SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

ProgramName:B. Tech	Assignment Type: Lab	AcademicYear:2025-2026

CourseCoordinatorName	Venkataramana Veeramsetty		
Instructor(s)Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
CourseCode	24CS002PC215	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week4 - Wednesday	Time(s)	
Duration	2 Hours	Applicableto Batches	
AssignmentNumber:8.3(Present assignment number)/24(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	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.	Week4 - Wednesday

- 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.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

Task Description#1

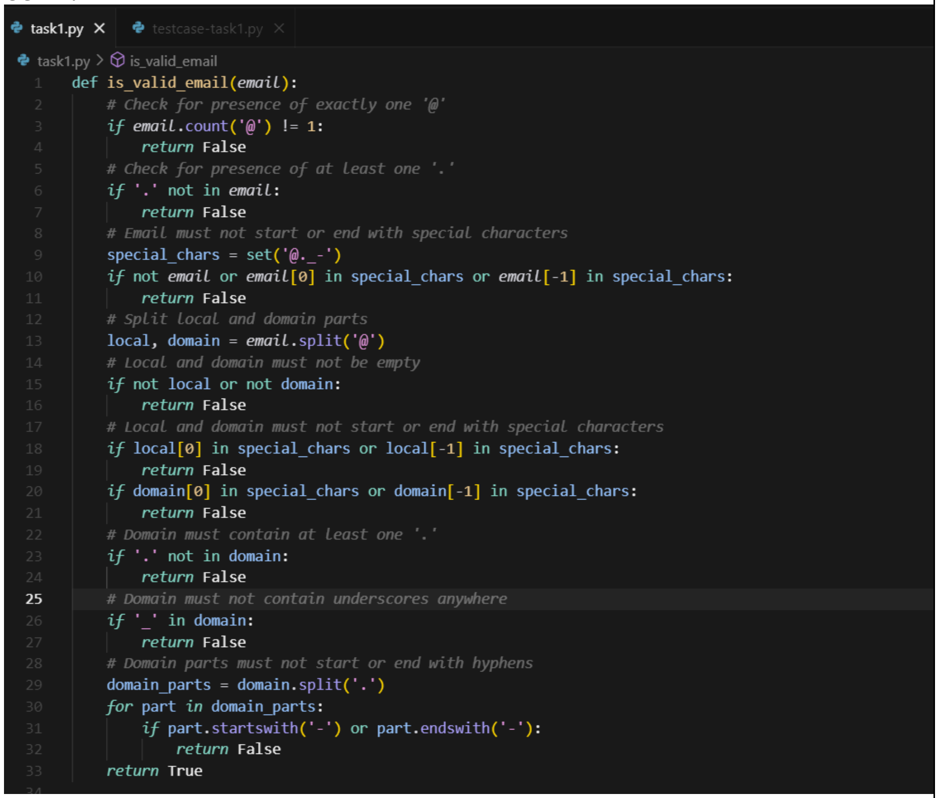
Use AI to generate test cases for is_valid_email(email) and then implement the validator function.

Requirements:

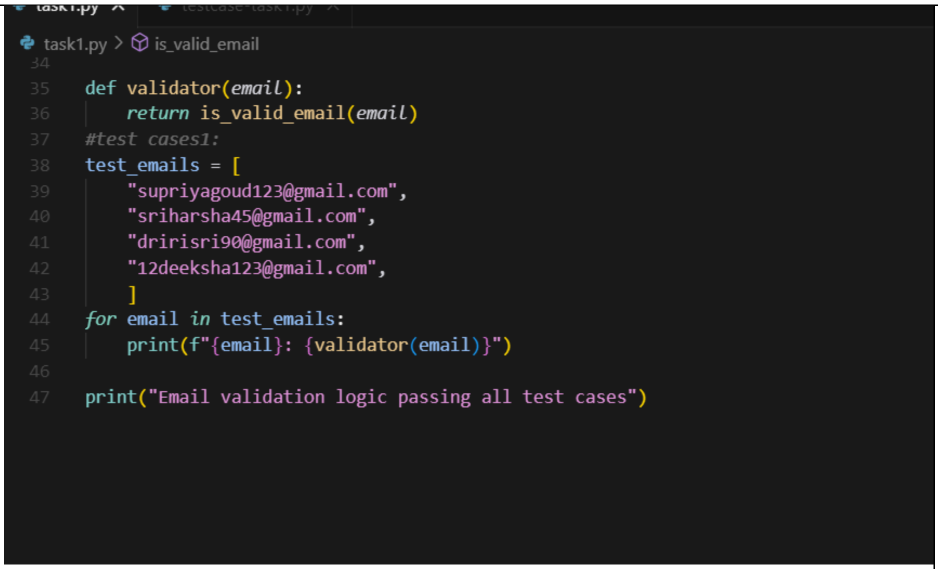
- Must contain @ and . characters.
- Must not start or end with special characters.
- Should not allow multiple @.

PROMT: *Generate a python code for is valid_email(email) and then implement the validator function. And it Must contain @ and . characters.Must not start or end with special characters.Should not allow multiple @.expected output Email validation logic passing all test cases.*

CODE:



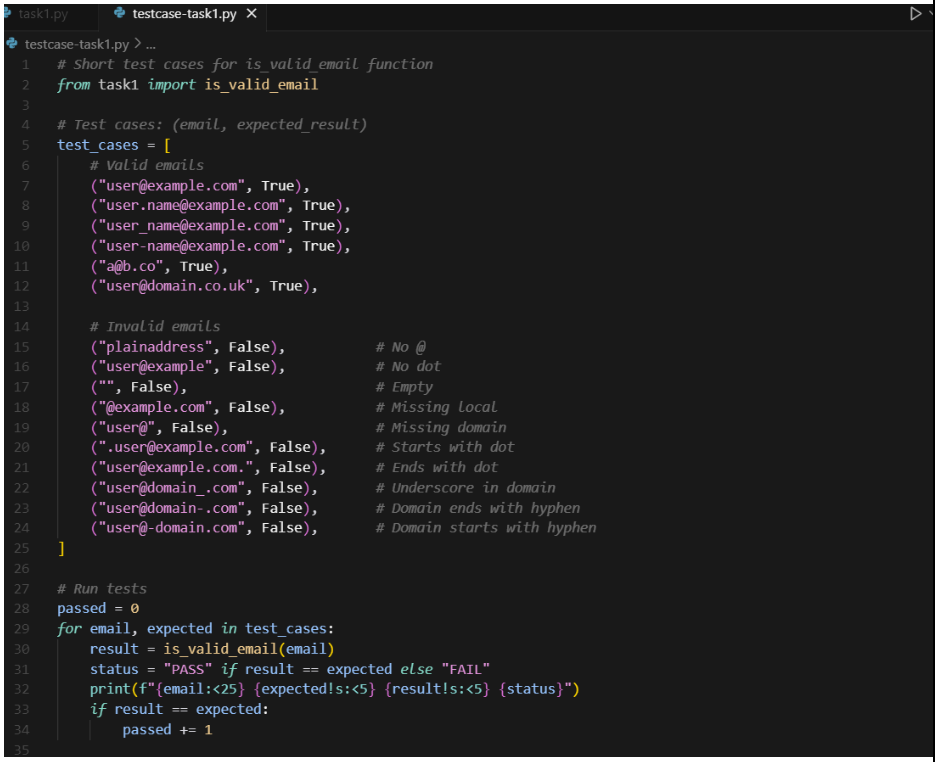
```python
def is_valid_email(email):
    # Check for presence of exactly one '@'
    if email.count('@') != 1:
        return False
    # Check for presence of at least one '.'
    if '.' not in email:
        return False
    # Email must not start or end with special characters
    special_chars = set('@._-')
    if not email or email[0] in special_chars or email[-1] in special_chars:
        return False
    # Split local and domain parts
    local, domain = email.split('@')
    # Local and domain must not be empty
    if not local or not domain:
        return False
    # Local and domain must not start or end with special characters
    if local[0] in special_chars or local[-1] in special_chars:
        return False
    if domain[0] in special_chars or domain[-1] in special_chars:
        return False
    # Domain must contain at least one '.'
    if '.' not in domain:
        return False
    # Domain must not contain underscores anywhere
    if '_' in domain:
        return False
    # Domain parts must not start or end with hyphens
    domain_parts = domain.split('.')
    for part in domain_parts:
        if part.startswith('-') or part.endswith('-'):
            return False
    return True
```



task1.py > is_valid_email

```python
def validator(email):
    return is_valid_email(email)
#test cases1:
test_emails = [
    "supriyagoud123@gmail.com",
    "sriharsha45@gmail.com",
    "dririsri90@gmail.com",
    "12deeksha123@gmail.com",
    ]
for email in test_emails:
    print(f"{email}: {validator(email)}")

print("Email validation logic passing all test cases")
```

