Course Title	AI Assisted Coding

Name: G. Snehith Roll Number: 2303A52324 Batch: 45

Assignment Number: 8.5(Present assignment number)/**24**(Total number of assignments)

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

Lab Objectives:

- To introduce students to test-driven development (TDD) using AI code generation tools.
- To enable the generation of test cases before writing code implementations.
- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

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:
  - Username length must be between 5 and 15 characters.
  - Must contain only alphabets and digits.
  - Must not start with a digit.
  - 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: Generate Python code using Test-Driven Development (TDD) for a function is_valid_username(username).

First, write at least 5 assert test cases.

Then implement the code.

Requirements:

Username length must be between 5 and 15 characters.

Must contain only alphabets and digits.

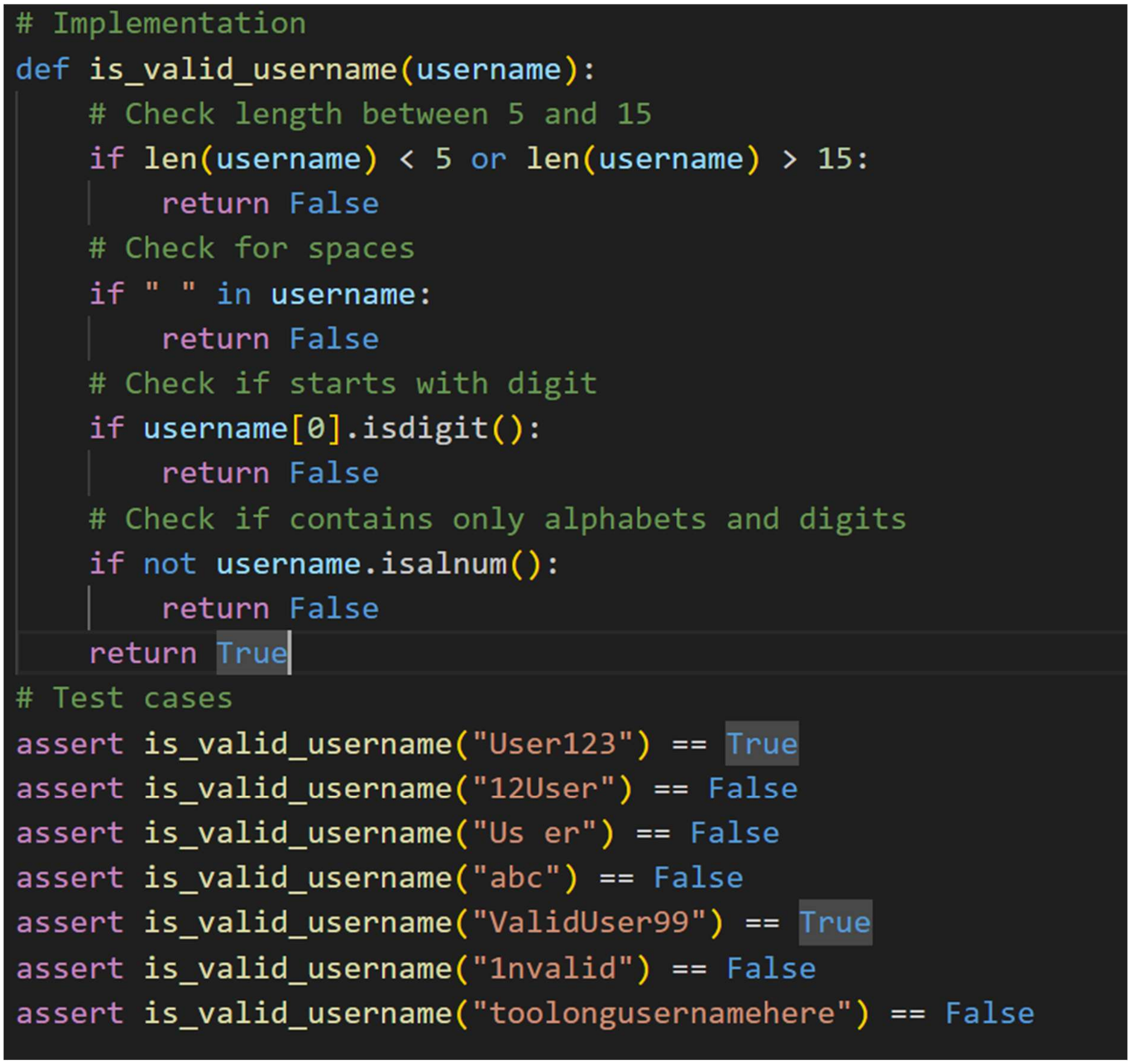
Must not start with a digit.

