

ASSIGNMENT-8.5

NAME:U.VIGNESH

H.NO-2303A51964

BATCH-24

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.

```

303A51964-ASSINGMENT-8.5.py > ...
def is_valid_username(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not username.isalnum():
        return False
    if username[0].isdigit():
        return False

    return True
# Test the function
assert is_valid_username("user123") == True
assert is_valid_username("123user") == False
assert is_valid_username("us er") == False

```

OUTPUT:

```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh> & C:/Users/VIGNESH/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/ASSESTENT CODING/vignesh/303A51964-ASSINGMENT-8.5.py"
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh>

```

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:

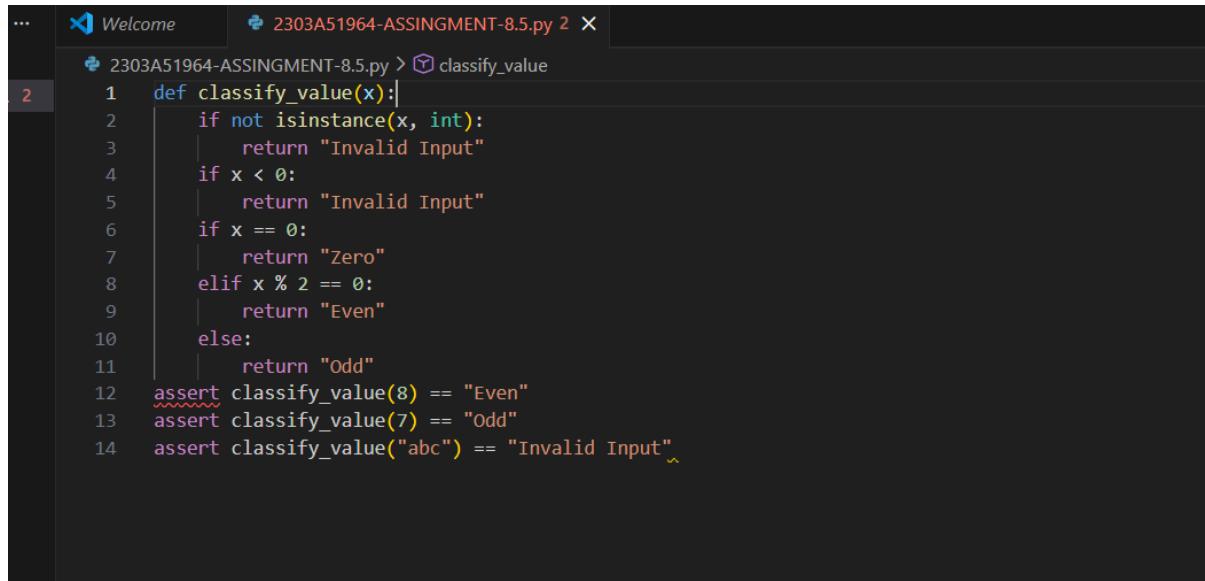
```
assert classify_value(8) == "Even"
```

```
assert classify_value(7) == "Odd"  
assert classify_value("abc") == "Invalid Input"
```

Expected Output #2:

- Function correctly classifying values and passing all test cases.

CODE:



The screenshot shows a code editor window with a dark theme. The title bar says "Welcome" and "2303A51964-ASSINGMENT-8.5.py 2". The code in the editor is as follows:

```
... Welcome 2303A51964-ASSINGMENT-8.5.py 2  
2303A51964-ASSINGMENT-8.5.py > classify_value  
1 def classify_value(x):  
2     if not isinstance(x, int):  
3         return "Invalid Input"  
4     if x < 0:  
5         return "Invalid Input"  
6     if x == 0:  
7         return "Zero"  
8     elif x % 2 == 0:  
9         return "Even"  
10    else:  
11        return "Odd"  
12 assert classify_value(8) == "Even"  
13 assert classify_value(7) == "Odd"  
14 assert classify_value("abc") == "Invalid Input"
```

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.

