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

ProgramName:M. Tech	**Assignment Type: Lab**	**AcademicYear:**2025-2026

CourseCoordinatorName	Venkataramana Veeramsetty		
CourseCode		**CourseTitle**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	10.11.2025	**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. • 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	Week4 - Wednesday

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

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").

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

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()

	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"** **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**	

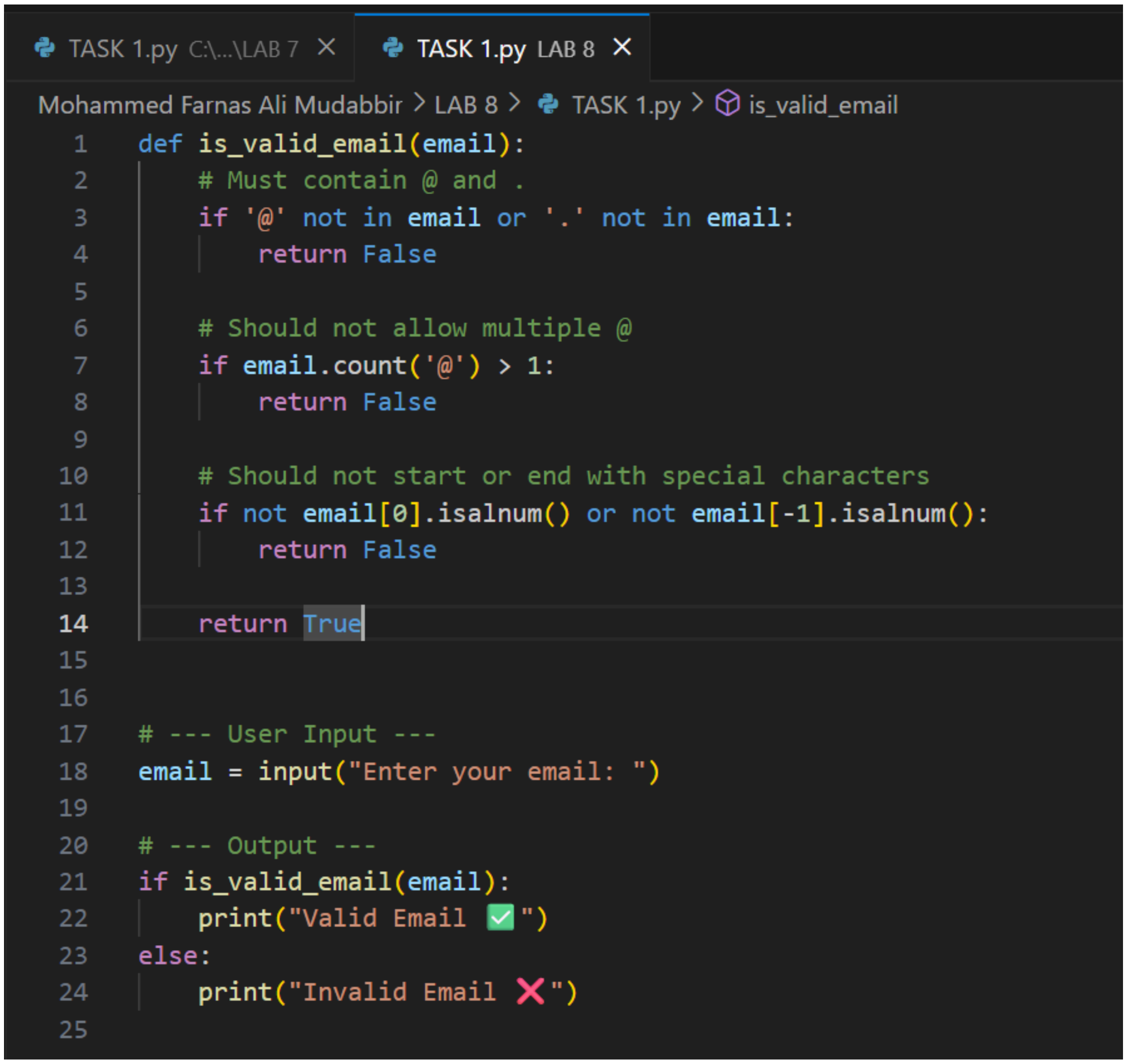
Task Description#1

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 @.

PROMPT: Generate test cases for is_valid_email(email) and implement the function. The email must contain @ and ., must not start or end with special characters, and should not allow multiple @.

