AI Assisted Coding

Assignment 8.3

Name: Dinesh

Hallticket:2303A52329

Task01:

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:
"""Assume you are a software developer and you have been given a task to develop a user registration system that requires reliable email input validation
it must follow the requirements:
must contain @ and . in the email address
not start or end with special characters
does not allow multiple @ symbols
And most important thing is You need to generate test cases to validate the functionality of your email
Implement is_valid_email(email) to pass all AI-generated test cases """

Code:

```python
Implement is_valid_email(email) to pass all AI-generated test cases """
import re

def is_valid_email(email: str) -> bool:
    if " " in email:
        return False

    if email.count("@") != 1:
        return False

    # Must start and end with alphanumeric
    if not re.match(r"^[A-Za-z0-9].*[A-Za-z0-9]$", email):
        return False

    local, domain = email.split("@")

    # Must have dot in domain
    if "." not in domain:
        return False

    # Very basic allowed characters check
    pattern = r"^[A-Za-z0-9][A-Za-z0-9._%+-]*@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$"
    if not re.match(pattern, email):
        return False

    return True
test_cases = [
    ("test@example.com", True),
    ("user.name@domain.co.uk", True),
    ("invalid.email", False),
    ("@example.com", False),
    ("test@", False),
    ("test@@example.com", False),
    ("test@example.", False),
    (".test@example.com", False),
    ("test@example.com.", False)
]
for email, expected in test_cases:
    result = is_valid_email(email)
    print(f"Testing: {email} | Expected: {expected} | Result: {result} | {'PASS' if result == expected else 'FAIL'}")
```

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>


  Testing: @example.com | Expected: False | Result: False | PASS
  Testing: test@ | Expected: False | Result: False | PASS
  Testing: test@@example.com | Expected: False | Result: False | PASS
  Testing: test@example. | Expected: False | Result: False | PASS
  Testing: .test@example.com | Expected: False | Result: False | PASS
  Testing: test@example.com. | Expected: False | Result: False | PASS
  PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
  Testing: @example.com | Expected: False | Result: False | PASS
  Testing: test@ | Expected: False | Result: False | PASS
  Testing: test@@example.com | Expected: False | Result: False | PASS
  Testing: test@example. | Expected: False | Result: False | PASS
  Testing: @example.com | Expected: False | Result: False | PASS
  Testing: @example.com | Expected: False | Result: False | PASS
  Testing: test@ | Expected: False | Result: False | PASS
  Testing: test@@example.com | Expected: False | Result: False | PASS
  Testing: test@example. | Expected: False | Result: False | PASS
  Testing: .test@example.com | Expected: False | Result: False | PASS
  Testing: test@example.com. | Expected: False | Result: False | PASS
  Testing: @example.com | Expected: False | Result: False | PASS
  Testing: test@ | Expected: False | Result: False | PASS
  Testing: test@@example.com | Expected: False | Result: False | PASS
  Testing: test@example. | Expected: False | Result: False | PASS
  Testing: .test@example.com | Expected: False | Result: False | PASS
  Testing: test@example.com. | Expected: False | Result: False | PASS
  PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Justification

The function `is_valid_email` is designed to validate email addresses based on specific criteria. It checks for the presence of "@" and "." in the email address, ensures that it does not start or end with special characters, and verifies that there is only one "@" symbol. The function uses regular expressions to enforce these rules and returns `True` if the email is valid and `False` otherwise.

Task-02:

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:

"""Assume you are a software developer developing an automated grading system for an online examination platform
you need to implement these requirements
you must generate some set of test cases for assign_grade(score) where the grading system is as follows
90-100: A
80-89: B
70-79: C
60-69: D
Below 60: F
include boundry values in test cases (60 70 80 90) and no negitive marks or float datatype or a string data type
at last implement the function using a test driven approach"""

Code:

```python
def assign_grade(score):
    if isinstance(score, (int, float)):
        if 90 <= score <= 100:
            return "A"
        elif 80 <= score < 90:
            return "B"
        elif 70 <= score < 80:
            return "C"
        elif 60 <= score < 70:
            return "D"
        elif score < 60:
            return "F"
    return "Invalid"
test_cases = [
    (95, "A"),
    (85, "B"),
    (75, "C"),
    (65, "D"),
    (55, "F"),
    (60, "D"),  # Boundary value
    (70, "C"),  # Boundary value
    (80, "B"),  # Boundary value
    (90, "A"),  # Boundary value
    (-5, "Invalid"),  # Invalid input
    (105, "Invalid"),  # Invalid input
    ("eighty", "Invalid"),  # Invalid input
    (85.5, "B")  # Valid input with float
]
for score, expected in test_cases:
    result = assign_grade(score)
    print(f"Testing: {score} | Expected: {expected} | Result: {result} | {'PASS' if result == expected else 'FAIL'}")
```