No spaces allowed.

Return True if valid, otherwise False.

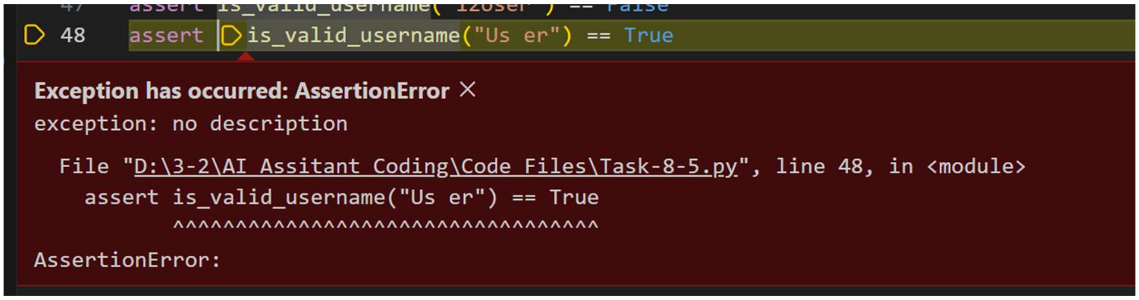
Ensure all test cases pass.

Code:



```python
# Implementation
def is_valid_username(username):
    # Check length between 5 and 15
    if len(username) < 5 or len(username) > 15:
        return False
    # Check for spaces
    if " " in username:
        return False
    # Check if starts with digit
    if username[0].isdigit():
        return False
    # Check if contains only alphabets and digits
    if not username.isalnum():
        return False
    return True
# 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("abc") == False
assert is_valid_username("ValidUser99") == True
assert is_valid_username("1nvalid") == False
assert is_valid_username("toolongusernamehere") == False
```

Output:



```
assert is_valid_username("12user") == False
assert is_valid_username("Us er") == True
```

Exception has occurred: AssertionError

```
exception: no description
  File "D:\3-2\AI Assitant Coding\Code Files\Task-8-5.py", line 48, in <module>
    assert is_valid_username("Us er") == True
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError:
```

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: Generate Python code using Test-Driven Development (TDD) for a function classify_value(x).

First, write at least 5 assert test cases.

Then implement the function using conditional logic and loops.

Requirements:

If input is an integer, return "Even" or "Odd".

