AI Assisted coding

Assignment-8.1

NAME : Akula Vivek

2403A51258

Batch-11

Task-1:

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

Prompt:

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

Code:

```
import re

def is_strong_password(password):
    if len(password) < 8:
        return False

    if ' ' in password:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'\d', password):
        return False
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return False
    return True

AI-generated assert test cases
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
assert is_strong_password("Abcdefg1") == False
assert is_strong_password("Abc @1234") == False
assert is_strong_password("A1@bcdef") == True

print("All test cases passed!")
```

output:

All test cases passed!

Task-2:

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"

Prompt:

Use AI to generate at least 3 assert test cases for a classify_number(n) function and implement it using loops

code:

```
def classify_number(n):
    valid_types = [int, float]
    is_valid = False
    for t in valid_types:
        if isinstance(n, t):
            is_valid = True
            break
    if not is_valid:
        return "Invalid input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number("text") == "Invalid input"
assert classify_number(None) == "Invalid input"
```

output:

All test cases passed!

Task-3:

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

Prompt:

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

Code:

```python
import re

def is_anagram(str1, str2):
    pattern = re.compile(r'[^\w]')
    cleaned1 = pattern.sub('', str1).lower()
    cleaned2 = pattern.sub('', str2).lower()
    return sorted(cleaned1) == sorted(cleaned2)

assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("", "") == True
assert is_anagram("abc", "abc") == True
assert is_anagram("A gentleman", "Elegant man") == True

print("All test cases passed!")
```

output:

All test cases passed!

Task-4:

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
```

Prompt:

Ask AI to generate at least 3 assert-based tests for an **Inventory** class with stock management and implement the class

Code:

```python
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    def remove_item(self, name, quantity):
        if name in self.stock and self.stock[name] >= quantity:
            self.stock[name] -= quantity
        else:
            self.stock[name] = 0

    def get_stock(self, name):
        return self.stock.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

inv.remove_item("Pen", 10)  # Removing more than available
assert inv.get_stock("Pen") == 0

inv.add_item("Notebook", 7)
assert inv.get_stock("Notebook") == 7

print("All Inventory test cases passed!")
```

output:

All Inventory test cases passed!

Task-5:

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"

Prompt:

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

Code:

```
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"

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"
assert validate_and_format_date("13/01/2023") == "Invalid Date"
assert validate_and_format_date("04/31/2023") == "Invalid Date"
assert validate_and_format_date("12/31/2023") == "2023-12-31"

print("All Date validation test cases passed!")
```

output:

All Date validation test cases passed!