Assignment – 8.5

Name : Yashaswini Nagireddi

Batch : 21

Ht.No : 2303A51418

1. Task Description (Username Validator – Apply AI in Authentication Context)

Promt:

```
# 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("User") == False
assert is_valid_username("User_123") == False
```

code:

13-02-2026.py > is_valid_username

```python
# 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("User") == False
# assert is_valid_username("User_123") == False
def is_valid_username(username):
    # Check if the username is between 5 and 20 characters
    if len(username) < 5 or len(username) > 20:
        return False
    # Check if the username starts with a letter
    if not username[0].isalpha():
        return False
    # Check if the username contains only letters and digits
    for char in username:
        if not char.isalnum():
            return False
    return True
# Example usage
print(is_valid_username("User123"))  # True
print(is_valid_username("12User"))   # False
print(is_valid_username("Us er"))    # False
print(is_valid_username("User"))     # False
print(is_valid_username("User_123")) # False

```

output:

```
bugpy\launcher' '57356' '--' 'c:\Users\dell\OneDrive\Desktop\AI\1
3-02-2026.py'
True
False
False
False
False
PS C:\Users\dell\OneDrive\Desktop\AI>
```

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.

2. Task: Even–Odd & Type Classification
promt:
assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value(0) == "Zero"
assert classify_value("abc") == "Invalid Input"
assert classify_value(2.5) == "Invalid Input"
Input:

13-02-2026.py > ...

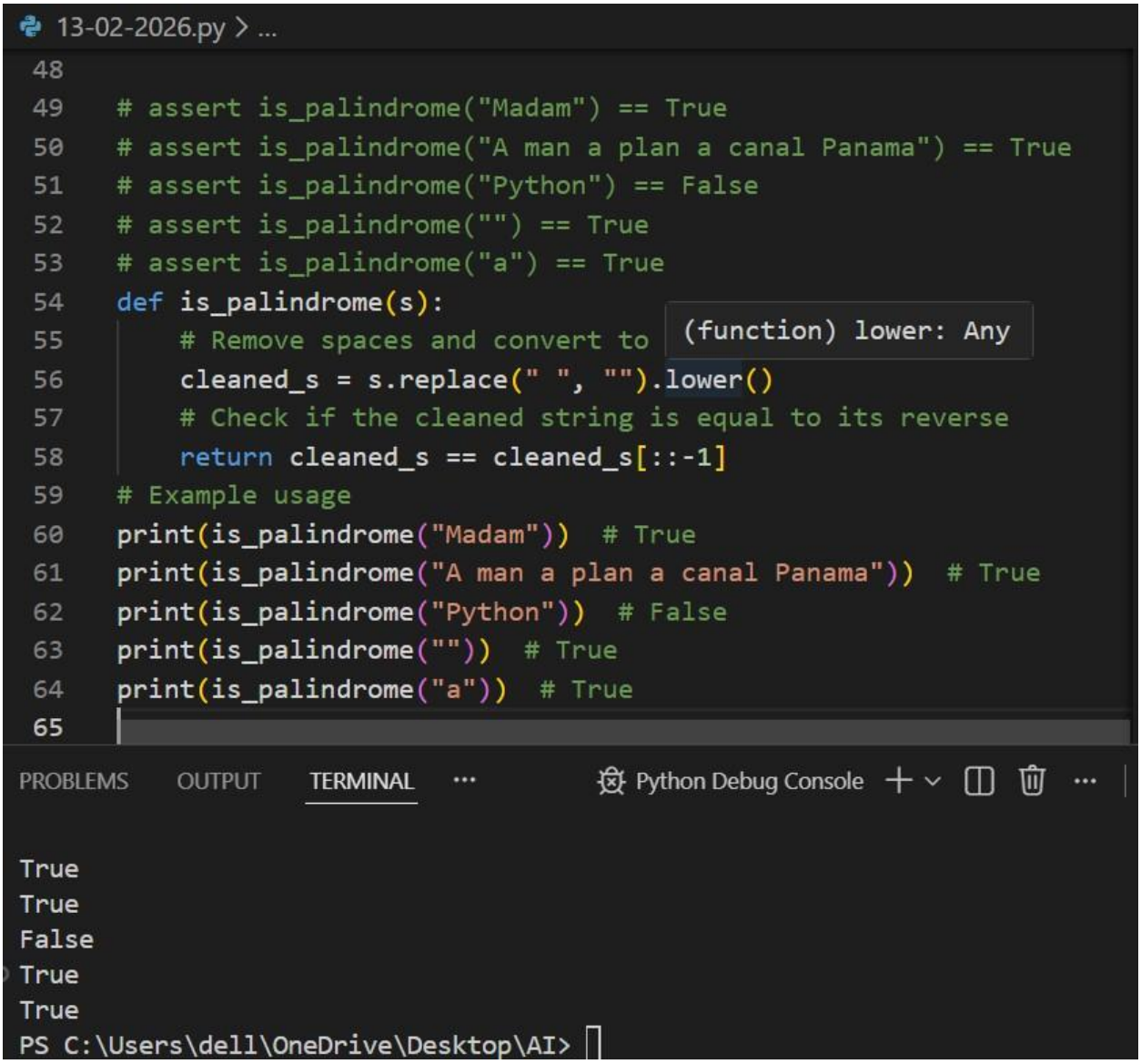
```python

# assert classify_value(8) == "Even"
#assert classify_value(7) == "Odd"
#assert classify_value(0) == "Zero"
#assert classify_value("abc") == "Invalid Input"
#assert classify_value(2.5) == "Invalid Input"
def classify_value(value):
    if isinstance(value, int):
        if value == 0:
            return "Zero"
        elif value % 2 == 0:
            return "Even"
        else:
            return "Odd"
    else:
        return "Invalid Input"
# Example usage
print(classify_value(8))      # "Even"
print(classify_value(7))      # "Odd"
print(classify_value(0))      # "Zero"
print(classify_value("abc"))  # "Invalid Input"
print(classify_value(2.5))    # "Invalid Input"


