# AI ASSISTED CODING

# ASSIGNMENT-8.1

**Name: Harshitha Guda**

**H.T.No: 2303A51102**

## 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:

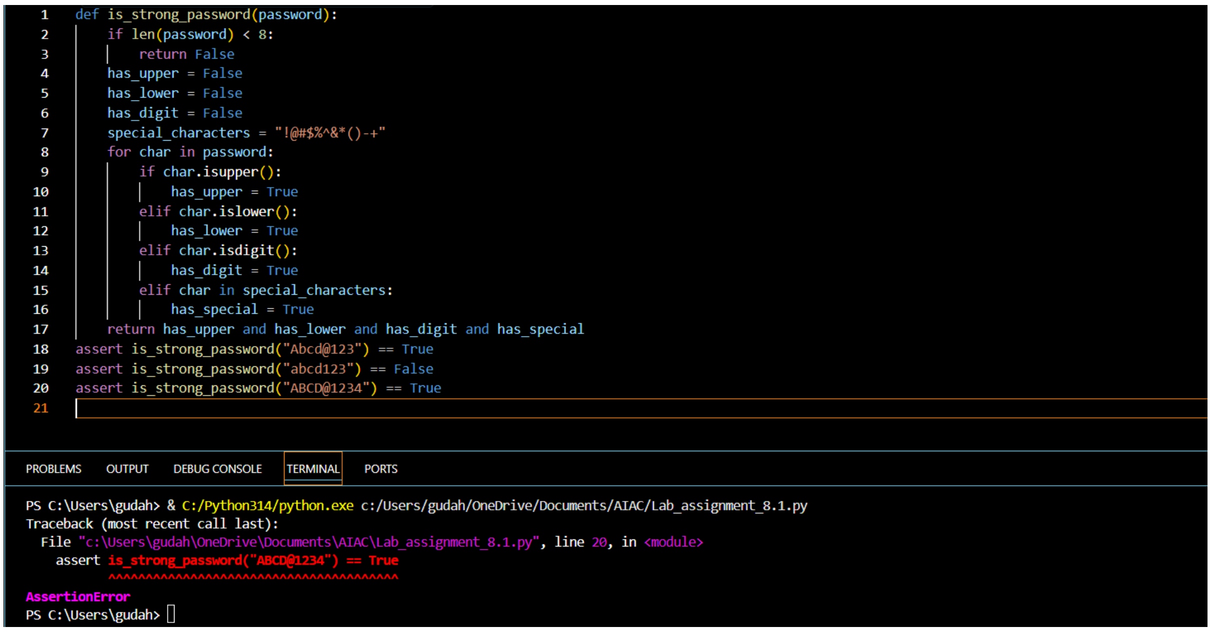
assert is_strong_password("Abcd@123") == True

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.



```
 1  def is_strong_password(password):
 2      if len(password) < 8:
 3          return False
 4      has_upper = False
 5      has_lower = False
 6      has_digit = False
 7      special_characters = "!@#$%^&*()-+"
 8      for char in password:
 9          if char.isupper():
10              has_upper = True
11          elif char.islower():
12              has_lower = True
13          elif char.isdigit():
14              has_digit = True
15          elif char in special_characters:
16              has_special = True
17      return has_upper and has_lower and has_digit and has_special
18  assert is_strong_password("Abcd@123") == True
19  assert is_strong_password("abcd123") == False
20  assert is_strong_password("ABCD@1234") == True
21  
```

```
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.1.py
Traceback (most recent call last):
  File "c:\Users\gudah\OneDrive\Documents\AIAC\Lab_assignment_8.1.py", line 20, in <module>
    assert is_strong_password("ABCD@1234") == True
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

AssertionError
PS C:\Users\gudah> 
```

**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:

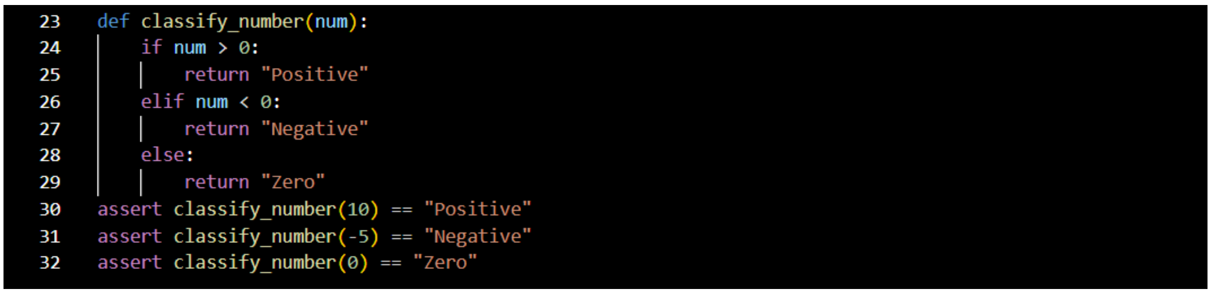
assert classify_number(10) == "Positive"

assert classify_number(-5) == "Negative"

assert classify_number(0) == "Zero"

Expected Output #2:

• Classification logic passing all assert tests.



```python
def classify_number(num):
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
```

**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:

o Ignore case, spaces, and punctuation.

o Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

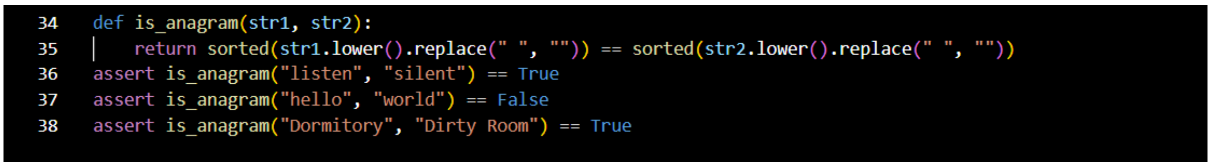
assert is_anagram("listen", "silent") == True