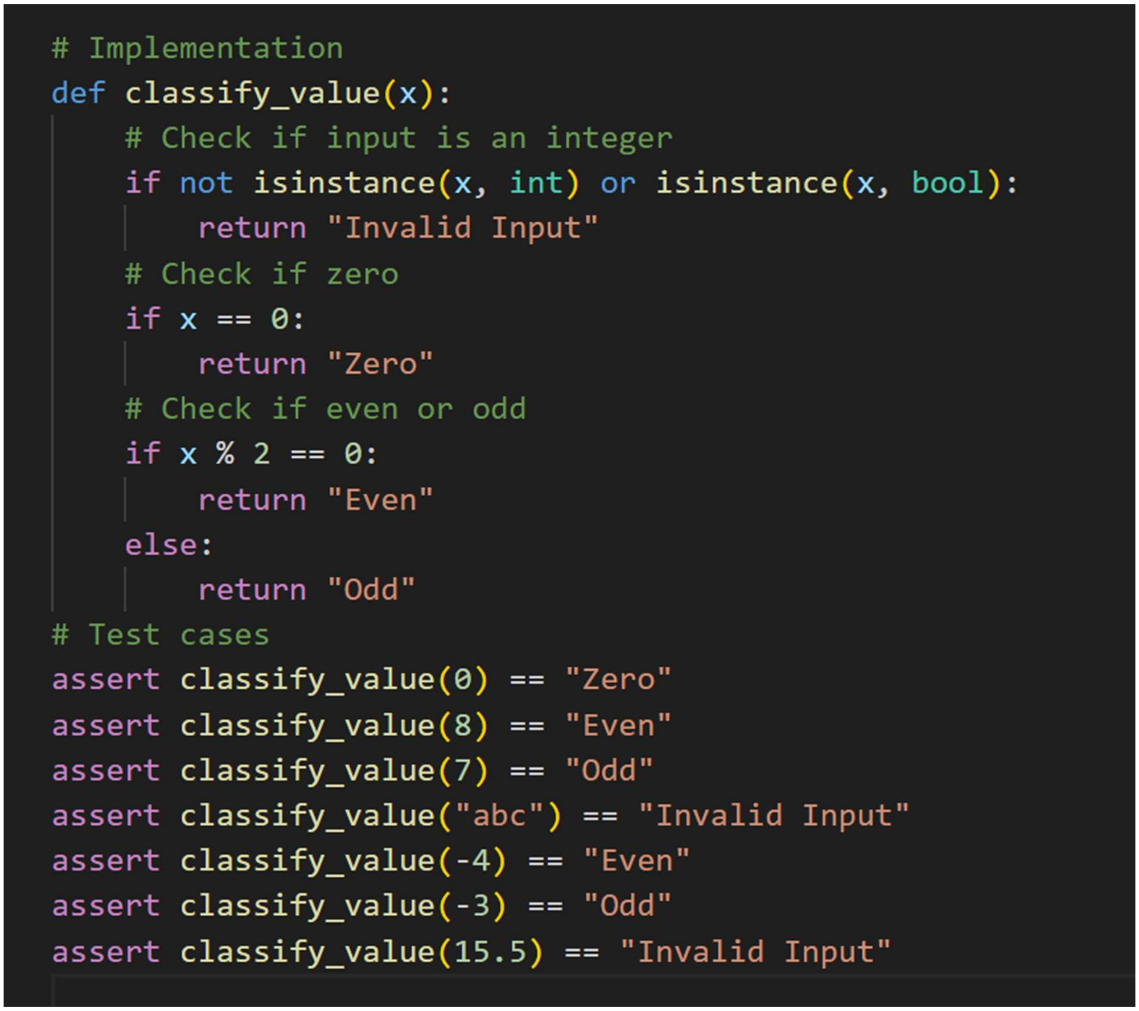
If input is 0, return "Zero".

If input is non-numeric, return "Invalid Input".

Ensure all test cases pass successfully.

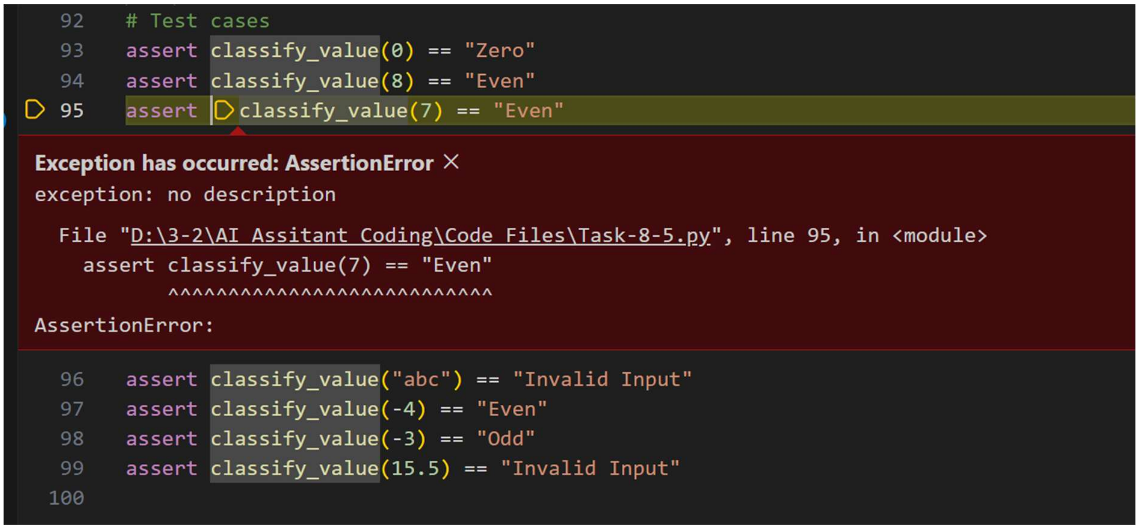
Provide clean and properly formatted Python code only.

