AI ASSISTED CODING

ASSIGNMENT-8.3

NAME:P.Sanwitha

ROLL NO:2403A52L09

BATCH:50

Task 1: Email Validation using TDD

Scenario

You are developing a user registration system that requires reliable email input validation.

Requirements

- **Must contain @ and . characters**
- **Must not start or end with special characters**
- **Should not allow multiple @ symbols**
- **AI should generate test cases covering valid and invalid email formats**
- **Implement is_valid_email(email) to pass all AI-generated test cases**

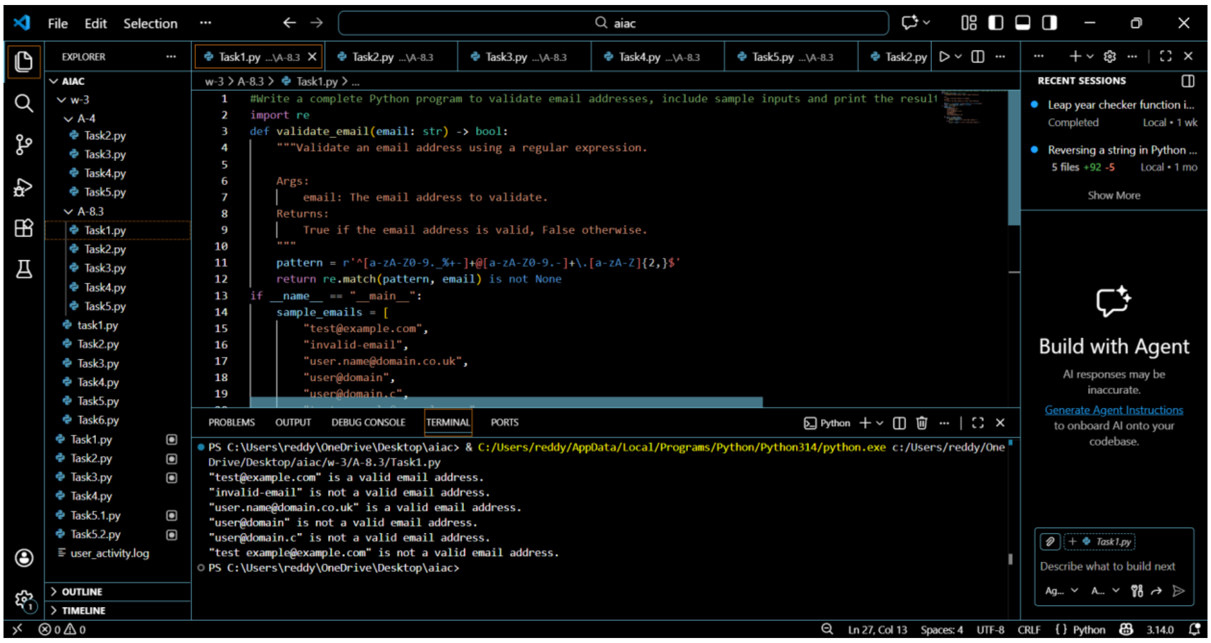
Expected Output

- **Python function for email validation**
- **All AI-generated test cases pass successfully**
- **Invalid email formats are correctly rejected**
- **Valid email formats return True**

PROMPT:

"Write a complete Python program to validate email addresses, include sample inputs and print the result so output is shown in VS Code".

OUTPUT:



```python
#Write a complete Python program to validate email addresses, include sample inputs and print the resul
import re
def validate_email(email: str) -> bool:
    """Validate an email address using a regular expression.

    Args:
        email: The email address to validate.
    Returns:
        True if the email address is valid, False otherwise.
    """
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
if __name__ == "__main__":
    sample_emails = [
        "test@example.com",
        "invalid-email",
        "user.name@domain.co.uk",
        "user@domain",
        "user@domain.c",
```

```
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
Drive/Desktop/aiac/w-3/A-8.3/Task1.py
"test@example.com" is a valid email address.
"invalid-email" is not a valid email address.
"user.name@domain.co.uk" is a valid email address.
"user@domain" is not a valid email address.
"user@domain.c" is not a valid email address.
"test example@example.com" is not a valid email address.
PS C:\Users\reddy\OneDrive\Desktop\aiac>
```

EXPLANATION:

The AI is given instructions without examples to validate email addresses. The program checks whether the email contains one @ symbol and a dot, and ensures it does not start or end with special characters. Using these simple rules, valid email addresses are accepted and invalid ones are rejected. This helps ensure correct email input.