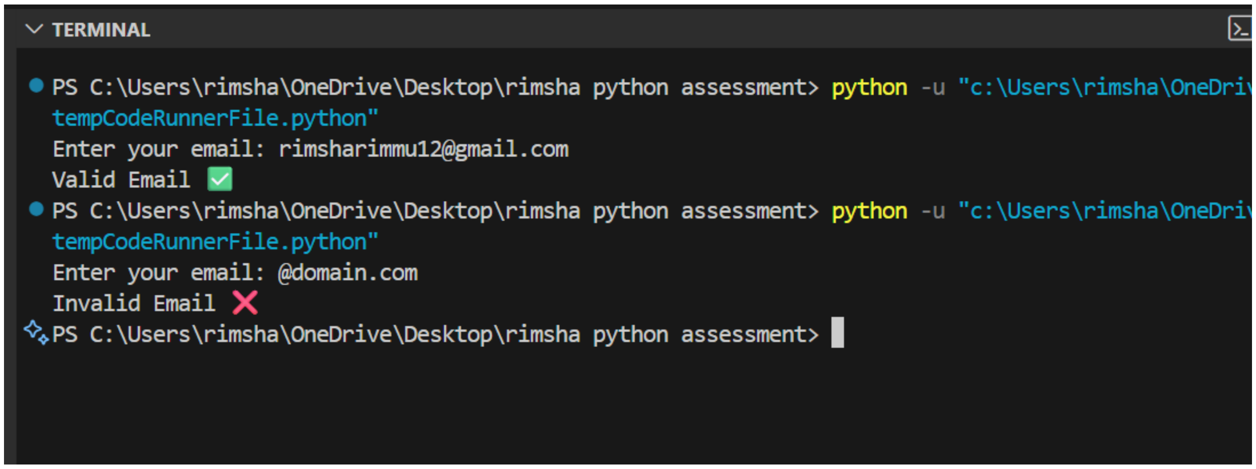


```
TASK 1.py C:\...\LAB 7 ×    TASK 1.py LAB 8 ×
Mohammed Farnas Ali Mudabbir > LAB 8 > TASK 1.py > is_valid_email
def is_valid_email(email):
    # Must contain @ and .
    if '@' not in email or '.' not in email:
        return False

    # Should not allow multiple @
    if email.count('@') > 1:
        return False

    # Should not start or end with special characters
    if not email[0].isalnum() or not email[-1].isalnum():
        return False

    return True


# --- User Input ---
email = input("Enter your email: ")

# --- Output ---
if is_valid_email(email):
    print("Valid Email ✅")
else:
    print("Invalid Email ❌")
```

Expected Output#1

- Email validation logic passing all test cases