Code:



```python
# Implementation
def classify_value(x):
    # Check if input is an integer
    if not isinstance(x, int) or isinstance(x, bool):
        return "Invalid Input"
    # Check if zero
    if x == 0:
        return "Zero"
    # Check if even or odd
    if x % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Test cases
assert classify_value(0) == "Zero"
assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("abc") == "Invalid Input"
assert classify_value(-4) == "Even"
assert classify_value(-3) == "Odd"
assert classify_value(15.5) == "Invalid Input"
```

Output:



```
 92  # Test cases
 93  assert classify_value(0) == "Zero"
 94  assert classify_value(8) == "Even"
 95  assert classify_value(7) == "Even"
```

Exception has occurred: AssertionError ×

```
exception: no description
  File "D:\3-2\AI Assitant Coding\Code Files\Task-8-5.py", line 95, in <module>
    assert classify_value(7) == "Even"
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError:
```

```
 96  assert classify_value("abc") == "Invalid Input"
 97  assert classify_value(-4) == "Even"
 98  assert classify_value(-3) == "Odd"
 99  assert classify_value(15.5) == "Invalid Input"
100
```

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.

Prompt: Generate Python code using Test-Driven Development (TDD) for a function is_palindrome(text).

First, write at least 5 assert test cases.

Then implement the function.

Requirements:

Ignore case, spaces, and punctuation.

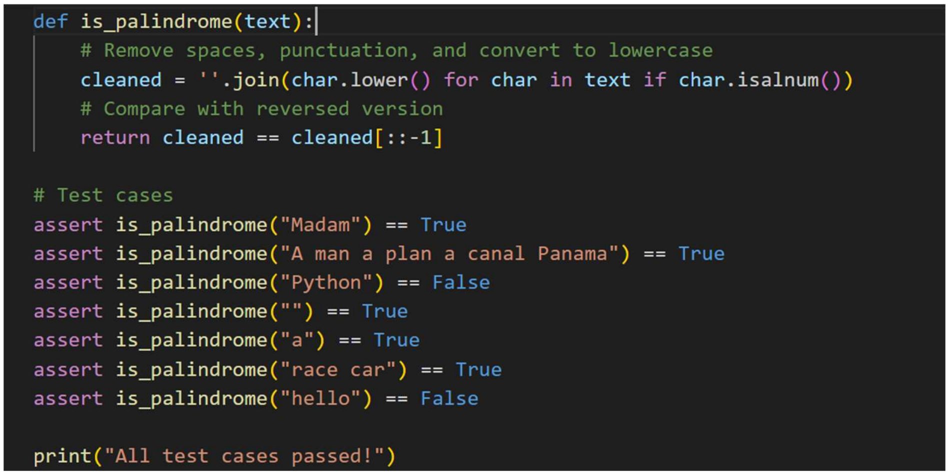
Handle edge cases like empty strings and single characters.

Return True if the text is a palindrome, otherwise False.

Ensure all test cases pass.

Provide clean and properly formatted Python code only.

Code:

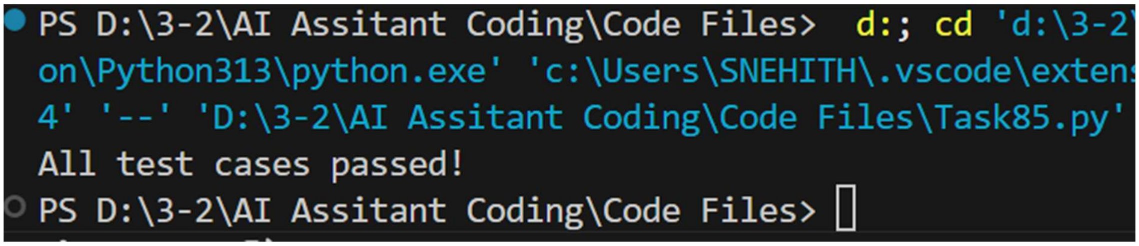


```python
def is_palindrome(text):
    # Remove spaces, punctuation, and convert to lowercase
    cleaned = ''.join(char.lower() for char in text if char.isalnum())
    # Compare with reversed version
    return cleaned == cleaned[::-1]

# Test cases
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
assert is_palindrome("race car") == True
assert is_palindrome("hello") == False

print("All test cases passed!")
```

