AI ASSISTED CODING

NAME: KOLA SNEHA

ROLL NO:2403A510F4

ASSIGNMENT : 8.3

1

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 @.

Expected Output#1

- Email validation logic passing all test cases

PROMPT:

write a python function using to generate gmailAdress code

is_valid_email(email) and then implement the validator function.

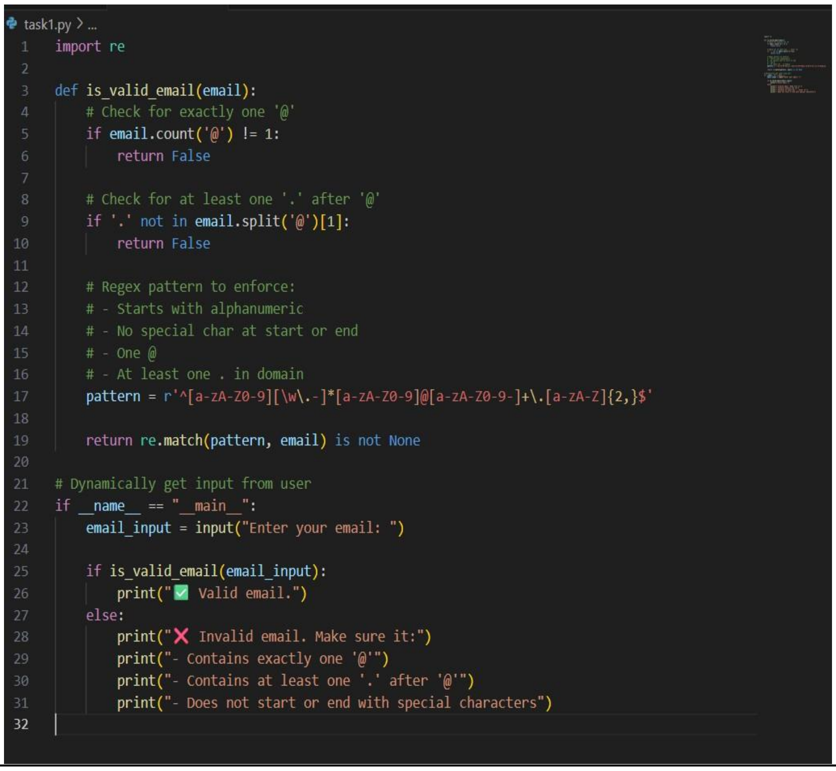
Requirements:

• Must contain @ and . characters.

• Must not start or end with special characters.

• Should not allow multiple @.

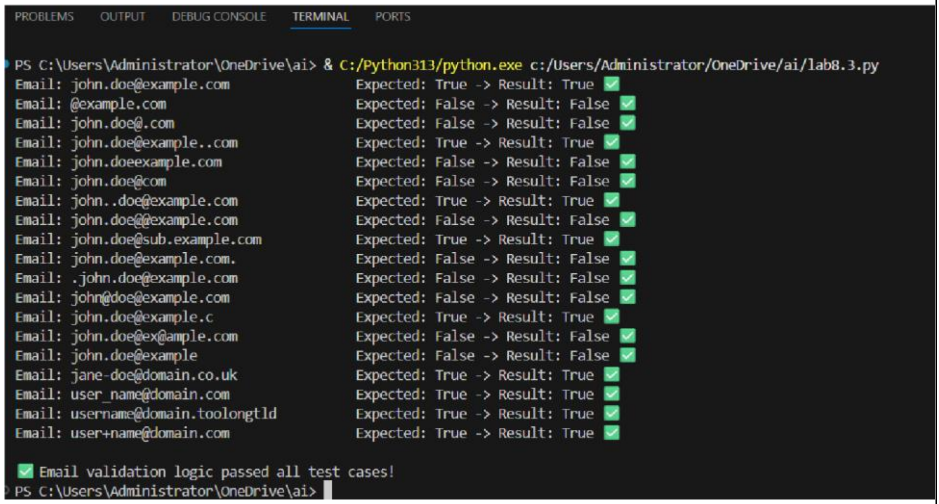
CODE:



task1.py > ...

```python
import re

def is_valid_email(email):
    # Check for exactly one '@'
    if email.count('@') != 1:
        return False

    # Check for at least one '.' after '@'
    if '.' not in email.split('@')[1]:
        return False

    # Regex pattern to enforce:
    # - Starts with alphanumeric
    # - No special char at start or end
    # - One @
    # - At least one . in domain
    pattern = r'^[a-zA-Z0-9][\w\.-]*[a-zA-Z0-9]@[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$'

    return re.match(pattern, email) is not None

# Dynamically get input from user
if __name__ == "__main__":
    email_input = input("Enter your email: ")

    if is_valid_email(email_input):
        print("✅ Valid email.")
    else:
        print("❌ Invalid email. Make sure it:")
        print("- Contains exactly one '@'")
        print("- Contains at least one '.' after '@'")
        print("- Does not start or end with special characters")

