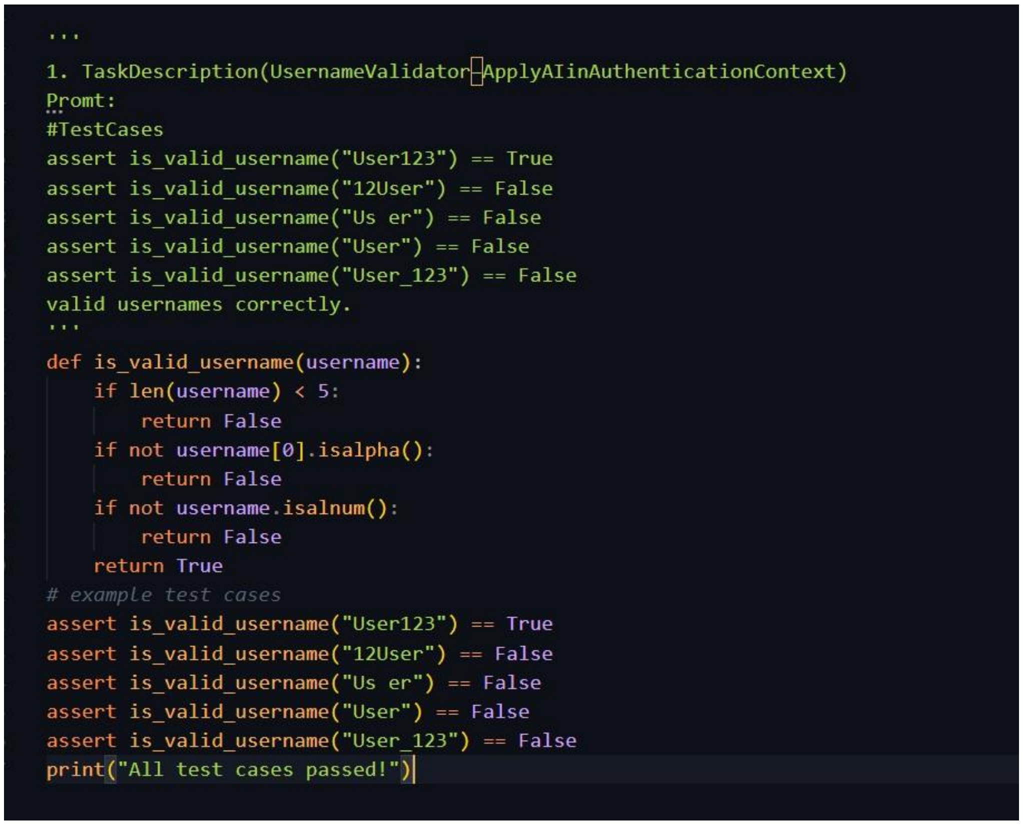
AI Assisted Coding
Assignment – 8.5

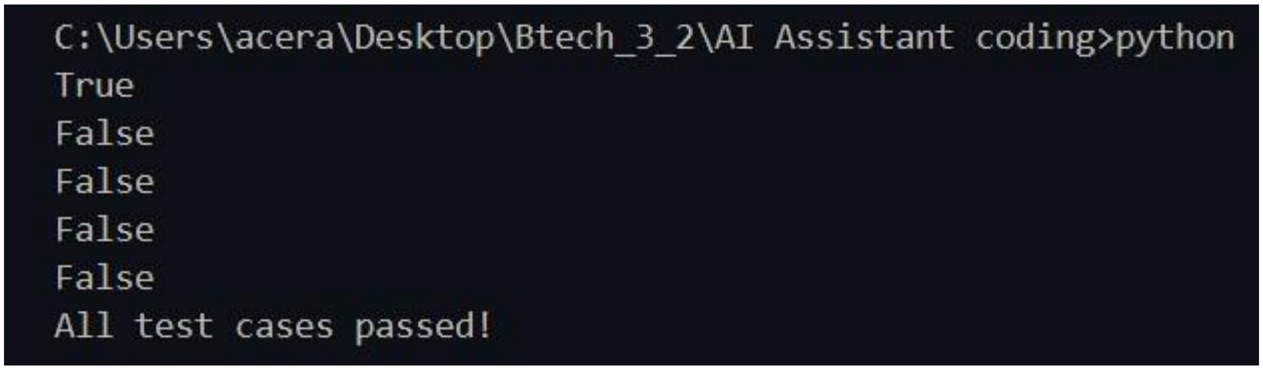
Name : N.Varshith
Batch : 21
Ht.No : 2303A51441
Task 1 :
TaskDescription(UsernameValidator–ApplyAIinAuthenticationContext)
Promt: #TestCases assert is_valid_username("User123") == True assert is_valid_username("12User") == False assert is_valid_username("Us er") == False assert is_valid_username("User") == False assertis_valid_username("User_123")==False valid usernames correctly.

