AI Assisted Coding

Lab Assignment 8.5

Name : V Sai Krishnachaithanya

Hall Ticket no : 2303A52154

Task Description #1 (Username Validator – Apply AI in Authentication Context)

• Task: Use AI to generate at least 3 assert test cases for a function is_valid_username(username) and then implement the function using Test-Driven Development principles.
• Requirements:
o Username length must be between 5 and 15 characters.
o Must contain only alphabets and digits.
o Must not start with a digit.
o No spaces allowed.
Example Assert Test Cases:
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
Expected Output #1:
• Username validation logic successfully passing all AI-generated test cases.

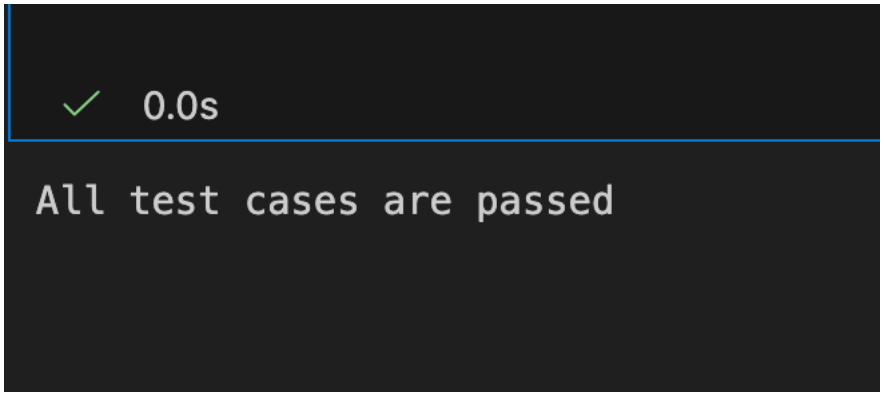
Prompt :

generate the python code for Username Validator where username length must be between 5 to 15 characters, no special characters are allowed,dont start with digit and no spaces are allowed.assign some assert test cases and output should be true for valid usernames and false for invalid usernames.and print all test caes are passed

```python
#generate the python code for Username Validator where username length must be between 5 to 15 characters, no special characters are allowed,dont

import re
def is_valid_username(username):
    # Check if the username length is between 5 and 15 characters
    if len(username) < 5 or len(username) > 15:
        return False

    # Check if the username starts with a digit
    if username[0].isdigit():
        return False

    # Check for special characters and spaces
    if re.search(r'[^a-zA-Z0-9]', username):
        return False

    return True
# Test cases
assert is_valid_username("validUser") == True
assert is_valid_username("user123") == True
assert is_valid_username("12user") == False  # Starts with a digit
assert is_valid_username("user name") == False  # Contains space
assert is_valid_username("user@name") == False  # Contains special character
assert is_valid_username("us") == False  # Too short
assert is_valid_username("thisisaverylongusername") == False  # Too long
print("All test cases are passed")
```

0.0s Python

Output:



Task Description #2 (Even–Odd & Type Classification – Apply AI for Robust Input Handling)

• Task: Use AI to generate at least 3 assert test cases for a function classify_value(x) and implement it using conditional logic and loops.

• Requirements:

If input is an integer, classify as "Even" or "Odd".

If input is 0, return "Zero".

If input is non-numeric, return "Invalid Input".

Example Assert Test Cases:

assert classify_value(8) == "Even"

assert classify_value(7) == "Odd"

assert classify_value("abc") == "Invalid Input"

Expected Output #2:
• Function correctly classifying values and passing all test cases.

Prompt:

generate a python function classify_value that takes an input value and classifies it as "Even", "Odd", or "zero". The function should return the classification as a string as invaild input. Include assert test cases to verify the correctness of the function.

```python
# generate a python function classify_value that takes an input value and classifies it
def classify_value(value):
    if not isinstance(value, int):
        return "Invalid input"
    if value == 0:
        return "Zero"
    elif value % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Test cases
assert classify_value(0) == "Zero"
assert classify_value(2) == "Even"
assert classify_value(3) == "Odd"
assert classify_value(-4) == "Even"
assert classify_value(-5) == "Odd"
assert classify_value("string") == "Invalid input"
assert classify_value(3.14) == "Invalid input"
print("All test cases are passed")
```

Output :

```
saikrishnachaitanya@Chaithanyas-MacBook-Pro AI_Assistant % python3 -m doctest classify.py -v
All test cases are passed
2 items had no tests:
    classify
    classify.classify_value
0 tests in 2 items.
0 passed and 0 failed.
Test passed.
saikrishnachaitanya@Chaithanyas-MacBook-Pro AI_Assistant %
```

Task Description #3 (Palindrome Checker – Apply AI for String Normalization)

• Task: Use AI to generate at least 3 assert test cases for a function is_palindrome(text) and implement the function.