Task 2: Grade Assignment using Loops

Scenario

You are building an automated grading system for an online examination platform.

Requirements

• AI should generate test cases for assign_grade(score) where:

– 90–100 → A

– 80–89 → B

– 70–79 → C

– 60–69 → D

– Below 60 → F

• Include boundary values (60, 70, 80, 90)

• Include invalid inputs such as -5, 105, "eighty"

• Implement the function using a test-driven approach

Expected Output

• Grade assignment function implemented in Python

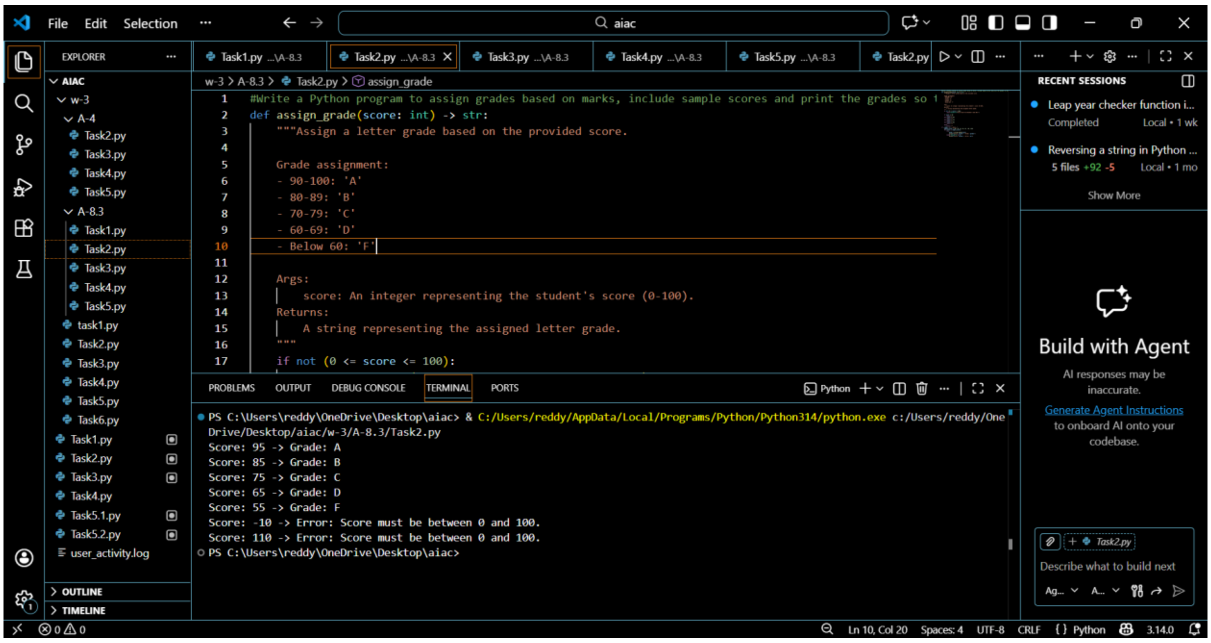
• Boundary values handled correctly

- **Invalid inputs handled gracefully**
- **All AI-generated test cases pass**

PROMPT:

Write a Python program to assign grades based on marks, include sample scores and print the grades so the program produces output.

OUTPUT:



```python
#Write a Python program to assign grades based on marks, include sample scores and print the grades so t
def assign_grade(score: int) -> str:
    """Assign a letter grade based on the provided score.

    Grade assignment:
    - 90-100: 'A'
    - 80-89: 'B'
    - 70-79: 'C'
    - 60-69: 'D'
    - Below 60: 'F'

    Args:
        score: An integer representing the student's score (0-100).
    Returns:
        A string representing the assigned letter grade.
    """
    if not (0 <= score <= 100):
```

```
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
Drive/Desktop/aiac/w-3/A-8.3/Task2.py
Score: 95 -> Grade: A
Score: 85 -> Grade: B
Score: 75 -> Grade: C
Score: 65 -> Grade: D
Score: 55 -> Grade: F
Score: -10 -> Error: Score must be between 0 and 100.
Score: 110 -> Error: Score must be between 0 and 100.
PS C:\Users\reddy\OneDrive\Desktop\aiac>
```

EXPLNATION:

The AI receives grading rules without sample inputs. The program uses conditional statements to assign grades based on the given score ranges. Boundary values and invalid inputs are handled correctly. This ensures accurate and reliable grade assignment.

