SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 8

Name of Student : JEEVAN SAI
Enrollment No. : 2303A51420
Batch No. 21

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 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

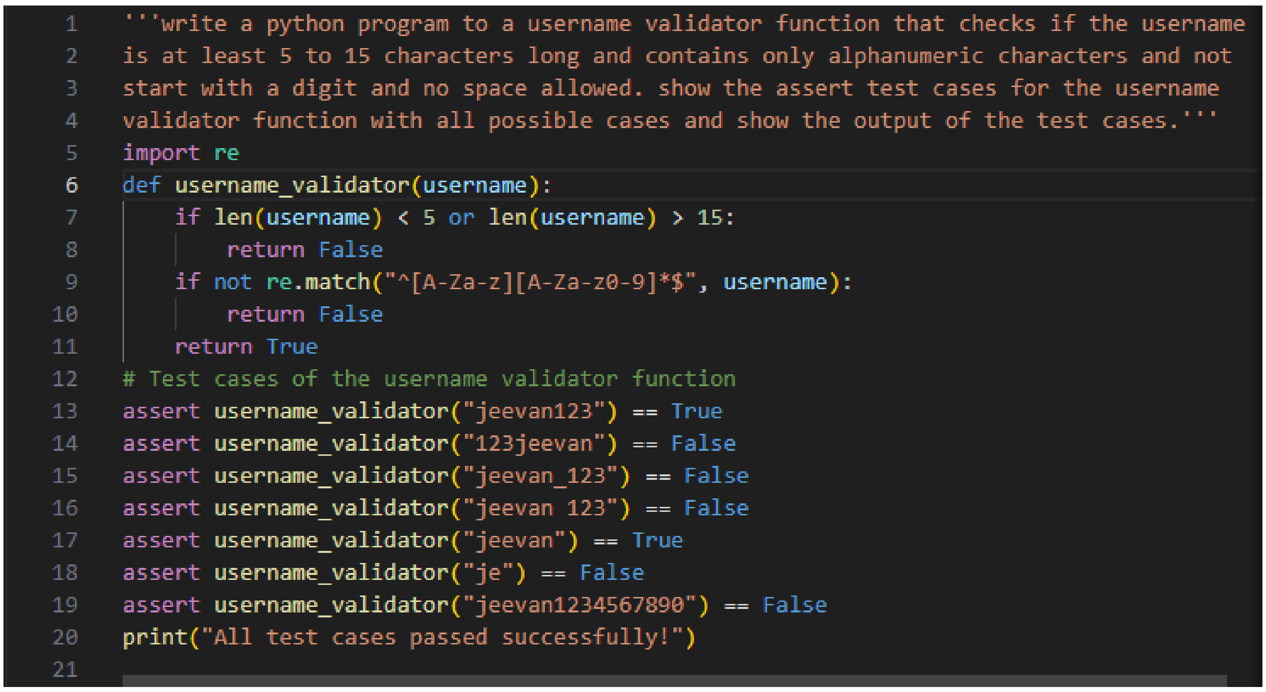
Expected Output #1:

• Username validation logic successfully passing all AI- generated test cases.

Prompt:-

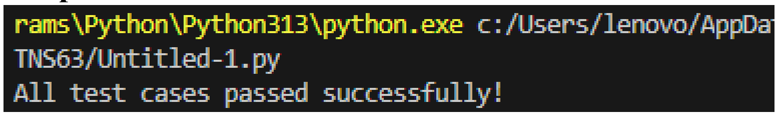
write a python program to a username validator function that checks if the username is at least 5 to 15 characters long and contains only alphanumeric characters and not start with a digit and no space allowed. show the assert test cases for the username validator function with all possible cases and show the output of the test cases.