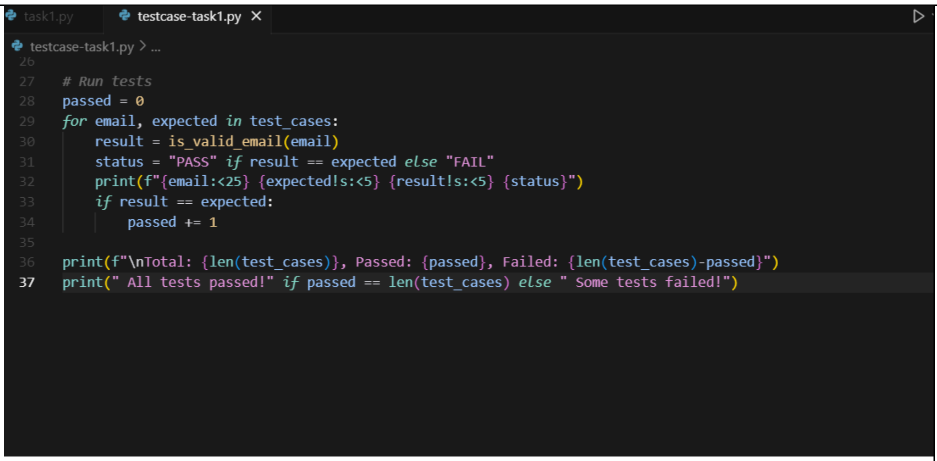
TEST-CASES:



task1.py testcase-task1.py

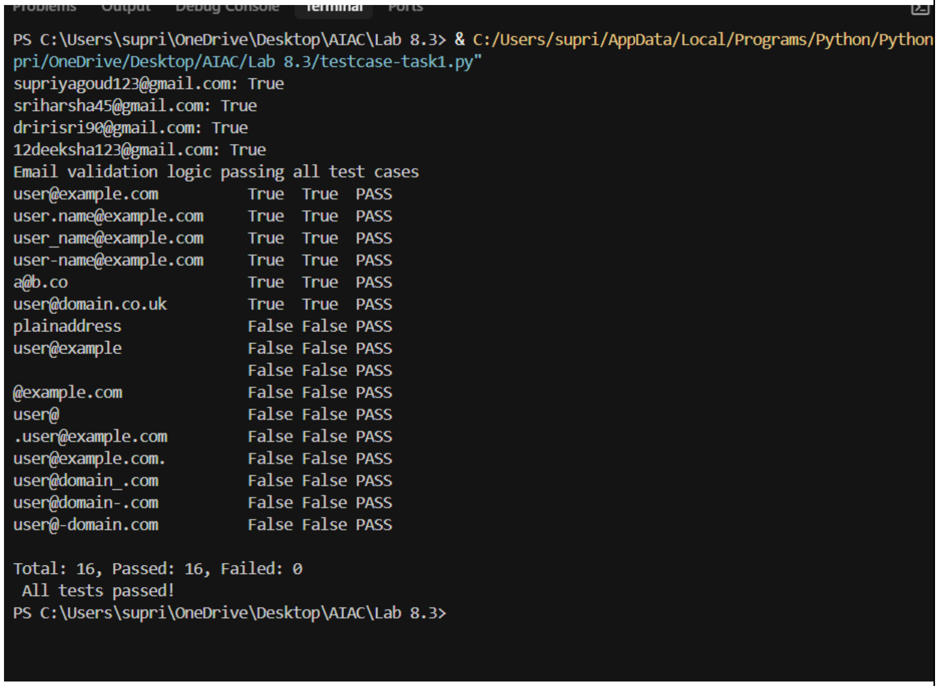
testcase-task1.py > ...

```python
# Short test cases for is_valid_email function
from task1 import is_valid_email

# Test cases: (email, expected_result)
test_cases = [
    # Valid emails
    ("user@example.com", True),
    ("user.name@example.com", True),
    ("user_name@example.com", True),
    ("user-name@example.com", True),
    ("a@b.co", True),
    ("user@domain.co.uk", True),

    # Invalid emails
    ("plainaddress", False),           # No @
    ("user@example", False),           # No dot
    ("", False),                       # Empty
    ("@example.com", False),           # Missing local
    ("user@", False),                  # Missing domain
    (".user@example.com", False),      # Starts with dot
    ("user@example.com.", False),      # Ends with dot
    ("user@domain_.com", False),       # Underscore in domain
    ("user@domain-.com", False),       # Domain ends with hyphen
    ("user@-domain.com", False),       # Domain starts with hyphen
]

# Run tests
passed = 0
for email, expected in test_cases:
    result = is_valid_email(email)
    status = "PASS" if result == expected else "FAIL"
    print(f"{email:<25} {expected!s:<5} {result!s:<5} {status}")
    if result == expected:
        passed += 1
```



```python
# Run tests
passed = 0
for email, expected in test_cases:
    result = is_valid_email(email)
    status = "PASS" if result == expected else "FAIL"
    print(f"{email:<25} {expected!s:<5} {result!s:<5} {status}")
    if result == expected:
        passed += 1

print(f"\nTotal: {len(test_cases)}, Passed: {passed}, Failed: {len(test_cases)-passed}")
print(" All tests passed!" if passed == len(test_cases) else " Some tests failed!")
```

OUTPUT:



```
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> & C:/Users/supri/AppData/Local/Programs/Python/Python
pri/OneDrive/Desktop/AIAC/Lab 8.3/testcase-task1.py"
supriyagoud123@gmail.com: True
sriharsha45@gmail.com: True
dririsri90@gmail.com: True
12deeksha123@gmail.com: True
Email validation logic passing all test cases
user@example.com          True  True  PASS
user.name@example.com     True  True  PASS
user_name@example.com     True  True  PASS
user-name@example.com     True  True  PASS
a@b.co                    True  True  PASS
user@domain.co.uk         True  True  PASS
plainaddress              False False PASS
user@example              False False PASS
                          False False PASS
@example.com              False False PASS
user@                     False False PASS
.user@example.com         False False PASS
user@example.com.         False False PASS
user@domain_.com          False False PASS
user@domain-.com          False False PASS
user@-domain.com          False False PASS

Total: 16, Passed: 16, Failed: 0
 All tests passed!
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3>
```

Expected Output#1

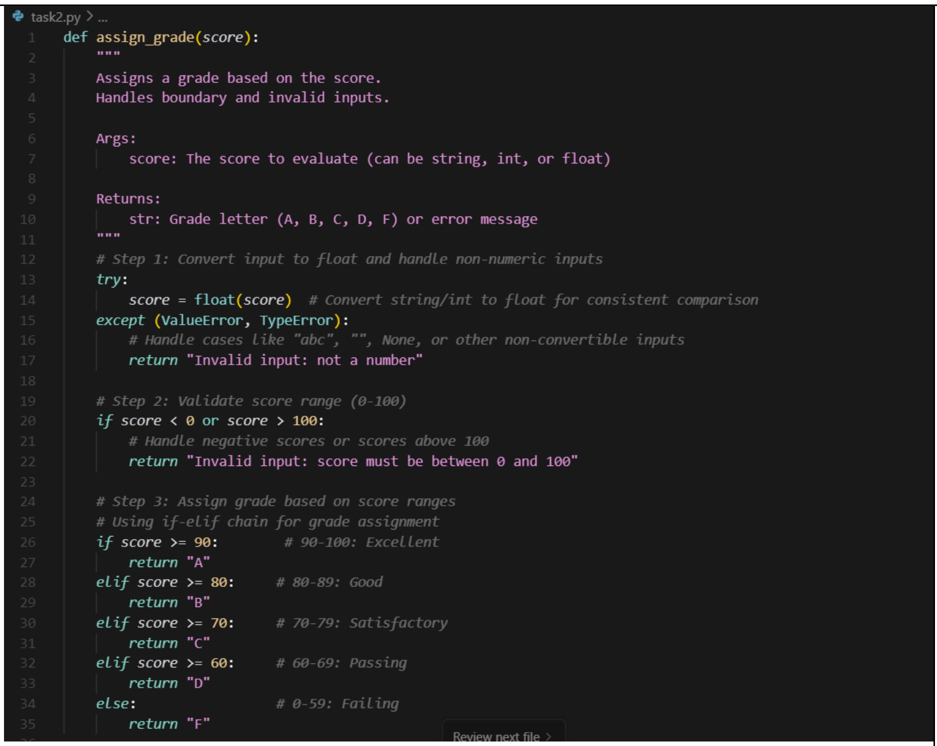
- Email validation logic passing all test cases

Task Description#2 (Loops)

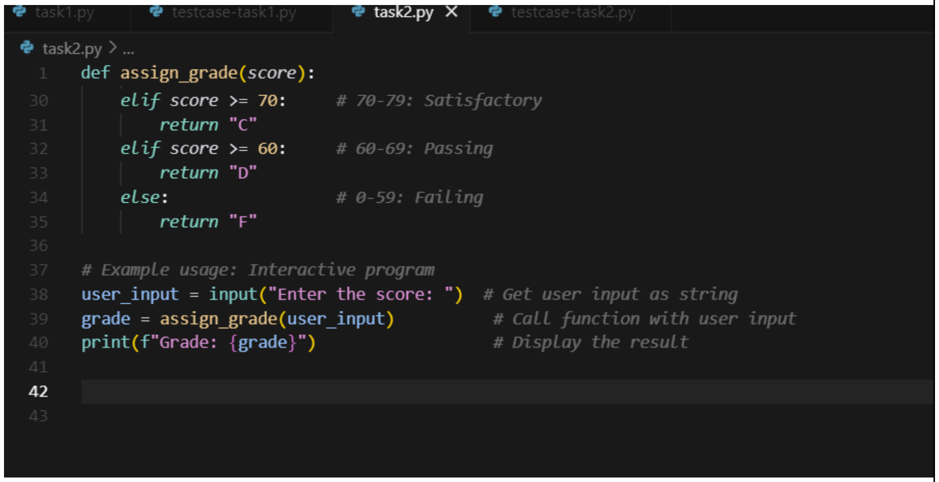
- Ask AI to generate test cases for assign_grade(score) function. Handle boundary and invalid inputs.
  Requirements
- AI should generate test cases for assign_grade(score) where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

PROMT: *Generate a pyhton code for assign_grade(score) .function Handle boundary and invalid inputs.* the input values given by the user.