Code:



```
'''
1. TaskDescription(UsernameValidator–ApplyAIinAuthenticationContext)
Promt:
#TestCases
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
valid usernames correctly.
'''
def is_valid_username(username):
    if len(username) < 5:
        return False
    if not username[0].isalpha():
        return False
    if not username.isalnum():
        return False
    return True
# example 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
print("All test cases passed!")
```

Output:

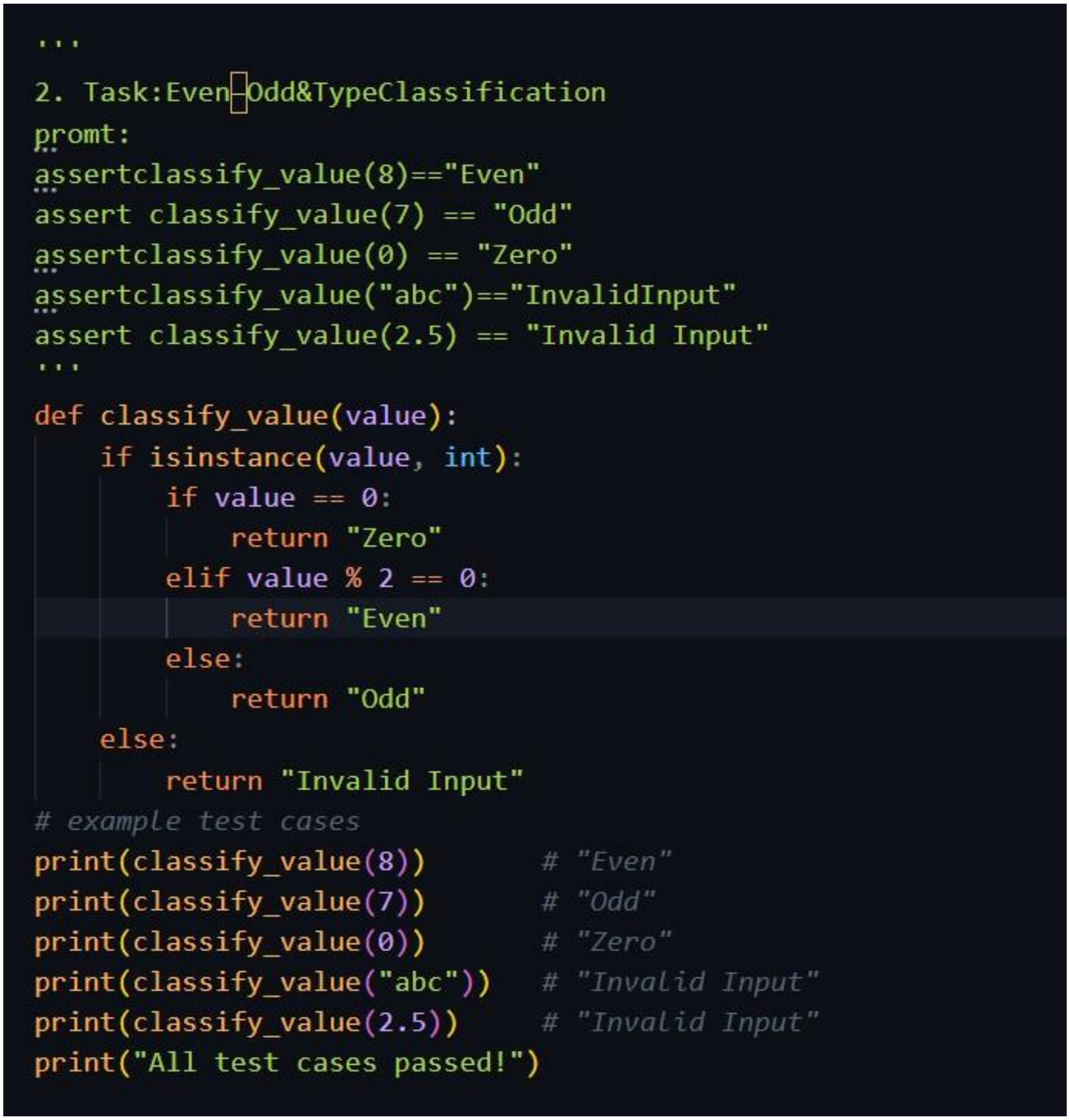


```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python
True
False
False
False
False
All test cases passed!
```