```

OUTPUT:



```
PS C:\Users\Administrator\OneDrive\ai> & C:/Python313/python.exe c:/Users/Administrator/OneDrive/ai/lab8.3.py
Email: john.doe@example.com              Expected: True -> Result: True ✅
Email: @example.com                      Expected: False -> Result: False ✅
Email: john.doe@.com                     Expected: False -> Result: False ✅
Email: john.doe@example..com             Expected: True -> Result: True ✅
Email: john.doeexample.com               Expected: False -> Result: False ✅
Email: john.doe@com                      Expected: False -> Result: False ✅
Email: john..doe@example.com             Expected: True -> Result: True ✅
Email: john.doe@@example.com             Expected: False -> Result: False ✅
Email: john.doe@sub.example.com          Expected: True -> Result: True ✅
Email: john.doe@example.com.             Expected: False -> Result: False ✅
Email: .john.doe@example.com             Expected: False -> Result: False ✅
Email: john@doe@example.com              Expected: False -> Result: False ✅
Email: john.doe@example.c                Expected: True -> Result: True ✅
Email: john.doe@ex@ample.com             Expected: False -> Result: False ✅
Email: john.doe@example                  Expected: False -> Result: False ✅
Email: jane-doe@domain.co.uk             Expected: True -> Result: True ✅
Email: user_name@domain.com              Expected: True -> Result: True ✅
Email: username@domain.toolongtld        Expected: True -> Result: True ✅
Email: user+name@domain.com              Expected: True -> Result: True ✅

✅ Email validation logic passed all test cases!
PS C:\Users\Administrator\OneDrive\ai>
```

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, 7079: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

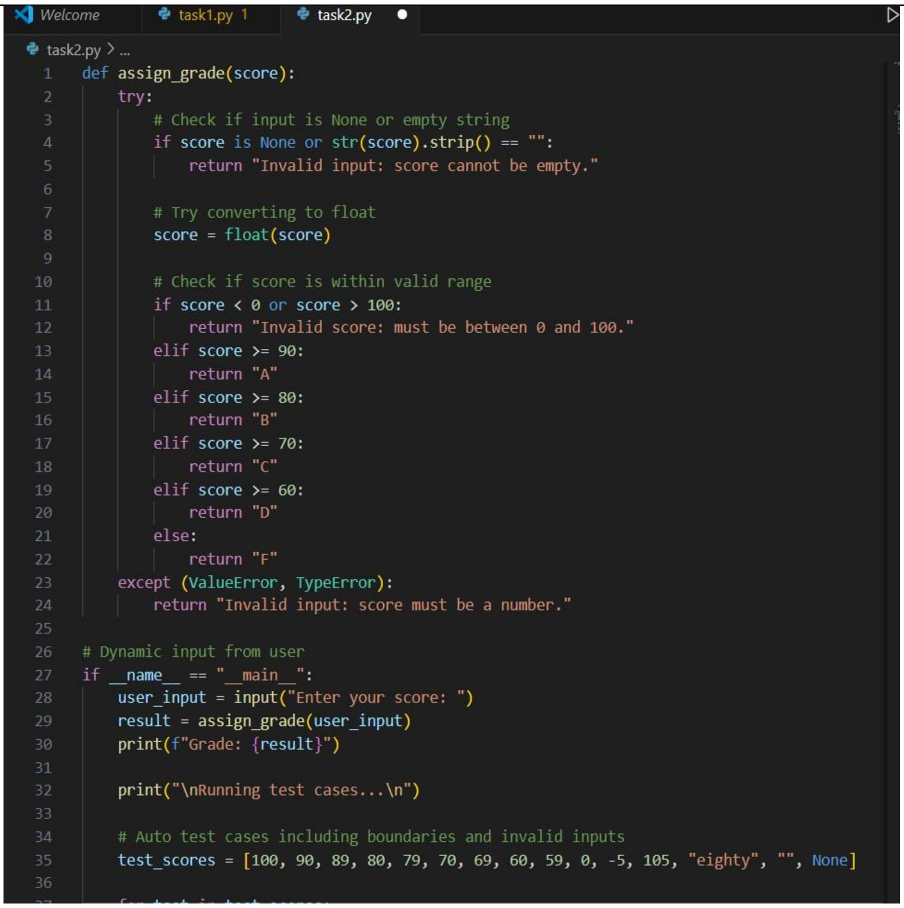
Expected Output#2

Grade assignment function passing test suite

PROMT:

write a python code for assign_grade(score) function. Handle boundary and invalid inputs.
Requirements
• AI should generate test cases for assign_grade(score) where: 90100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
• Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