Task 3: Sentence Palindrome Checker

Scenario

You are developing a text-processing utility to analyze sentences.

Requirements

- **AI should generate test cases for is_sentence_palindrome(sentence)**
- **Ignore case, spaces, and punctuation**
- **Test both palindromic and non-palindromic sentences**
- **Example:**

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

Expected Output

- **Function correctly identifies sentence palindromes**

• Case and punctuation are ignored

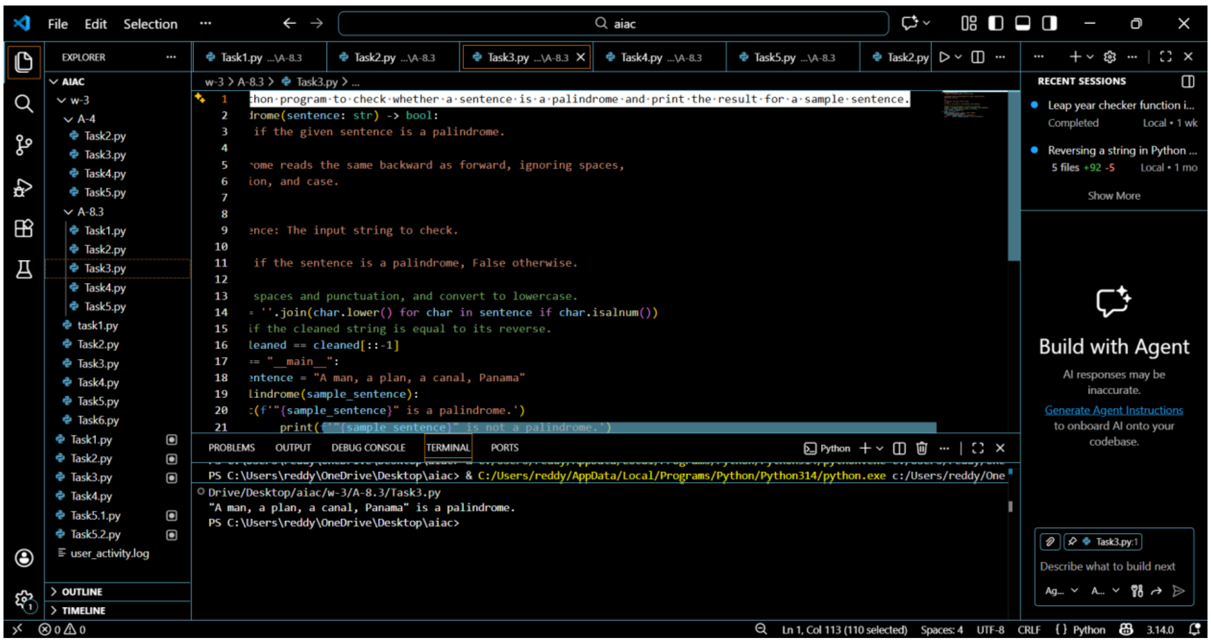
• Returns True or False accurately

• All AI-generated test cases pass

PROMPT:

Write a Python program to check whether a sentence is a palindrome and print the result for a sample sentence.

OUTPUT:



```
thon program to check whether a sentence is a palindrome and print the result for a sample sentence.
drome(sentence: str) -> bool:
 if the given sentence is a palindrome.

rome reads the same backward as forward, ignoring spaces,
ion, and case.


ence: The input string to check.

 if the sentence is a palindrome, False otherwise.

 spaces and punctuation, and convert to lowercase.
= ''.join(char.lower() for char in sentence if char.isalnum())
if the cleaned string is equal to its reverse.
leaned == cleaned[::-1]
== "__main__":
entence = "A man, a plan, a canal, Panama"
lindrome(sample_sentence):
:(f'"{sample_sentence}" is a palindrome.')
    print(f'"{sample_sentence}" is not a palindrome.')
```

```
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
Drive/Desktop/aiac/w-3/A-8.3/Task3.py
"A man, a plan, a canal, Panama" is a palindrome.
PS C:\Users\reddy\OneDrive\Desktop\aiac>
```

EXPLANTION:

The AI is instructed to identify sentence palindromes without examples. The program removes spaces, punctuation, and converts letters to lowercase before checking. It then compares the cleaned sentence with its reverse. This allows correct identification of palindromic and non-palindromic sentences.

