Lab-8.5

Course : AI Assisted Coding

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

Student Details:

Name	: Mohammed Sabir
Hall Ticket No	: 2303A51506
Batch	: 22

Task-01

Final Optimal Prompt:

write a python program to to give user name as a string and print is in true or false

the name string should be in 5 to 15 characters long

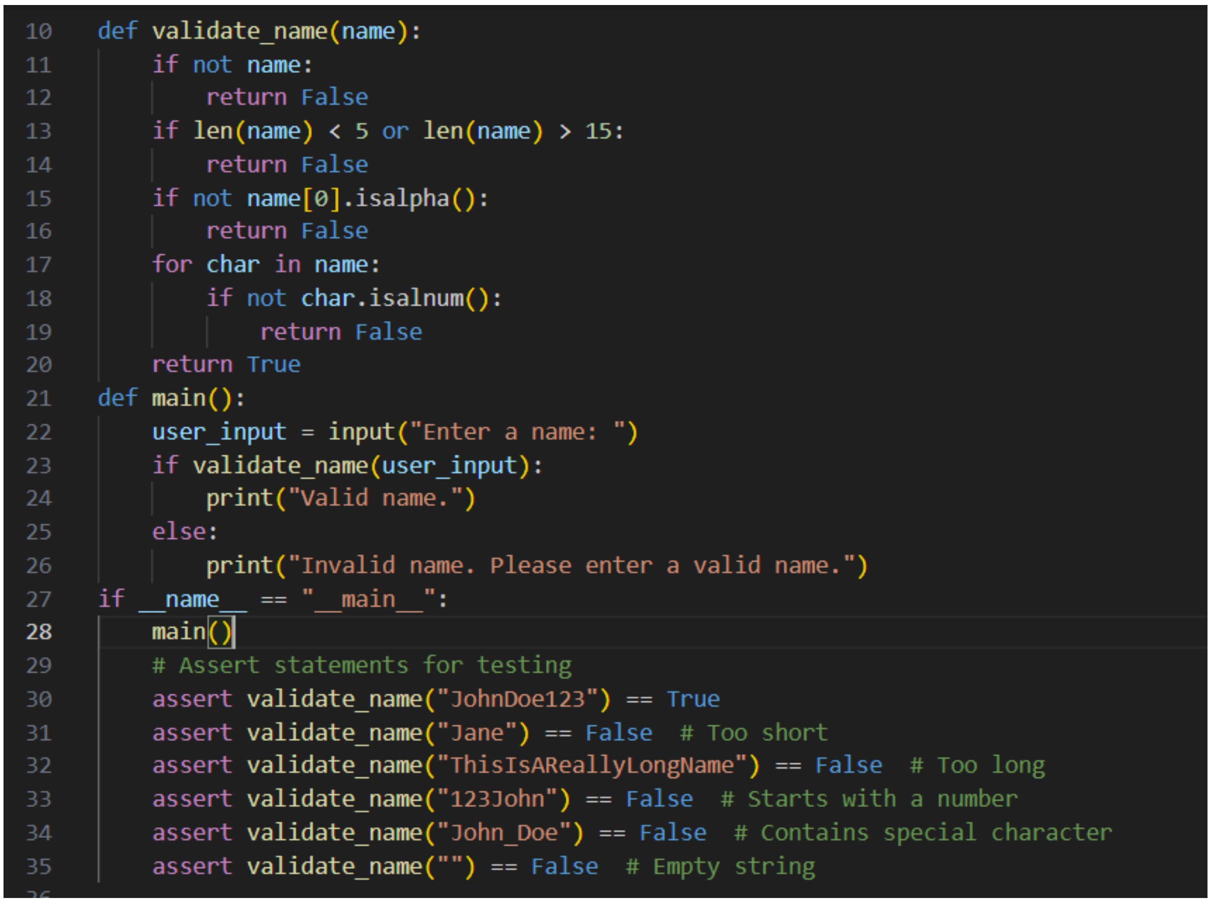
the name should contains only alphabets and should not be empty in the last after name u can give integers

