

## 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:

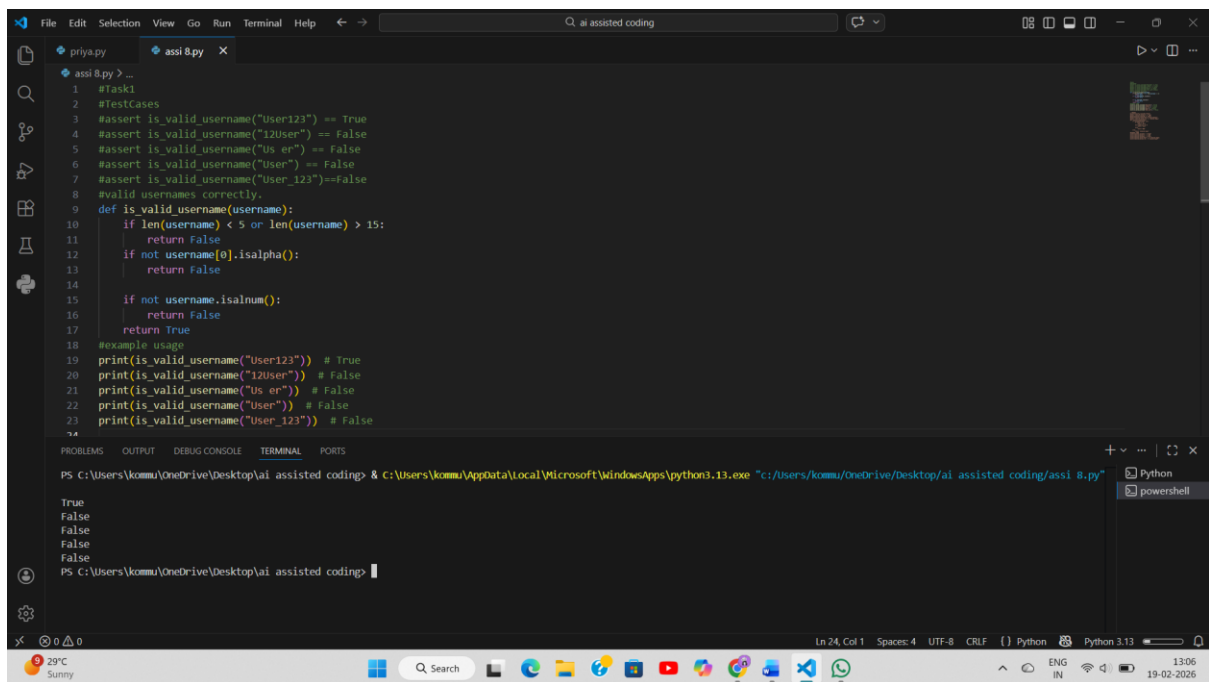
```
assert is_valid_username("User123") == True
```

```
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.



```
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
18 #example usage
19 print(is_valid_username("User123")) # True
20 print(is_valid_username("12User")) # False
21 print(is_valid_username("Us er")) # False
22 print(is_valid_username("User")) # False
23 print(is_valid_username("User_123")) # False
24
```

```
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"
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:

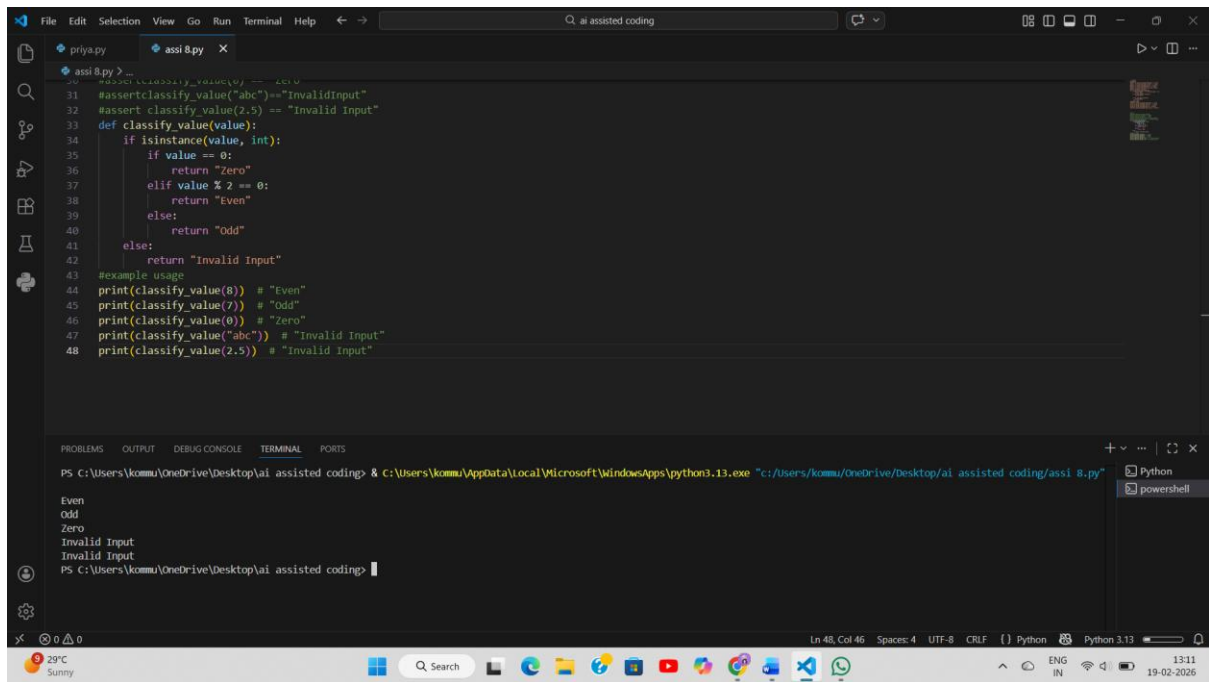
```
assert classify_value(8) == "Even"
```

```
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 a Visual Studio Code editor window with a file named 'assi 8.py' open. The code defines a function 'classify\_value' that takes a value and returns 'Zero', 'Even', 'Odd', or 'Invalid Input' based on conditional logic. The function is tested with various inputs: 8 (Even), 7 (Odd), 0 (Zero), 'abc' (Invalid Input), and 2.5 (Invalid Input). The terminal at the bottom shows the output of these tests: 'Even', 'Odd', 'Zero', 'Invalid Input', 'Invalid Input'. The status bar at the bottom indicates the file is at line 48, column 46, using UTF-8 encoding and CRLF line endings, with Python 3.13 selected as the interpreter.

```
30 #assert classify_value(0) == "Zero"
31 #assert classify_value("abc") == "Invalid Input"
32 #assert classify_value(2.5) == "Invalid Input"
33 def classify_value(value):
34     if isinstance(value, int):
35         if value == 0:
36             return "Zero"
37         elif value % 2 == 0:
38             return "Even"
39         else:
40             return "Odd"
41     else:
42         return "Invalid Input"
43 #example usage
44 print(classify_value(8)) # "Even"
45 print(classify_value(7)) # "Odd"
46 print(classify_value(0)) # "Zero"
47 print(classify_value("abc")) # "Invalid Input"
48 print(classify_value(2.5)) # "Invalid Input"
```

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"

Even  
Odd  
Zero  
Invalid Input  
Invalid Input  
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

Ln 48, Col 46 Spaces: 4 UTF-8 CRLF Python Python 3.13

**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:

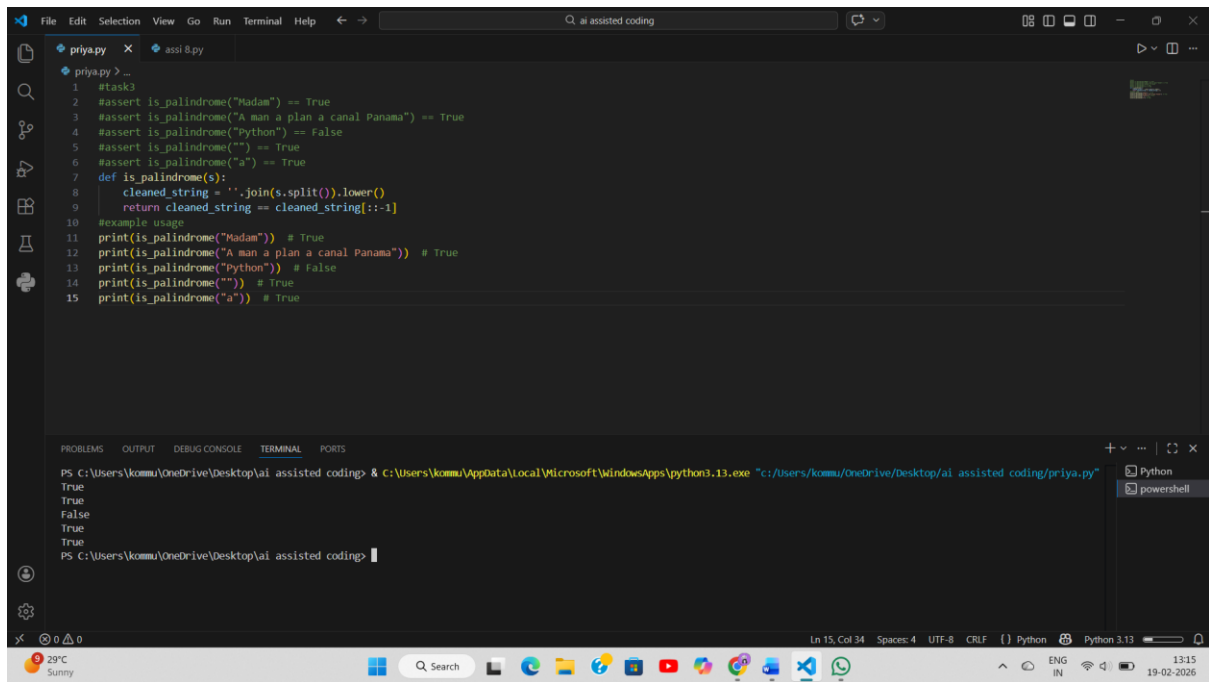
```
assert is_palindrome("Madam") == True
```