Task 4: ShoppingCart Class

Scenario

You are designing a basic shopping cart module for an e-commerce application.

Requirements

• AI should generate test cases for the ShoppingCart class

• Class must include the following methods:

– add_item(name, price)

– remove_item(name)

– total_cost()

• Validate correct addition, removal, and cost calculation

• Handle empty cart scenarios

Expected Output

• Fully implemented ShoppingCart class

• All methods pass AI-generated test cases

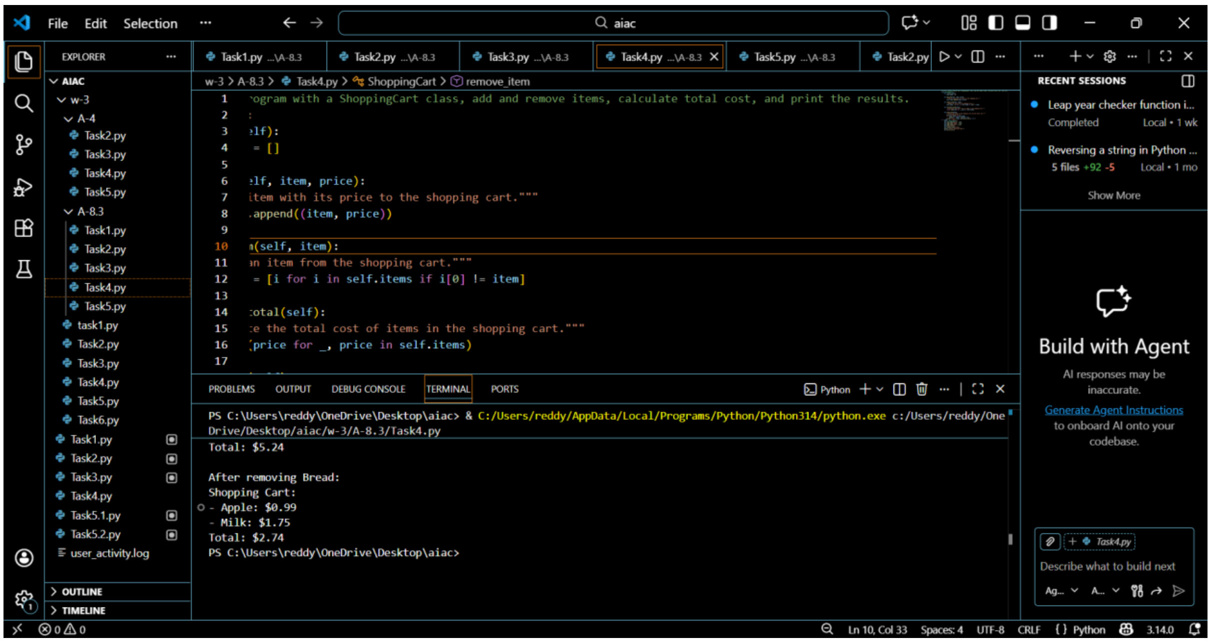
• Total cost is calculated accurately

• Items are added and removed correctly

PROMPT:

Write a Python program with a ShoppingCart class, add and remove items, calculate total cost, and print the results.

OUTPUT:



```
ogram with a ShoppingCart class, add and remove items, calculate total cost, and print the results.
:
elf):
 = []

elf, item, price):
item with its price to the shopping cart."""
.append((item, price))

n(self, item):
n item from the shopping cart."""
 = [i for i in self.items if i[0] != item]

otal(self):
e the total cost of items in the shopping cart."""
price for _, price in self.items)
```

```
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
Drive/Desktop/aiac/w-3/A-8.3/Task4.py
Total: $5.24

After removing Bread:
Shopping Cart:
- Apple: $0.99
- Milk: $1.75
Total: $2.74
PS C:\Users\reddy\OneDrive\Desktop\aiac>
```

EXPLANATION:

The AI is asked to create a shopping cart system without sample code. The program allows adding and removing items and calculates the total cost. It also handles empty cart cases properly. This ensures the shopping cart works correctly.

Task 5: Date Format Conversion

Scenario

You are creating a utility function to convert date formats for reports.