the name should can not start with a number or special character

in the middle of a name there can be no space or special character

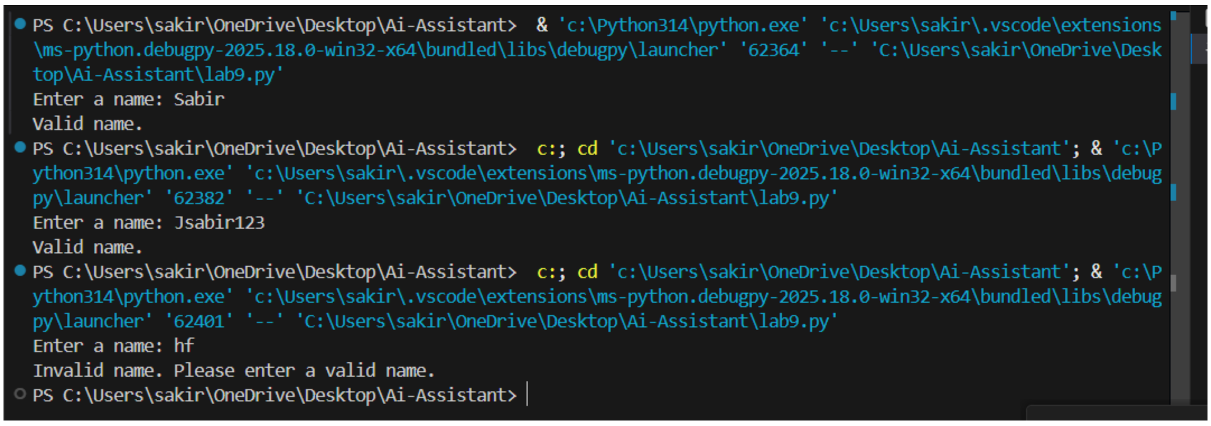
give at least 3 assert statements to test the function

Code Screenshot :



```python
def validate_name(name):
    if not name:
        return False
    if len(name) < 5 or len(name) > 15:
        return False
    if not name[0].isalpha():
        return False
    for char in name:
        if not char.isalnum():
            return False
    return True
def main():
    user_input = input("Enter a name: ")
    if validate_name(user_input):
        print("Valid name.")
    else:
        print("Invalid name. Please enter a valid name.")
if __name__ == "__main__":
    main()
    # Assert statements for testing
    assert validate_name("JohnDoe123") == True
    assert validate_name("Jane") == False  # Too short
    assert validate_name("ThisIsAReallyLongName") == False  # Too long
    assert validate_name("123John") == False  # Starts with a number
    assert validate_name("John_Doe") == False  # Contains special character
    assert validate_name("") == False  # Empty string
```