```

Output:

```
25.18.0-win32-x64\bundled\libs\debugpy\launcher' '59098' '--' 'c:\Users\dell\One
rive\Desktop\AI\13-02-2026.py'
True
False
False
False
False
Even
Odd
Zero
Invalid Input
Invalid Input
```

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 3: Palindrome Checker
Promt:
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
assert is_palindrome("a") == True
input:
output:



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.

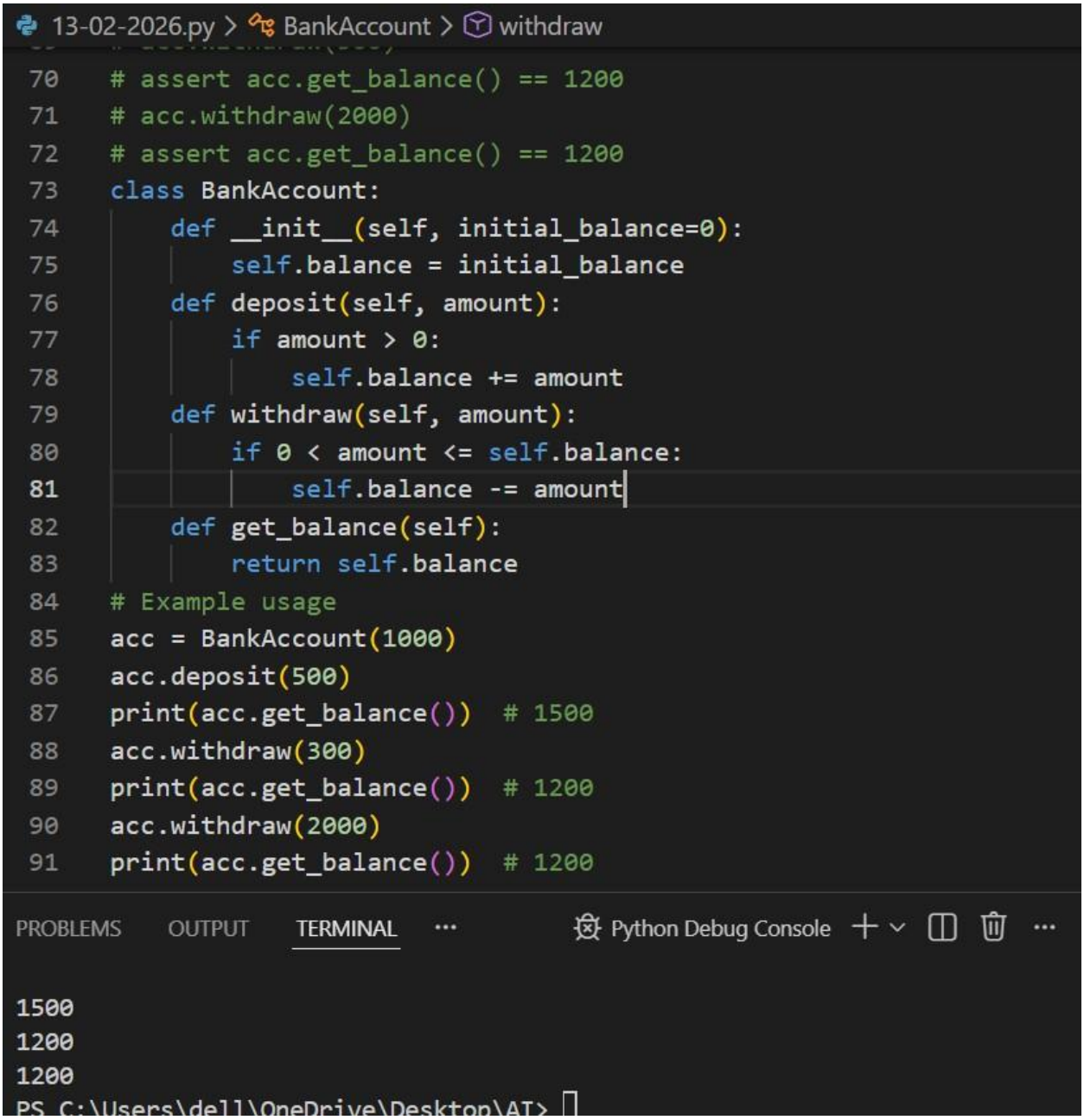