Code:-



```
'''write a python program to a username validator function that checks if the username
is at least 5 to 15 characters long and contains only alphanumeric characters and not
start with a digit and no space allowed. show the assert test cases for the username
validator function with all possible cases and show the output of the test cases.'''
import re
def username_validator(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not re.match("^[A-Za-z][A-Za-z0-9]*$", username):
        return False
    return True
# Test cases of the username validator function
assert username_validator("jeevan123") == True
assert username_validator("123jeevan") == False
assert username_validator("jeevan_123") == False
assert username_validator("jeevan 123") == False
assert username_validator("jeevan") == True
assert username_validator("je") == False
assert username_validator("jeevan1234567890") == False
print("All test cases passed successfully!")
```

Output:-



```
rams\Python\Python313\python.exe c:/Users/lenovo/AppDa
TNS63/Untitled-1.py
All test cases passed successfully!
```

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"

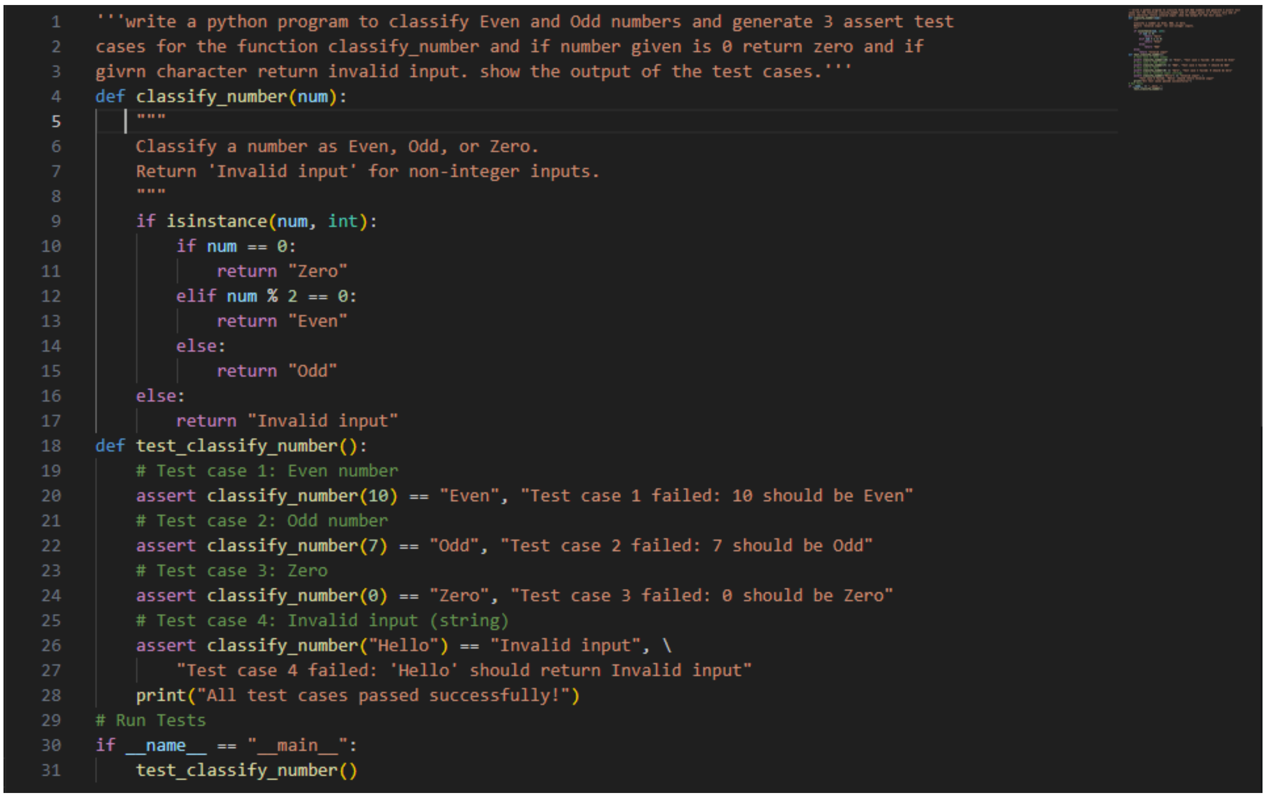
Expected Output #2:

- **Function correctly classifying values and passing all test cases.**

Prompt:-

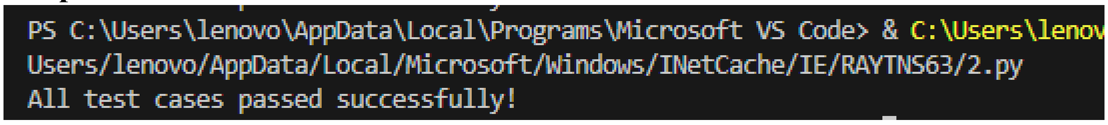
write a python program to classify Even and Odd numbers and generate 3 assert test cases for the function classify_number and if number given is 0 return zero and if givrn character return invalid input. show the output of the test cases.