Output Screenshot:



```
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  & 'c:\Python314\python.exe' 'c:\Users\sakir\.vscode\extensions
\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62364' '--' 'C:\Users\sakir\OneDrive\Desk
top\Ai-Assistant\lab9.py'
Enter a name: Sabir
Valid name.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\P
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '62382' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a name: Jsabir123
Valid name.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\P
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '62401' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a name: hf
Invalid name. Please enter a valid name.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

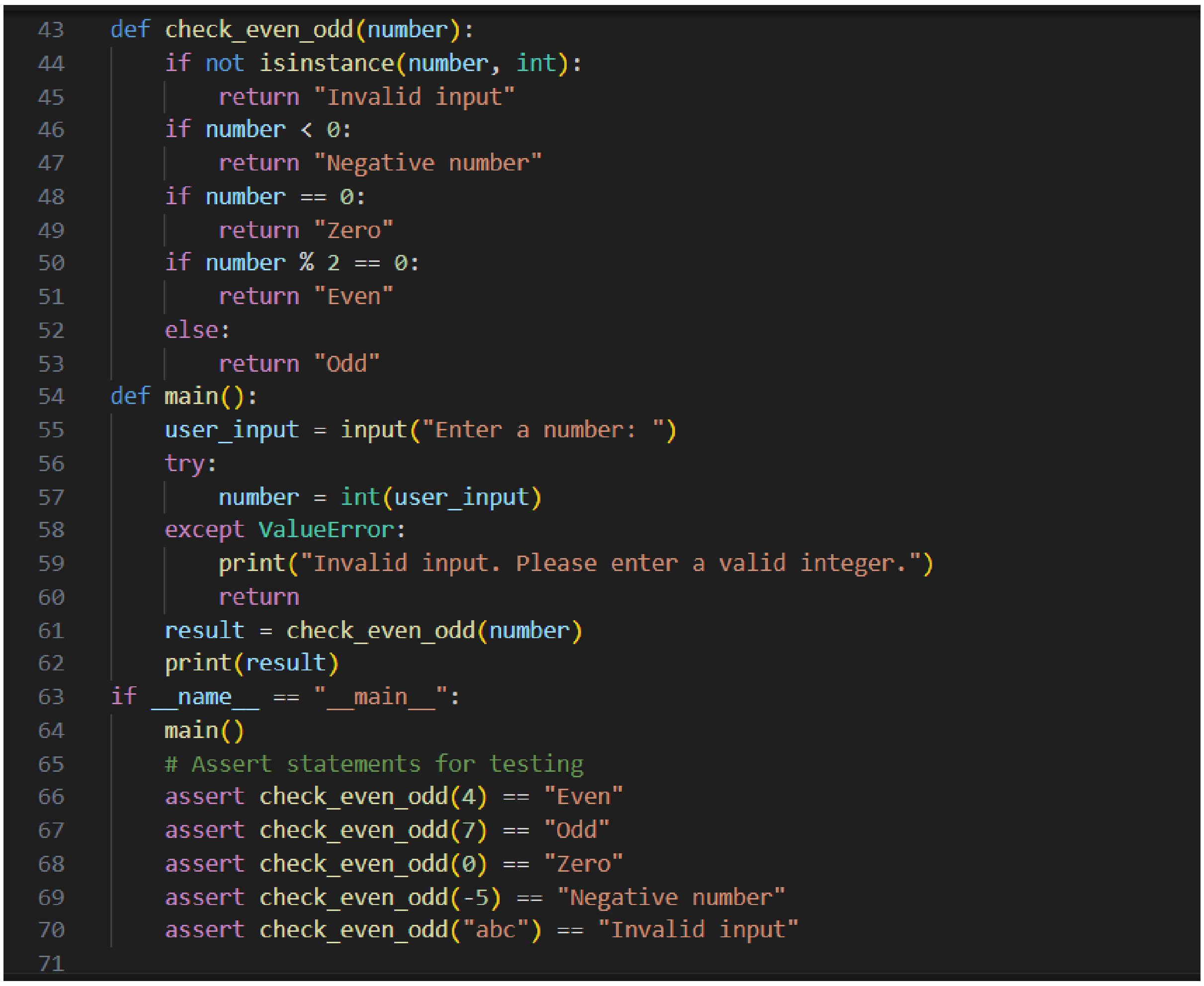
The program validates a name based on specific rules to ensure proper formatting. It first checks whether the input is empty, returning False if no name is provided. Then, it verifies that the length of the name is between 5 and 15 characters. The program also ensures that the first character is an alphabet using isalpha(), preventing names from starting with numbers or symbols. Additionally, it checks that every character in the name is alphanumeric, disallowing spaces and special characters. The main() function manages user interaction, and assert statements confirm correctness. Overall, the program demonstrates effective validation logic and structured coding.

Task-02

Final Optimal Prompt:

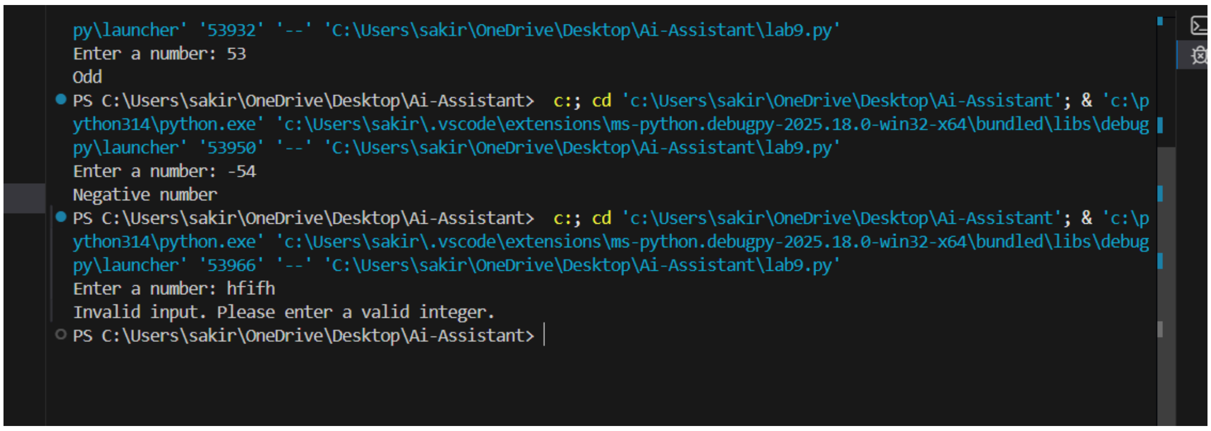
write a python program code to check the number is even or odd using conditional logic and loopsif input is zero it should return zeroif input is non numeric then return invalid inputif input is negative then return negative numbergive at least 3 assert statements to test the function

Code Screenshot :