Output:



```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2
on\Python313\python.exe' 'c:\Users\SNEHITH\.vscode\exten
4' '--' 'D:\3-2\AI Assitant Coding\Code Files\Task85.py'
All test cases passed!
PS D:\3-2\AI Assitant Coding\Code Files> 
```

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:
  - deposit(amount)
  - withdraw(amount)
  - 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: Generate Python code using Test-Driven Development (TDD) for a BankAccount class.

First, write at least 5 assert-based test cases.

Then implement the class.

The class should include:

deposit(amount)

withdraw(amount)

get_balance()

Ensure:

Deposits increase balance.

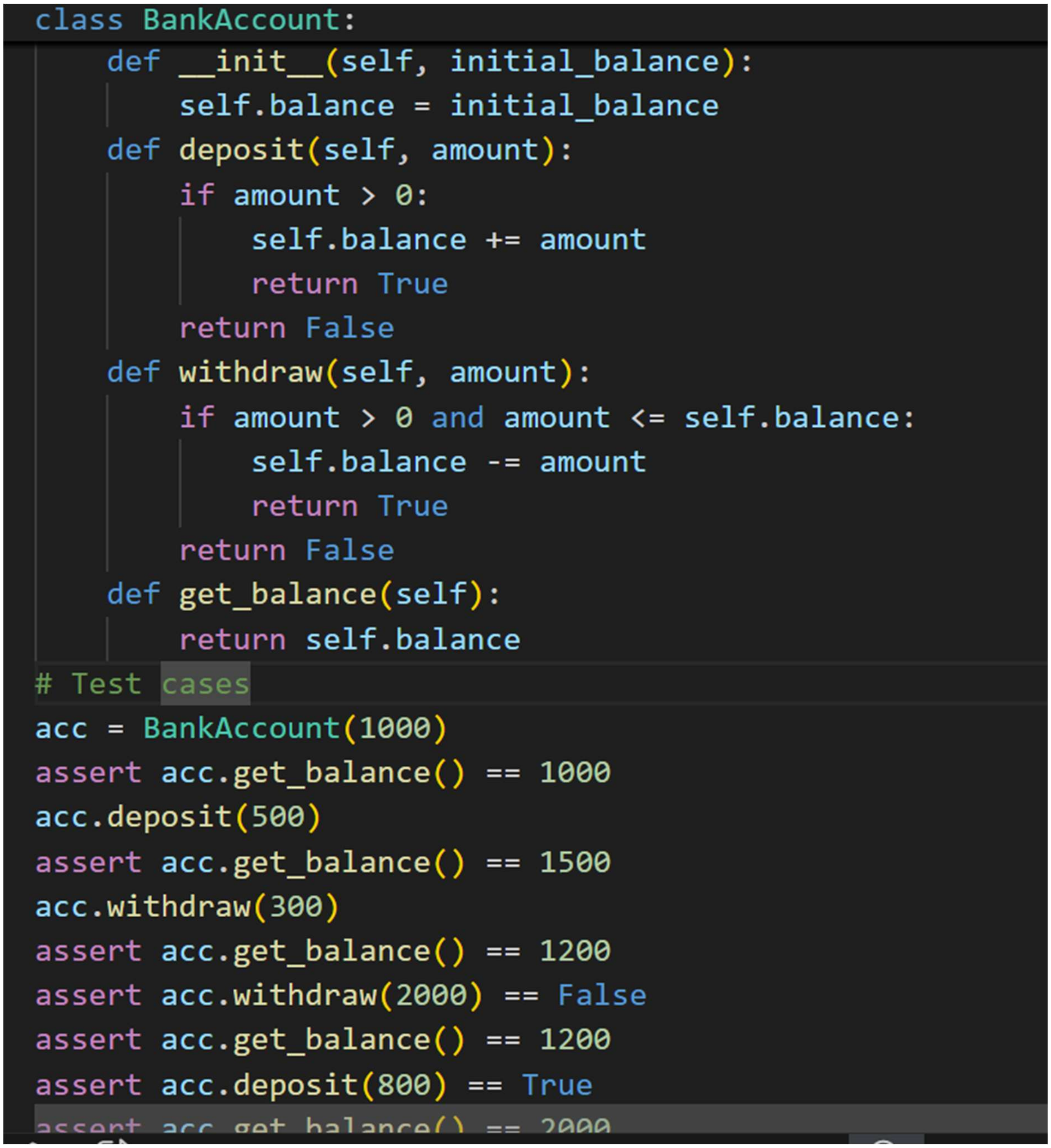
Withdrawals decrease balance (handle insufficient balance properly).