Observation:
AI-generated assert test cases helped define the user’s name 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.

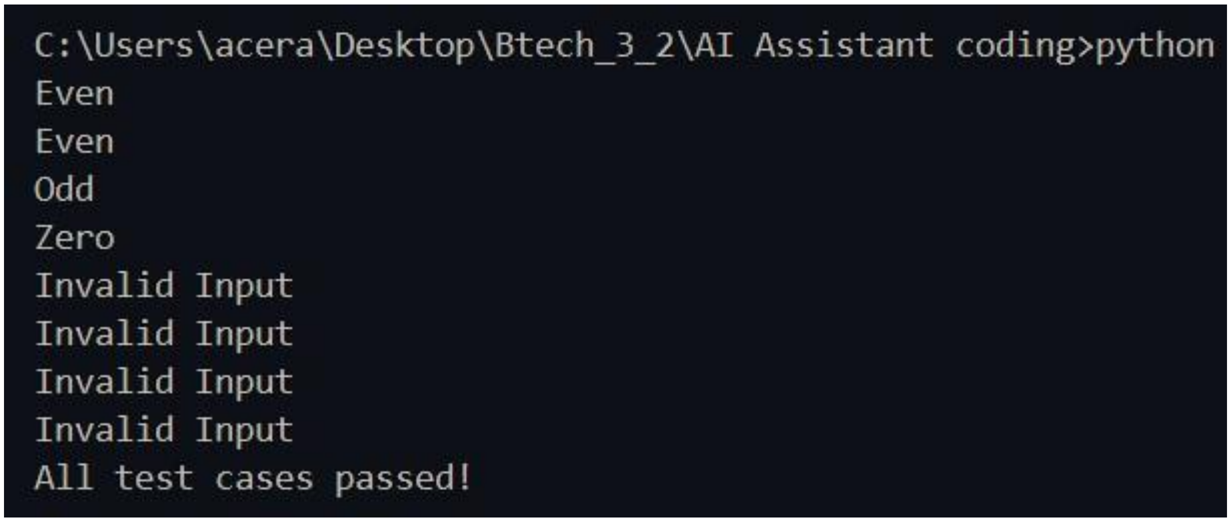
2.Task: Even–Odd&TypeClassification
promt: assertclassify_value(8)=="Even"
assert classify_value(7) == "Odd"
assertclassify_value(0) == "Zero"
assertclassify_value("abc")=="InvalidInput"
assert classify_value(2.5) == "Invalid Input"

Code:



```
'''
2. Task:Even–Odd&TypeClassification
promt:
assertclassify_value(8)=="Even"
assert classify_value(7) == "Odd"
assertclassify_value(0) == "Zero"
assertclassify_value("abc")=="InvalidInput"
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 test cases
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"
print("All test cases passed!")
```

Output:



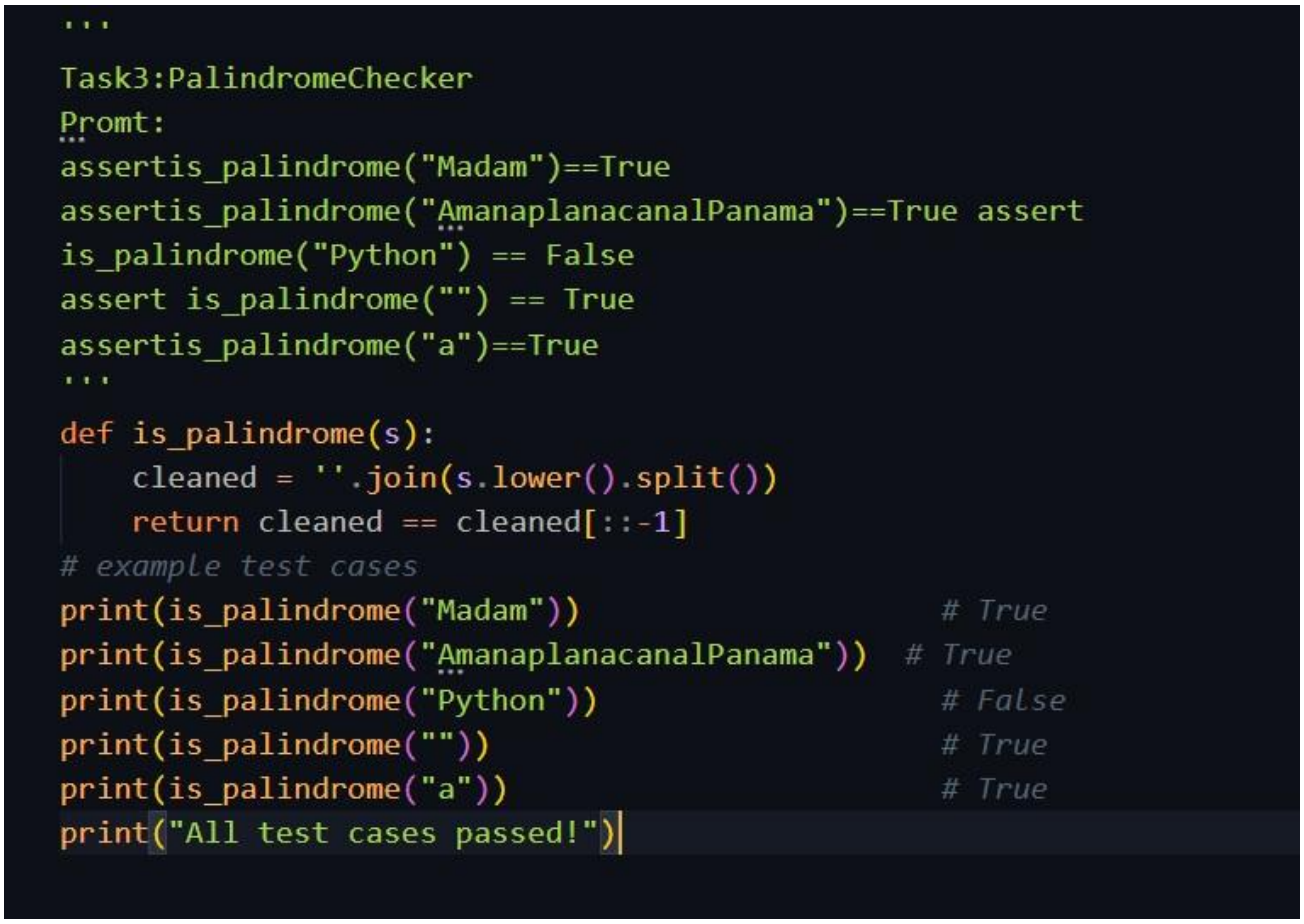
```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python
Even
Even
Odd
Zero
Invalid Input
Invalid Input
Invalid Input
Invalid Input
All test cases passed!
```

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.