Code:-



```python
'''write a python program to classify Even and Odd numbers and generate 3 assert test
cases for the function classify_number and if number given is 0 return zero and if
givrn character return invalid input. show the output of the test cases.'''
def classify_number(num):
    """
    Classify a number as Even, Odd, or Zero.
    Return 'Invalid input' for non-integer inputs.
    """
    if isinstance(num, int):
        if num == 0:
            return "Zero"
        elif num % 2 == 0:
            return "Even"
        else:
            return "Odd"
    else:
        return "Invalid input"
def test_classify_number():
    # Test case 1: Even number
    assert classify_number(10) == "Even", "Test case 1 failed: 10 should be Even"
    # Test case 2: Odd number
    assert classify_number(7) == "Odd", "Test case 2 failed: 7 should be Odd"
    # Test case 3: Zero
    assert classify_number(0) == "Zero", "Test case 3 failed: 0 should be Zero"
    # Test case 4: Invalid input (string)
    assert classify_number("Hello") == "Invalid input", \
        "Test case 4 failed: 'Hello' should return Invalid input"
    print("All test cases passed successfully!")
# Run Tests
if __name__ == "__main__":
    test_classify_number()
```

Output:-



```
PS C:\Users\lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Users\lenov
Users/lenovo/AppData/Local/Microsoft/Windows/INetCache/IE/RAYTNS63/2.py
All test cases passed successfully!
```

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:**

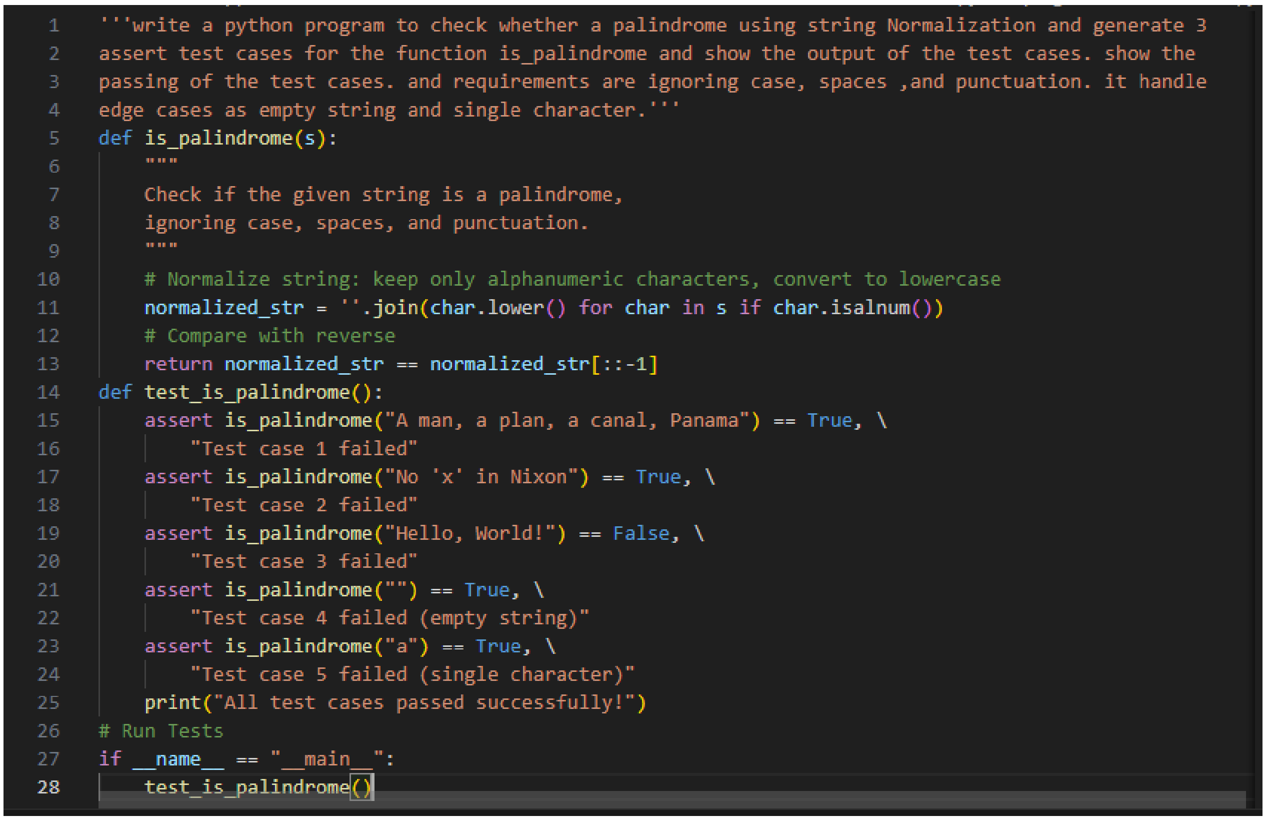
o Ignore case, spaces, and punctuation. o Handle edge cases such as empty strings and single characters.