Initial balance is set through the constructor.

Make sure all test cases pass.

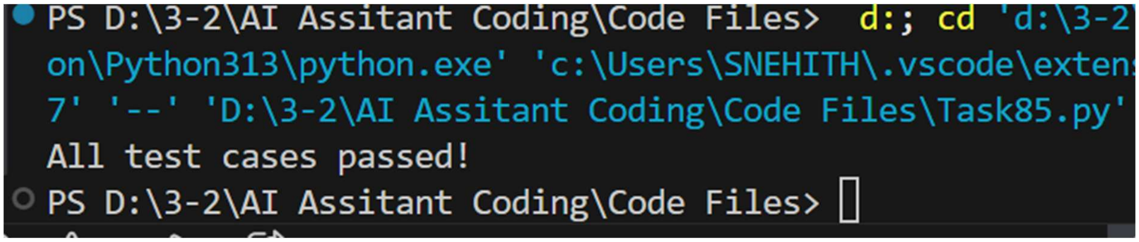
Provide clean and properly formatted Python code only.

Code:



```python
class BankAccount:
    def __init__(self, initial_balance):
        self.balance = initial_balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False
    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            return True
        return False
    def get_balance(self):
        return self.balance
# Test cases
acc = BankAccount(1000)
assert acc.get_balance() == 1000
acc.deposit(500)
assert acc.get_balance() == 1500
acc.withdraw(300)
assert acc.get_balance() == 1200
assert acc.withdraw(2000) == False
assert acc.get_balance() == 1200
assert acc.deposit(800) == True
assert acc.get_balance() == 2000
```

Output:



```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2
on\Python313\python.exe' 'c:\Users\SNEHITH\.vscode\exten
7' '--' 'D:\3-2\AI Assitant Coding\Code Files\Task85.py'
All test cases passed!
PS D:\3-2\AI Assitant Coding\Code Files> 
```

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:
  - Must contain @ and .
  - Must not start or end with special characters.
  - 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: Generate Python code using Test-Driven Development (TDD) for a function validate_email(email).

First, write at least 5 assert test cases.

Then implement the function.

Requirements:

Email must contain @ and .