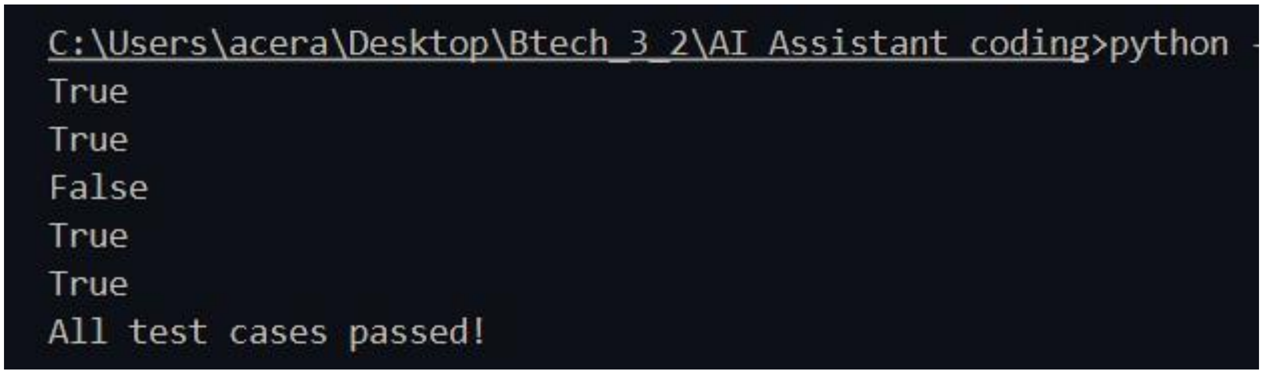
Task3:PalindromeChecker Promt:
assertis_palindrome("Madam")==Tru
e
assertis_palindrome("AmanaplanacanalPanama")==True assert
is_palindrome("Python") == False assert
is_palindrome("") == True
assertis_palindrome("a")==True

Code:



```
'''
Task3:PalindromeChecker
Promt:
assertis_palindrome("Madam")==True
assertis_palindrome("AmanaplanacanalPanama")==True assert
is_palindrome("Python") == False
assert is_palindrome("") == True
assertis_palindrome("a")==True
'''
def is_palindrome(s):
    cleaned = ''.join(s.lower().split())
    return cleaned == cleaned[::-1]
# example test cases
print(is_palindrome("Madam"))                    # True
print(is_palindrome("AmanaplanacanalPanama"))  # True
print(is_palindrome("Python"))                   # False
print(is_palindrome(""))                         # True
print(is_palindrome("a"))                        # True
print("All test cases passed!")
```

Output:



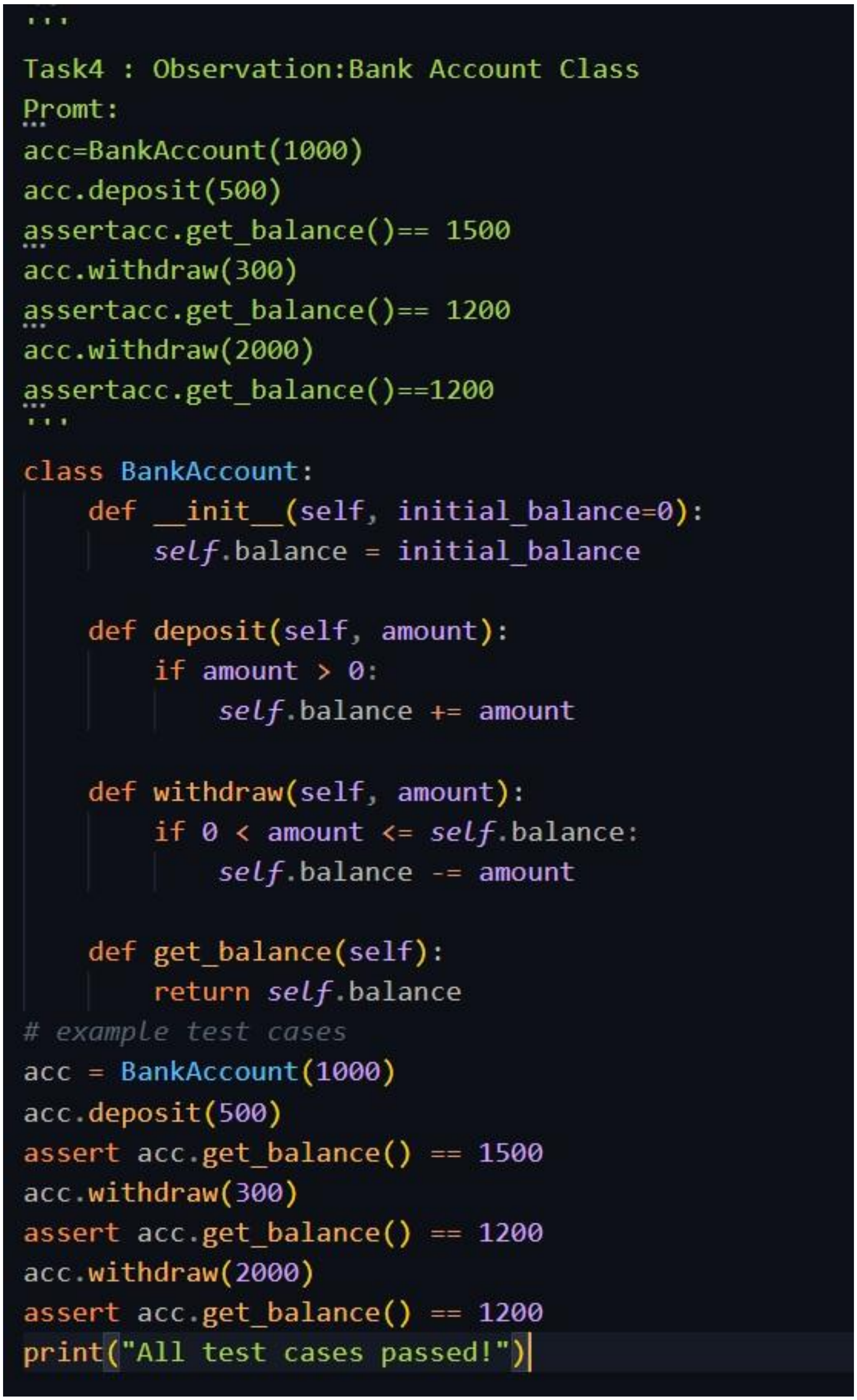
```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python
True
True
False
True
True
All test cases passed!
```