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.**

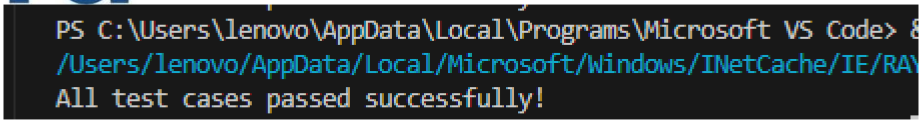
Prompt:- write a python program to check whether a palindrome using string Normalization and generate 3 assert test cases for the function is_palindrome and show the output of the test cases. show the passing of the test cases. and requirements are ignoring case, spaces ,and punctuation. it handle edge cases as empty string and single character.

Code:-



```python
'''write a python program to check whether a palindrome using string Normalization and generate 3
assert test cases for the function is_palindrome and show the output of the test cases. show the
passing of the test cases. and requirements are ignoring case, spaces ,and punctuation. it handle
edge cases as empty string and single character.'''
def is_palindrome(s):
    """
    Check if the given string is a palindrome,
    ignoring case, spaces, and punctuation.
    """
    # Normalize string: keep only alphanumeric characters, convert to lowercase
    normalized_str = ''.join(char.lower() for char in s if char.isalnum())
    # Compare with reverse
    return normalized_str == normalized_str[::-1]
def test_is_palindrome():
    assert is_palindrome("A man, a plan, a canal, Panama") == True, \
        "Test case 1 failed"
    assert is_palindrome("No 'x' in Nixon") == True, \
        "Test case 2 failed"
    assert is_palindrome("Hello, World!") == False, \
        "Test case 3 failed"
    assert is_palindrome("") == True, \
        "Test case 4 failed (empty string)"
    assert is_palindrome("a") == True, \
        "Test case 5 failed (single character)"
    print("All test cases passed successfully!")
# Run Tests
if __name__ == "__main__":
    test_is_palindrome()
```

Output:-



```
PS C:\Users\lenovo\AppData\Local\Programs\Microsoft VS Code> &
/Users/lenovo/AppData/Local/Microsoft/Windows/INetCache/IE/RAY
All test cases passed successfully!
```