Practical output:

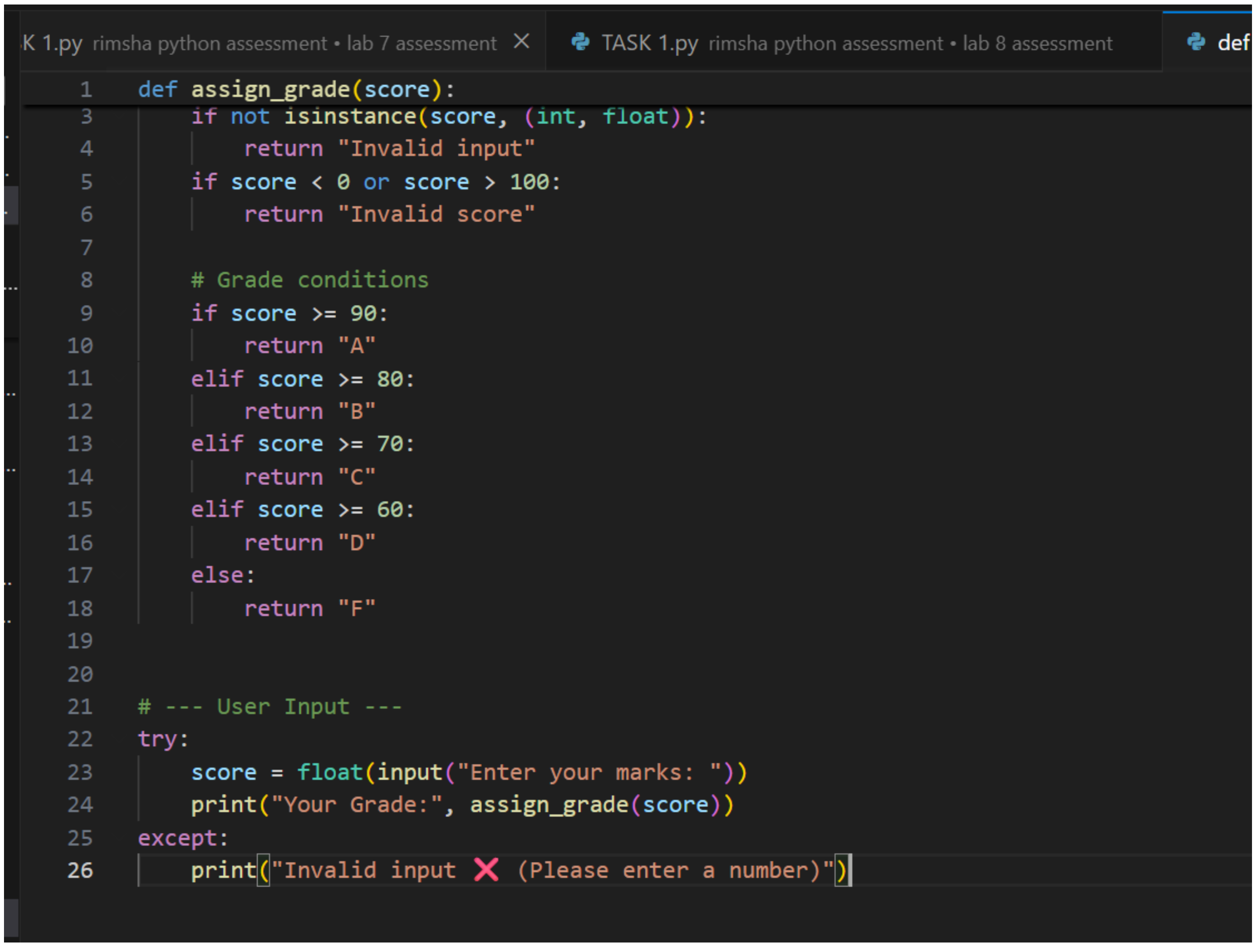


```
TERMINAL
● PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDri
  tempCodeRunnerFile.python"
  Enter your email: rimsharimmu12@gmail.com
  Valid Email ✅
● PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDri
  tempCodeRunnerFile.python"
  Enter your email: @domain.com
  Invalid Email ❌
  PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

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").

PROMPT: Write a Python program to take user input for score and assign grade (A–F). Handle invalid inputs and show test cases.

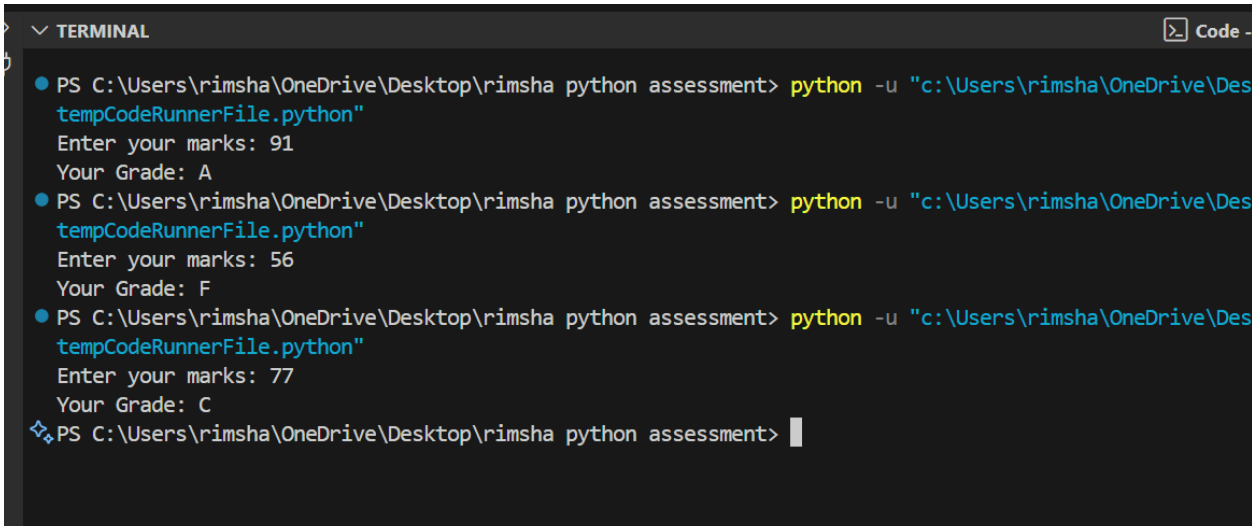

K 1.py rimsha python assessment • lab 7 assessment
TASK 1.py rimsha python assessment • lab 8 assessment
def