CODE:

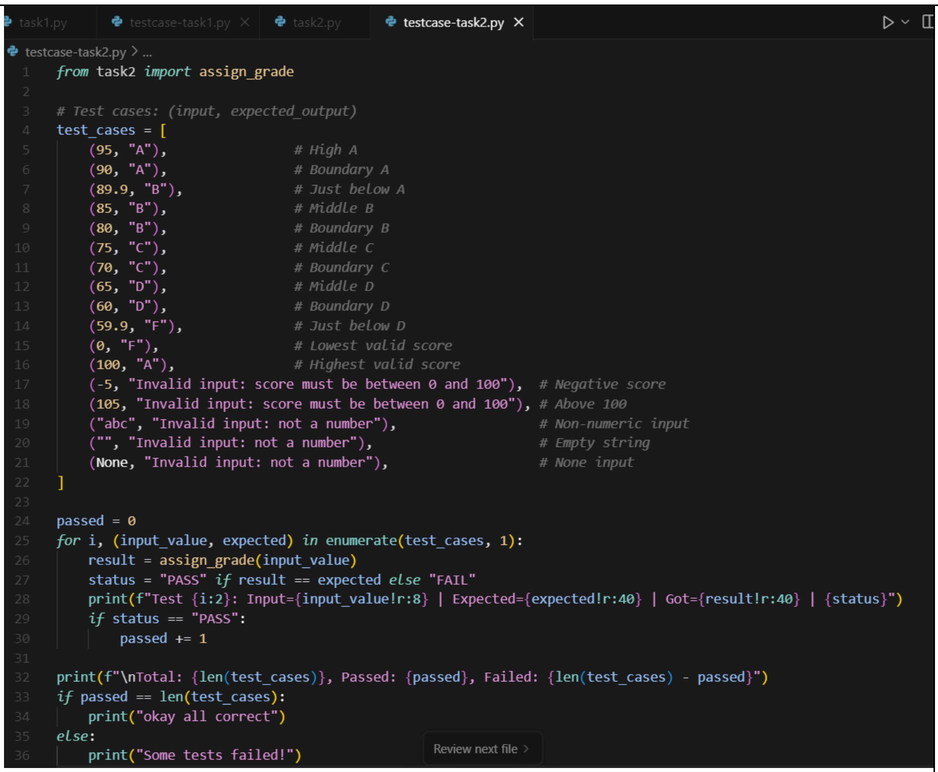


```
task2.py > ...
def assign_grade(score):
    """
    Assigns a grade based on the score.
    Handles boundary and invalid inputs.

    Args:
        score: The score to evaluate (can be string, int, or float)

    Returns:
        str: Grade letter (A, B, C, D, F) or error message
    """
    # Step 1: Convert input to float and handle non-numeric inputs
    try:
        score = float(score)  # Convert string/int to float for consistent comparison
    except (ValueError, TypeError):
        # Handle cases like "abc", "", None, or other non-convertible inputs
        return "Invalid input: not a number"

    # Step 2: Validate score range (0-100)
    if score < 0 or score > 100:
        # Handle negative scores or scores above 100
        return "Invalid input: score must be between 0 and 100"

    # Step 3: Assign grade based on score ranges
    # Using if-elif chain for grade assignment
    if score >= 90:        # 90-100: Excellent
        return "A"
    elif score >= 80:      # 80-89: Good
        return "B"
    elif score >= 70:      # 70-79: Satisfactory
        return "C"
    elif score >= 60:      # 60-69: Passing
        return "D"
    else:                  # 0-59: Failing
        return "F"
```



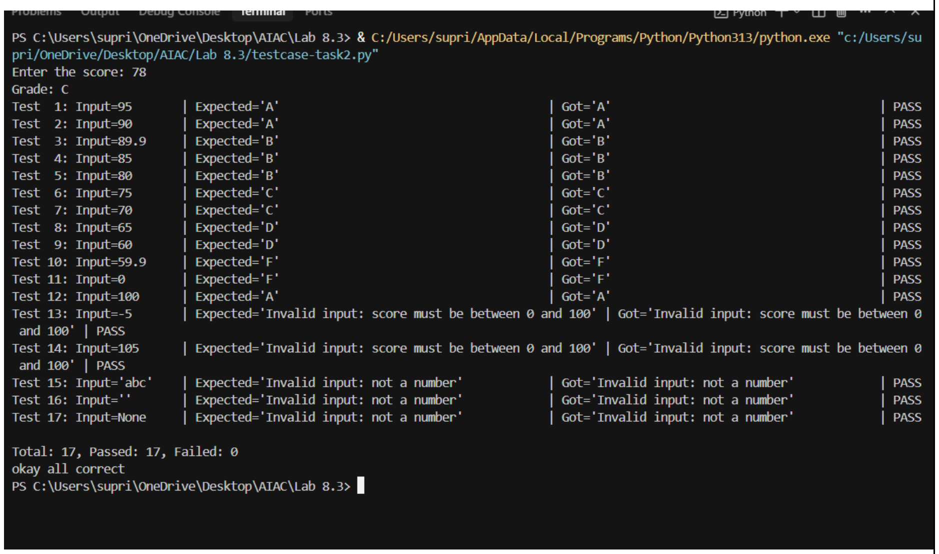
```
task1.py   testcase-task1.py   task2.py ×   testcase-task2.py
task2.py > ...
def assign_grade(score):
    elif score >= 70:      # 70-79: Satisfactory
        return "C"
    elif score >= 60:      # 60-69: Passing
        return "D"
    else:                  # 0-59: Failing
        return "F"

# Example usage: Interactive program
user_input = input("Enter the score: ")  # Get user input as string
grade = assign_grade(user_input)           # Call function with user input
print(f"Grade: {grade}")                   # Display the result
```

TEST-CASES:



```python
from task2 import assign_grade

# Test cases: (input, expected_output)
test_cases = [
    (95, "A"),                # High A
    (90, "A"),                # Boundary A
    (89.9, "B"),              # Just below A
    (85, "B"),                # Middle B
    (80, "B"),                # Boundary B
    (75, "C"),                # Middle C
    (70, "C"),                # Boundary C
    (65, "D"),                # Middle D
    (60, "D"),                # Boundary D
    (59.9, "F"),              # Just below D
    (0, "F"),                 # Lowest valid score
    (100, "A"),               # Highest valid score
    (-5, "Invalid input: score must be between 0 and 100"),  # Negative score
    (105, "Invalid input: score must be between 0 and 100"), # Above 100
    ("abc", "Invalid input: not a number"),                  # Non-numeric input
    ("", "Invalid input: not a number"),                     # Empty string
    (None, "Invalid input: not a number"),                   # None input
]

passed = 0
for i, (input_value, expected) in enumerate(test_cases, 1):
    result = assign_grade(input_value)
    status = "PASS" if result == expected else "FAIL"
    print(f"Test {i:2}: Input={input_value!r:8} | Expected={expected!r:40} | Got={result!r:40} | {status}")
    if status == "PASS":
        passed += 1

print(f"\nTotal: {len(test_cases)}, Passed: {passed}, Failed: {len(test_cases) - passed}")
if passed == len(test_cases):
    print("okay all correct")
else:
    print("Some tests failed!")
```

