

ASSIGNMENT_8.1

HT.NO:2303A51731

Task Description #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:

- o Password must have at least 8 characters.
- o Must include uppercase, lowercase, digit, and special character.
- o 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.

CODE:

```

❸ practice.py > ...
1  #Task Description #1 (Password Strength Validator - Apply AI in Security Context)
2  #Task: Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.
3  """
4      o Password must have at least 8 characters.
5      o Must include uppercase, lowercase, digit, and special character.
6
7      o Must not contain spaces.
8  Example Assert Test Cases:
9  assert is_strong_password("Abcd@123") == True
10 assert is_strong_password("abcd123") == False
11 • Password validation logic passing all AI-generated test cases"""
12 #fix the code below
13
14 import re
15 def is_strong_password(password):
16     if len(password) < 8:
17         return False
18     if ' ' in password:
19         return False
20     if not re.search(r'[A-Z]', password):
21         return False
22     if not re.search(r'[a-z]', password):
23         return False
24     if not re.search(r'[0-9]', password):
25         return False
26     if not re.search(r'[\W_]', password):
27         return False
28     return True
29 # AI-generated test cases
30 assert is_strong_password("Abcd@123") == True
31 assert is_strong_password("abcd123") == False
32 assert is_strong_password("ABCD@1234") == False
33 assert is_strong_password("A1b2C3d4l!") == True
34 assert is_strong_password("Short1!") == False
35 assert is_strong_password("NoSpecialChar1") == False
36 assert is_strong_password("With Space1@") == False
37 assert is_strong_password("ValidPass#9") == True
38 assert is_strong_password("nouppercase@") == False
39 assert is_strong_password("NOUPERCASE1@") == False
40 assert is_strong_password("NoDigit@") == False
41 print("All test cases passed!")
42
43

```

OUTPUT:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
❸ PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/Activate.ps1"
❸ PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/python.exe" "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/practice.py"
❸ All test cases passed!
❸ (.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING>

```

Task Description #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:
 - o Classify numbers as Positive, Negative, or Zero.
 - o Handle invalid inputs like strings and None.

- o 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 #2:

- Classification logic passing all assert tests.

CODE:

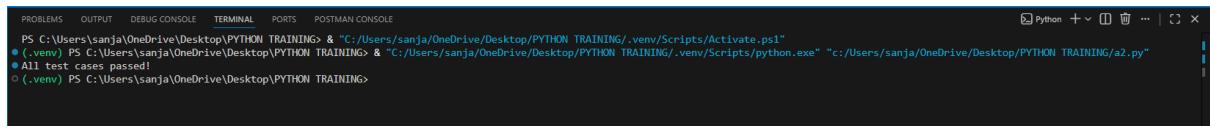
```

  practice.py      a2.py      X
  a2.py > ...

1   #Task Description #2 (Number Classification with Loops - ApplyAI for Edge Case Handling)
2   # Task: Use AI to generate at least 3 assert test cases for a
3   #classify_number(n) function. Implement using loops.
4   # Requirements:
5   """o Classify numbers as Positive, Negative, or Zero.
6   o Handle invalid inputs like strings and None.
7   o Include boundary conditions (-1, 0, 1).
8   Example Assert Test Cases:
9   assert classify_number(10) == "Positive"
10  assert classify_number(-5) == "Negative"
11  assert classify_number(0) == "Zero"
12  Expected Output #2:
13  • Classification logic passing all assert tests."""
14 def classify_number(n):
15     if not isinstance(n, (int, float)):
16         return "Invalid input"
17     if n > 0:
18         return "Positive"
19     elif n < 0:
20         return "Negative"
21     else:
22         return "Zero"
23 # Assert Test Cases
24 assert classify_number(10) == "Positive"
25 assert classify_number(-5) == "Negative"
26 assert classify_number(0) == "Zero"
27 assert classify_number(1) == "Positive"
28 assert classify_number(-1) == "Negative"
29 assert classify_number("string") == "Invalid input"
30 assert classify_number(-0.0001) == "Negative"
31 print(["All test cases passed!"])
32
33
34

```

OUTPUT:



A screenshot of a terminal window titled "Python". The window shows the following command and its output:

```
PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/Activate.ps1"
● (.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/python.exe" "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/a2.py"
● All test cases passed!
○ (.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING>
```

Task Description #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 #3:

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

CODE:

```
practice.py x a2.py x
C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING\practice.py
1 # Task Description #3 (Anagram Checker - Apply AI for StringAnalysis)
2 # Task: Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.
3 """
4 Requirements:
5 o Ignore case, spaces, and punctuation.
6 o Handle edge cases (empty strings, identical words).
7 Example Assert Test Cases:
8 assert is_anagram("listen", "silent") == True
9 assert is_anagram("hello", "world") == False
10 assert is_anagram("Dormitory", "Dirty Room") == True
11 Expected Output #3:
12 • Function correctly identifying anagrams and passing all AI-generated tests."""
13
14
15 def is_anagram(str1, str2):
16     # Remove non-alphabetic characters and convert to lowercase
17     cleaned1 = ''.join(c.lower() for c in str1 if c.isalpha())
18     cleaned2 = ''.join(c.lower() for c in str2 if c.isalpha())
19     # Check if sorted characters are equal
20     return sorted(cleaned1) == sorted(cleaned2)
21
22 # Assert Test Cases
23 assert is_anagram("listen", "silent") == True
24 assert is_anagram("hello", "world") == False
25 assert is_anagram("Dormitory", "Dirty Room") == True
26 assert is_anagram("Astronomer", "Moon starer") == True
27 assert is_anagram("The Morse Code", "Here come dots!") == True
28 assert is_anagram("", "") == True # Edge case: both strings empty
29 assert is_anagram("a", "A") == True # Edge case: single character, different cases
30 assert is_anagram("abc", "ab") == False # Edge case: different lengths
31 assert is_anagram("123", "321") == True # Edge case: numeric characters ignored
32 assert is_anagram("A gentleman", "Elegant man") == True # Edge case: spaces and cases ignored
33 print("All test cases passed!")
```

OUTPUT:

Task Description #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:

- o add_item(name, quantity)
 - o remove_item(name, quantity)
 - o 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  
inv.add_item("Book", 3)  
assert inv.get_stock("Book") == 3
```

Expected Output #4:

- Fully functional class passing all assertions.

CODE:

```

practice.py a2.py
a2.py > Inventory

1 # Task Description #4 (Inventory Class - Apply AI to Simulate Real- World Inventory System)
2 #Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
3 #+ Methods:
4 """add_item(name, quantity)
5 o remove_item(name, quantity)
6 o get_stock(name)
7 Example Assert Test Cases:
8 inv = Inventory()
9 inv.add_item("Pen", 10)
10 assert inv.get_stock("Pen") == 10
11 inv.remove_item("Pen", 5)
12 assert inv.get_stock("Pen") == 5
13 inv.add_item("Book", 3)
14 assert inv.get_stock("Book") == 3
15 Expected Output #4:
16 • Fully functional class passing all assertions"""
17 #use assert Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
18 class Inventory:
19     def __init__(self):
20         self.stock = {}
21
22     def add_item(self, name, quantity):
23         if name in self.stock:
24             self.stock[name] += quantity
25         else:
26             self.stock[name] = quantity
27
28     def remove_item(self, name, quantity):
29         if name in self.stock and self.stock[name] >= quantity:
30             self.stock[name] -= quantity
31         else:
32             raise ValueError("Not enough stock to remove")
33
34     def get_stock(self, name):
35         return self.stock.get(name, 0)
36
37 # Example Assert Test Cases:
38 inv = Inventory()
39 inv.add_item("Pen", 10)
40 assert inv.get_stock("Pen") == 10
41 inv.remove_item("Pen", 5)
42 assert inv.get_stock("Pen") == 5
43 inv.add_item("Book", 3)
44 assert inv.get_stock("Book") == 3
45 # Additional Assert Test Cases:
46 inv.add_item("Notebook", 7)
47 assert inv.get_stock("Notebook") == 7
48 inv.remove_item("Notebook", 2)
49 assert inv.get_stock("Notebook") == 5
50 inv.add_item("Eraser", 4)
51 assert inv.get_stock("Eraser") == 4
52 inv.remove_item("Eraser", 4)
53 assert inv.get_stock("Eraser") == 0
54 print("All tests passed!")
55

```

OUTPUT:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
● PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/Activate.ps1"
(venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/python.exe" "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/a2.py"
● All tests passed!
(venv) PS C:\Users\Sanja\OneDrive\Desktop\PYTHON TRAINING>

```

Task Description #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:

- o Validate "MM/DD/YYYY" format.
 - o Handle invalid dates.
 - o 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 #5:

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

CODE:

```
actions... py > ...
1 # Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)
2 # Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.
3 """
4     Requirements:
5         o Validate "MM/DD/YYYY" format.
6         o Handle invalid dates.
7         o Convert valid dates to "YYYY-MM-DD".
8 Example Assert Test Cases:
9 assert validate_and_format_date("10/15/2023") == "2023-10-15"
10 assert validate_and_format_date("02/30/2023") == "Invalid Date"
11 assert validate_and_format_date("01/01/2024") == "2024-01-01"
12 Expected Output #5:
13     • Function passes all AI-generated assertions and handles edge
14     cases.
15 """
16 from datetime import datetime
17 def validate_and_format_date(date_str):
18     try:
19         # Parse the date string to a datetime object
20         date_obj = datetime.strptime(date_str, "%m/%d/%Y")
21         # Return the date in "YYYY-MM-DD" format
22         return date_obj.strftime("%Y-%m-%d")
23     except ValueError:
24         # If parsing fails, return "Invalid Date"
25         return "Invalid Date"
26 # AI-generated assert test cases
27 assert validate_and_format_date("10/15/2023") == "2023-10-15"
28 assert validate_and_format_date("02/30/2023") == "Invalid Date"
29 assert validate_and_format_date("01/01/2024") == "2024-01-01"
30
31 print("All test cases passed!")
```

OUTPUT

A screenshot of a terminal window titled "Terminal". The window shows command-line output for a Python test suite. The output includes several lines of text indicating test cases passed and failed, along with file paths and command-line arguments. The terminal has a light blue background with white text and a dark blue header bar.