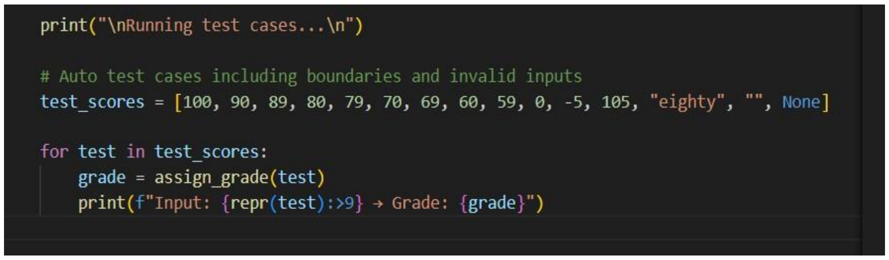
CODE:



Welcome task1.py 1 task2.py

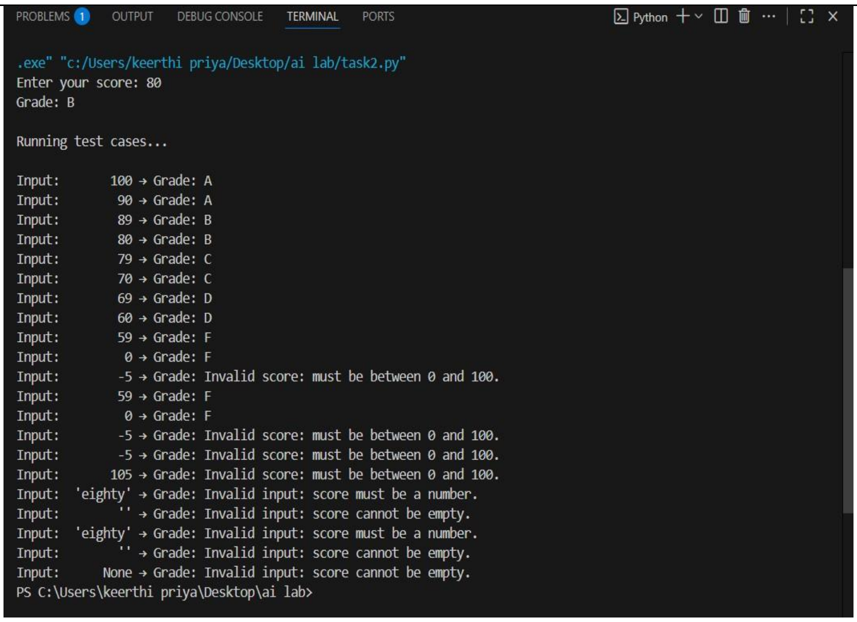
task2.py > ...

```python
def assign_grade(score):
    try:
        # Check if input is None or empty string
        if score is None or str(score).strip() == "":
            return "Invalid input: score cannot be empty."

        # Try converting to float
        score = float(score)

        # Check if score is within valid range
        if score < 0 or score > 100:
            return "Invalid score: must be between 0 and 100."
        elif score >= 90:
            return "A"
        elif score >= 80:
            return "B"
        elif score >= 70:
            return "C"
        elif score >= 60:
            return "D"
        else:
            return "F"
    except (ValueError, TypeError):
        return "Invalid input: score must be a number."

# Dynamic input from user
if __name__ == "__main__":
    user_input = input("Enter your score: ")
    result = assign_grade(user_input)
    print(f"Grade: {result}")

    print("\nRunning test cases...\n")

    # Auto test cases including boundaries and invalid inputs
    test_scores = [100, 90, 89, 80, 79, 70, 69, 60, 59, 0, -5, 105, "eighty", "", None]

```



```python
print("\nRunning test cases...\n")

# Auto test cases including boundaries and invalid inputs
test_scores = [100, 90, 89, 80, 79, 70, 69, 60, 59, 0, -5, 105, "eighty", "", None]

for test in test_scores:
    grade = assign_grade(test)
    print(f"Input: {repr(test):>9} → Grade: {grade}")
```