• Requirements:
Ignore case, spaces, and punctuation.
Handle edge cases such as empty strings and single characters.
Example Assert Test Cases:
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Python") == False
Expected Output #3:
• Function correctly identifying palindromes and passing all AI-generated tests.

Prompt:

```python
import re

def is_palindrome(s):
    # Remove spaces and punctuation, and convert to lowercase
    cleaned_s = re.sub(r'[^a-zA-Z0-9]', '', s).lower()
    # Check if the cleaned string is equal to its reverse
    return cleaned_s == cleaned_s[::-1]

# Test cases: (input, expected)
test_cases = [
    ("", True),  # Empty string
    ("a", True),  # Single character
    ("A man, a plan, a canal, Panama", True),
    ("No 'x' in Nixon", True),
    ("Hello", False),
    ("Was it a car or a cat I saw?", True),
    ("Not a palindrome", False),
]

all_passed = True
for inp, expected in test_cases:
    result = is_palindrome(inp)
    print(f"Input: {repr(inp)} -> Output: {result} (Expected: {expected})")
    try:
        assert result == expected
    except AssertionError:
        all_passed = False
        print(f"Assertion failed for input: {repr(inp)}")

if all_passed:
    print("All test cases are passed")
else:
    raise AssertionError("Some test cases failed")
```

✓ 0.0s

Output:

```
✓ 0.0s
Input: '' -> Output: True (Expected: True)
Input: 'a' -> Output: True (Expected: True)
Input: 'A man, a plan, a canal, Panama' -> Output: True (Expected: True)
Input: "No 'x' in Nixon" -> Output: True (Expected: True)
Input: 'Hello' -> Output: False (Expected: False)
Input: 'Was it a car or a cat I saw?' -> Output: True (Expected: True)
Input: 'Not a palindrome' -> Output: False (Expected: False)
All test cases are passed
```

Task Description #4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)

• Task: Ask AI to generate at least 3 assert-based test cases for a BankAccount class and then implement the class.

• Methods:

deposit(amount)

withdraw(amount)

get_balance()

Example Assert Test Cases:

acc = BankAccount(1000)

acc.deposit(500)

assert acc.get_balance() == 1500

acc.withdraw(300)

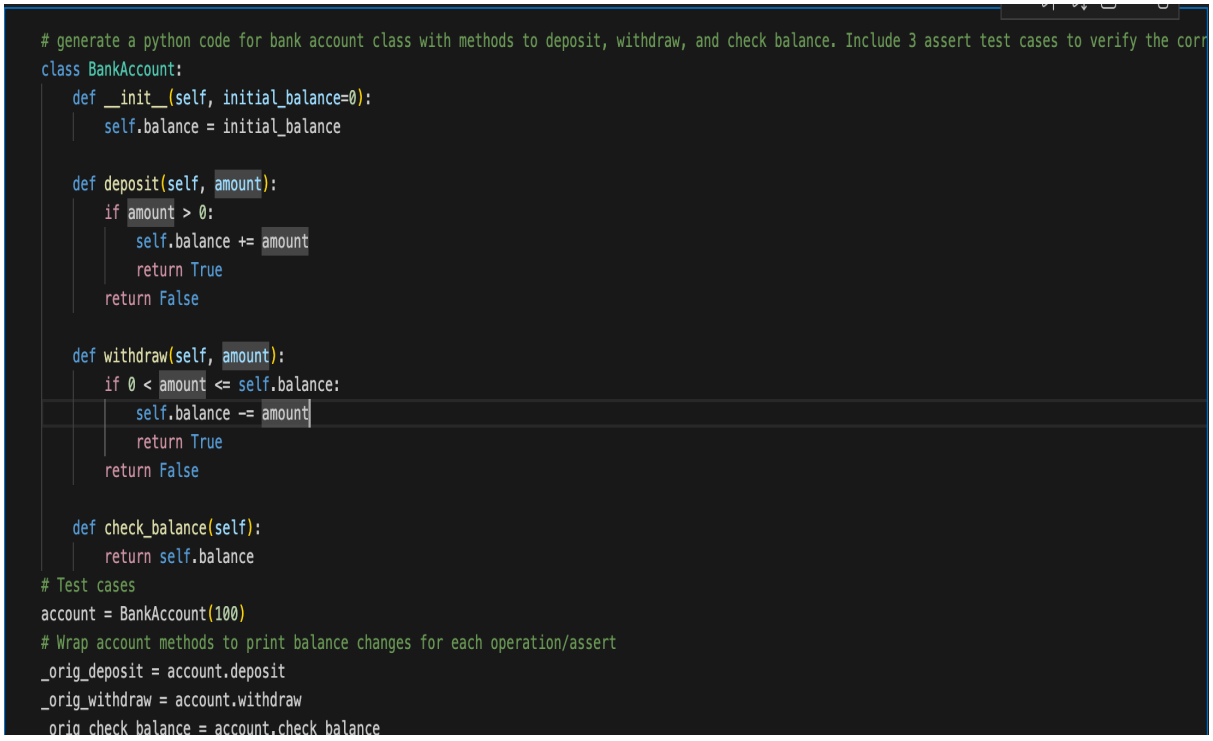
assert acc.get_balance() == 1200

Expected Output #4:

• Fully functional class that passes all AI-generated assertions.

