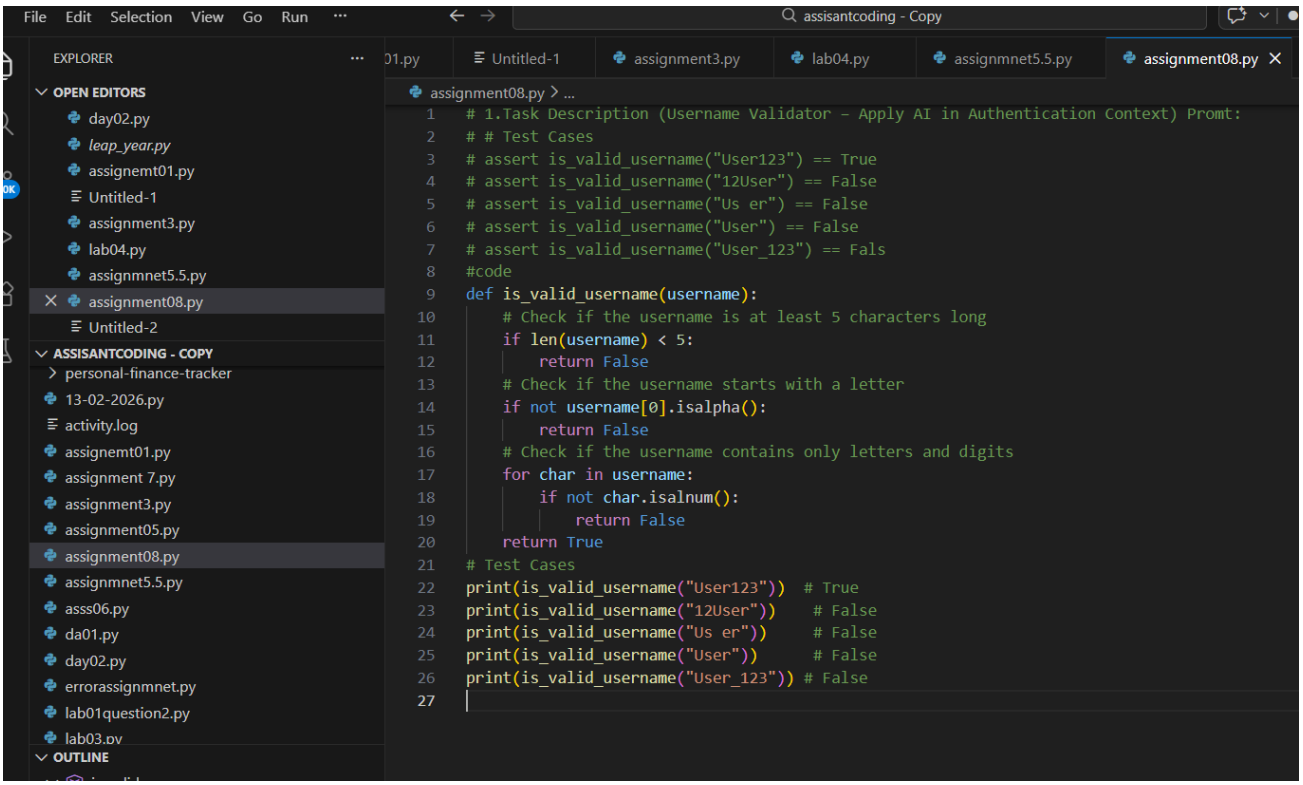
Assignment–8.5

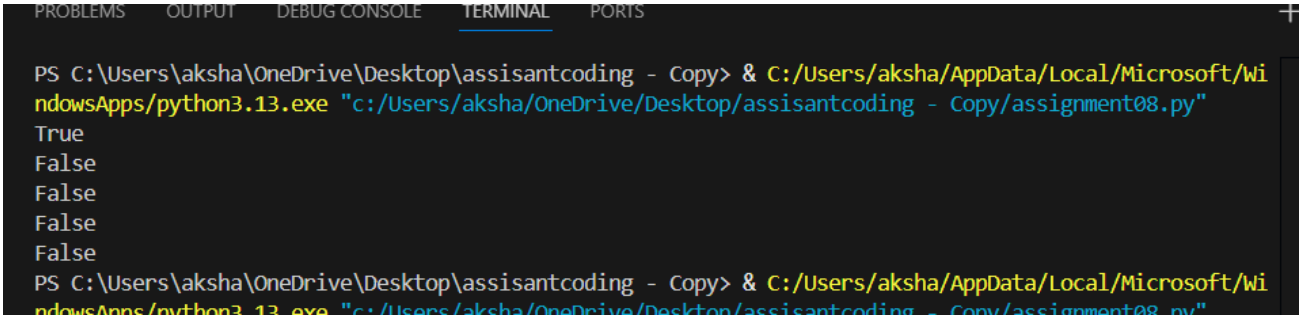
Name : J.Akshaya
Batch : 22
Ht.No : 2303A51629

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:



Output:



Observation:

AI-generated assert test cases helped define the user 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"