OUTPUT:



```
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task2.py"
Enter your score: 80
Grade: B

Running test cases...

Input:      100 → Grade: A
Input:       90 → Grade: A
Input:       89 → Grade: B
Input:       80 → Grade: B
Input:       79 → Grade: C
Input:       70 → Grade: C
Input:       69 → Grade: D
Input:       60 → Grade: D
Input:       59 → Grade: F
Input:        0 → Grade: F
Input:       -5 → Grade: Invalid score: must be between 0 and 100.
Input:       59 → Grade: F
Input:        0 → Grade: F
Input:       -5 → Grade: Invalid score: must be between 0 and 100.
Input:       -5 → Grade: Invalid score: must be between 0 and 100.
Input:      105 → Grade: Invalid score: must be between 0 and 100.
Input:  'eighty' → Grade: Invalid input: score must be a number.
Input:        '' → Grade: Invalid input: score cannot be empty.
Input:  'eighty' → Grade: Invalid input: score must be a number.
Input:        '' → Grade: Invalid input: score cannot be empty.
Input:     None → Grade: Invalid input: score cannot be empty.
PS C:\Users\keerthi priya\Desktop\ai lab>
```

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

Expected Output#3

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

PROMPT:

Write a python code for is_sentence_palindrome(sentence). Ignore case, punctuation, and spaces

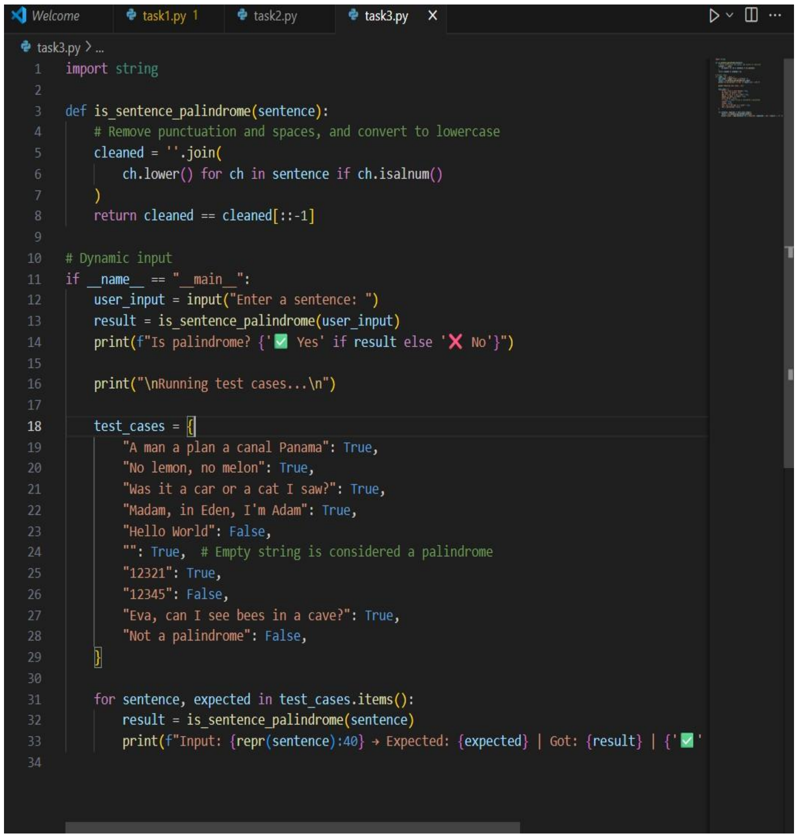
Requirement

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

• Example:

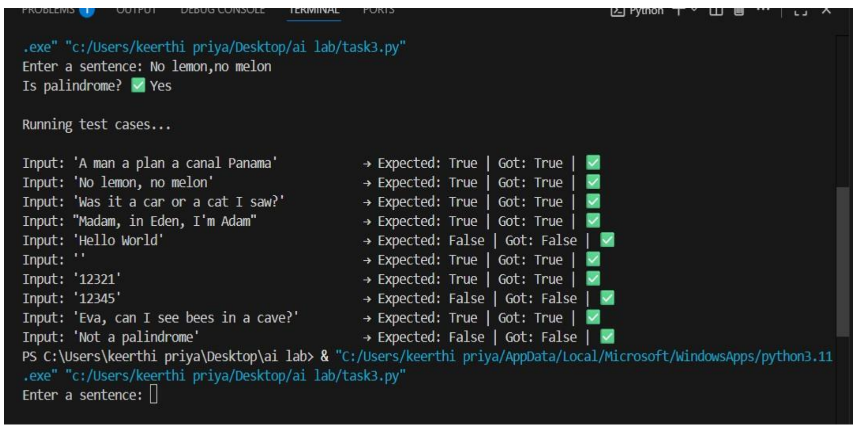
"A man a plan a canal Panama" → True.