```python
def assign_grade(score):
    if not isinstance(score, (int, float)):
        return "Invalid input"
    if score < 0 or score > 100:
        return "Invalid score"

    # Grade conditions
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"


# --- User Input ---
try:
    score = float(input("Enter your marks: "))
    print("Your Grade:", assign_grade(score))
except:
    print("Invalid input ❌ (Please enter a number)")
```

Expected Output#2

Grade assignment function passing test suite

Practical output:



```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDrive\Des
tempCodeRunnerFile.python"
Enter your marks: 91
Your Grade: A
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDrive\Des
tempCodeRunnerFile.python"
Enter your marks: 56
Your Grade: F
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDrive\Des
tempCodeRunnerFile.python"
Enter your marks: 77
Your Grade: C
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

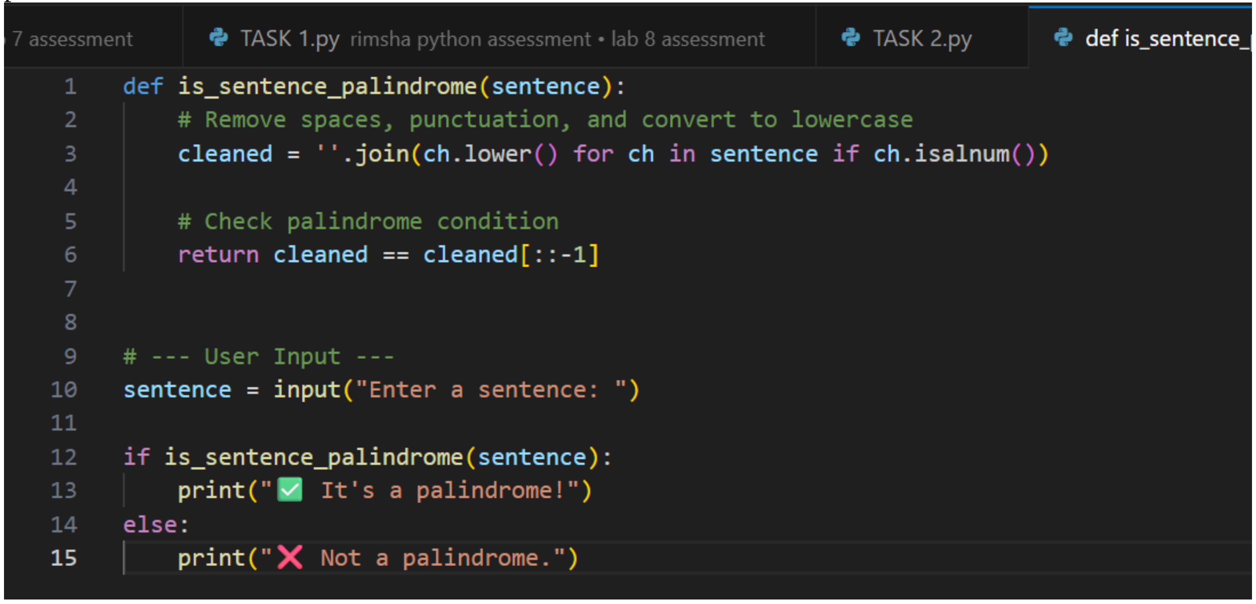
Task Description#3

- 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

PROMPT: Write a Python program to check if a sentence is a palindrome, ignoring case, spaces, and punctuation.



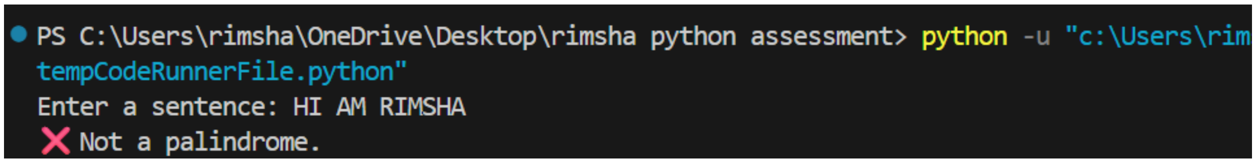
```python
def is_sentence_palindrome(sentence):
    # Remove spaces, punctuation, and convert to lowercase
    cleaned = ''.join(ch.lower() for ch in sentence if ch.isalnum())

    # Check palindrome condition
    return cleaned == cleaned[::-1]


# --- User Input ---
sentence = input("Enter a sentence: ")

if is_sentence_palindrome(sentence):
    print("✅ It's a palindrome!")
else:
    print("❌ Not a palindrome.")
```

Expected Output#3

- Function returns True/False for cleaned sentences

Implement the function to pass AI-generated tests

Practical output:



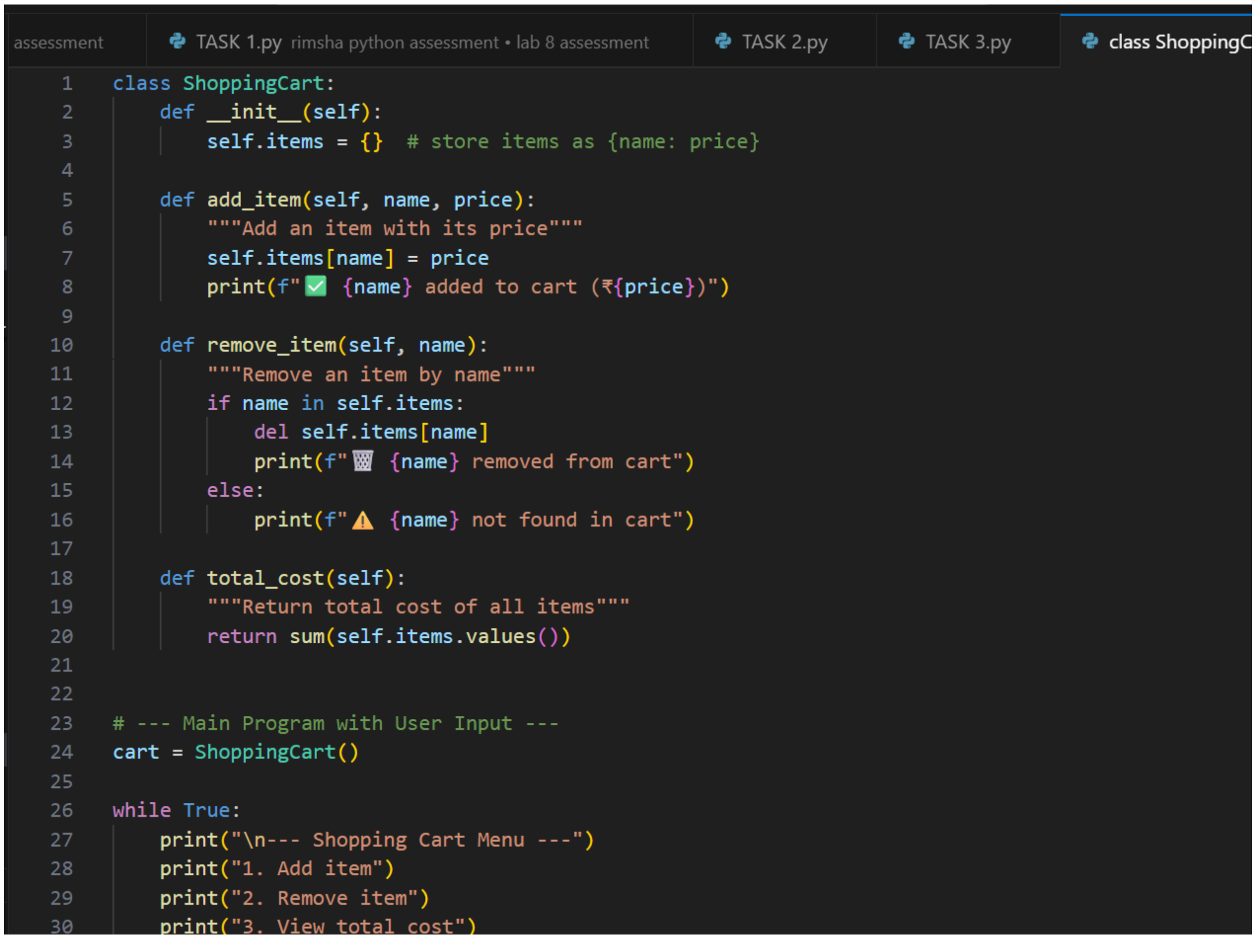
```
● PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rim
  tempCodeRunnerFile.python"
  Enter a sentence: HI AM RIMSHA
  ❌ Not a palindrome.
```