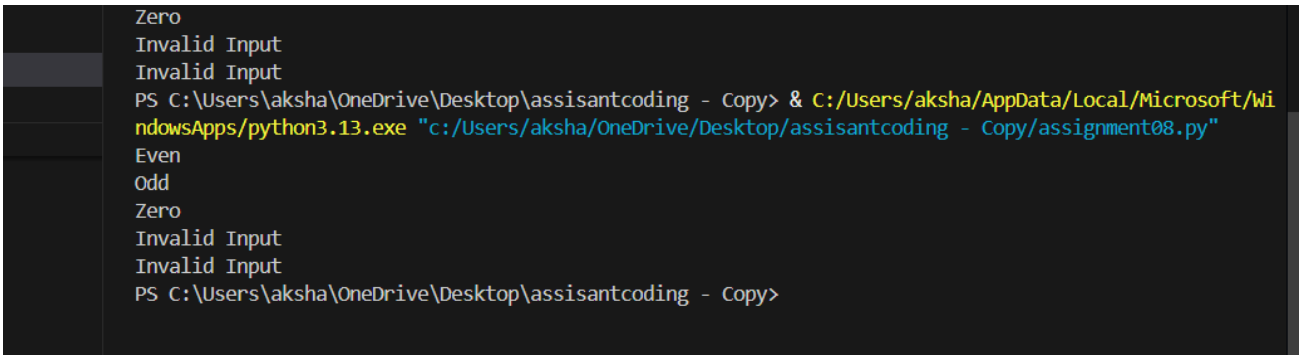
Input:

```python

# Task 2: 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"
def classify_value(value):
    # Check if the value is an integer
    if not isinstance(value, int):
        return "Invalid Input"
    # Check if the value is zero
    if value == 0:
        return "Zero"
    # Check if the value is even or odd
    elif value % 2 == 0:
        return "Even"
    else:
        return "Odd"
# 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

```

Output:



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")==True
assertis_palindrome("AmanaplanacanalPanama")==True assert is_palindrome("Python") == False
assert is_palindrome("") == True
assertis_palindrome("a")==True

Code:

```python
# for task2
#
# 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
def is_palindrome(s):
    # Remove spaces and convert to lowercase
    cleaned_s = s.replace(" ", "").lower()
    # Check if the cleaned string is equal to its reverse
    return cleaned_s == cleaned_s[::-1]
# Test Cases
print(is_palindrome("Madam"))  # True
print(is_palindrome("A man a plan a canal Panama"))  # True
print(is_palindrome("Python"))  # False
print(is_palindrome(""))  # True
print(is_palindrome("a"))  # True
print(is_palindrome("No 'x' in Nixon"))  # True
```

Output:

```
False
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Desktop/assisantcoding - Copy/assignment08.py"
True
True
False
True
True
False
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

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:

```python
# 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

class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds. Withdrawal denied.")
        elif amount <= 0:
            print("Withdrawal amount must be positive.")
        else:
            self.balance -= amount
```

```
    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds. Withdrawal denied.")
        elif amount <= 0:
            print("Withdrawal amount must be positive.")
        else:
            self.balance -= amount

    def get_balance(self):
        return self.balance
# Test Cases
acc = BankAccount(1000)
acc.deposit(500)
print(acc.get_balance())  # 1500
acc.withdraw(300)
print(acc.get_balance())  # 1200
acc.withdraw(2000)
print(acc.get_balance())  # 1200 (unchanged due to insufficient funds)
```

Output:

```
True
False
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/Ap
pData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/aksha/OneDrive/Des
ktop/assisantcoding - Copy/assignment08.py"
1500
1200
Insufficient funds. Withdrawal denied.
1200
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

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

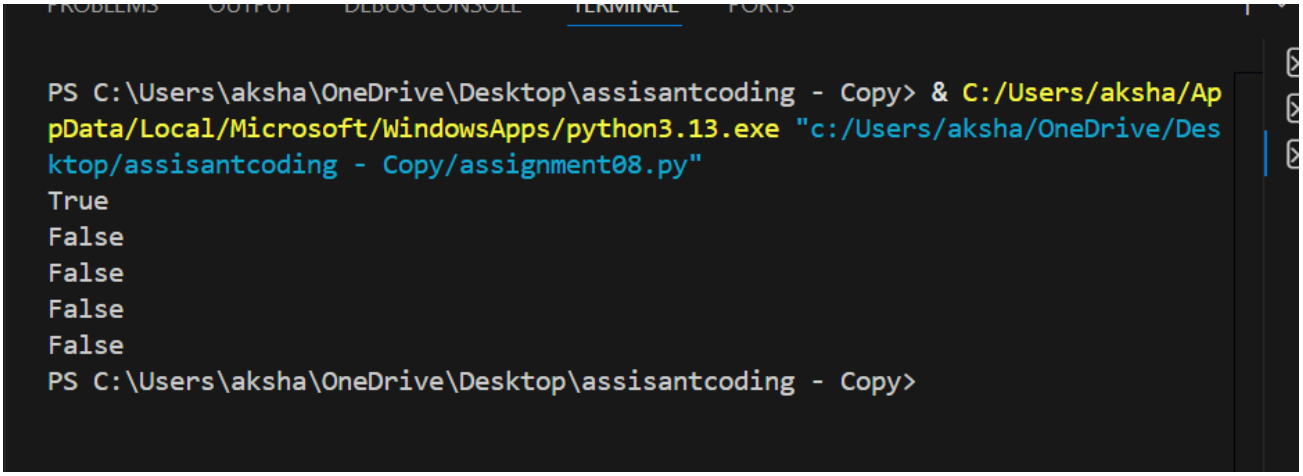
Code:

assignment08.py > ...

```python
# 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
def validate_email(email):
    # Check if the email contains exactly one '@' symbol
    if email.count('@') != 1:
        return False
    local_part, domain_part = email.split('@')
    # Check if the local part is not empty
    if not local_part:
        return False
    # Check if the domain part contains at least one '.' symbol
    if '.' not in domain_part:
        return False
    # Check if the domain part has a valid format (e.g., "example.com")
    domain_name, extension = domain_part.rsplit('.', 1)
    if not domain_name or not extension:
        return False
    return True
# 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
```

Ln 142, Col 49 Spaces: 4 UTF-8 CRLF

Output:



Observation:

AI test cases guided the validation rules for email format.The function correctly checked for required symbols and invalid formats . Edgecases such as missing symbols and improper placement were handled effectively, improving data validation reliability