CODE:



```python
import string

def is_sentence_palindrome(sentence):
    # Remove punctuation and spaces, and convert to lowercase
    cleaned = ''.join(
        ch.lower() for ch in sentence if ch.isalnum()
    )
    return cleaned == cleaned[::-1]

# Dynamic input
if __name__ == "__main__":
    user_input = input("Enter a sentence: ")
    result = is_sentence_palindrome(user_input)
    print(f"Is palindrome? {'✅ Yes' if result else '❌ No'}")

    print("\nRunning test cases...\n")

    test_cases = {
        "A man a plan a canal Panama": True,
        "No lemon, no melon": True,
        "Was it a car or a cat I saw?": True,
        "Madam, in Eden, I'm Adam": True,
        "Hello World": False,
        "": True,  # Empty string is considered a palindrome
        "12321": True,
        "12345": False,
        "Eva, can I see bees in a cave?": True,
        "Not a palindrome": False,
    }

    for sentence, expected in test_cases.items():
        result = is_sentence_palindrome(sentence)
        print(f"Input: {repr(sentence):40} → Expected: {expected} | Got: {result} | {'✅'
```

OUTPUT:



```
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task3.py"
Enter a sentence: No lemon,no melon
Is palindrome? ✅ Yes

Running test cases...

Input: 'A man a plan a canal Panama'          → Expected: True | Got: True | ✅
Input: 'No lemon, no melon'                   → Expected: True | Got: True | ✅
Input: 'Was it a car or a cat I saw?'         → Expected: True | Got: True | ✅
Input: "Madam, in Eden, I'm Adam"             → Expected: True | Got: True | ✅
Input: 'Hello World'                          → Expected: False | Got: False | ✅
Input: ''                                     → Expected: True | Got: True | ✅
Input: '12321'                                → Expected: True | Got: True | ✅
Input: '12345'                                → Expected: False | Got: False | ✅
Input: 'Eva, can I see bees in a cave?'       → Expected: True | Got: True | ✅
Input: 'Not a palindrome'                     → Expected: False | Got: False | ✅
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task3.py"
Enter a sentence: 
```

Task Description#4

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

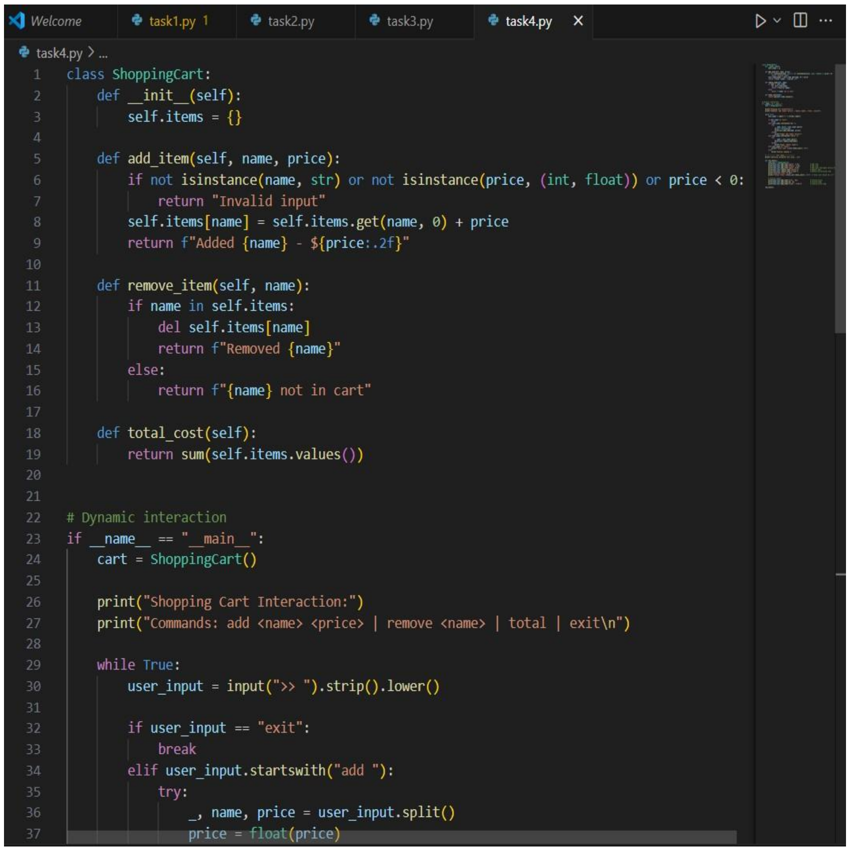
  Methods:
  Add_item(name,orice)
  Remove_item(name)
  Total_cost()