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:**

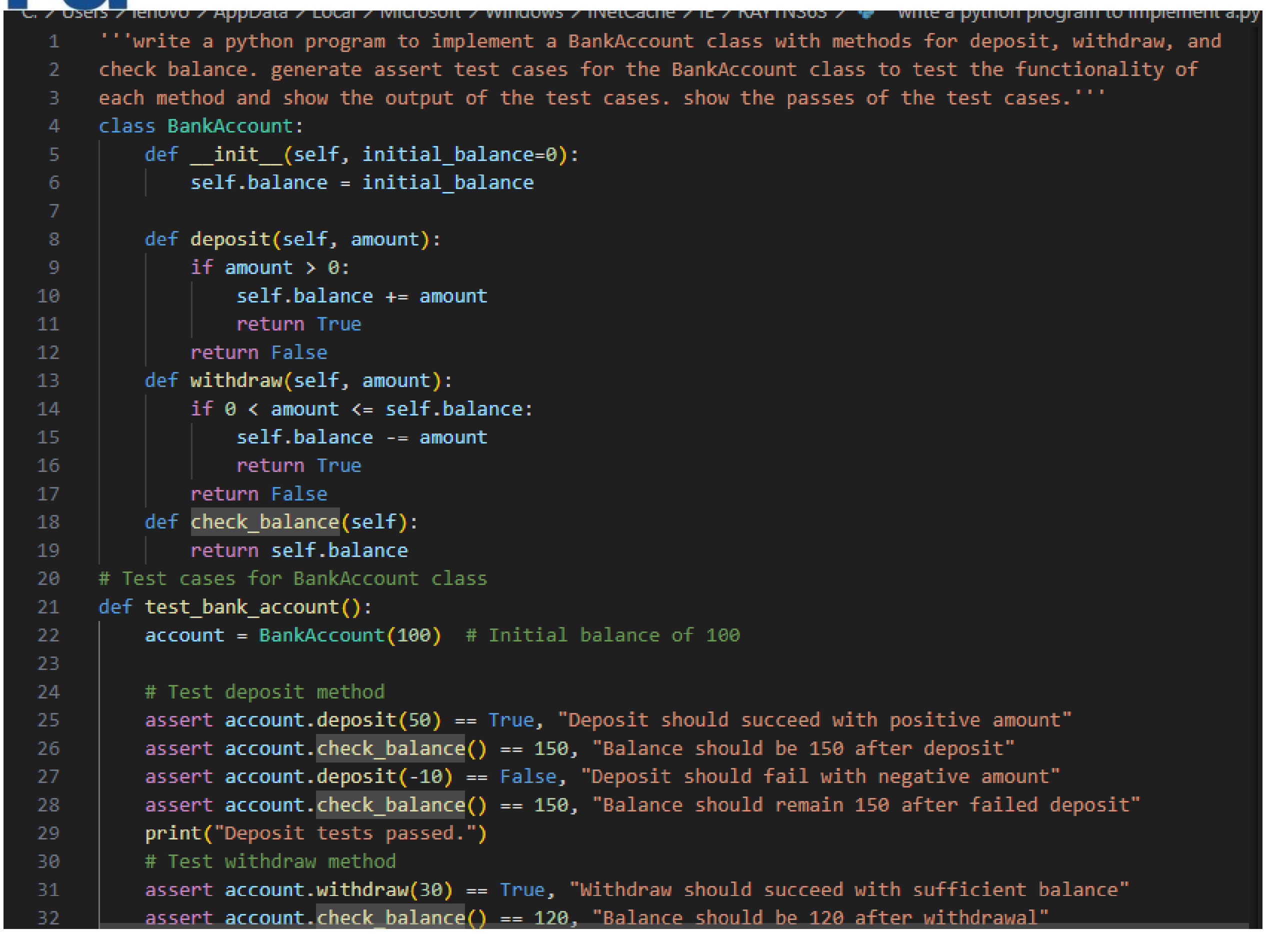
o deposit(amount) o withdraw(amount)
o get_balance()

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.**

Prompt:- write a python program to implement a BankAccount class with methods for deposit, withdraw, and check balance. generate assert test cases for the BankAccount class to test the functionality of each method and show the output of the test cases. show the passes of the test cases.