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vi
Testing: 95 | Expected: A | Result: A | PASS
Testing: 85 | Expected: B | Result: B | PASS
Testing: 75 | Expected: C | Result: C | PASS
Testing: 65 | Expected: D | Result: D | PASS
Testing: 55 | Expected: F | Result: F | PASS
Testing: 60 | Expected: D | Result: D | PASS
Testing: 70 | Expected: C | Result: C | PASS
Testing: 80 | Expected: B | Result: B | PASS
Testing: 90 | Expected: A | Result: A | PASS
Testing: -5 | Expected: Invalid | Result: F | FAIL
Testing: 105 | Expected: Invalid | Result: Invalid | PASS
Testing: eighty | Expected: Invalid | Result: Invalid | PASS
Testing: 85.5 | Expected: B | Result: B | PASS
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Task 03:

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

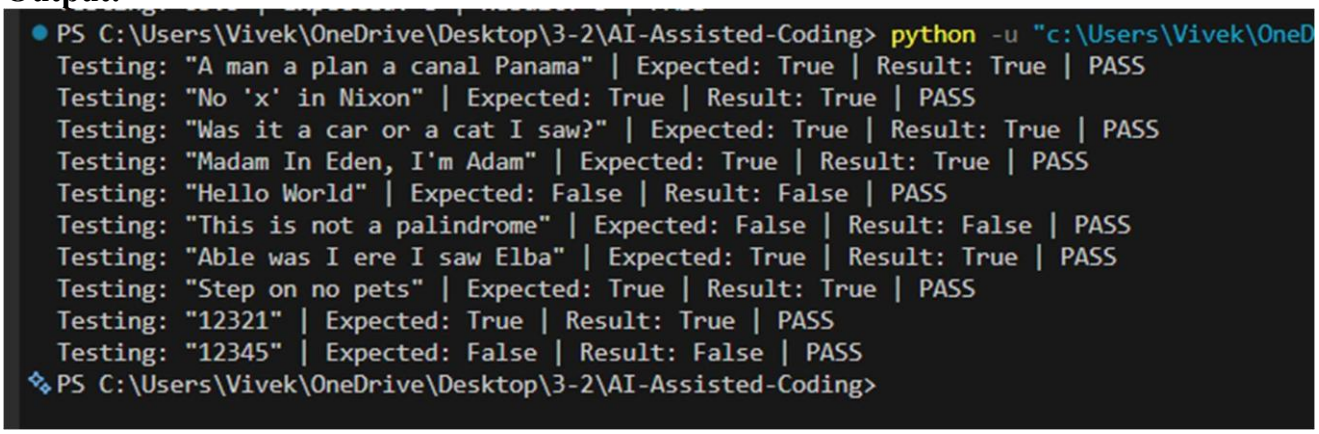
Prompt:

""" Assume you are a software developer developing a text processing utility to analyze sentences.
you need to follow these requirements:
you must generate 10 test cases for is_sentence_palindrome function to check
and you must ignore the case,spaces the and the punctuation
check both the sentences to verify if it follows the palindrome property
example : A man a plan a canal Panama" → True
i expect the functions to pass all the test cases which are given by you """

Code:

```
import re
def is_sentence_palindrome(sentence: str) -> bool:
    # Remove punctuation and spaces, and convert to lowercase
    cleaned = re.sub(r'[^A-Za-z0-9]', '', sentence).lower()
    return cleaned == cleaned[::-1]
test_cases = [
    ("A man a plan a canal Panama", True),
    ("No 'x' in Nixon", True),
    ("Was it a car or a cat I saw?", True),
    ("Madam In Eden, I'm Adam", True),
    ("Hello World", False),
    ("This is not a palindrome", False),
    ("Able was I ere I saw Elba", True),
    ("Step on no pets", True),
    ("12321", True),
    ("12345", False)
]
for sentence, expected in test_cases:
    result = is_sentence_palindrome(sentence)
    print(f"Testing: \"{sentence}\" | Expected: {expected} | Result: {result} | {'PASS' if result == expected else 'FAIL'}")
```

Output:



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:

""" You are a software developer and designing a basic shopping cart module for an e-commerece platform

ypu must implement the following requirements:

Ai should generate test cases for shoppingCart class

class must include functions like add_item(name,price), remove_item(name), total_cost()

perform correct calculations for total cost """