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.



```python
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ", "")) == sorted(str2.lower().replace(" ", ""))
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
```

**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)

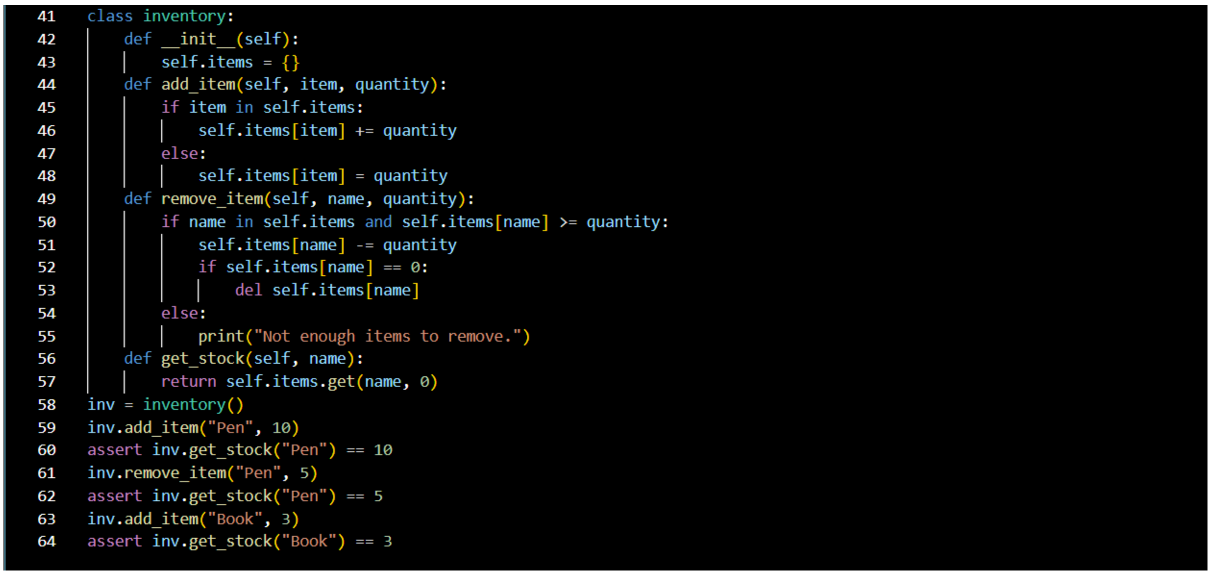
assert inv.get_stock("Pen") == 5

inv.add_item("Book", 3)

assert inv.get_stock("Book") == 3

Expected Output #4:

• Fully functional class passing all assertions.



```python
class inventory:
    def __init__(self):
        self.items = {}
    def add_item(self, item, quantity):
        if item in self.items:
            self.items[item] += quantity
        else:
            self.items[item] = quantity
    def remove_item(self, name, quantity):
        if name in self.items and self.items[name] >= quantity:
            self.items[name] -= quantity
            if self.items[name] == 0:
                del self.items[name]
        else:
            print("Not enough items to remove.")
    def get_stock(self, name):
        return self.items.get(name, 0)
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
```

**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:

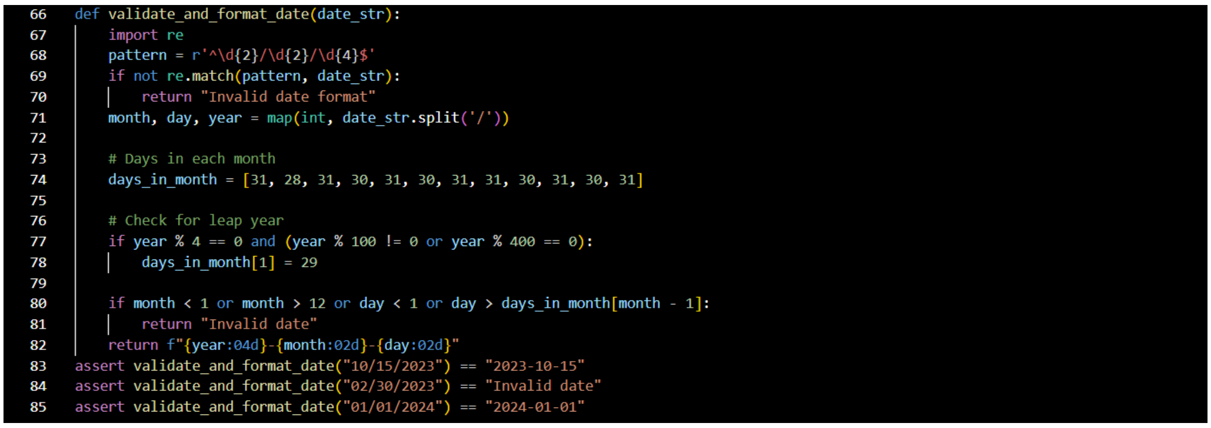
assert validate_and_format_date("10/15/2023") == "2023-10-15"

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.



```python
def validate_and_format_date(date_str):
    import re
    pattern = r'^\d{2}/\d{2}/\d{4}$'
    if not re.match(pattern, date_str):
        return "Invalid date format"
    month, day, year = map(int, date_str.split('/'))

    # Days in each month
    days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    # Check for leap year
    if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
        days_in_month[1] = 29

    if month < 1 or month > 12 or day < 1 or day > days_in_month[month - 1]:
        return "Invalid date"
    return f"{year:04d}-{month:02d}-{day:02d}"
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"
```