OUTPUT:



```
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/su
pri/OneDrive/Desktop/AIAC/Lab 8.3/testcase-task2.py"
Enter the score: 78
Grade: C
Test  1: Input=95       | Expected='A'                                      | Got='A'                                      | PASS
Test  2: Input=90       | Expected='A'                                      | Got='A'                                      | PASS
Test  3: Input=89.9     | Expected='B'                                      | Got='B'                                      | PASS
Test  4: Input=85       | Expected='B'                                      | Got='B'                                      | PASS
Test  5: Input=80       | Expected='B'                                      | Got='B'                                      | PASS
Test  6: Input=75       | Expected='C'                                      | Got='C'                                      | PASS
Test  7: Input=70       | Expected='C'                                      | Got='C'                                      | PASS
Test  8: Input=65       | Expected='D'                                      | Got='D'                                      | PASS
Test  9: Input=60       | Expected='D'                                      | Got='D'                                      | PASS
Test 10: Input=59.9     | Expected='F'                                      | Got='F'                                      | PASS
Test 11: Input=0        | Expected='F'                                      | Got='F'                                      | PASS
Test 12: Input=100      | Expected='A'                                      | Got='A'                                      | PASS
Test 13: Input=-5       | Expected='Invalid input: score must be between 0 and 100' | Got='Invalid input: score must be between 0
 and 100' | PASS
Test 14: Input=105      | Expected='Invalid input: score must be between 0 and 100' | Got='Invalid input: score must be between 0
 and 100' | PASS
Test 15: Input='abc'    | Expected='Invalid input: not a number'            | Got='Invalid input: not a number'            | PASS
Test 16: Input=''       | Expected='Invalid input: not a number'            | Got='Invalid input: not a number'            | PASS
Test 17: Input=None     | Expected='Invalid input: not a number'            | Got='Invalid input: not a number'            | PASS

Total: 17, Passed: 17, Failed: 0
okay all correct
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> 
```

Expected Output#2

Grade assignment function passing test suite

Task Description#3

- Generate test cases using AI for is_sentence_palindrome(sentence). Ignore case, punctuation, and spaces

Requirement

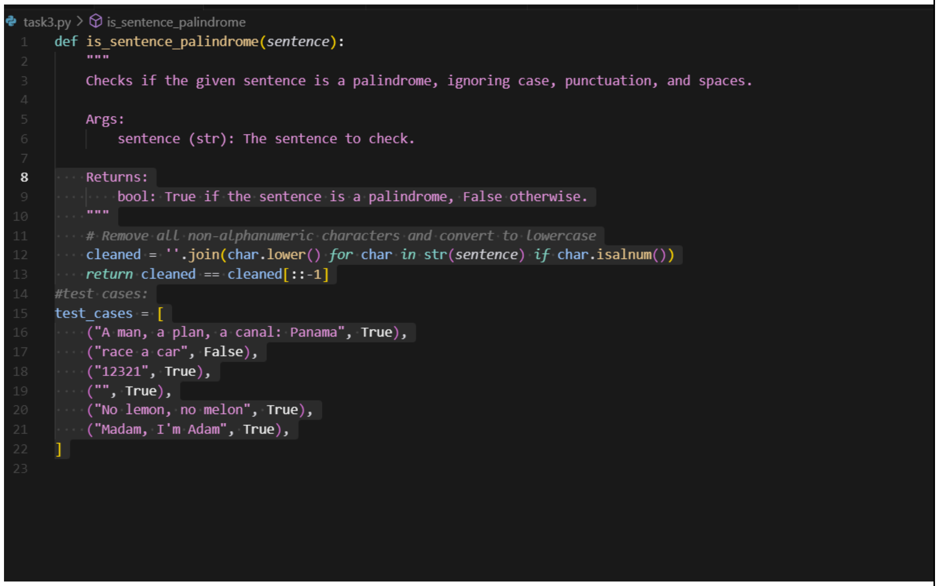
- Ask AI to create test cases for is_sentence_palindrome(sentence) (ignores case, spaces, and punctuation).

- Example:

"A man a plan a canal Panama" → True

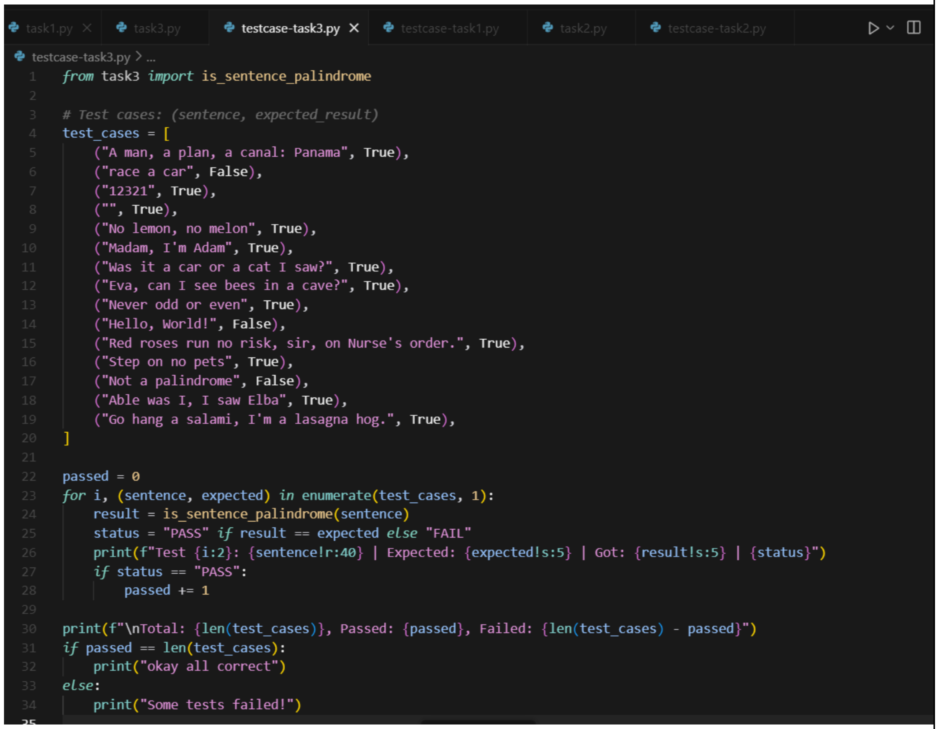
- ***PROMT:*** *Generate a python function for is_sentence_palindrome(sentence).Ignore case, punctuation, and spaces and the input values given by the user.*

CODE:



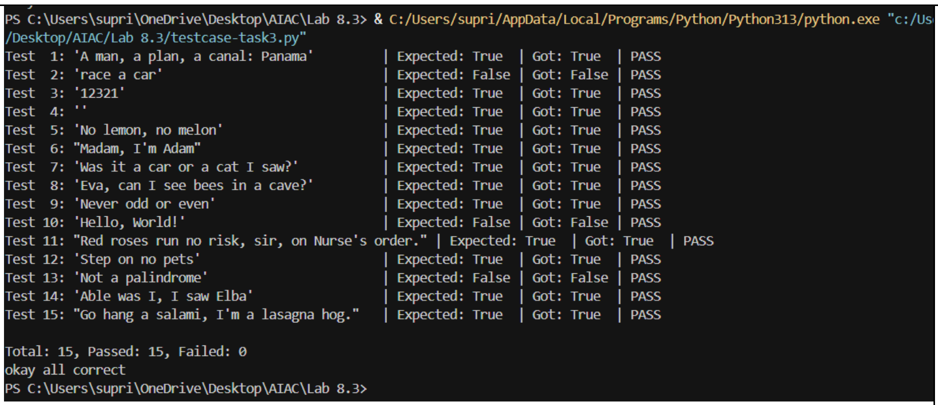
```python
def is_sentence_palindrome(sentence):
    """
    Checks if the given sentence is a palindrome, ignoring case, punctuation, and spaces.

    Args:
        sentence (str): The sentence to check.

    Returns:
        bool: True if the sentence is a palindrome, False otherwise.
    """
    # Remove all non-alphanumeric characters and convert to lowercase
    cleaned = ''.join(char.lower() for char in str(sentence) if char.isalnum())
    return cleaned == cleaned[::-1]
#test cases:
test_cases = [
    ("A man, a plan, a canal: Panama", True),
    ("race a car", False),
    ("12321", True),
    ("", True),
    ("No lemon, no melon", True),
    ("Madam, I'm Adam", True),
]
```

TEST-CASES:



```python
from task3 import is_sentence_palindrome

# Test cases: (sentence, expected_result)
test_cases = [
    ("A man, a plan, a canal: Panama", True),
    ("race a car", False),
    ("12321", True),
    ("", True),
    ("No lemon, no melon", True),
    ("Madam, I'm Adam", True),
    ("Was it a car or a cat I saw?", True),
    ("Eva, can I see bees in a cave?", True),
    ("Never odd or even", True),
    ("Hello, World!", False),
    ("Red roses run no risk, sir, on Nurse's order.", True),
    ("Step on no pets", True),
    ("Not a palindrome", False),
    ("Able was I, I saw Elba", True),
    ("Go hang a salami, I'm a lasagna hog.", True),
]

passed = 0
for i, (sentence, expected) in enumerate(test_cases, 1):
    result = is_sentence_palindrome(sentence)
    status = "PASS" if result == expected else "FAIL"
    print(f"Test {i:2}: {sentence!r:40} | Expected: {expected!s:5} | Got: {result!s:5} | {status}")
    if status == "PASS":
        passed += 1

print(f"\nTotal: {len(test_cases)}, Passed: {passed}, Failed: {len(test_cases) - passed}")
if passed == len(test_cases):
    print("okay all correct")
else:
    print("Some tests failed!")
```