Task Description#4

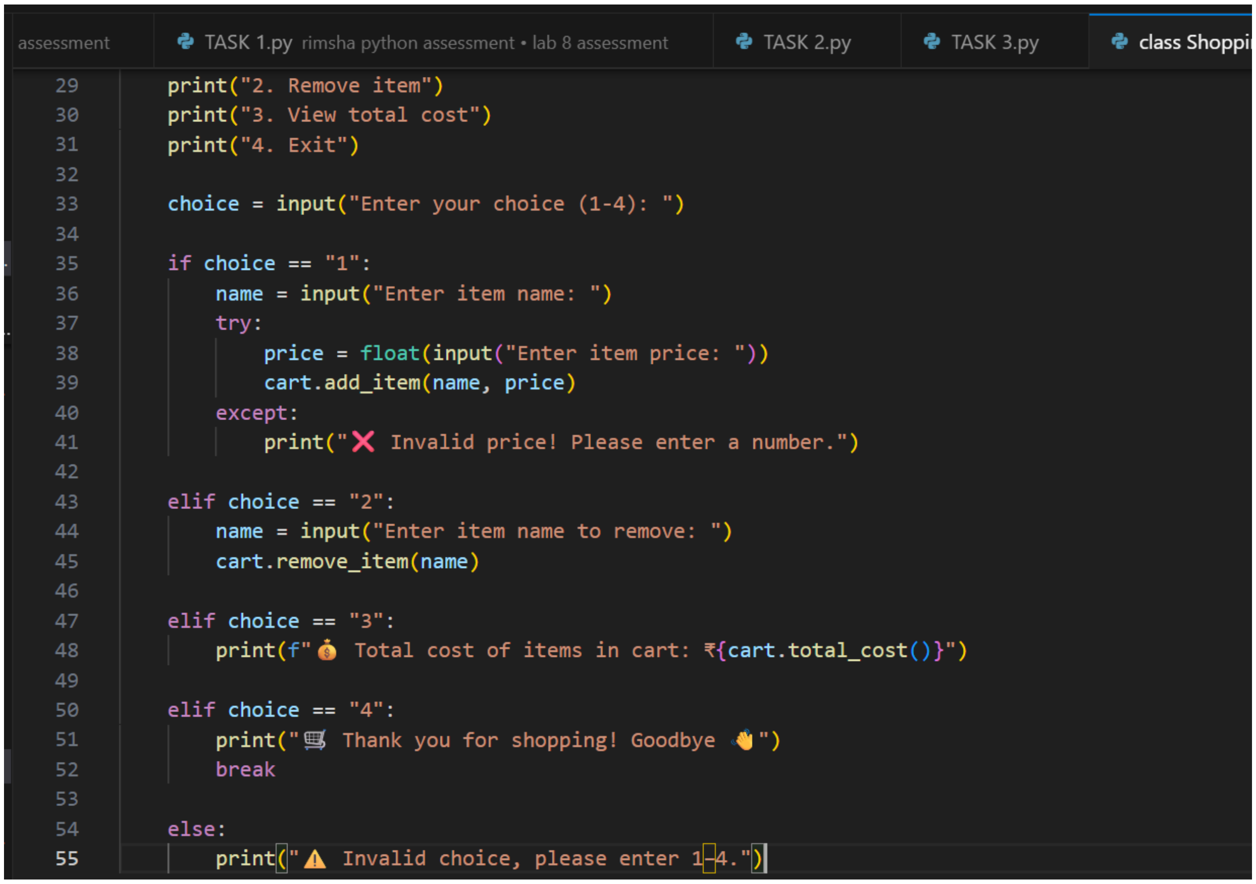
- Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

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

PROMPT: Write a Python program for a ShoppingCart class with methods add_item(name, price), remove_item(name), and total_cost()



```python
class ShoppingCart:
    def __init__(self):
        self.items = {}  # store items as {name: price}

    def add_item(self, name, price):
        """Add an item with its price"""
        self.items[name] = price
        print(f"✅ {name} added to cart (₹{price})")

    def remove_item(self, name):
        """Remove an item by name"""
        if name in self.items:
            del self.items[name]
            print(f"🗑 {name} removed from cart")
        else:
            print(f"⚠ {name} not found in cart")

    def total_cost(self):
        """Return total cost of all items"""
        return sum(self.items.values())


# --- Main Program with User Input ---
cart = ShoppingCart()

while True:
    print("\n--- Shopping Cart Menu ---")
    print("1. Add item")
    print("2. Remove item")
    print("3. View total cost")
```