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

Task4 : Observation:Bank Account Class
Promt: acc=BankAccount(1000)
acc.deposit(500)
assertacc.get_balance()== 1500

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

acc.withdraw(2000)
assertacc.get_balance()==1200 Code:



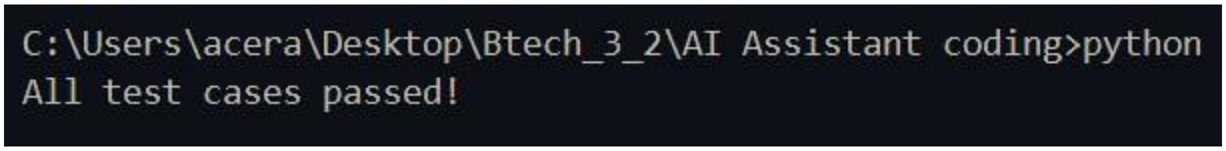
```
'''
Task4 : Observation:Bank Account Class
Promt:
acc=BankAccount(1000)
acc.deposit(500)
assertacc.get_balance()== 1500
acc.withdraw(300)
assertacc.get_balance()== 1200
acc.withdraw(2000)
assertacc.get_balance()==1200
'''
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount

    def get_balance(self):
        return self.balance
# example test cases
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
print("All test cases passed!")
```

Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python
All test cases passed!
```

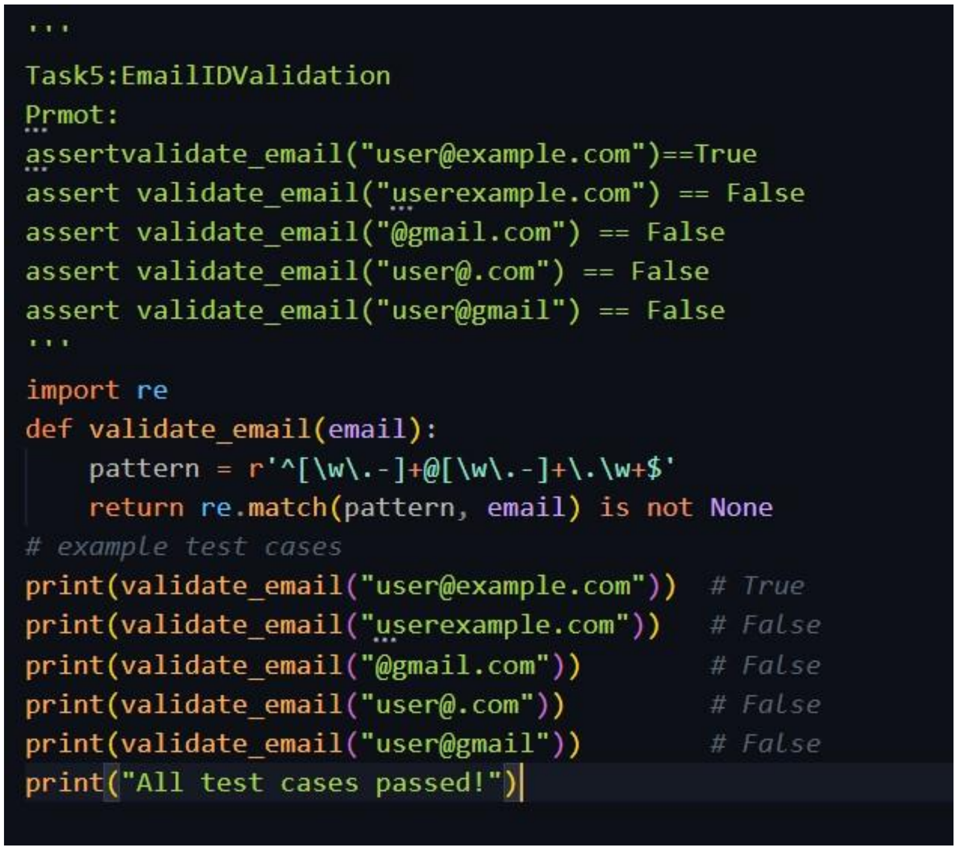
Observation:

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.

Task5:EmailIDValidation Prmot:

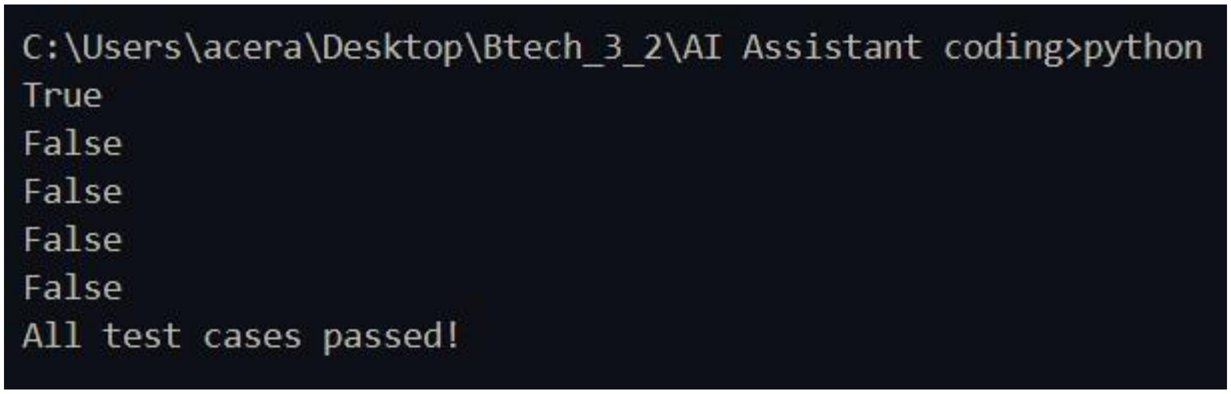
assertvalidate_email("user@example.com")==True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False assert
validate_email("user@.com") == False
assertvalidate_email("user@gmail")==False

Code:



```
'''
Task5:EmailIDValidation
Prmot:
assertvalidate_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
'''
import re
def validate_email(email):
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return re.match(pattern, email) is not None
# example test cases
print(validate_email("user@example.com"))  # True
print(validate_email("userexample.com"))   # False
print(validate_email("@gmail.com"))        # False
print(validate_email("user@.com"))         # False
print(validate_email("user@gmail"))        # False
print("All test cases passed!")
```

Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python
True
False
False
False
False
All test cases passed!
```

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 formats were handled effectively, improving data validation reliability