Requirements

• AI should generate test cases for convert_date_format(date_str)

• Input format must be "YYYY-MM-DD"

• Output format must be "DD-MM-YYYY"

• Example:

– "2023-10-15" → "15-10-2023"

Expected Output

• Date conversion function implemented in Python

• Correct format conversion for all valid inputs

• All AI-generated test cases pass successfully

Note: Report should be submitted as a word document for all tasks in a

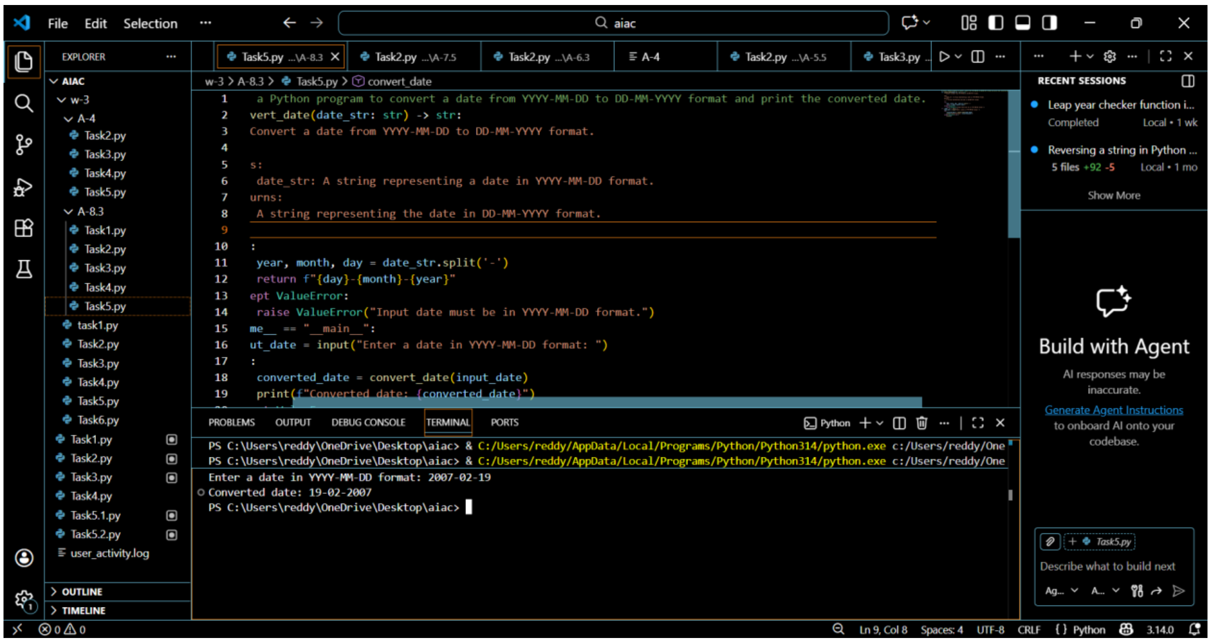
single document with prompts, comments & code explanation, and output

and if required, screenshots.

PROMPT:

Write a Python program to convert a date from YYYY-MM-DD to DD-MM-YYYY format and print the converted date.

OUTPUT:



```
 a Python program to convert a date from YYYY-MM-DD to DD-MM-YYYY format and print the converted date.
vert_date(date_str: str) -> str:
Convert a date from YYYY-MM-DD to DD-MM-YYYY format.

s:
 date_str: A string representing a date in YYYY-MM-DD format.
urns:
 A string representing the date in DD-MM-YYYY format.


:
 year, month, day = date_str.split('-')
 return f"{day}-{month}-{year}"
ept ValueError:
 raise ValueError("Input date must be in YYYY-MM-DD format.")
me__ == "__main__":
ut_date = input("Enter a date in YYYY-MM-DD format: ")
:
 converted_date = convert_date(input_date)
 print(f"Converted date: {converted_date}")
```

```
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
PS C:\Users\reddy\OneDrive\Desktop\aiac> & C:/Users/reddy/AppData/Local/Programs/Python/Python314/python.exe c:/Users/reddy/One
Enter a date in YYYY-MM-DD format: 2007-02-19
Converted date: 19-02-2007
PS C:\Users\reddy\OneDrive\Desktop\aiac>
```

EXPLANATION:

The AI is given the required input and output date formats. The program splits the date string and rearranges the parts into the new format. This converts dates accurately. The solution uses basic string operations to achieve the result.