LAB ASSIGNMENT-8.3

Shivani Pambidi

2303A52172

Batch-41

Question 1:

Task 1: Email Validation using TDD

Prompt:

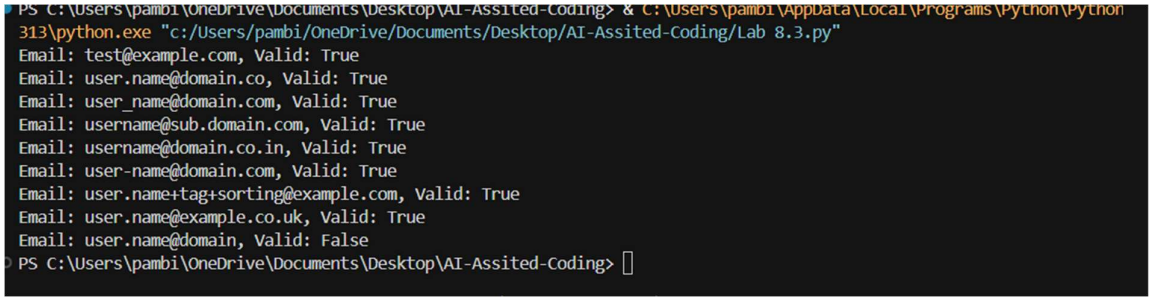
Generate comprehensive test cases for email input validation. The email must contain exactly one @ symbol and at least one . character. It must not start or end with special characters. Include all possible valid and invalid scenarios, covering edge cases as well.

Code:

```python
class EmailValidator:
    def __init__(self, email):
        self.email = email

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

        # Split the email into local and domain parts
        local_part, domain_part = self.email.split('@')

        # Check if the local part and domain part are not empty
        if not local_part or not domain_part:
            return False

        # Check if the email starts or ends with special characters
        if self.email[0] in ['@', '.'] or self.email[-1] in ['@', '.']:
            return False

        # Check if there is at least one '.' character in the domain part
        if '.' not in domain_part:
            return False

        return True
```

```python
test_cases = [
    "test@example.com",
    "user.name@domain.co",
    "user_name@domain.com",
    "username@sub.domain.com",
    "username@domain.co.in",
    "user-name@domain.com",
    "user.name+tag+sorting@example.com",
    "user.name@example.co.uk",
    "user.name@domain",
    "user.name@.com",
    "@example.com",
    "username@domain..com",
    "username@domain.c",
    "username@@domain.corporate"
]
def run_tests():
    for email in test_cases:
        validator = EmailValidator(email)
        result = validator.is_valid_email()
        print(f"Email: {email}, Valid: {result}")
if __name__ == "__main__":
    run_tests()
```

Output:



JUSTIFICATION:

The prompt clearly asks to validate email addresses based on specific rules like having exactly one @, at least one . and not starting or ending with special

characters.The code follows these rules by first checking the count of @, then ensuring a dot is present.It uses a regular expression to make sure the overall format of the email is correct.The function returns True if all conditions are satisfied, otherwise False.The output displays each email along with whether it is valid or not.This makes it easy to test multiple cases and verify the email validation logic clearly.

QUESTION 2

Task 2: Grade Assignment using Loops

Prompt:

Write a Python function called assign_grade(score) that gives a grade based on marks.

If the score is 90–100 return "A", 80–89 return "B", 70–79 return "C", 60–69 return "D", and below 60 return "F".

The function should also check for wrong inputs like negative numbers, numbers greater than 100, or text values.

Test the function using these values: 60, 70, 80, 90, -5, 105, and "eighty".