OUTPUT:


```
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe "c:/Us
/Desktop/AIAC/Lab 8.3/testcase-task3.py"
Test  1: 'A man, a plan, a canal: Panama'         | Expected: True  | Got: True  | PASS
Test  2: 'race a car'                             | Expected: False | Got: False | PASS
Test  3: '12321'                                  | Expected: True  | Got: True  | PASS
Test  4: ''                                       | Expected: True  | Got: True  | PASS
Test  5: 'No lemon, no melon'                     | Expected: True  | Got: True  | PASS
Test  6: "Madam, I'm Adam"                        | Expected: True  | Got: True  | PASS
Test  7: 'Was it a car or a cat I saw?'           | Expected: True  | Got: True  | PASS
Test  8: 'Eva, can I see bees in a cave?'         | Expected: True  | Got: True  | PASS
Test  9: 'Never odd or even'                      | Expected: True  | Got: True  | PASS
Test 10: 'Hello, World!'                          | Expected: False | Got: False | PASS
Test 11: "Red roses run no risk, sir, on Nurse's order." | Expected: True  | Got: True  | PASS
Test 12: 'Step on no pets'                        | Expected: True  | Got: True  | PASS
Test 13: 'Not a palindrome'                       | Expected: False | Got: False | PASS
Test 14: 'Able was I, I saw Elba'                 | Expected: True  | Got: True  | PASS
Test 15: "Go hang a salami, I'm a lasagna hog."   | Expected: True  | Got: True  | PASS

Total: 15, Passed: 15, Failed: 0
okay all correct
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3>
```

Expected Output#3

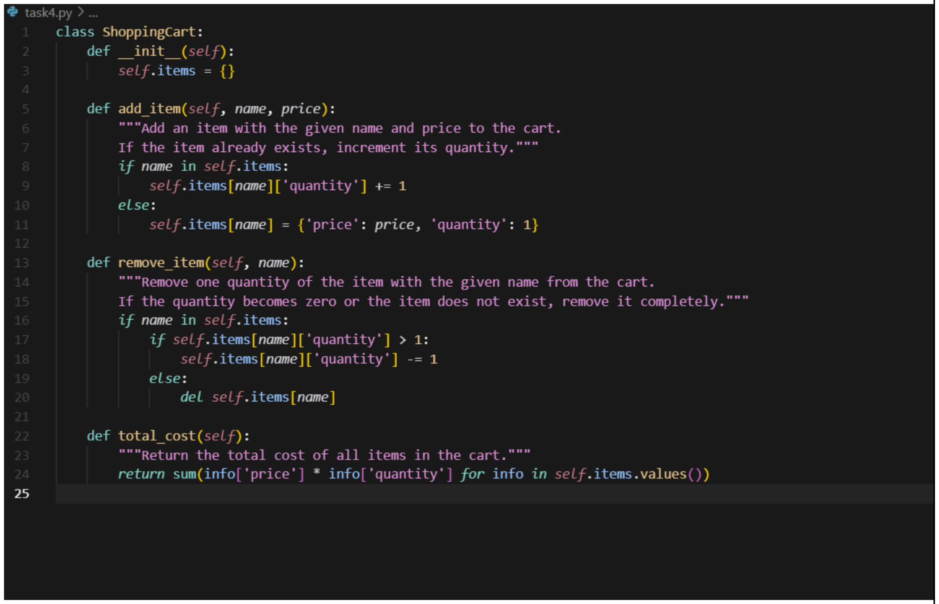
- Function returns True/False for cleaned sentences
- Implement the function to pass AI-generated tests.

Task Description#4

- Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

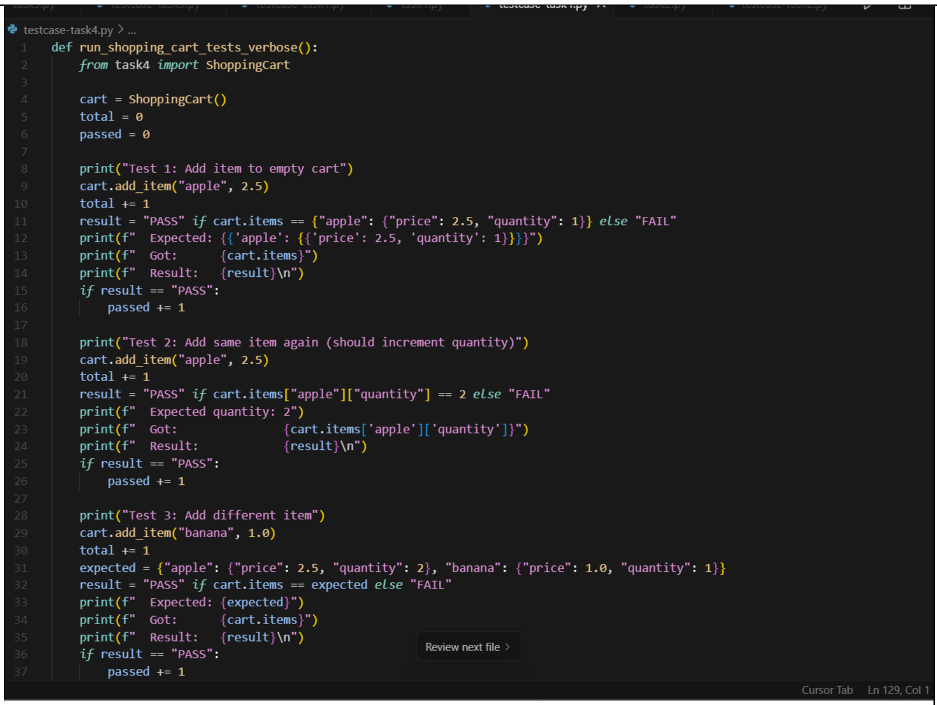
  Methods:
  Add_item(name,orice)
  Remove_item(name)
  Total_cost()

PROMT: *Generate a python function for a ShoppingCart class (add_item, remove_item, total_cost).the methods are Add_item(name,orice)Remove_item(name)Total_cost().*

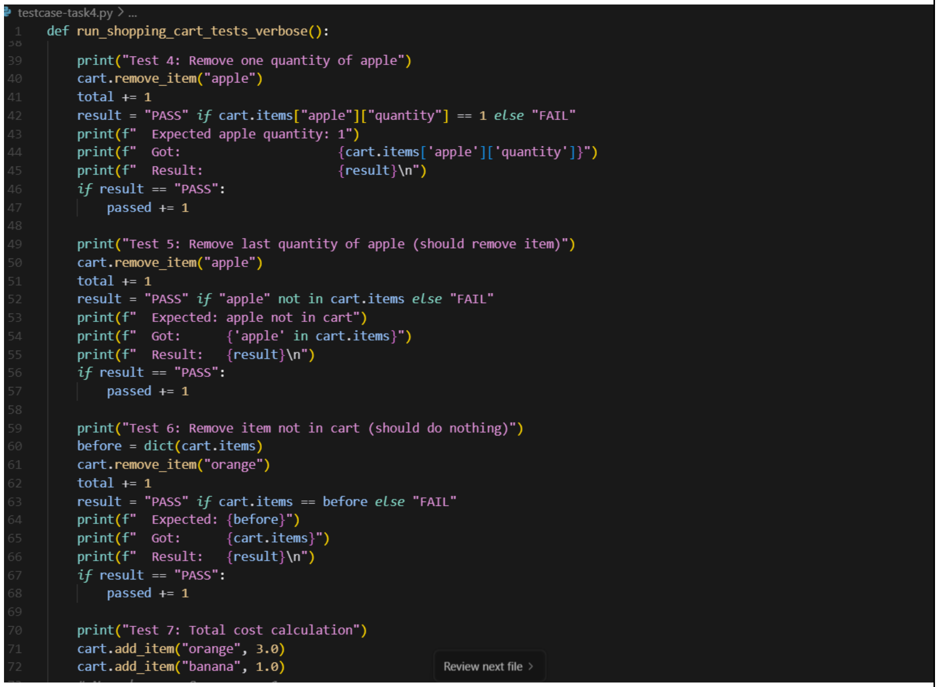


```
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        """Add an item with the given name and price to the cart.
        If the item already exists, increment its quantity."""
        if name in self.items:
            self.items[name]['quantity'] += 1
        else:
            self.items[name] = {'price': price, 'quantity': 1}

    def remove_item(self, name):
        """Remove one quantity of the item with the given name from the cart.
        If the quantity becomes zero or the item does not exist, remove it completely."""
        if name in self.items:
            if self.items[name]['quantity'] > 1:
                self.items[name]['quantity'] -= 1
            else:
                del self.items[name]

    def total_cost(self):
        """Return the total cost of all items in the cart."""
        return sum(info['price'] * info['quantity'] for info in self.items.values())
```