```python
def check_even_odd(number):
    if not isinstance(number, int):
        return "Invalid input"
    if number < 0:
        return "Negative number"
    if number == 0:
        return "Zero"
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
def main():
    user_input = input("Enter a number: ")
    try:
        number = int(user_input)
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
        return
    result = check_even_odd(number)
    print(result)
if __name__ == "__main__":
    main()
    # Assert statements for testing
    assert check_even_odd(4) == "Even"
    assert check_even_odd(7) == "Odd"
    assert check_even_odd(0) == "Zero"
    assert check_even_odd(-5) == "Negative number"
    assert check_even_odd("abc") == "Invalid input"

```

Output Screenshot:



```
py\launcher' '53932' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a number: 53
Odd
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\p
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '53950' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a number: -54
Negative number
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\p
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '53966' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a number: hfifh
Invalid input. Please enter a valid integer.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program determines whether a given number is even, odd, zero, negative, or invalid. It first checks if the input is an integer using isinstance(), ensuring type safety. If the number is negative, it returns "Negative number," and if it is zero, it returns "Zero." For positive numbers, it uses the modulus operator (%) to check divisibility by 2 to identify even or odd numbers. The main() function handles user input and uses exception handling (try-except) to manage invalid entries. Assert statements are included for testing. Overall, the program demonstrates proper validation, error handling, and logical structure.

Task-03

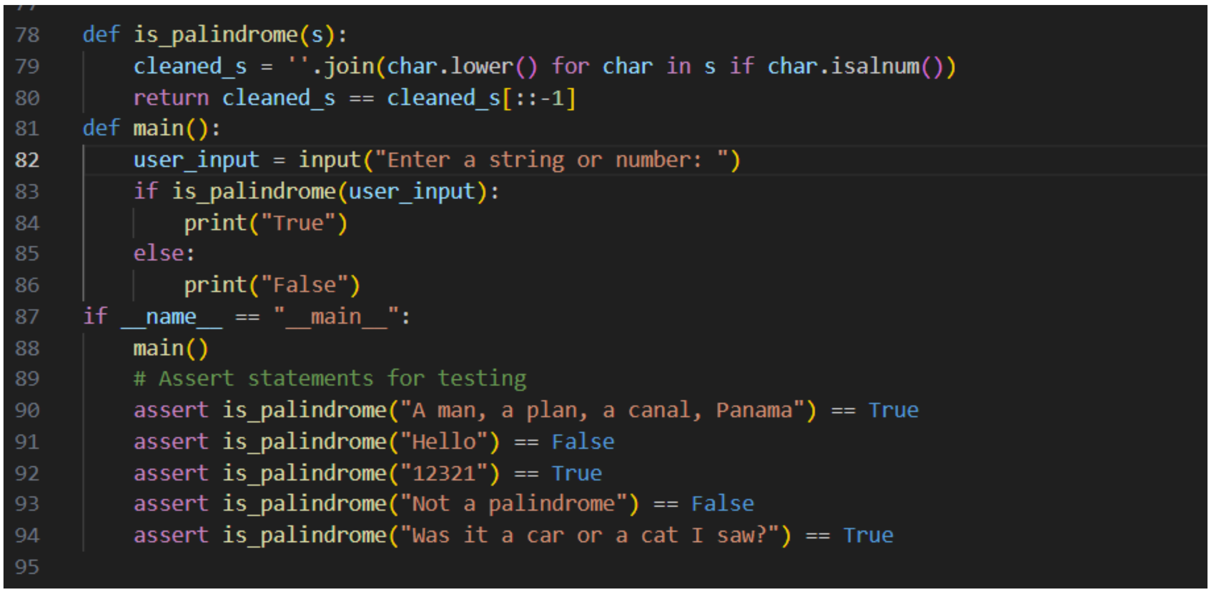
Final Optimal Prompt:

write a program to check the give number or string is a palinderom or not return only true or false

a palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization).