```
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.



```
File Edit Selection View Go Run Terminal Help
ai assisted coding

priya.py
1 #task3
2 #assert is_palindrome("Madam") == True
3 #assert is_palindrome("A man a plan a canal Panama") == True
4 #assert is_palindrome("python") == False
5 #assert is_palindrome("") == True
6 #assert is_palindrome("a") == True
7 def is_palindrome(s):
8     cleaned_string = ''.join(s.split()).lower()
9     return cleaned_string == cleaned_string[::-1]
10 #example usage
11 print(is_palindrome("Madam")) # True
12 print(is_palindrome("A man a plan a canal Panama")) # True
13 print(is_palindrome("python")) # False
14 print(is_palindrome("")) # True
15 print(is_palindrome("a")) # True

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"
True
True
False
True
True
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> |
```

**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)

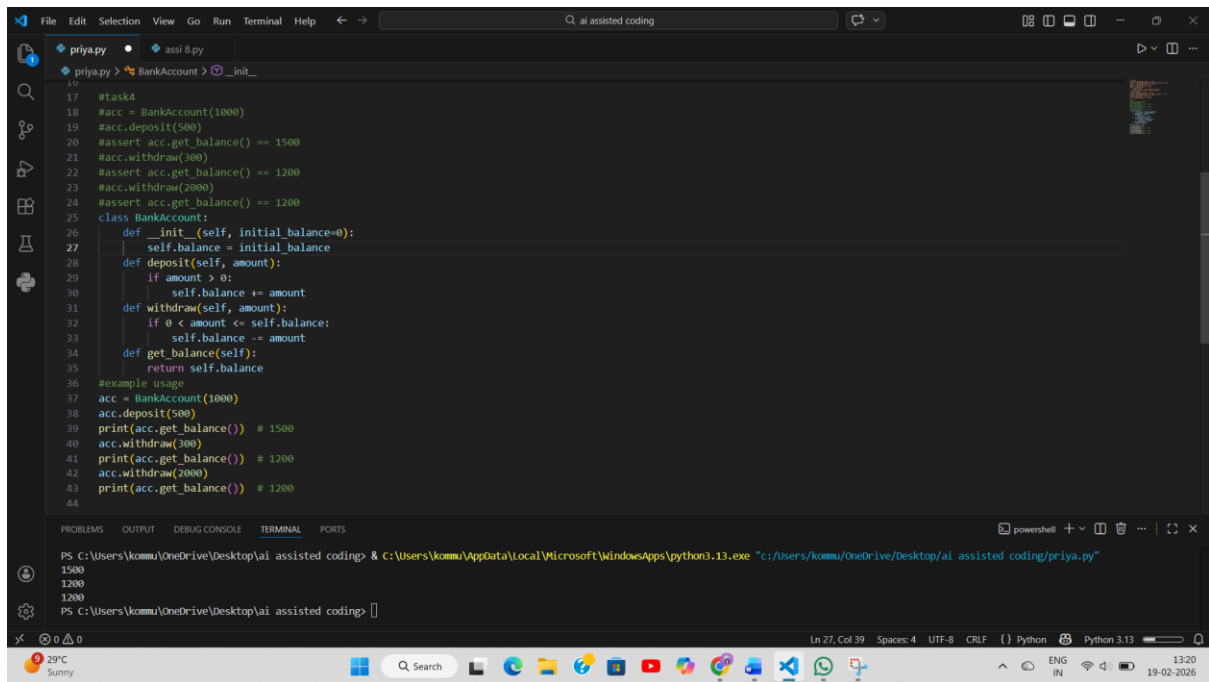
- **Task:** Ask AI to generate at least 3 assert-based test cases for a BankAccount class and then implement the class.
- **Methods:**
  - o deposit(amount)
  - o withdraw(amount)
  - o get\_balance()