Code:-

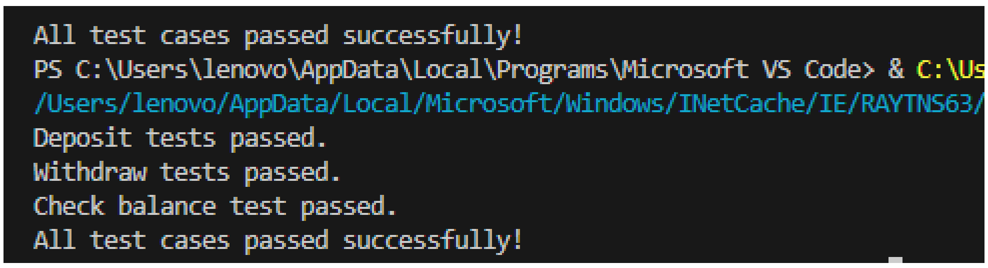


```
C: > Users > lenovo > AppData > Local > Microsoft > Windows > INetCache > IE > RAYTNS63 > write a python program to implement a.py
'''write a python program to implement a BankAccount class with methods for deposit, withdraw, and
check balance. generate assert test cases for the BankAccount class to test the functionality of
each method and show the output of the test cases. show the passes of the test cases.'''
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return True
        return False
    def check_balance(self):
        return self.balance
# Test cases for BankAccount class
def test_bank_account():
    account = BankAccount(100)  # Initial balance of 100

    # Test deposit method
    assert account.deposit(50) == True, "Deposit should succeed with positive amount"
    assert account.check_balance() == 150, "Balance should be 150 after deposit"
    assert account.deposit(-10) == False, "Deposit should fail with negative amount"
    assert account.check_balance() == 150, "Balance should remain 150 after failed deposit"
    print("Deposit tests passed.")
    # Test withdraw method
    assert account.withdraw(30) == True, "Withdraw should succeed with sufficient balance"
    assert account.check_balance() == 120, "Balance should be 120 after withdrawal"
```

Output:-



```
All test cases passed successfully!
PS C:\Users\lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Us
/Users/lenovo/AppData/Local/Microsoft/Windows/INetCache/IE/RAYTNS63/
Deposit tests passed.
Withdraw tests passed.
Check balance test passed.
All test cases passed successfully!
```

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

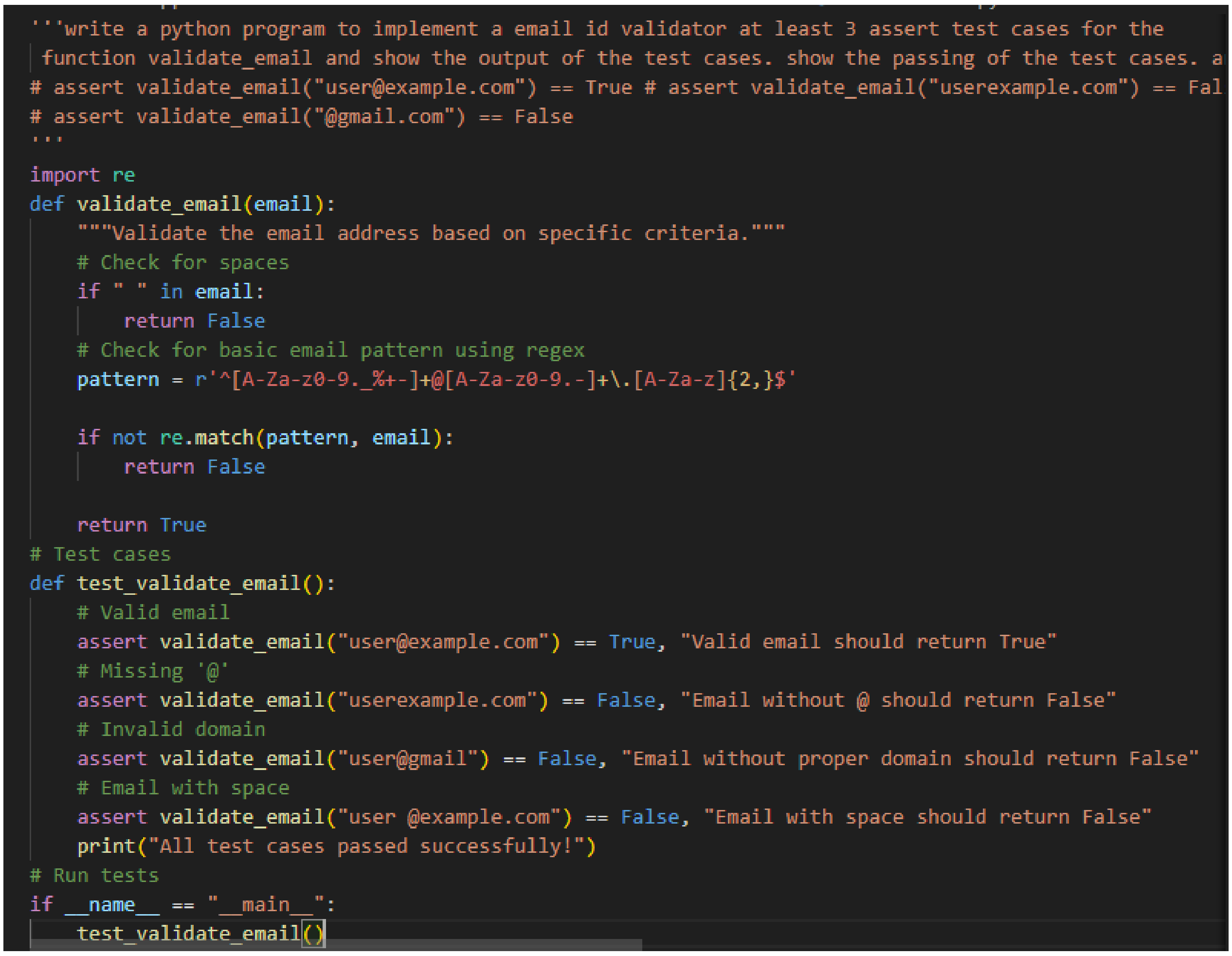
validate_email("@gmail.com") == False Expected Output #5:

- **Email validation function passing all AI-generated test cases and handling edge cases correctly.**

Prompt:-

write a python program to implement a email id validator at least 3 assert test cases for the function validate_email and show the output of the test cases. show the passing of the test cases. and requirements are it should contain "@" symbol, a valid domain name, and no spaces allowed.
# assert validate_email("user@example.com") == True
# assert validate_email("userexample.com") == False
# assert validate_email("@gmail.com") == False

Code:-

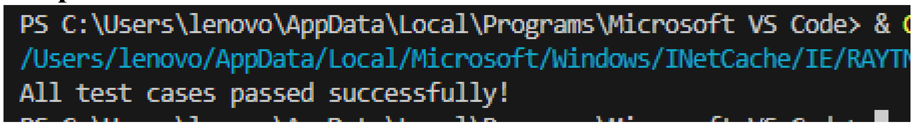


```python
'''write a python program to implement a email id validator at least 3 assert test cases for the
 function validate_email and show the output of the test cases. show the passing of the test cases. a
# assert validate_email("user@example.com") == True # assert validate_email("userexample.com") == Fal
# assert validate_email("@gmail.com") == False
'''
import re
def validate_email(email):
    """Validate the email address based on specific criteria."""
    # Check for spaces
    if " " in email:
        return False
    # Check for basic email pattern using regex
    pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'

    if not re.match(pattern, email):
        return False

    return True
# Test cases
def test_validate_email():
    # Valid email
    assert validate_email("user@example.com") == True, "Valid email should return True"
    # Missing '@'
    assert validate_email("userexample.com") == False, "Email without @ should return False"
    # Invalid domain
    assert validate_email("user@gmail") == False, "Email without proper domain should return False"
    # Email with space
    assert validate_email("user @example.com") == False, "Email with space should return False"
    print("All test cases passed successfully!")
# Run tests
if __name__ == "__main__":
    test_validate_email()
```

Output:-



```
PS C:\Users\lenovo\AppData\Local\Programs\Microsoft VS Code> & C
/Users/lenovo/AppData/Local/Microsoft/Windows/INetCache/IE/RAYTM
All test cases passed successfully!
```