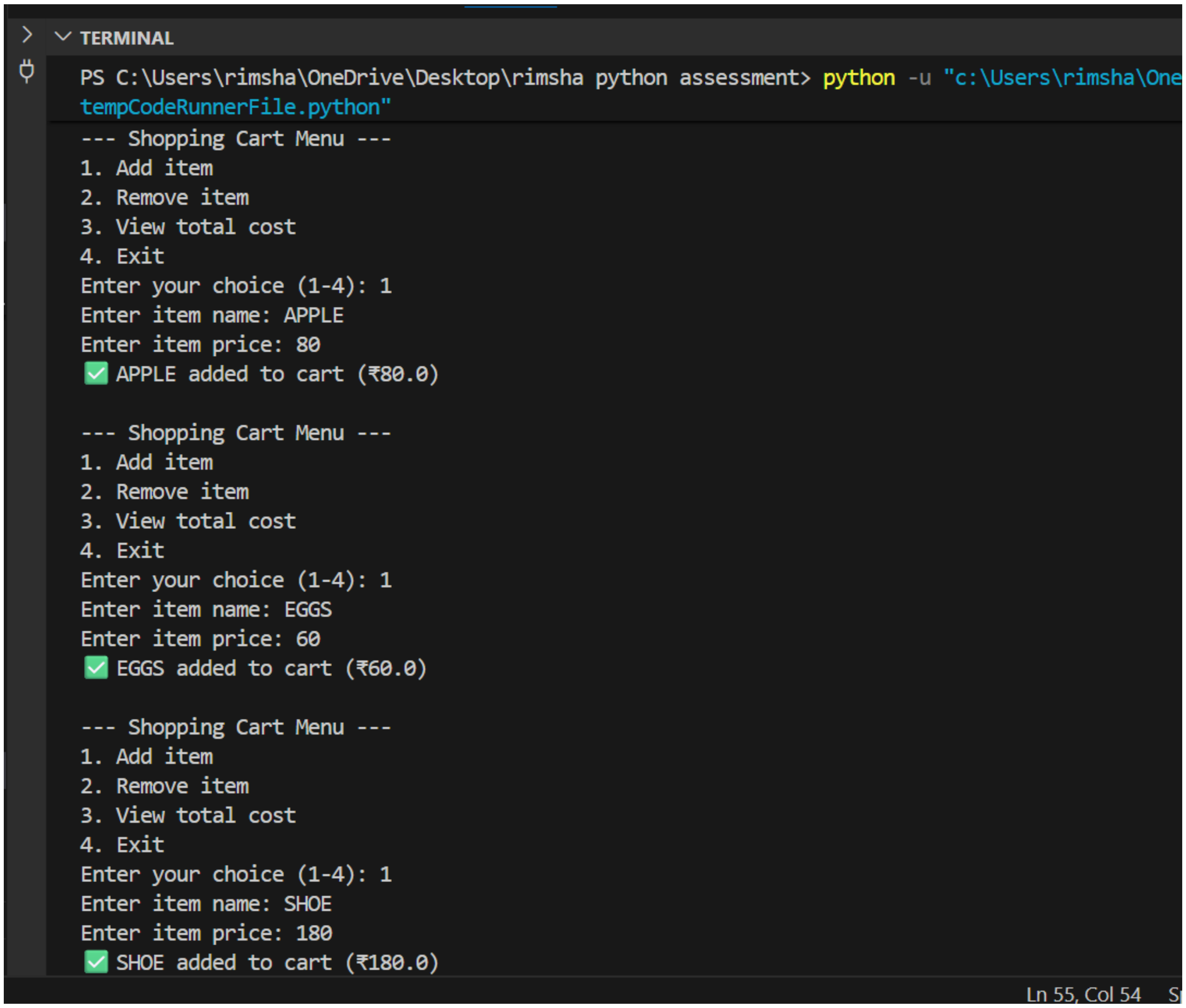


```python
    print("2. Remove item")
    print("3. View total cost")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == "1":
        name = input("Enter item name: ")
        try:
            price = float(input("Enter item price: "))
            cart.add_item(name, price)
        except:
            print("❌ Invalid price! Please enter a number.")

    elif choice == "2":
        name = input("Enter item name to remove: ")
        cart.remove_item(name)

    elif choice == "3":
        print(f"💰 Total cost of items in cart: ₹{cart.total_cost()}")

    elif choice == "4":
        print("🛒 Thank you for shopping! Goodbye 👋")
        break

    else:
        print("⚠️ Invalid choice, please enter 1-4.")
```