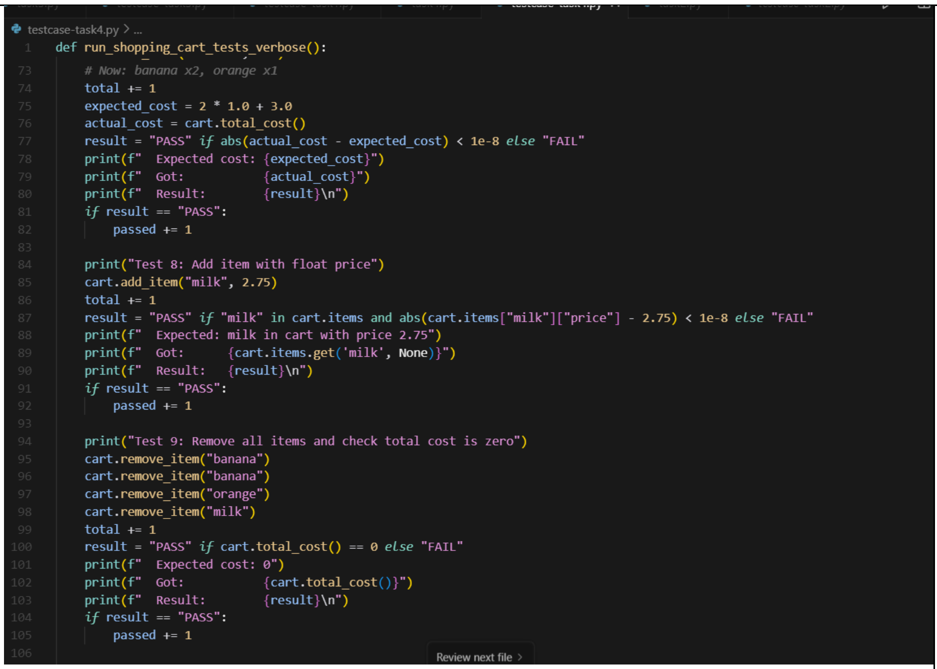
TEST-CASES*:*



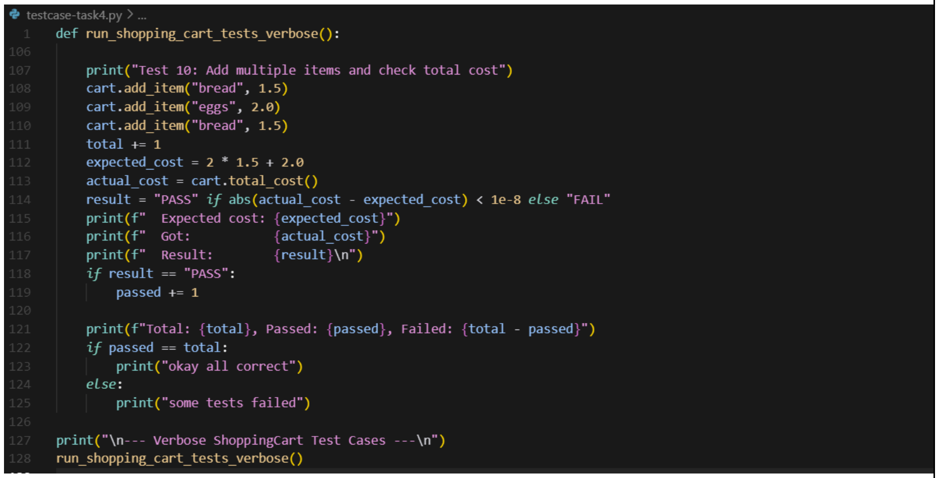
```python
def run_shopping_cart_tests_verbose():
    from task4 import ShoppingCart

    cart = ShoppingCart()
    total = 0
    passed = 0

    print("Test 1: Add item to empty cart")
    cart.add_item("apple", 2.5)
    total += 1
    result = "PASS" if cart.items == {"apple": {"price": 2.5, "quantity": 1}} else "FAIL"
    print(f"  Expected: {{'apple': {{'price': 2.5, 'quantity': 1}}}}")
    print(f"  Got:      {cart.items}")
    print(f"  Result:   {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 2: Add same item again (should increment quantity)")
    cart.add_item("apple", 2.5)
    total += 1
    result = "PASS" if cart.items["apple"]["quantity"] == 2 else "FAIL"
    print(f"  Expected quantity: 2")
    print(f"  Got:               {cart.items['apple']['quantity']}")
    print(f"  Result:            {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 3: Add different item")
    cart.add_item("banana", 1.0)
    total += 1
    expected = {"apple": {"price": 2.5, "quantity": 2}, "banana": {"price": 1.0, "quantity": 1}}
    result = "PASS" if cart.items == expected else "FAIL"
    print(f"  Expected: {expected}")
    print(f"  Got:      {cart.items}")
    print(f"  Result:   {result}\n")
    if result == "PASS":
        passed += 1
```



```python
def run_shopping_cart_tests_verbose():

    print("Test 4: Remove one quantity of apple")
    cart.remove_item("apple")
    total += 1
    result = "PASS" if cart.items["apple"]["quantity"] == 1 else "FAIL"
    print(f"  Expected apple quantity: 1")
    print(f"  Got:                     {cart.items['apple']['quantity']}")
    print(f"  Result:                  {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 5: Remove last quantity of apple (should remove item)")
    cart.remove_item("apple")
    total += 1
    result = "PASS" if "apple" not in cart.items else "FAIL"
    print(f"  Expected: apple not in cart")
    print(f"  Got:      {'apple' in cart.items}")
    print(f"  Result:   {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 6: Remove item not in cart (should do nothing)")
    before = dict(cart.items)
    cart.remove_item("orange")
    total += 1
    result = "PASS" if cart.items == before else "FAIL"
    print(f"  Expected: {before}")
    print(f"  Got:      {cart.items}")
    print(f"  Result:   {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 7: Total cost calculation")
    cart.add_item("orange", 3.0)
    cart.add_item("banana", 1.0)
```

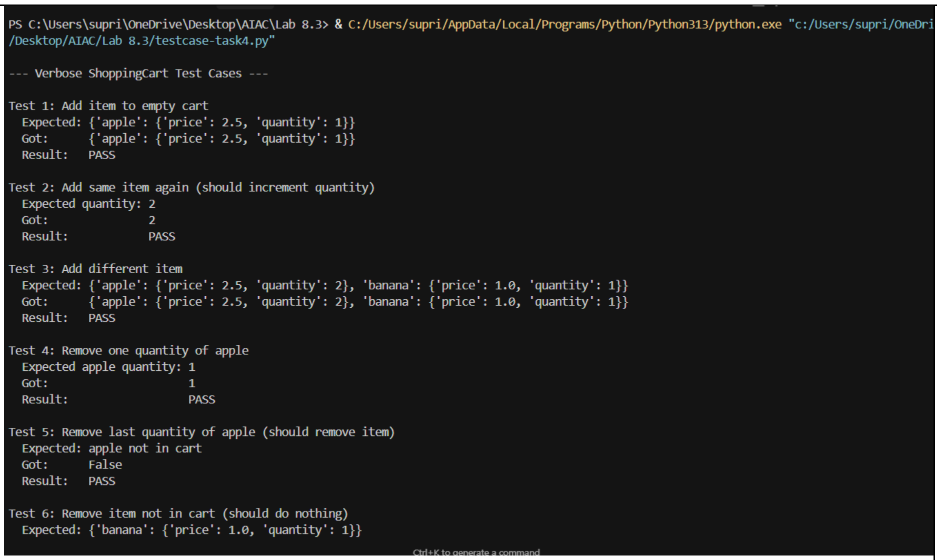


```python
def run_shopping_cart_tests_verbose():
    # Now: banana x2, orange x1
    total += 1
    expected_cost = 2 * 1.0 + 3.0
    actual_cost = cart.total_cost()
    result = "PASS" if abs(actual_cost - expected_cost) < 1e-8 else "FAIL"
    print(f"  Expected cost: {expected_cost}")
    print(f"  Got:           {actual_cost}")
    print(f"  Result:        {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 8: Add item with float price")
    cart.add_item("milk", 2.75)
    total += 1
    result = "PASS" if "milk" in cart.items and abs(cart.items["milk"]["price"] - 2.75) < 1e-8 else "FAIL"
    print(f"  Expected: milk in cart with price 2.75")
    print(f"  Got:      {cart.items.get('milk', None)}")
    print(f"  Result:   {result}\n")
    if result == "PASS":
        passed += 1

    print("Test 9: Remove all items and check total cost is zero")
    cart.remove_item("banana")
    cart.remove_item("banana")
    cart.remove_item("orange")
    cart.remove_item("milk")
    total += 1
    result = "PASS" if cart.total_cost() == 0 else "FAIL"
    print(f"  Expected cost: 0")
    print(f"  Got:           {cart.total_cost()}")
    print(f"  Result:        {result}\n")
    if result == "PASS":
        passed += 1

