

## AI ASSISTANT CODING

### ASSIGNMENT-8

Name: Kommu Madhupriya

Hallticket:2303A51583

Batch:22

---

#### **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.

The screenshot shows a code editor with two tabs: 'priya.py' and 'assi 8.py'. The 'assi 8.py' tab contains Python code for testing a function named 'is\_valid\_username'. The code includes assertions for various username inputs like 'User123', '12User', 'Us er', 'User\_123', and 'User\_123'. It also includes a function definition and some example usage print statements. The terminal below shows the execution of the script and its output, which matches the expected results from the test cases.

```

File Edit Selection View Go Run Terminal Help ← → Q ai assisted coding
priya.py assi 8.py
1 #Task1
2 #TestCases
3 #assert is_valid_username("User123") == True
4 #assert is_valid_username("12User") == False
5 #assert is_valid_username("Us er") == False
6 #assert is_valid_username("User") == False
7 #assert is_valid_username("User_123") == False
8 #valid usernames correctly.
9 def is_valid_username(username):
10     if len(username) < 5 or len(username) > 15:
11         return False
12     if not username[0].isalpha():
13         return False
14     if not username.isalnum():
15         return False
16     return True
17 #example usage
18 print(is_valid_username("User123")) # True
19 print(is_valid_username("12User")) # False
20 print(is_valid_username("Us er")) # False
21 print(is_valid_username("User")) # False
22 print(is_valid_username("User_123")) # False
23
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:\Users\kommu\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/assi 8.py"
Python
powershell
True
False
False
False
False
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

```

**Observation:** AI-generated assert test cases helped define the username validation rules before coding. By writing tests first, the function was implemented to satisfy all constraints such as length limits, allowed characters, and starting character rules. This ensured the function was reliable and handled invalid usernames correctly.

## 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"

```

### Expected Output #2:

- Function correctly classifying values and passing all test cases.

The screenshot shows the VS Code interface with two files open: 'priya.py' and 'assi 8.py'. The 'assi 8.py' file contains the following code:

```

1  #!/usr/bin/python
2  def classify_value(value):
3      if value == 0:
4          return "Zero"
5      elif value % 2 == 0:
6          return "Even"
7      else:
8          return "Odd"
9
10 #example usage
11 print(classify_value(8)) # "Even"
12 print(classify_value(7)) # "Odd"
13 print(classify_value(0)) # "Zero"
14 print(classify_value("abc")) # "Invalid Input"
15 print(classify_value(2.5)) # "Invalid Input"

```

The terminal window shows the output of running the script:

```

PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:\Users\kommu\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/assi 8.py"
Even
Odd
Zero
Invalid Input
Invalid Input
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

```

The status bar at the bottom indicates the file is saved, the Python version is 3.13, and the date is 19-02-2026.

**Observation:** AI-assisted test cases guided the classification of different inputs such as integers, zero, and non-numeric values. The function correctly used conditional logic to identify even, odd, zero, and invalid inputs, improving robustness and error handling.

---

### 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.

The screenshot shows a code editor interface with two files open: `priya.py` and `ass18.py`. The `priya.py` file contains a function `is_palindrome` that checks if a string is a palindrome. The `ass18.py` file contains test cases for this function. Below the code editor is a terminal window showing the output of running `priya.py`. The terminal output is as follows:

```

PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:\Users\kommu\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/priya.py"
True
True
False
True
True
True
True
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

```

The terminal also shows the system tray with a weather icon indicating 29°C and sunny conditions.

**Observation:** AI-generated tests helped identify edge cases like spaces, punctuation, and case differences. String normalization techniques were applied to ensure accurate palindrome detection. The function successfully handled empty strings and single-character inputs.

---

#### 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.

```

File Edit Selection View Go Run Terminal Help ← → Q ai assisted coding
priya.py ● assi 8.py
priya.py > ⚡ BankAccount > __init__
17 #task4
18 #acc = BankAccount(1000)
19 #acc.deposit(500)
20 #assert acc.get_balance() == 1500
21 #acc.withdraw(300)
22 #assert acc.get_balance() == 1200
23 #acc.withdraw(2000)
24 #assert acc.get_balance() == 1200
25 class BankAccount:
26     def __init__(self, initial_balance=0):
27         self.balance = initial_balance
28     def deposit(self, amount):
29         if amount > 0:
30             self.balance += amount
31     def withdraw(self, amount):
32         if 0 < amount <= self.balance:
33             self.balance -= amount
34     def get_balance(self):
35         return self.balance
36 #example usage
37 acc = BankAccount(1000)
38 acc.deposit(500)
39 print(acc.get_balance()) # 1500
40 acc.withdraw(300)
41 print(acc.get_balance()) # 1200
42 acc.withdraw(2000)
43 print(acc.get_balance()) # 1200
44
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & c:\users\kommu\appdata\local\microsoft\windowsapps\python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/priya.py"
1500
1200
1200
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> []
In 27, Col 39 Spaces: 4 UTF-8 CRLF {} Python 🐍 Python 3.13
29°C Sunny
Search

```

**Observation:** AI-generated test cases helped design object-oriented methods before implementation. The class correctly handled deposits, withdrawals, and balance retrieval. Test-driven development ensured correct behavior and reduced logical errors in financial operations.

---

### 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:

- o Must contain @ and .
- o Must not start or end with special characters.
- o 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.

The screenshot shows a code editor interface with a dark theme. On the left, there are two tabs: 'priya.py' and 'assi 8.py'. The 'priya.py' tab is active, displaying the following Python code:

```
1 #task5
2 #assert validate_email("user@example.com") == True
3 #assert validate_email("user@example.com") == False
4 #assert validate_email("@gmail.com") == False
5 #assert validate_email("user@.com") == False
6 #assert validate_email("user@gmail") == False
7 def validate_email(email):
8     if not isinstance(email,str):
9         return False
10    if "@" not in email or "." not in email:
11        return False
12    if email[0] == "@" or email[-1] == ".":
13        return False
14    return True
15 #example usage
16 print(validate_email("user@example.com")) # Should return True
17 print(validate_email("user@example.com")) # Should return False
18 print(validate_email("@gmail.com")) # Should return False
19 print(validate_email("user@.com")) # Should return False
20 print(validate_email("user@gmail")) # Should return False
```

Below the code editor, there is a terminal window titled 'powershell' showing the execution of the script:

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:\Users\kommu\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/priya.py"
True
False
False
True
False
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

The system tray at the bottom of the screen shows the date and time as '19-02-2026' and '13:27'.

**Observation:** AI test cases guided the validation rules for email format. The function correctly checked for required symbols and invalid formats. Edge cases such as missing symbols and improper placement were handled effectively, improving data validation reliability.