

School of Computer Science and Artificial Intelligence  
**LAB ASSIGNMENT # 8.5**

Program : B. Tech (CSE)  
Specialization  
Course Title : AI 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 AI 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 using Test-Driven Development so that all tests pass.

```

ai_coding.py > ...
 8 #o Must contain only alphabets and digits.
 9 #o Must not start with a digit.
10 #o No spaces allowed.
11 #Example Assert Test Cases:
12 #assert is_valid_username("User123") == True
13 #assert is_valid_username("12User") == False#
14 #assert is_valid_username("Us er") == False
15 #Expected Output #1:
16 #• Username validation logic successfully passing all AI-
17 #generated test cases.
18 def is_valid_username(username):
19     # Check length
20     if not (5 <= len(username) <= 15):
21         return False
22     # Check if starts with a digit
23     if username[0].isdigit():
24         return False
25     # Check for allowed characters and spaces
26     if not username.isalnum():
27         return False
28     return True
29 # AI-generated test cases
30 assert is_valid_username("User123") == True
31 assert is_valid_username("12User") == False
32 assert is_valid_username("Us er") == False
33 assert is_valid_username("ValidUser1") == True
34 assert is_valid_username("TooLongUsername123") == False
35 assert is_valid_username("Short") == True
36 assert is_valid_username("No$pecialChar") == False
37 assert is_valid_username("iStartsWithDigit") == False
38 assert is_valid_username("NormalUser") == True
39 assert is_valid_username("User Name") == False
40 assert is_valid_username("User_123") == False
41 assert is_valid_username("User") == False
42 assert is_valid_username("ThisUsernameIsWayTooLong") == False
43 assert is_valid_username("User12345") == True
44 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:

- 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"

```

```
ai_coding.py > ...
1  #Task Description #2 (Even-Odd & Type Classification - Apply
2  #AI for Robust Input Handling)
3  #• Task: Use AI to generate at least 3 assert test cases for a
4  #function classify_value(x) and implement it using conditional
5  #logic and loops.
6  #• Requirements:
7  #o If input is an integer, classify as "Even" or "Odd".
8  #o If input is 0, return "Zero".
9  #o If input is non-numeric, return "Invalid Input".
10 def classify_value(x):
11     if isinstance(x, int):
12         if x == 0:
13             return "Zero"
14         elif x % 2 == 0:
15             return "Even"
16         else:
17             return "Odd"
18     else:
19         return "Invalid Input"
20 # Test cases
21 def test_classify_value():
22     assert classify_value(4) == "Even", "Test Case 1 Failed"
23     assert classify_value(7) == "Odd", "Test Case 2 Failed"
24     assert classify_value(0) == "Zero", "Test Case 3 Failed"
25     assert classify_value("hello") == "Invalid Input", "Test Case 4 Failed"
26     assert classify_value(3.5) == "Invalid Input", "Test Case 5 Failed"
27     print("All test cases passed!")
28 test_classify_value()
29 # AI-generated test cases
30 assert classify_value(-2) == "Even", "Test Case 6 Failed"
31 assert classify_value(15) == "Odd", "Test Case 7 Failed"
32 assert classify_value([]) == "Invalid Input", "Test Case 8 Failed"
33 assert classify_value(None) == "Invalid Input", "Test Case 9 Failed"
34 assert classify_value(100) == "Even", "Test Case 10 Failed"
35 assert classify_value(-7) == "Odd", "Test Case 11 Failed"
36 print("All AI-generated test cases passed!")
37
```

```
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>
```

### 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
```

```
ai_coding.py > ...
1  #Task Description #3 (Palindrome Checker – Apply AI for
2  #String Normalization)
3  #• Task: Use AI to generate at least 3 assert test cases for a
4  #function is_palindrome(text) and implement the function.
5  #• Requirements:
6  #◦ Ignore case, spaces, and punctuation.
7  #◦ Handle edge cases such as empty strings and single
8  #characters.
9  #Example Assert Test Cases:
10 #assert is_palindrome("Madam") == True
11 #assert is_palindrome("A man a plan a canal Panama") ==
12 #True
13 #assert is_palindrome("Python") == False
14 #Expected Output #3:
15 #• Function correctly identifying palindromes and passing all
16 #AI-generated tests.
17 import string
18 def is_palindrome(text):
19     # Normalize the text: convert to lowercase, remove spaces and punctuation
20     normalized_text = ''.join(char.lower() for char in text if char.isalnum())
21     # Check if the normalized text is equal to its reverse
22     return normalized_text == normalized_text[::-1]
23 # AI-generated test cases
24 assert is_palindrome("Madam") == True
25 assert is_palindrome("A man a plan a canal Panama") == True
26 assert is_palindrome("Python") == False
27 assert is_palindrome("") == True # Edge case: empty string
28 assert is_palindrome("A") == True # Edge case: single character
29 assert is_palindrome("No 'x' in Nixon") == True # Palindrome with punctuation
30 print("All test cases passed!")
```

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

## 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 > ...
1 3#Task Description #4 (BankAccount Class – Apply AI for
2 #Object-Oriented Test-Driven Development)
3 #• Task: Ask AI to generate at least 3 assert-based test cases for
4 #a BankAccount class and then implement the class.
5 #• Methods:
6 #o deposit(amount)
7 #o withdraw(amount)
8 #o get_balance()
9 #Example Assert Test Cases:
10 #acc = BankAccount(1000)
11 #acc.deposit(500)
12 #assert acc.get_balance() == 1500
13 #acc.withdraw(300)
14 #assert acc.get_balance() == 1200
15 #Expected Output #4:
16 #• Fully functional class that passes all AI-generated assertions.
17 class BankAccount:
18     def __init__(self, initial_balance=0):
19         self.balance = initial_balance
20
21     def deposit(self, amount):
22         if amount > 0:
23             self.balance += amount
24
25     def withdraw(self, amount):
26         if 0 < amount <= self.balance:
27             self.balance -= amount
28
29     def get_balance(self):
30         return self.balance
31 # AI-generated test cases
32 acc = BankAccount(1000)
33 acc.deposit(500)
34 assert acc.get_balance() == 1500, "Test Case 1 Failed"
35 acc.withdraw(300)
36 assert acc.get_balance() == 1200, "Test Case 2 Failed"
37 acc.withdraw(2000) # Attempt to withdraw more than balance
```

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with icons for file operations like New, Open, Save, and Find. The main area displays a Python script named `ai_coding.py`. The code contains several assert statements testing various deposit and withdraw operations on a `BankAccount` object. At the bottom of the code, a print statement confirms all tests passed.

```
ai_coding.py > ...
37 acc.withdraw(2000) # Attempt to withdraw more than balance
38 assert acc.get_balance() == 1200, "Test Case 3 Failed"
39 acc.deposit(-100) # Attempt to deposit a negative amount
40 assert acc.get_balance() == 1200, "Test Case 4 Failed"
41 # Additional AI-generated test case
42 acc2 = BankAccount()
43 assert acc2.get_balance() == 0, "Test Case 5 Failed"
44 acc2.deposit(100)
45 assert acc2.get_balance() == 100, "Test Case 6 Failed"
46 acc2.withdraw(50)
47 assert acc2.get_balance() == 50, "Test Case 7 Failed"
48 print("All test cases passed!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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