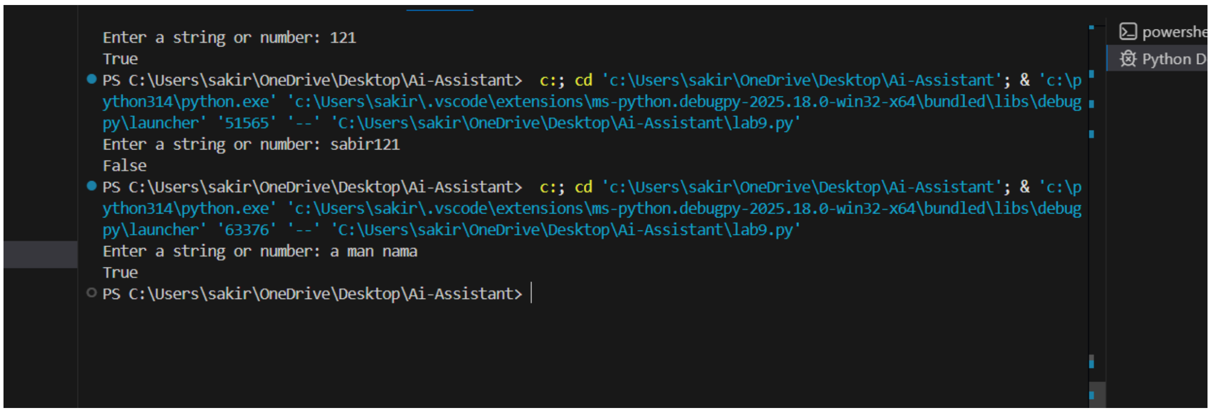
give at least 3 assert statements to test the function

Code Screenshot :



```python
def is_palindrome(s):
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned_s == cleaned_s[::-1]
def main():
    user_input = input("Enter a string or number: ")
    if is_palindrome(user_input):
        print("True")
    else:
        print("False")
if __name__ == "__main__":
    main()
    # Assert statements for testing
    assert is_palindrome("A man, a plan, a canal, Panama") == True
    assert is_palindrome("Hello") == False
    assert is_palindrome("12321") == True
    assert is_palindrome("Not a palindrome") == False
    assert is_palindrome("Was it a car or a cat I saw?") == True
```