Task 4 Observation: BankAccount Class

Promt:

```
acc = BankAccount(1000)
acc.deposit(500)
assert acc.get_balance() == 1500

acc.withdraw(300)
assert acc.get_balance() == 1200

acc.withdraw(2000)
assert acc.get_balance() == 1200
```

input:

output:



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

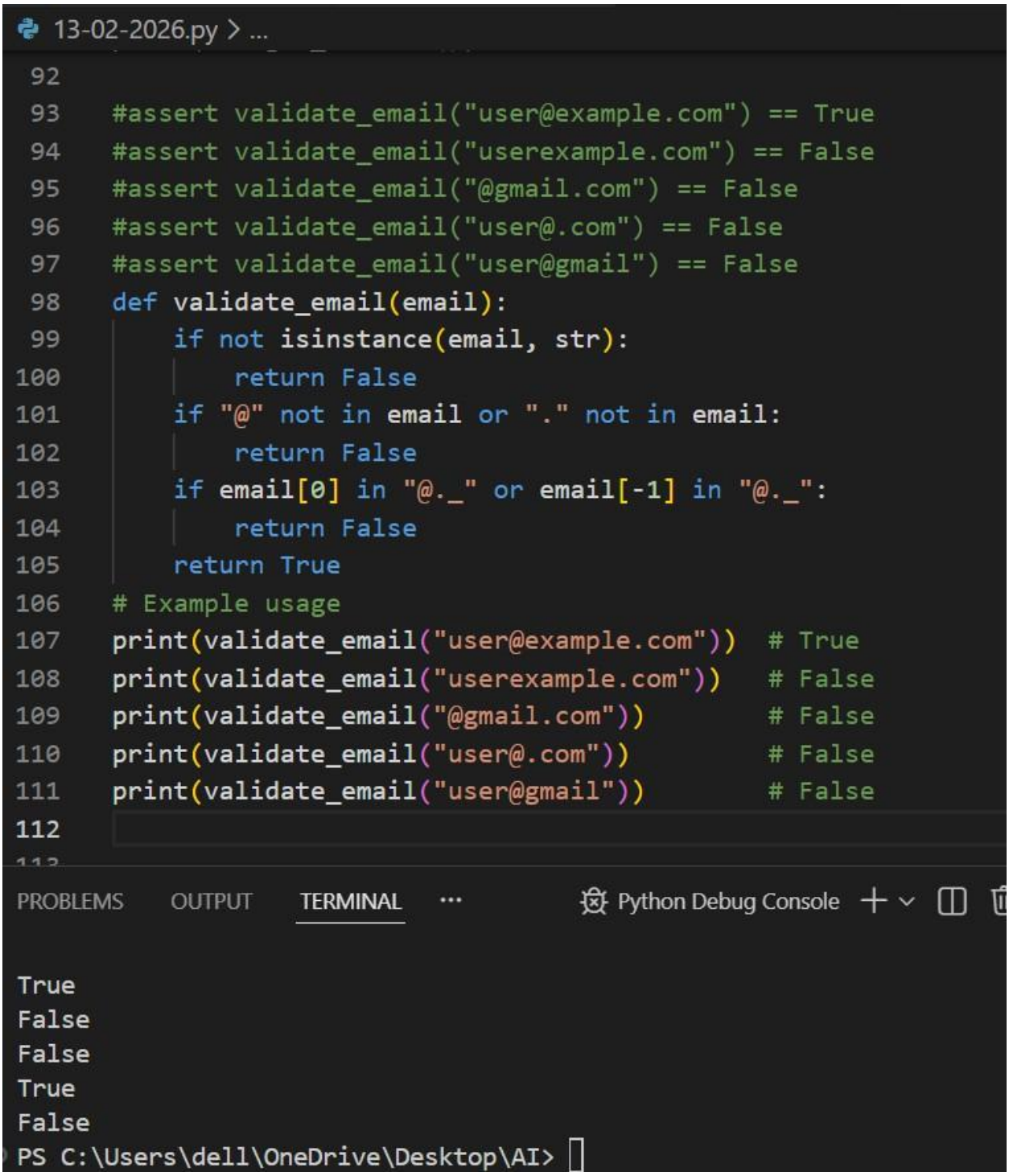
Task 5 : Email ID Validation

Prmot:

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

Input:

Output:



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.