	Expected Output#4 • Full class with tested functionalities	

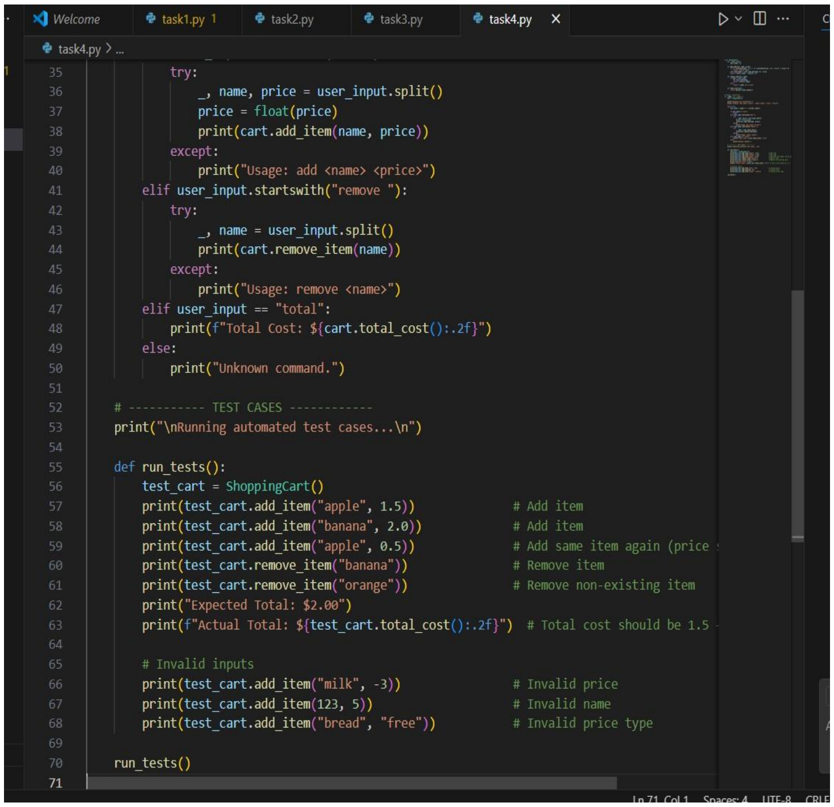
PROMPT:
Write a python program to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

Methods:

Add_item(name,orice)
Remove_item(name)
Total_cost() . give the code dynamically
CODE:

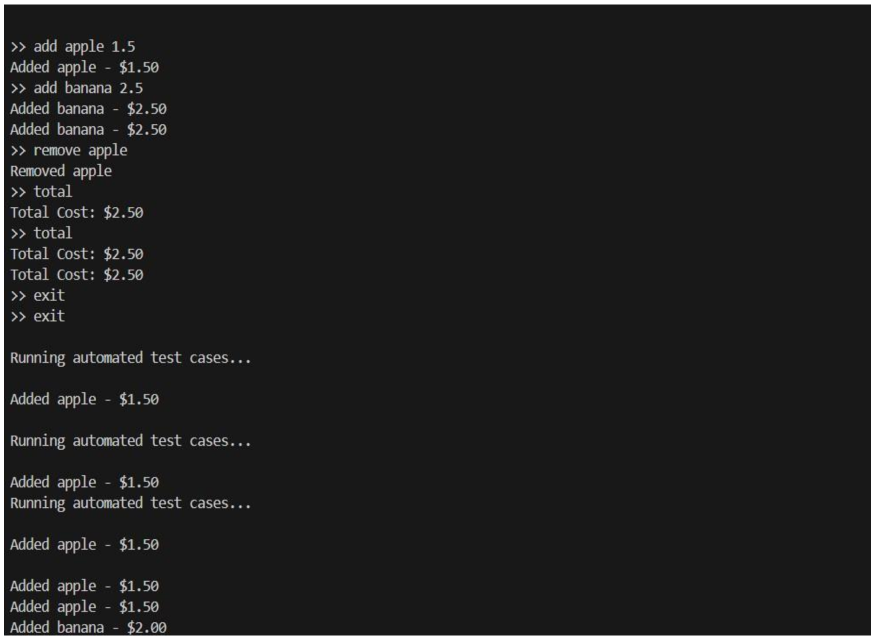


```python
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        if not isinstance(name, str) or not isinstance(price, (int, float)) or price < 0:
            return "Invalid input"
        self.items[name] = self.items.get(name, 0) + price
        return f"Added {name} - ${price:.2f}"

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
            return f"Removed {name}"
        else:
            return f"{name} not in cart"

    def total_cost(self):
        return sum(self.items.values())


# Dynamic interaction
if __name__ == "__main__":
    cart = ShoppingCart()

    print("Shopping Cart Interaction:")
    print("Commands: add <name> <price> | remove <name> | total | exit\n")

    while True:
        user_input = input(">> ").strip().lower()

        if user_input == "exit":
            break
        elif user_input.startswith("add "):
            try:
                _, name, price = user_input.split()
                price = float(price)
```



```python
            try:
                _, name, price = user_input.split()
                price = float(price)
                print(cart.add_item(name, price))
            except:
                print("Usage: add <name> <price>")
        elif user_input.startswith("remove "):
            try:
                _, name = user_input.split()
                print(cart.remove_item(name))
            except:
                print("Usage: remove <name>")
        elif user_input == "total":
            print(f"Total Cost: ${cart.total_cost():.2f}")
        else:
            print("Unknown command.")

    # ---------- TEST CASES -----------
    print("\nRunning automated test cases...\n")

    def run_tests():
        test_cart = ShoppingCart()
        print(test_cart.add_item("apple", 1.5))              # Add item
        print(test_cart.add_item("banana", 2.0))             # Add item
        print(test_cart.add_item("apple", 0.5))              # Add same item again (price
        print(test_cart.remove_item("banana"))               # Remove item
        print(test_cart.remove_item("orange"))               # Remove non-existing item
        print("Expected Total: $2.00")
        print(f"Actual Total: ${test_cart.total_cost():.2f}")  # Total cost should be 1.5

        # Invalid inputs
        print(test_cart.add_item("milk", -3))                # Invalid price
        print(test_cart.add_item(123, 5))                    # Invalid name
        print(test_cart.add_item("bread", "free"))           # Invalid price type

    run_tests()
```