Output Screenshot:



```
Enter a string or number: 121
True
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\p
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '51565' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a string or number: sabir121
False
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\p
ython314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debug
py\launcher' '63376' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter a string or number: a man nama
True
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program checks whether a given string or number is a palindrome using a clean and efficient approach. It first removes all non-alphanumeric characters and converts the remaining characters to lowercase using a generator expression. This ensures that spaces, punctuation, and case differences do not affect the result. The cleaned string is then compared with its reverse using slicing ([::-1]). If both are equal, the function returns True; otherwise, it returns False. The main() function handles user input, and assert statements are included to test correctness. Overall, the program demonstrates good string handling and logical implementation.

Task-04

Final Optimal Prompt:

write a python program to check the bank account class with the following attributes and methods:

Attributes:

- account_number (string)

- account_holder (string)

- balance (float)

Methods:

- deposit(amount): adds the specified amount to the balance

- withdraw(amount): subtracts the specified amount from the balance if sufficient funds are available, otherwise returns an error message

- get_balance(): returns the current balance

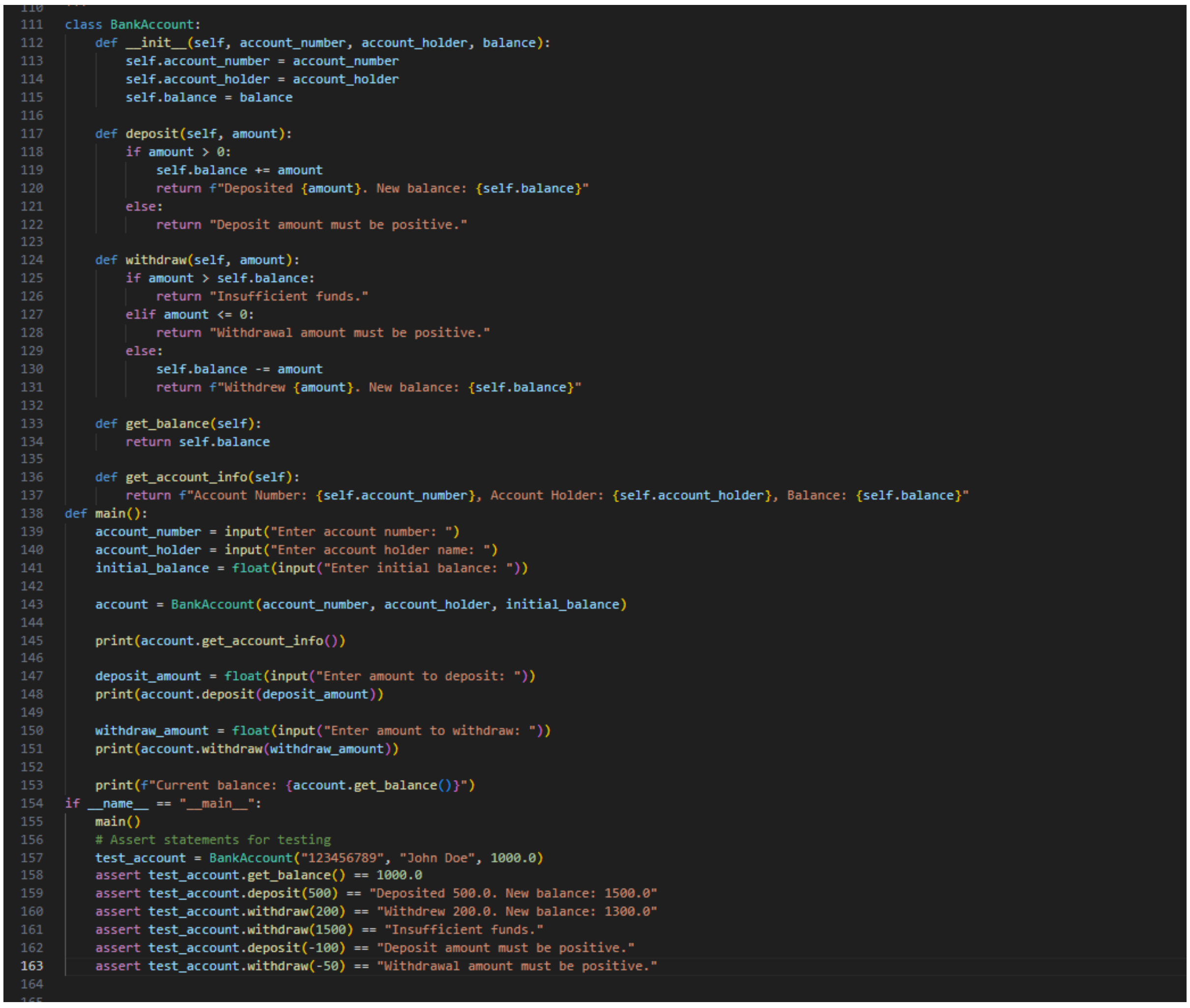
- get_account_info(): returns a string with the account number, account holder, and balance

give at least 3 assert statements to test the class and its methods

user shoulbe be give the inputs for account number, account holder and initial balance when creating an instance of the BankAccount class

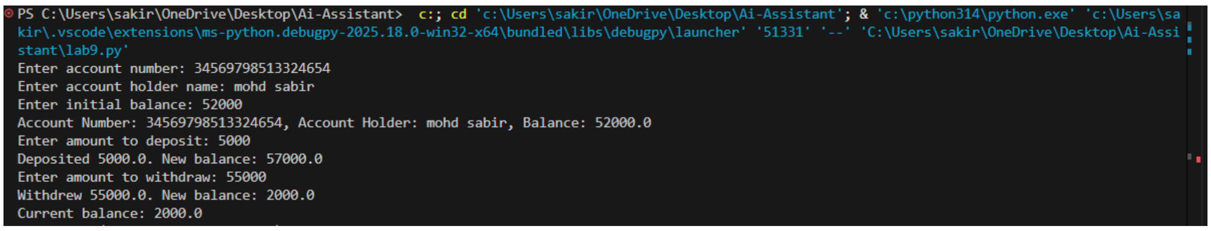
---

Code Screenshot :



```python
class BankAccount:
    def __init__(self, account_number, account_holder, balance):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposited {amount}. New balance: {self.balance}"
        else:
            return "Deposit amount must be positive."