Expected Output#4

- Full class with tested functionalities

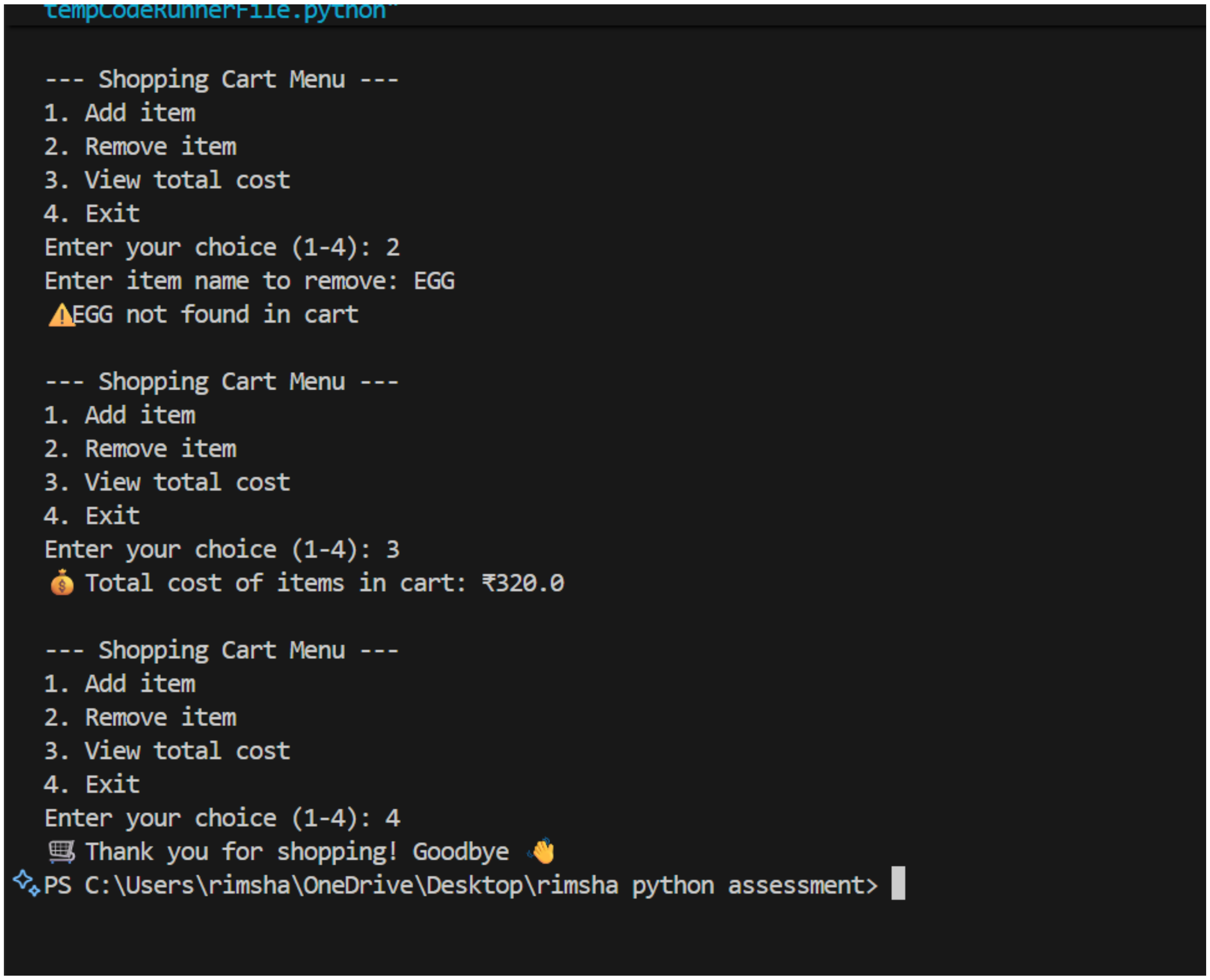
Practical output:



TERMINAL

```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\One
tempCodeRunnerFile.python"
--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 1
Enter item name: APPLE
Enter item price: 80
✅ APPLE added to cart (₹80.0)

--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 1
Enter item name: EGGS
Enter item price: 60
✅ EGGS added to cart (₹60.0)

--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 1
Enter item name: SHOE
Enter item price: 180
✅ SHOE added to cart (₹180.0)
```