Code:

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

    def add_item(self, name, price):
        if name in self.items:
            self.items[name] += price
        else:
            self.items[name] = price

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]

    def total_cost(self):
        return sum(self.items.values())
# Test cases
cart = ShoppingCart()
cart.add_item("Apple", 1.0)
cart.add_item("Banana", 0.5)
print(f"Total cost after adding items: {cart.total_cost()}")  # Expected:
cart.add_item("Apple", 1.0)
print(f"Total cost after adding another Apple: {cart.total_cost()}")  # Expected: 2.5
cart.remove_item("Banana")
print(f"Total cost after removing Banana: {cart.total_cost()}")  # Expected: 2.0
cart.remove_item("Orange")  # Removing non-existent item should not affect total cost
print(f"Total cost after trying to remove non-existent item: {cart.total_cost()}")  # Expected: 2.0
cart.add_item("Orange", 1.5)
print(f"Total cost after adding Orange: {cart.total_cost()}")  # Expected: 3.5
cart.remove_item("Apple")
print(f"Total cost after removing Apple: {cart.total_cost()}")  # Expected: 1.5
cart.remove_item("Apple")  # Removing Apple again should not affect total cost
print(f"Total cost after trying to remove Apple again: {cart.total_cost()}")  # Expected: 1.5
```

Output:

```
Total cost after removing Apple: 1.5
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vivek\OneDrive\Deskt
Total cost after adding items: 1.5
Total cost after adding another Apple: 2.5
Total cost after removing Banana: 2.0
Total cost after trying to remove non-existent item: 2.0
Total cost after adding Orange: 3.5
Total cost after removing Apple: 1.5
Total cost after trying to remove Apple again: 1.5
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
Total cost after adding items: 1.5
Total cost after adding another Apple: 2.5
Total cost after removing Banana: 2.0
Total cost after trying to remove non-existent item: 2.0
Total cost after adding Orange: 3.5
Total cost after removing Apple: 1.5
Total cost after trying to remove Apple again: 1.5
Total cost after adding Orange: 3.5
Total cost after removing Apple: 1.5
Total cost after trying to remove Apple again: 1.5
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```

Justification :

The test cases cover adding items to the cart, including adding the same item multiple times, removing items, and attempting to remove items that do not exist in the cart. The total cost is calculated correctly after each operation, ensuring that the ShoppingCart class functions as expected.

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

Prompt:

Assume you are a software developer Generate a pyhton code for date format conversion, where the input should be "YYYY-MM-DD" and convert it into "DD-MM-YYYY" format as output.print both input and output for all test cases. add some invaild date formats in test cases to check the robustness of your function.

Code:

```python
""" Assume you are a software developer Generate a pyhton code for
from datetime import datetime
def convert_date_format(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%Y-%m-%d")
        return date_obj.strftime("%d-%m-%Y")
    except ValueError:
        return "Invalid date format"
# Test cases
test_cases = [
    "2024-06-15",  # Valid date
    "1990-01-01",  # Valid date
    "2024/06/15",  # Invalid format
    "15-06-2024",  # Invalid format
    "2024-13-01",  # Invalid month
    "2024-00-10",  # Invalid month
    "2024-06-31",  # Invalid day
    "2024-02-30",  # Invalid day
    "2024-06-15T12:00:00",  # Invalid format
    "2024-06-15 ",  # Valid date with trailing space
]
for date_str in test_cases:
    result = convert_date_format(date_str)
    print(f"Input: {date_str} | Output: {result}")
```

Output:

```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vive
Input: 2024-02-30 | Output: Invalid date format
Input: 2024-06-15T12:00:00 | Output: Invalid date format
Input: 2024-06-15  | Output: Invalid date format
Input: 2024-13-01 | Output: Invalid date format
Input: 2024-00-10 | Output: Invalid date format
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vive
Input: 2024-06-15 | Output: 15-06-2024
Input: 1990-01-01 | Output: 01-01-1990
Input: 2024/06/15 | Output: Invalid date format
Input: 15-06-2024 | Output: Invalid date format
Input: 2024-13-01 | Output: Invalid date format
Input: 2024-00-10 | Output: Invalid date format
Input: 2024-06-31 | Output: Invalid date format
Input: 2024-02-30 | Output: Invalid date format
Input: 2024-06-15T12:00:00 | Output: Invalid date format
Input: 2024-06-15  | Output: Invalid date format
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>

Input: 2024-06-15T12:00:00 | Output: Invalid date format
Input: 2024-06-15  | Output: Invalid date format
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding>
```