```

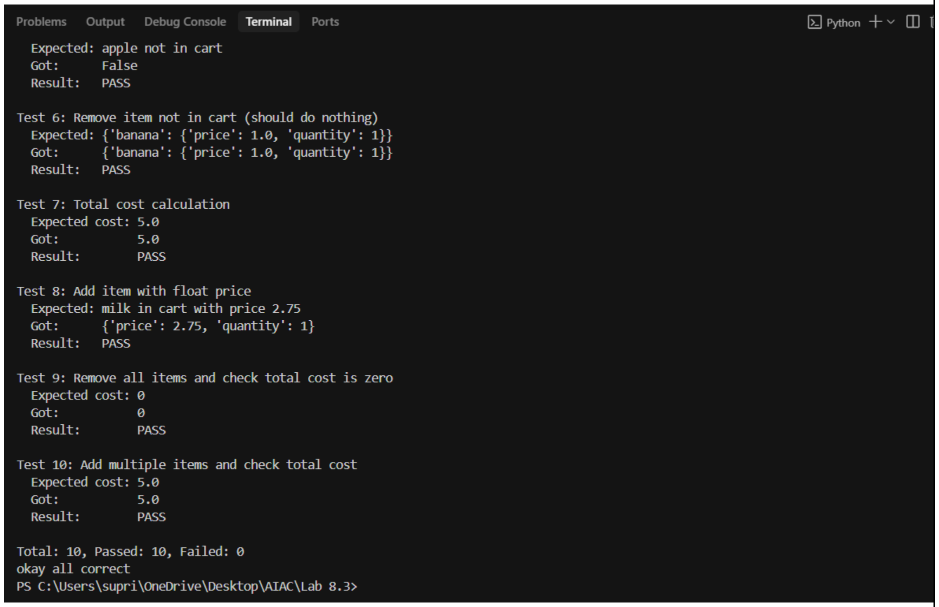


```python
def run_shopping_cart_tests_verbose():

    print("Test 10: Add multiple items and check total cost")
    cart.add_item("bread", 1.5)
    cart.add_item("eggs", 2.0)
    cart.add_item("bread", 1.5)
    total += 1
    expected_cost = 2 * 1.5 + 2.0
    actual_cost = cart.total_cost()
    result = "PASS" if abs(actual_cost - expected_cost) < 1e-8 else "FAIL"
    print(f"  Expected cost: {expected_cost}")
    print(f"  Got:           {actual_cost}")
    print(f"  Result:        {result}\n")
    if result == "PASS":
        passed += 1

    print(f"Total: {total}, Passed: {passed}, Failed: {total - passed}")
    if passed == total:
        print("okay all correct")
    else:
        print("some tests failed")

print("\n--- Verbose ShoppingCart Test Cases ---\n")
run_shopping_cart_tests_verbose()
```

OUTPUT:



```
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/supri/OneDri
/Desktop/AIAC/Lab 8.3/testcase-task4.py"

--- Verbose ShoppingCart Test Cases ---

Test 1: Add item to empty cart
  Expected: {'apple': {'price': 2.5, 'quantity': 1}}
  Got:      {'apple': {'price': 2.5, 'quantity': 1}}
  Result:   PASS

Test 2: Add same item again (should increment quantity)
  Expected quantity: 2
  Got:               2
  Result:            PASS

Test 3: Add different item
  Expected: {'apple': {'price': 2.5, 'quantity': 2}, 'banana': {'price': 1.0, 'quantity': 1}}
  Got:      {'apple': {'price': 2.5, 'quantity': 2}, 'banana': {'price': 1.0, 'quantity': 1}}
  Result:   PASS

Test 4: Remove one quantity of apple
  Expected apple quantity: 1
  Got:                     1
  Result:                  PASS

Test 5: Remove last quantity of apple (should remove item)
  Expected: apple not in cart
  Got:      False
  Result:   PASS

Test 6: Remove item not in cart (should do nothing)
  Expected: {'banana': {'price': 1.0, 'quantity': 1}}
```



```
  Expected: apple not in cart
  Got:      False
  Result:   PASS

Test 6: Remove item not in cart (should do nothing)
  Expected: {'banana': {'price': 1.0, 'quantity': 1}}
  Got:      {'banana': {'price': 1.0, 'quantity': 1}}
  Result:   PASS

Test 7: Total cost calculation
  Expected cost: 5.0
  Got:           5.0
  Result:        PASS

Test 8: Add item with float price
  Expected: milk in cart with price 2.75
  Got:      {'price': 2.75, 'quantity': 1}
  Result:   PASS

Test 9: Remove all items and check total cost is zero
  Expected cost: 0
  Got:           0
  Result:        PASS

Test 10: Add multiple items and check total cost
  Expected cost: 5.0
  Got:           5.0
  Result:        PASS

Total: 10, Passed: 10, Failed: 0
okay all correct
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3>
```

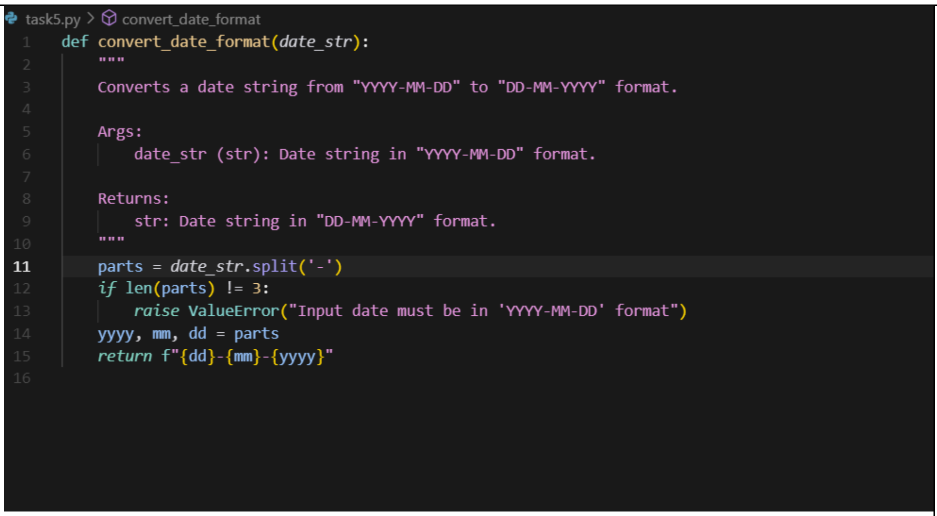
Expected Output#4

- Full class with tested functionalities

Task Description#5

- Use AI to write test cases for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".
  Example: "2023-10-15" → "15-10-2023"

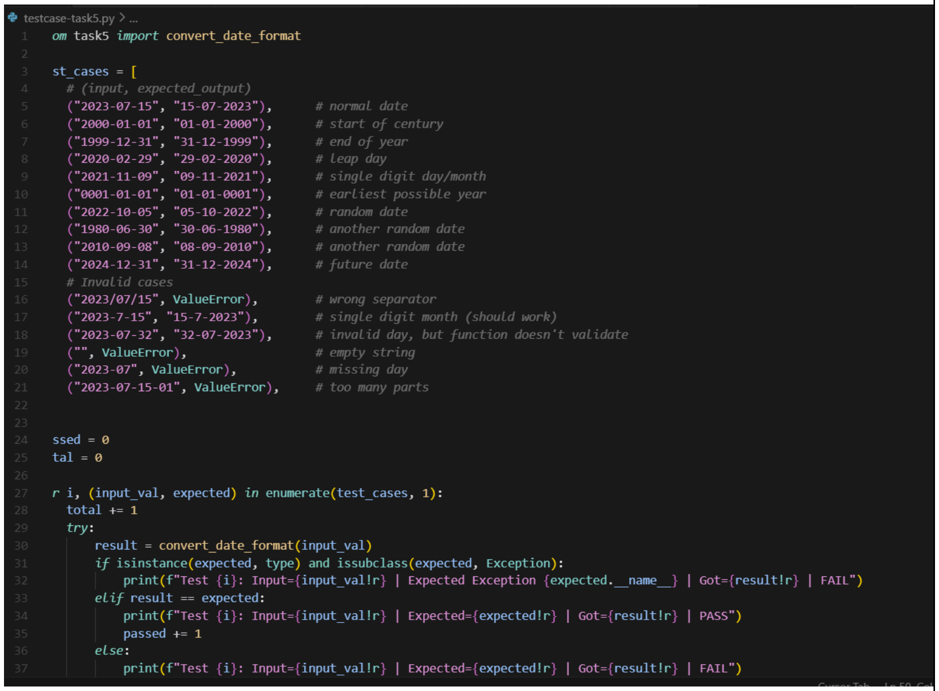
PROMT: *Generate a python function for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".* **For *Example: "2023-10-15" → "15-10-2023"***