The screenshot shows a code editor window with a dark theme. The file is named '2303A51964-ASSINGMENT-8.5.py'. The code defines a function 'is_palindrome' that takes a string 'text' and returns True if it is a palindrome. It normalizes the text by removing non-alphanumeric characters and converting them to lowercase. The function then checks if the normalized string is equal to its reverse. Below the function definition, there is a section titled '# Test cases' containing four assert statements. The first three assertions pass, while the fourth assertion for 'Python' fails, as indicated by the red squiggle under 'False'.

```
def is_palindrome(text):
    normalized = ""
    for char in text:
        if char.isalnum():
            normalized += char.lower()
    return normalized == normalized[::-1]

# Test cases
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("No lemon, no melon!") == True
assert is_palindrome("Python") == False
```

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)

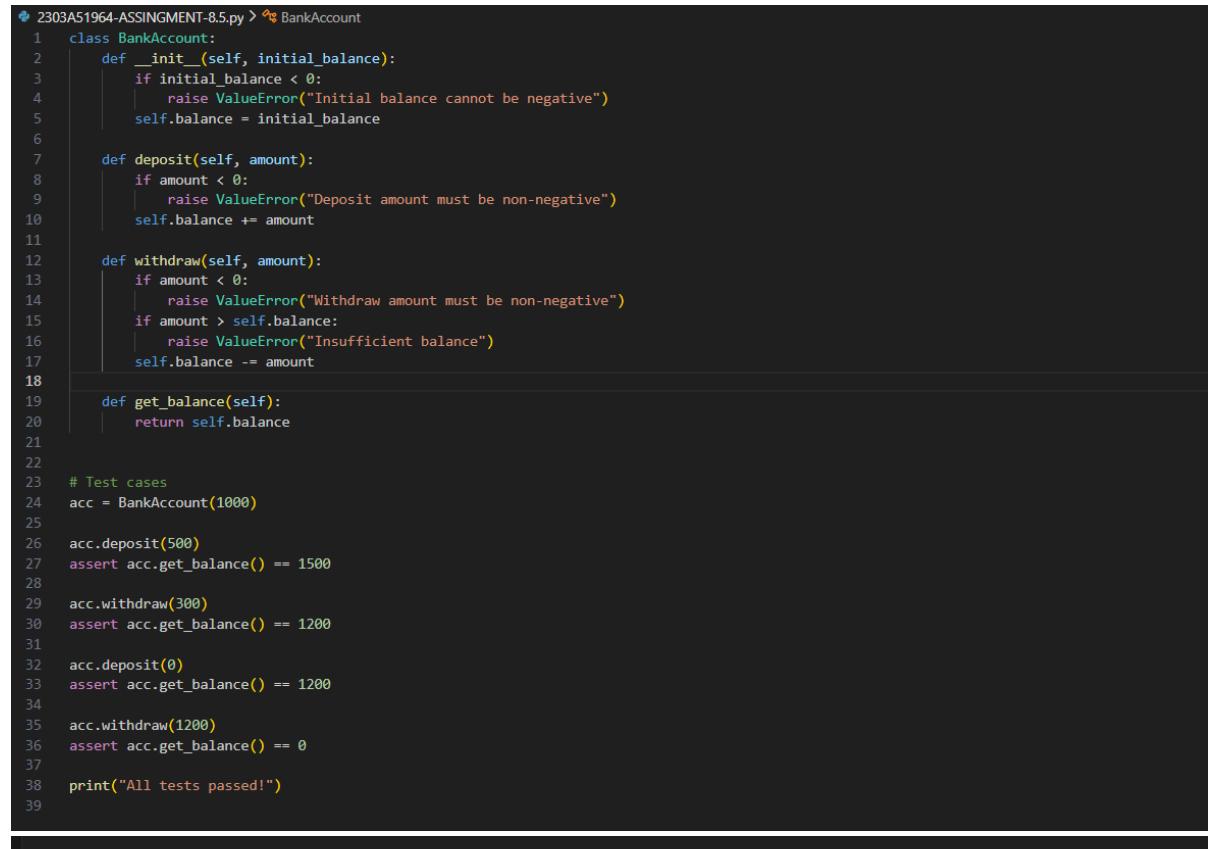
assert acc.get_balance() == 1200

```

Expected Output #4:

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

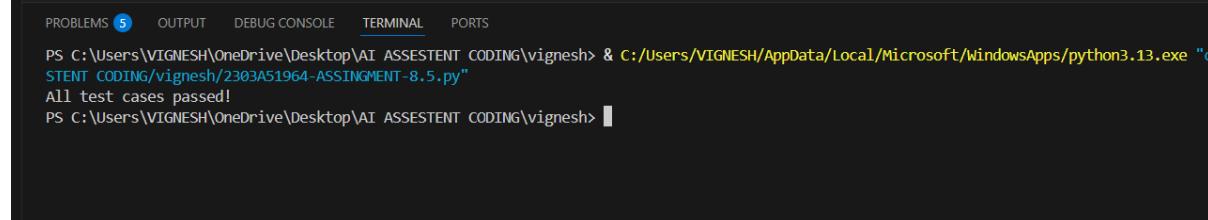
CODE:



```

1  2303A51964-ASSINGMENT-8.5.py > 📈 BankAccount
2
3  class BankAccount:
4      def __init__(self, initial_balance):
5          if initial_balance < 0:
6              raise ValueError("Initial balance cannot be negative")
7          self.balance = initial_balance
8
9      def deposit(self, amount):
10         if amount < 0:
11             raise ValueError("Deposit amount must be non-negative")
12         self.balance += amount
13
14     def withdraw(self, amount):
15         if amount < 0:
16             raise ValueError("Withdraw amount must be non-negative")
17         if amount > self.balance:
18             raise ValueError("Insufficient balance")
19         self.balance -= amount
20
21
22
23     # Test cases
24     acc = BankAccount(1000)
25
26     acc.deposit(500)
27     assert acc.get_balance() == 1500
28
29     acc.withdraw(300)
30     assert acc.get_balance() == 1200
31
32     acc.deposit(0)
33     assert acc.get_balance() == 1200
34
35     acc.withdraw(1200)
36     assert acc.get_balance() == 0
37
38     print("All tests passed!")
39

```



```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh> & C:/Users/VIGNESH/AppData/Local/Microsoft/WindowsApps/python3.13.exe "C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh\2303A51964-ASSINGMENT-8.5.py"
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh> All test cases passed!
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh>

```

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 window with a dark theme. The file being edited is named "2303A51964-ASSINGMENT-8.5.py". The code defines a function `validate_email` that checks if an email address is valid based on specific rules. It includes three test cases using the `assert` statement to verify the function's behavior with different email addresses.

```
def validate_email(email):
    if '@' not in email or '.' not in email:
        return False
    at_index = email.index('@')
    dot_index = email.rindex('.')
    if at_index < 1 or dot_index < at_index + 2 or dot_index >= len(email) - 1:
        return False
    return True

# Test cases for the validate_email function
assert validate_email("user@example.com") == True, "Test case 1 failed"
assert validate_email("userexample.com") == False, "Test case 2 failed"
assert validate_email("user@.com") == False, "Test case 3 failed"
print("All test cases passed!")
```

The screenshot shows a terminal window with a dark theme. The command `python3.13.exe` is used to run the script "2303A51964-ASSINGMENT-8.5.py". The output indicates that all test cases passed successfully.

```
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh> & C:/Users/VIGNESH/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh\2303A51964-ASSINGMENT-8.5.py"
All tests passed!
PS C:\Users\VIGNESH\OneDrive\Desktop\AI ASSESTENT CODING\vignesh>
```