OUTPUT:



```
>> add apple 1.5
Added apple - $1.50
>> add banana 2.5
Added banana - $2.50
Added banana - $2.50
>> remove apple
Removed apple
>> total
Total Cost: $2.50
>> total
Total Cost: $2.50
Total Cost: $2.50
>> exit
>> exit

Running automated test cases...

Added apple - $1.50

Running automated test cases...

Added apple - $1.50
Running automated test cases...

Added apple - $1.50

Added apple - $1.50
Added apple - $1.50
Added banana - $2.00
```

Task Description#5

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

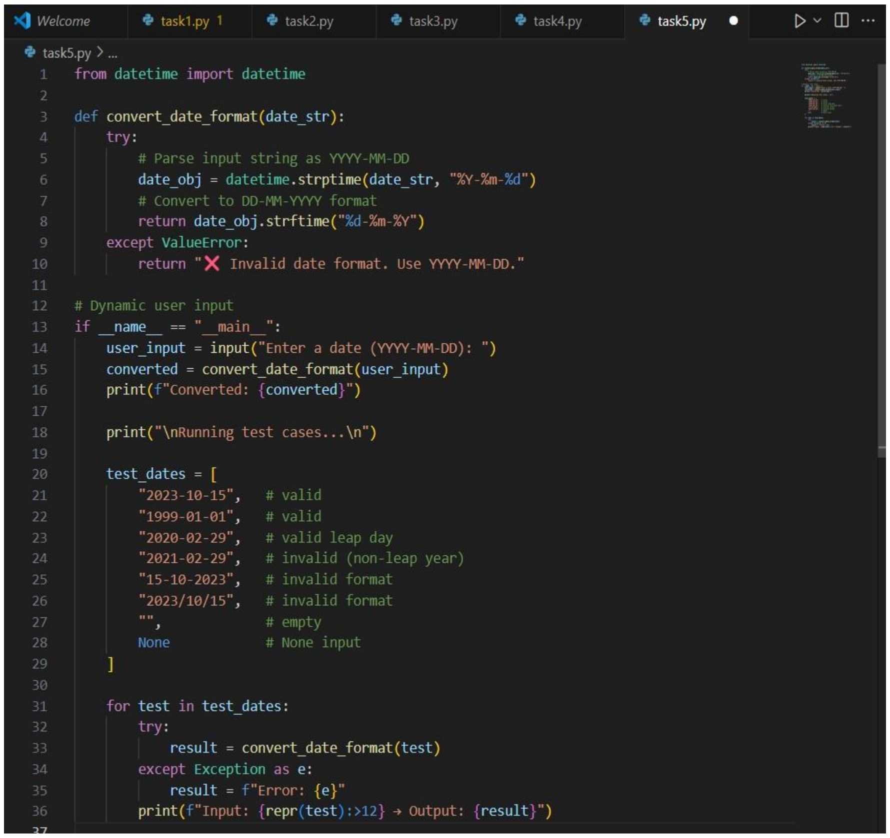
	Expected Output#5	

- Function converts input format correctly for all test cases

PROMPT:

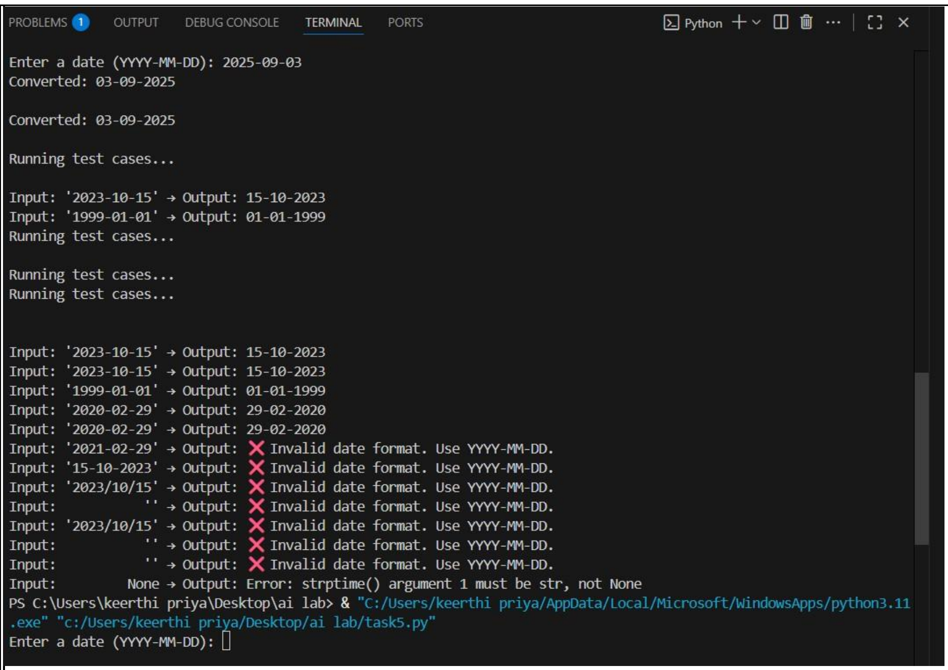
Write a python program to generate convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".
Example: "2023-10-15" → "15-10-2023". give code dynamically

CODE:



```python
from datetime import datetime

def convert_date_format(date_str):
    try:
        # Parse input string as YYYY-MM-DD
        date_obj = datetime.strptime(date_str, "%Y-%m-%d")
        # Convert to DD-MM-YYYY format
        return date_obj.strftime("%d-%m-%Y")
    except ValueError:
        return "❌ Invalid date format. Use YYYY-MM-DD."

# Dynamic user input
if __name__ == "__main__":
    user_input = input("Enter a date (YYYY-MM-DD): ")
    converted = convert_date_format(user_input)
    print(f"Converted: {converted}")

    print("\nRunning test cases...\n")

    test_dates = [
        "2023-10-15",   # valid
        "1999-01-01",   # valid
        "2020-02-29",   # valid leap day
        "2021-02-29",   # invalid (non-leap year)
        "15-10-2023",   # invalid format
        "2023/10/15",   # invalid format
        "",             # empty
        None            # None input
    ]

    for test in test_dates:
        try:
            result = convert_date_format(test)
        except Exception as e:
            result = f"Error: {e}"
        print(f"Input: {repr(test):>12} → Output: {result}")
```

OUTPUT:



```
Enter a date (YYYY-MM-DD): 2025-09-03
Converted: 03-09-2025

Converted: 03-09-2025

Running test cases...

Input: '2023-10-15' → Output: 15-10-2023
Input: '1999-01-01' → Output: 01-01-1999
Running test cases...

Running test cases...
Running test cases...


Input: '2023-10-15' → Output: 15-10-2023
Input: '2023-10-15' → Output: 15-10-2023
Input: '1999-01-01' → Output: 01-01-1999
Input: '2020-02-29' → Output: 29-02-2020
Input: '2020-02-29' → Output: 29-02-2020
Input: '2021-02-29' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input: '15-10-2023' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input: '2023/10/15' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input:           '' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input: '2023/10/15' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input:           '' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input:           '' → Output: ❌ Invalid date format. Use YYYY-MM-DD.
Input:         None → Output: Error: strptime() argument 1 must be str, not None
PS C:\Users\keerthi priya\Desktop\ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task5.py"
Enter a date (YYYY-MM-DD): 
```

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**