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds."
        elif amount <= 0:
            return "Withdrawal amount must be positive."
        else:
            self.balance -= amount
            return f"Withdrew {amount}. New balance: {self.balance}"

    def get_balance(self):
        return self.balance

    def get_account_info(self):
        return f"Account Number: {self.account_number}, Account Holder: {self.account_holder}, Balance: {self.balance}"
def main():
    account_number = input("Enter account number: ")
    account_holder = input("Enter account holder name: ")
    initial_balance = float(input("Enter initial balance: "))

    account = BankAccount(account_number, account_holder, initial_balance)

    print(account.get_account_info())

    deposit_amount = float(input("Enter amount to deposit: "))
    print(account.deposit(deposit_amount))

    withdraw_amount = float(input("Enter amount to withdraw: "))
    print(account.withdraw(withdraw_amount))

    print(f"Current balance: {account.get_balance()}")
if __name__ == "__main__":
    main()
    # Assert statements for testing
    test_account = BankAccount("123456789", "John Doe", 1000.0)
    assert test_account.get_balance() == 1000.0
    assert test_account.deposit(500) == "Deposited 500.0. New balance: 1500.0"
    assert test_account.withdraw(200) == "Withdrew 200.0. New balance: 1300.0"
    assert test_account.withdraw(1500) == "Insufficient funds."
    assert test_account.deposit(-100) == "Deposit amount must be positive."
    assert test_account.withdraw(-50) == "Withdrawal amount must be positive."
```

Output Screenshot:



```
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\python314\python.exe' 'c:\Users\sakir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51331' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assistant\lab9.py'
Enter account number: 345697985133246654
Enter account holder name: mohd sabir
Enter initial balance: 52000
Account Number: 345697985133246654, Account Holder: mohd sabir, Balance: 52000.0
Enter amount to deposit: 5000
Deposited 5000.0. New balance: 57000.0
Enter amount to withdraw: 55000
Withdrew 55000.0. New balance: 2000.0
Current balance: 2000.0
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The given program demonstrates a simple implementation of Object-Oriented Programming in Python using a BankAccount class. It encapsulates account details such as account number, account holder name, and balance within a single class. The methods deposit() and withdraw() include proper validation to ensure that transactions are positive and that withdrawals do not exceed the available balance. The get_balance() and get_account_info() methods provide controlled access to account data. The main() function handles user interaction, while assert statements are used for testing functionality. Overall, the program shows good structure, validation logic, and practical use of OOP concepts

---

Tack-05

Final Optimal Prompt:

write a python program to create a email validator function that checks if the given email address is valid or not. The function should return true if the email is valid and false if it is not. A valid email address should have the following characteristics:

- It should contain exactly one "@" symbol.