task5.py > convert_date_format

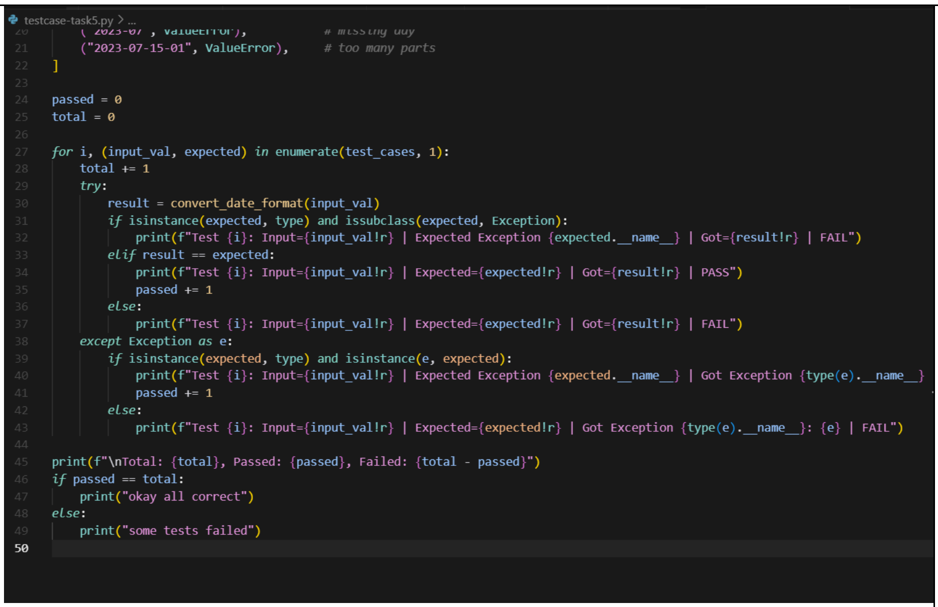
```python
def convert_date_format(date_str):
    """
    Converts a date string from "YYYY-MM-DD" to "DD-MM-YYYY" format.

    Args:
        date_str (str): Date string in "YYYY-MM-DD" format.

    Returns:
        str: Date string in "DD-MM-YYYY" format.
    """
    parts = date_str.split('-')
    if len(parts) != 3:
        raise ValueError("Input date must be in 'YYYY-MM-DD' format")
    yyyy, mm, dd = parts
    return f"{dd}-{mm}-{yyyy}"
```

TEST-CASES:



testcase-task5.py > ...

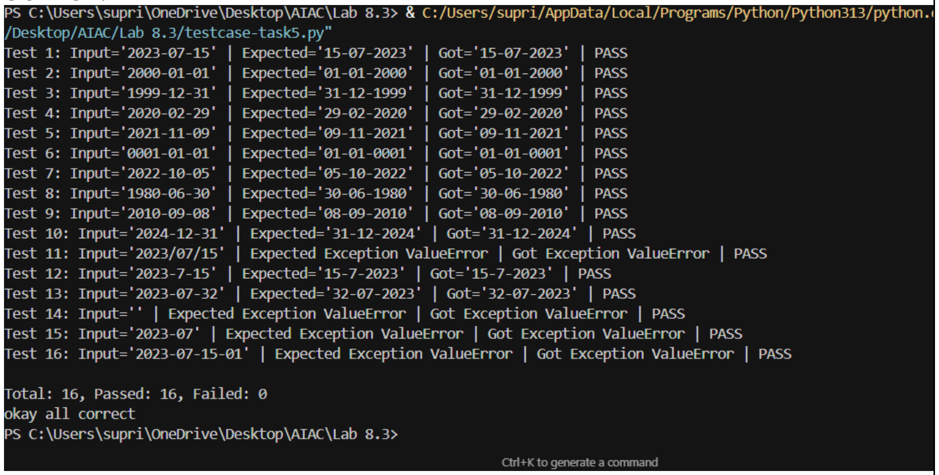
```python
om task5 import convert_date_format

st_cases = [
  # (input, expected_output)
  ("2023-07-15", "15-07-2023"),      # normal date
  ("2000-01-01", "01-01-2000"),      # start of century
  ("1999-12-31", "31-12-1999"),      # end of year
  ("2020-02-29", "29-02-2020"),      # leap day
  ("2021-11-09", "09-11-2021"),      # single digit day/month
  ("0001-01-01", "01-01-0001"),      # earliest possible year
  ("2022-10-05", "05-10-2022"),      # random date
  ("1980-06-30", "30-06-1980"),      # another random date
  ("2010-09-08", "08-09-2010"),      # another random date
  ("2024-12-31", "31-12-2024"),      # future date
  # Invalid cases
  ("2023/07/15", ValueError),        # wrong separator
  ("2023-7-15", "15-7-2023"),        # single digit month (should work)
  ("2023-07-32", "32-07-2023"),      # invalid day, but function doesn't validate
  ("", ValueError),                  # empty string
  ("2023-07", ValueError),           # missing day
  ("2023-07-15-01", ValueError),     # too many parts


ssed = 0
tal = 0

r i, (input_val, expected) in enumerate(test_cases, 1):
  total += 1
  try:
      result = convert_date_format(input_val)
      if isinstance(expected, type) and issubclass(expected, Exception):
          print(f"Test {i}: Input={input_val!r} | Expected Exception {expected.__name__} | Got={result!r} | FAIL")
      elif result == expected:
          print(f"Test {i}: Input={input_val!r} | Expected={expected!r} | Got={result!r} | PASS")
          passed += 1
      else:
          print(f"Test {i}: Input={input_val!r} | Expected={expected!r} | Got={result!r} | FAIL")
```



testcase-task5.py > ...

```python
    ("2023-07", ValueError),            # missing day
    ("2023-07-15-01", ValueError),      # too many parts
]

passed = 0
total = 0

for i, (input_val, expected) in enumerate(test_cases, 1):
    total += 1
    try:
        result = convert_date_format(input_val)
        if isinstance(expected, type) and issubclass(expected, Exception):
            print(f"Test {i}: Input={input_val!r} | Expected Exception {expected.__name__} | Got={result!r} | FAIL")
        elif result == expected:
            print(f"Test {i}: Input={input_val!r} | Expected={expected!r} | Got={result!r} | PASS")
            passed += 1
        else:
            print(f"Test {i}: Input={input_val!r} | Expected={expected!r} | Got={result!r} | FAIL")
    except Exception as e:
        if isinstance(expected, type) and isinstance(e, expected):
            print(f"Test {i}: Input={input_val!r} | Expected Exception {expected.__name__} | Got Exception {type(e).__name__}
            passed += 1
        else:
            print(f"Test {i}: Input={input_val!r} | Expected={expected!r} | Got Exception {type(e).__name__}: {e} | FAIL")

print(f"\nTotal: {total}, Passed: {passed}, Failed: {total - passed}")
if passed == total:
    print("okay all correct")
else:
    print("some tests failed")

```

OUTPUT:



```
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.
/Desktop/AIAC/Lab 8.3/testcase-task5.py"
Test 1: Input='2023-07-15' | Expected='15-07-2023' | Got='15-07-2023' | PASS
Test 2: Input='2000-01-01' | Expected='01-01-2000' | Got='01-01-2000' | PASS
Test 3: Input='1999-12-31' | Expected='31-12-1999' | Got='31-12-1999' | PASS
Test 4: Input='2020-02-29' | Expected='29-02-2020' | Got='29-02-2020' | PASS
Test 5: Input='2021-11-09' | Expected='09-11-2021' | Got='09-11-2021' | PASS
Test 6: Input='0001-01-01' | Expected='01-01-0001' | Got='01-01-0001' | PASS
Test 7: Input='2022-10-05' | Expected='05-10-2022' | Got='05-10-2022' | PASS
Test 8: Input='1980-06-30' | Expected='30-06-1980' | Got='30-06-1980' | PASS
Test 9: Input='2010-09-08' | Expected='08-09-2010' | Got='08-09-2010' | PASS
Test 10: Input='2024-12-31' | Expected='31-12-2024' | Got='31-12-2024' | PASS
Test 11: Input='2023/07/15' | Expected Exception ValueError | Got Exception ValueError | PASS
Test 12: Input='2023-7-15' | Expected='15-7-2023' | Got='15-7-2023' | PASS
Test 13: Input='2023-07-32' | Expected='32-07-2023' | Got='32-07-2023' | PASS
Test 14: Input='' | Expected Exception ValueError | Got Exception ValueError | PASS
Test 15: Input='2023-07' | Expected Exception ValueError | Got Exception ValueError | PASS
Test 16: Input='2023-07-15-01' | Expected Exception ValueError | Got Exception ValueError | PASS

Total: 16, Passed: 16, Failed: 0
okay all correct
PS C:\Users\supri\OneDrive\Desktop\AIAC\Lab 8.3>
```

Expected Output#5

- Function converts input format correctly for all test cases

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	**Max Marks**
Task #1	0.5
Task #2	0.5
Task #3	0.5
Task #4	0.5
Task #5	0.5
Total	**2.5 Marks**