Must not start or end with special characters

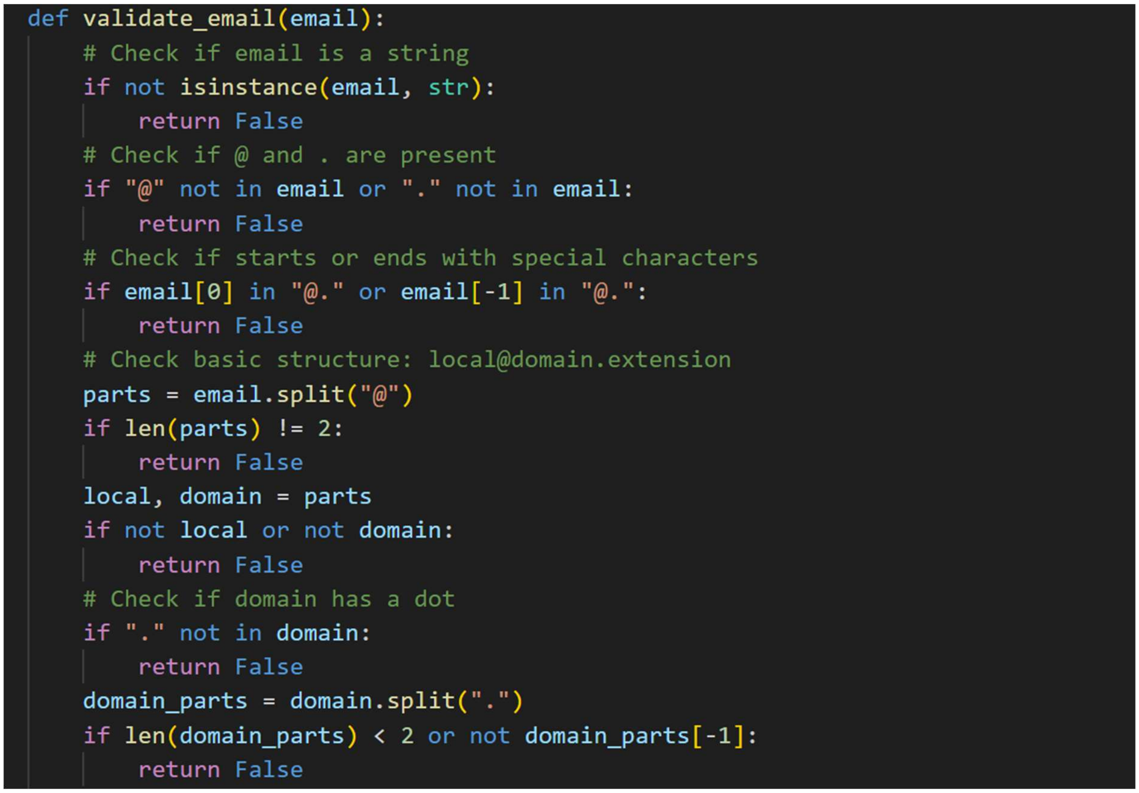
Handle invalid formats gracefully

Return True if valid, otherwise False

Ensure all test cases pass successfully.

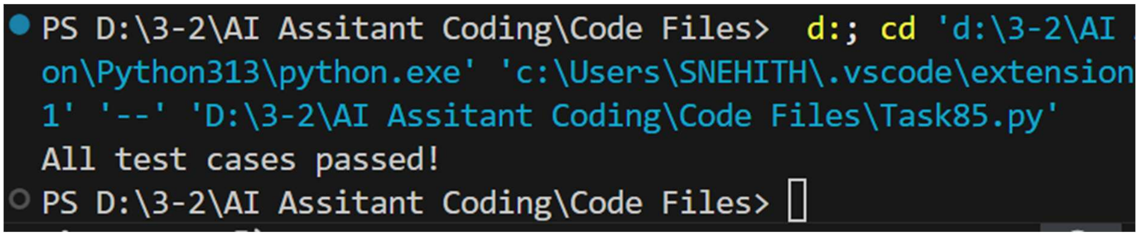
Provide clean and properly formatted Python code only.

Code:



```
def validate_email(email):
    # Check if email is a string
    if not isinstance(email, str):
        return False
    # Check if @ and . are present
    if "@" not in email or "." not in email:
        return False
    # Check if starts or ends with special characters
    if email[0] in "@." or email[-1] in "@.":
        return False
    # Check basic structure: local@domain.extension
    parts = email.split("@")
    if len(parts) != 2:
        return False
    local, domain = parts
    if not local or not domain:
        return False
    # Check if domain has a dot
    if "." not in domain:
        return False
    domain_parts = domain.split(".")
    if len(domain_parts) < 2 or not domain_parts[-1]:
        return False
```

Output:



```
PS D:\3-2\AI Assitant Coding\Code Files>  d:; cd 'd:\3-2\AI
on\Python313\python.exe' 'c:\Users\SNEHITH\.vscode\extension
1' '--' 'D:\3-2\AI Assitant Coding\Code Files\Task85.py'
All test cases passed!
PS D:\3-2\AI Assitant Coding\Code Files> 
```