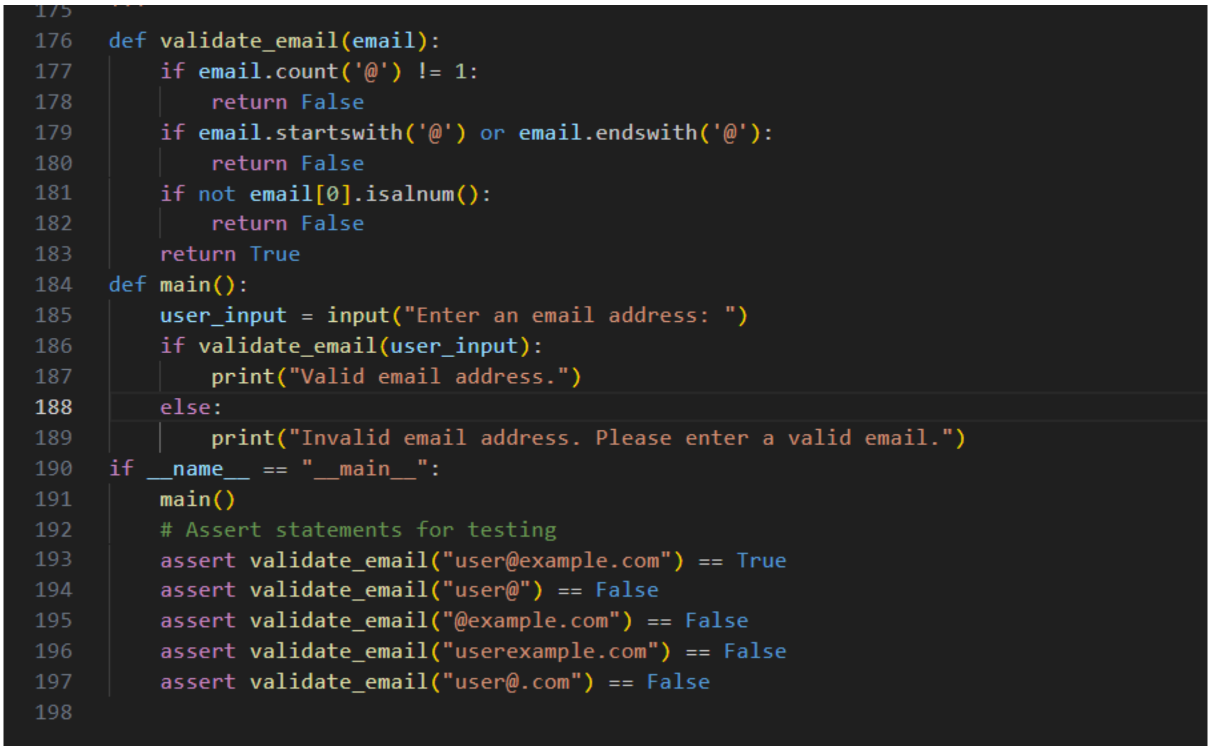
- The "@" symbol should not be the first or last character of the email address.

it must not start with special characters

it should handle invalid formats gracefully and return false for any invalid email address.

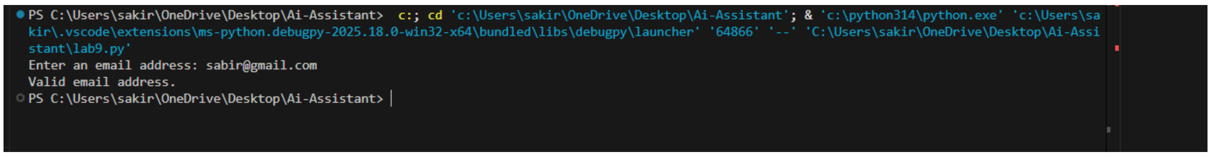
give at least 3 assert statements to test the function

Code Screenshot :



```python
def validate_email(email):
    if email.count('@') != 1:
        return False
    if email.startswith('@') or email.endswith('@'):
        return False
    if not email[0].isalnum():
        return False
    return True
def main():
    user_input = input("Enter an email address: ")
    if validate_email(user_input):
        print("Valid email address.")
    else:
        print("Invalid email address. Please enter a valid email.")
if __name__ == "__main__":
    main()
    # Assert statements for testing
    assert validate_email("user@example.com") == True
    assert validate_email("user@") == False
    assert validate_email("@example.com") == False
    assert validate_email("userexample.com") == False
    assert validate_email("user@.com") == False
```

Output Screenshot:



```
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant>  c:; cd 'c:\Users\sakir\OneDrive\Desktop\Ai-Assistant'; & 'c:\python314\python.exe' 'c:\Users\sa
kir\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64866' '--' 'C:\Users\sakir\OneDrive\Desktop\Ai-Assi
stant\lab9.py'
Enter an email address: sabir@gmail.com
Valid email address.
PS C:\Users\sakir\OneDrive\Desktop\Ai-Assistant> 
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :

The program is designed to check whether an email address is valid based on basic formatting rules. It verifies that the email contains exactly one '@' symbol and ensures it is not placed at the beginning or end of the address. The first character must be alphanumeric, which prevents invalid formats. The email is then split into local and domain parts, and the domain is checked to confirm it contains a dot and does not start or end with one. The use of functions and assert statements shows good structure, testing, and logical implementation of email validation.