Prompt:

generate a python code for bank account class with methods to deposit, withdraw, and check balance. Include 3 assert test cases to verify the correctness of the class methods. and print the results of the test cases.



```python
# generate a python code for bank account class with methods to deposit, withdraw, and check balance. Include 3 assert test cases to verify the corr
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return True
        return False

    def check_balance(self):
        return self.balance
# Test cases
account = BankAccount(100)
# Wrap account methods to print balance changes for each operation/assert
_orig_deposit = account.deposit
_orig_withdraw = account.withdraw
_orig_check_balance = account.check_balance
```

```python
def deposit(amount):
    before = _orig_check_balance()
    res = _orig_deposit(amount)
    after = _orig_check_balance()
    if res:
        print(f"Deposited {amount}: balance {before} -> {after}")
    else:
        print(f"Deposit of {amount} failed: balance unchanged {before}")
    return res

def withdraw(amount):
    before = _orig_check_balance()
    res = _orig_withdraw(amount)
    after = _orig_check_balance()
    if res:
        print(f"Withdrew {amount}: balance {before} -> {after}")
    else:
        print(f"Withdrawal of {amount} failed: balance unchanged {before}")
    return res

def check_balance():
    bal = _orig_check_balance()
    print(f"Checked balance: {bal}")
    return bal

account.deposit = deposit
account.withdraw = withdraw
account.check_balance = check_balance

assert account.check_balance() == 100  # Initial balance
assert account.deposit(50) == True  # Deposit money
assert account.check_balance() == 150  # Balance after deposit
assert account.withdraw(30) == True  # Withdraw money
assert account.check_balance() == 120  # Balance after withdrawal
assert account.withdraw(200) == False  # Withdraw more than balance
assert account.check_balance() == 120  # Balance should remain unchanged
print("All test cases are passed")
```

Output:

```
✓ 0.0s

Checked balance: 100
Deposited 50: balance 100 -> 150
Checked balance: 150
Withdrew 30: balance 150 -> 120
Checked balance: 120
Withdrawal of 200 failed: balance unchanged 120
Checked balance: 120
All test cases are passed
```

Task Description #5 (Email ID Validation – Apply AI for Data Validation)

• Task: Use AI to generate at least 3 assert test cases for a function validate_email(email) and implement the function.

• Requirements:

Must contain @ and .

Must not start or end with special characters.

Should handle invalid formats gracefully.

Example Assert Test Cases:
assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
Expected Output #5:
• Email validation function passing all AI-generated test cases
and handling edge cases correctly

Prompt:

generate a python function validate_email . my requirements are email should contain @ and not start or end with special characters. Include 3 assert test cases to verify the correctness of the function.

```python
def validate_email(email):
    """
    Validate an email according to:
    - must be a string
    - must contain exactly one '@'
    - must not contain spaces
    - must not start or end with a non-alphanumeric character
    - must have at least one character on each side of '@'
    """
    if not isinstance(email, str):
        return False
    if ' ' in email:
        return False
    if email.count('@') != 1:
        return False
    if len(email) < 3:
        return False
    if not email[0].isalnum() or not email[-1].isalnum():
        return False
    local, domain = email.split('@', 1)
    if not local or not domain:
        return False
    return True

# Test cases
assert validate_email("user@example.com") == True
assert validate_email("user.name+tag@sub.domain.co") == True
assert validate_email(".user@example.com") == False

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

Python

Output:

```
✓ 0.0s

Input: 'user@example.com' -> Output: True (Expected: True)
Input: 'user.name+tag@sub.domain.co' -> Output: True (Expected: True)
Input: '.user@example.com' -> Output: False (Expected: False)
Input: 'user@.com' -> Output: False (Expected: False)
Input: 'user@' -> Output: False (Expected: False)
Input: '@example.com' -> Output: False (Expected: False)
Input: 'user name@example.com' -> Output: False (Expected: False)
Input: 'user@exa mple.com' -> Output: False (Expected: False)
Input: 'us@example.com' -> Output: True (Expected: True)
All test cases are passed
```