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.



```
16 #task4
17 #acc = BankAccount(1000)
18 #acc.deposit(500)
19 #assert acc.get_balance() == 1500
20 #acc.withdraw(300)
21 #assert acc.get_balance() == 1200
22 #acc.withdraw(2000)
23 #assert acc.get_balance() == 1200
24 class BankAccount:
25     def __init__(self, initial_balance=0):
26         self.balance = initial_balance
27     def deposit(self, amount):
28         if amount > 0:
29             self.balance += amount
30     def withdraw(self, amount):
31         if 0 < amount <= self.balance:
32             self.balance -= amount
33     def get_balance(self):
34         return self.balance
35 #example usage
36 acc = BankAccount(1000)
37 acc.deposit(500)
38 print(acc.get_balance()) # 1500
39 acc.withdraw(300)
40 print(acc.get_balance()) # 1200
41 acc.withdraw(2000)
42 print(acc.get_balance()) # 1200
43
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>

**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:**

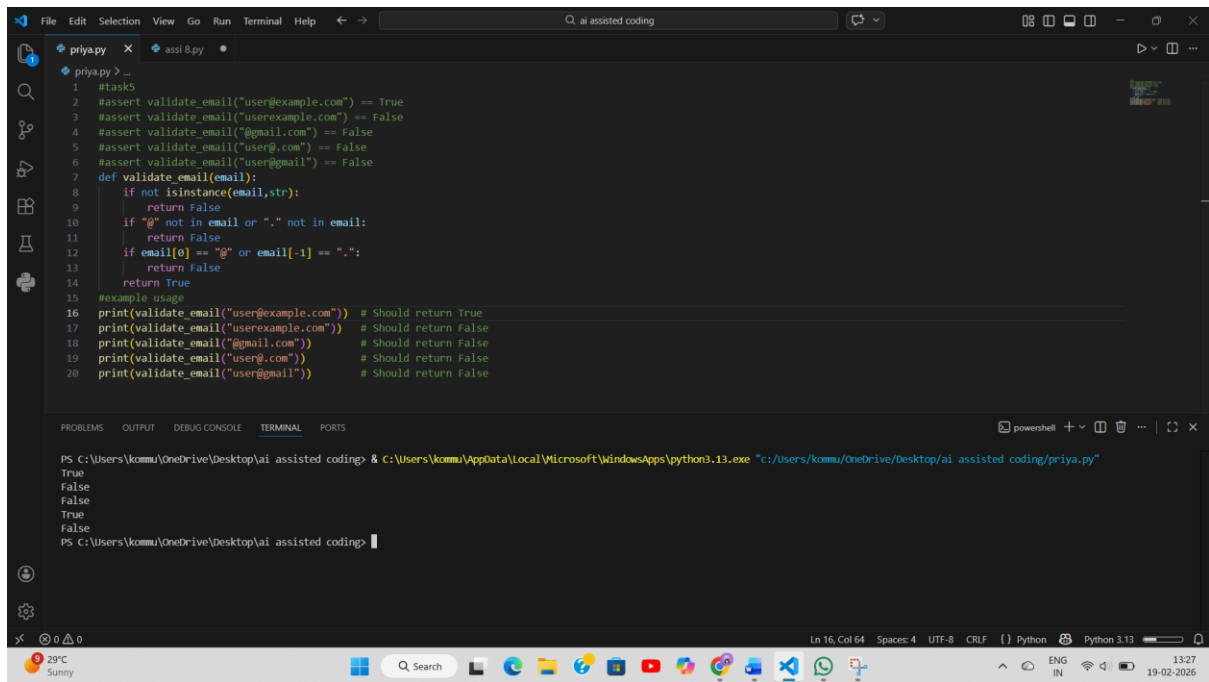
```
assert validate_email("user@example.com") == True
```

```
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 image shows a Visual Studio Code editor window with a Python file named `priya.py`. The script defines a `validate_email` function and includes several test cases using `assert` and `print` statements. The terminal at the bottom shows the command to run the script using `python3.13.exe`, and the output displays the results of the validation tests.

```
1 #task5
2 #assert validate_email("user@example.com") == True
3 #assert validate_email("userexample.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("userexample.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
```

Terminal Output:

```
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>
```

**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.