Ln 55, Col 54

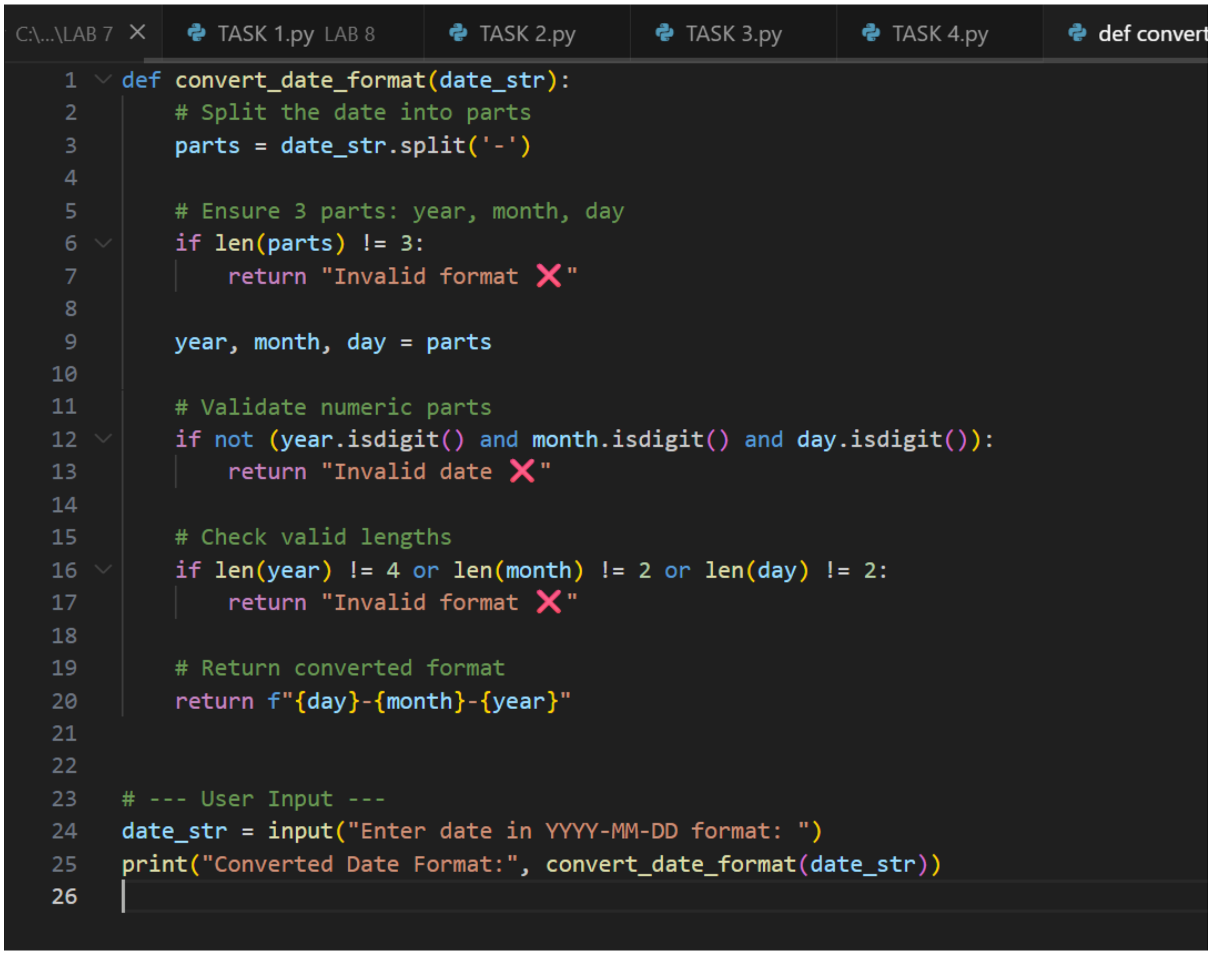


```
tempCodeRunnerFile.python

--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 2
Enter item name to remove: EGG
⚠️EGG not found in cart

--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 3
💰 Total cost of items in cart: ₹320.0

--- Shopping Cart Menu ---
1. Add item
2. Remove item
3. View total cost
4. Exit
Enter your choice (1-4): 4
🛒 Thank you for shopping! Goodbye 👋
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```

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"

PROMPT: Write a Python program to convert date from "YYYY-MM-DD" to "DD-MM-YYYY"

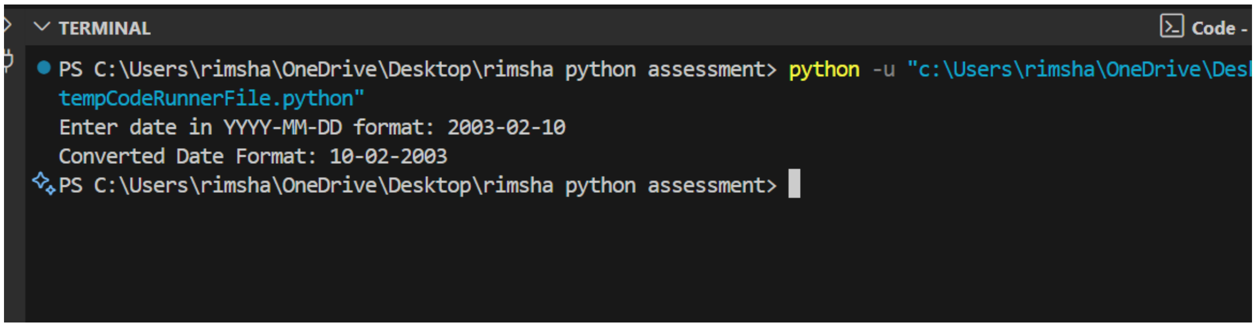


```python
def convert_date_format(date_str):
    # Split the date into parts
    parts = date_str.split('-')

    # Ensure 3 parts: year, month, day
    if len(parts) != 3:
        return "Invalid format ❌"

    year, month, day = parts

    # Validate numeric parts
    if not (year.isdigit() and month.isdigit() and day.isdigit()):
        return "Invalid date ❌"

    # Check valid lengths
    if len(year) != 4 or len(month) != 2 or len(day) != 2:
        return "Invalid format ❌"

    # Return converted format
    return f"{day}-{month}-{year}"


# --- User Input ---
date_str = input("Enter date in YYYY-MM-DD format: ")
print("Converted Date Format:", convert_date_format(date_str))
```

Expected Output#5

- Function converts input format correctly for all test cases

Practical output:



```
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment> python -u "c:\Users\rimsha\OneDrive\Des
tempCodeRunnerFile.python"
Enter date in YYYY-MM-DD format: 2003-02-10
Converted Date Format: 10-02-2003
PS C:\Users\rimsha\OneDrive\Desktop\rimsha python assessment>
```