sru SR UNIVERSITY

School of Computer Science and Artificial Intelligence

LAB ASSIGNMENT # 8.5

Program	: B. Tech (CSE)
Specialization	
Course Title	: Al Assisted Coding
Course Code	: 23CS002PC304
Semester	11
Academic Session	: 2025-2026
Name of Student	:.P.ESHWAR
Enrollment No.	: 2403A51L26
Batch No.	51
Date	: 30/01/26

Submission Starts here

Screenshots:

Task Description #1 (Username Validator — Apply Al in Authentication Context)

Prompt:

Generate at least 3 assert-based test cases for a Python function is_valid_username(username) with the following rules: - Username length must be between 5 and 15 characters. - Only alphabets and digits allowed. - Username must not start with a digit. - No spaces allowed. Then implement the function usin Test-Driven Develo ment so that all tests ass.

SR UNIVERSITY

```python
#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.
def is_valid_username(username):
    # Check length
    if not (5 <= len(username) <= 15):
        return False
    # Check if starts with a digit
    if username[0].isdigit():
        return False
    # Check for allowed characters and spaces
    if not username.isalnum():
        return False
    return True
# AI-generated test cases
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
assert is_valid_username("ValidUser1") == True
assert is_valid_username("TooLongUsername123") == False
assert is_valid_username("Short") == True
assert is_valid_username("No$pecialChar") == False
assert is_valid_username("1StartsWithDigit") == False
assert is_valid_username("NormalUser") == True
assert is_valid_username("User Name") == False
assert is_valid_username("User_123") == False
assert is_valid_username("User") == False
assert is_valid_username("ThisUsernameIsWayTooLong") == False
assert is_valid_username("User12345") == True
print("All test cases passed!")
```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/
All test cases passed!
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

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:

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

o If input is 0, return "Zero".

o 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"

SR UNIVERSITY

ai_coding.py > ...

```python
#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:
#o If input is an integer, classify as "Even" or "Odd".
#o If input is 0, return "Zero".
#o If input is non-numeric, return "Invalid Input".
def classify_value(x):
    if isinstance(x, int):
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    else:
        return "Invalid Input"
# Test cases
def test_classify_value():
    assert classify_value(4) == "Even", "Test Case 1 Failed"
    assert classify_value(7) == "Odd", "Test Case 2 Failed"
    assert classify_value(0) == "Zero", "Test Case 3 Failed"
    assert classify_value("hello") == "Invalid Input", "Test Case 4 Failed"
    assert classify_value(3.5) == "Invalid Input", "Test Case 5 Failed"
    print("All test cases passed!")
test_classify_value()
# AI-generated test cases
assert classify_value(-2) == "Even", "Test Case 6 Failed"
assert classify_value(15) == "Odd", "Test Case 7 Failed"
assert classify_value([]) == "Invalid Input", "Test Case 8 Failed"
assert classify_value(None) == "Invalid Input", "Test Case 9 Failed"
assert classify_value(100) == "Even", "Test Case 10 Failed"
assert classify_value(-7) == "Odd", "Test Case 11 Failed"
print("All AI-generated test cases passed!")

```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/py
All test cases passed!
All AI-generated test cases passed!
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

SR UNIVERSITY

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:

o Ignore case, spaces, and punctuation.

o 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

ai_coding.py > ...

```
#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:
#o Ignore case, spaces, and punctuation.
#o 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.
import string
def is_palindrome(text):
    # Normalize the text: convert to lowercase, remove spaces and punctuation
    normalized_text = ''.join(char.lower() for char in text if char.isalnum())
    # Check if the normalized text is equal to its reverse
    return normalized_text == normalized_text[::-1]
# AI-generated test cases
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Python") == False
assert is_palindrome("") == True  # Edge case: empty string
assert is_palindrome("A") == True  # Edge case: single character
assert is_palindrome("No 'x' in Nixon") == True  # Palindrome with punctuation
print("All test cases passed!")

```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\pythor
All test cases passed!
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

SR UNIVERSITY

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:**

o deposit(amount)

o withdraw(amount)

o 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

ai_coding.py > ...

```python
3#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:
#o deposit(amount)
#o withdraw(amount)
#o 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.
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

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

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

    def get_balance(self):
        return self.balance
# AI-generated test cases
acc = BankAccount(1000)
acc.deposit(500)
assert acc.get_balance() == 1500, "Test Case 1 Failed"
acc.withdraw(300)
assert acc.get_balance() == 1200, "Test Case 2 Failed"
acc.withdraw(2000)  # Attempt to withdraw more than balance
```

SR UNIVERSITY