return False
    if not any(c.islower() for c in password):
        return False
    if not any(c.isdigit() for c in password):
        return False
    if not any(c in "!@#$%^&*()-+" for c in password):
        return False
    assert is_strong_password("Abcd@123") == True
    assert is_strong_password("abcd123") == False
    assert is_strong_password("ABCD@1234") == True
```

Output:

The screenshot shows a terminal window with a dark theme. The terminal tab is active, showing the command `python ass_1_1.py` being run. The output shows the script's logic and the results of the three assert statements. The first two assertions fail with `AssertionError`, while the third one passes.

```
PS C:\Users\Himes\Documents\Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_1.py
Traceback (most recent call last):
  File "c:/Users/Himes/Documents/Dev/Python/ass_1_1.py", line 15, in <module>
    assert is_strong_password("Abcd@123") == True
                                         ^
AssertionError
PS C:\Users\Himes\Documents\Dev\Python>
```

2. (Number Classification with Loops – Apply AI for Edge Case Handling)

Task:

Use AI to generate at least 3 assert test cases for a `classify_number(n)` function.
Implement using loops.

Requirements:

- Classify numbers as Positive, Negative, or Zero.
- Handle invalid inputs like strings and None.
- Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

- `assert classify_number(10) == "Positive"`
- `assert classify_number(-5) == "Negative"`

- assert classify_number(0) == "Zero"

Expected Output:

Classification logic passing all assert tests.

```

ass_1_1.py ass_1_2.py
ass_1_2.py > ...
def classify_number(n):
    try:
        num = int(n)
    except (TypeError, ValueError):
        return "Enter a valid integer"
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    return "Zero"

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("123") == "Positive"
assert classify_number("-456") == "Negative"
assert classify_number("Hars") == "Enter a valid integer"

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_2.py
PS C:\Users\Himes\Documents\.Dev\Python>

```

3. (Anagram Checker – Apply AI for String Analysis)

Task:

Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.

Requirements:

- Ignore case, spaces, and punctuation.
- Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

- assert is_anagram("listen", "silent") == True
- assert is_anagram("hello", "world") == False
- assert is_anagram("Dormitory", "Dirty Room") == True

Expected Output:

Function correctly identifying anagrams and passing all AI-generated tests.

```

ass_1_1.py ass_1_2.py ass_1_3.py
ass_1_3.py > ...
def is_anagram(str1, str2):
    s1 = ""
    s2 = ""
    for s in str1:
        if s.isalpha():
            s1 += s
    for s in str2:
        if s.isalpha():
            s2 += s
    return sorted(s1) == sorted(s2)

assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_2.py
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_3.py
Traceback (most recent call last):
  File "<*>:1", line 1, in <module>
    assert is_anagram("Dormitory", "Dirty Room") == True
      ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
PS C:\Users\Himes\Documents\.Dev\Python>

```

4. (Inventory Class – Apply AI to Simulate Real-World Inventory System)

Task:

Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

Methods:

- add_item(name, quantity)
- remove_item(name, quantity)
- get_stock(name)

Example Assert Test Cases:

- inv = Inventory()
- inv.add_item("Pen", 10)
- assert inv.get_stock("Pen") == 10
- inv.remove_item("Pen", 5)
- assert inv.get_stock("Pen") == 5

```
ass_1_1.py ass_1_2.py ass_1_3.py ass_1_4.py x
ass_1_4.py >...
1  class Inventory:
2      def __init__(self):
3          self.Inventory = {}
4
5      def add_item(self, name, quantity):
6          if quantity <= 0:
7              return
8          if name in self.Inventory:
9              self.Inventory[name] += quantity
10         else:
11             self.Inventory[name] = quantity
12
13     def remove_item(self, name, quantity):
14         if quantity <= 0:
15             return
16         if name in self.Inventory:
17             if quantity >= self.Inventory[name]:
18                 self.Inventory[name] = 0
19             else:
20                 self.Inventory[name] -= quantity
21
22     def get_stock(self, name):
23         return self.Inventory.get(name, 0)
24
25 inv = Inventory()
26 inv.add_item("Pen", 10)
27 assert inv.get_stock("Pen") == 10
28 inv.remove_item("Pen", 5)
29 assert inv.get_stock("Pen") == 5
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_4.py
PS C:\Users\Himes\Documents\.Dev\Python>
```

5. (Date Validation & Formatting – Apply AI for Data Validation)

Task:

Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

Requirements:

- Validate "MM/DD/YYYY" format.
- Handle invalid dates.
- Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

- assert validate_and_format_date("10/15/2023") == "2023-10-15"
- assert validate_and_format_date("02/30/2023") == "Invalid Date"

- assert validate_and_format_date("01/01/2024") == "2024-01-01"

Expected Output:

Function passes all AI-generated assertions and handles edge cases.

```
ass_1_5.py <-- ass_1_2.py ass_1_3.py ass_1_4.py ass_1_5.py > ...
1 import datetime
2
3 def validate_and_format_date(date_str):
4     """
5         Validates a date string in 'MM/DD/YYYY' format, converts valid dates to 'YYYY-MM-DD',
6         and handles invalid dates/edge cases.
7
8     Args:
9         date_str (str): Input date string to validate and format
10
11    Returns:
12        str: Formatted date string if valid, otherwise "Invalid Date"
13    """
14    try:
15        # Validate format and parse date using strict MM/DD/YYYY pattern
16        date_obj = datetime.datetime.strptime(date_str, '%m/%d/%Y')
17        # Format valid dates to YYYY-MM-DD
18        return date_obj.strftime('%Y-%m-%d')
19    except ValueError:
20        return "Invalid Date"
21
22 # Test cases
23 assert validate_and_format_date("10/15/2023") == "2023-10-15"
24 assert validate_and_format_date("02/30/2023") == "Invalid Date"
25 assert validate_and_format_date("01/01/2024") == "2024-01-01"
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_5.py
PS C:\Users\Himes\Documents\.Dev\Python>
```