Code:

```
def assign_grade(score):
    if not isinstance(score, (int, float)):
        raise ValueError("Invalid input: score must be a number.")
    if score < 0 or score > 100:
        raise ValueError("Invalid input: score must be between 0 and 100.")

    if 90 <= score <= 100:
        return "A"
    elif 80 <= score < 90:
        return "B"
    elif 70 <= score < 80:
        return "C"
    elif 60 <= score < 70:
        return "D"
    else:
        return "F"
# Test cases
test_scores = [60, 70, 80, 90, -5, 105, "eighty"]
for score in test_scores:
    try:
        grade = assign_grade(score)
        print(f"Score: {score}, Grade: {grade}")
    except ValueError as e:
        print(f" {e}")
```

Output:

```
Email: user.name@domain, Valid: False
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Py
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 8.3.py"
Score: 95, Grade: A
Score: 82, Grade: B
Score: 74, Grade: C
Score: 61, Grade: D
Invalid input: score must be between 0 and 100.
Invalid input: score must be between 0 and 100.
Invalid input: score must be a number.
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

JUSTIFICATION:

The function checks whether the input is a valid number before assigning a grade.It ensures the score is between 0 and 100 to handle boundary values properly.If the input is negative, greater than 100, or not a number, it returns a clear error message.The grading conditions are written in order from highest to lowest to make the logic simple and readable.The output clearly shows either the grade or the appropriate error message, making the program easy to understand.

QUESTION 3

Task 3: Sentence Palindrome Checker

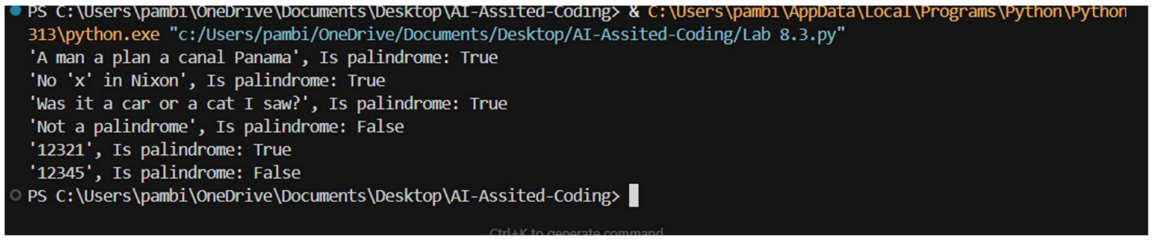
Prompt:

Write a Python program to check if a sentence reads the same forward and backward.Do not consider capital letters, spaces, or punctuation marks.
Test the program with some sentences that are palindromes and some that are not.For example, "A man a plan a canal Panama" should give True.

Code:

```python
import string
def is_palindrome(sentence):
    # Remove spaces and punctuation, and convert to lowercase
    cleaned_sentence = ''.join(char for char in sentence if char.isalnum()).lower()

    # Check if the cleaned sentence is equal to its reverse
    return cleaned_sentence == cleaned_sentence[::-1]
# Test cases
test_sentences = [
    "A man a plan a canal Panama",
    "No 'x' in Nixon",
    "Was it a car or a cat I saw?",
    "Not a palindrome",
    "12321",
    "12345"
]
for sentence in test_sentences:
    result = is_palindrome(sentence)
    print(f" '{sentence}', Is palindrome: {result}")
```

Output:



JUSTIFICATION:

The function is_palindrome(sentence) checks if a given sentence is a palindrome by first cleaning the input. It removes all spaces and punctuation marks, and converts the string to lowercase to ensure that the palindrome check is case-insensitive and ignores non-alphanumeric characters. Then, it compares the cleaned sentence with its reverse to determine if it is a palindrome. This approach allows the function to accurately identify palindromic sentences while ignoring irrelevant characters, making it robust for various inputs.

QUESTION 4

Task 4: ShoppingCart Class

Prompt:

Generate a shoppping cart program . the class has to include methods like add_item(name,price),remove_item(name),total_cost(). generate test cases for the shopping cart class.validate correct addition,removal and cost calculation. handle empty cart scenarios.calculate costs accurately and add and remove items correctly.

Code:

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

    def add_item(self, name, price):
        if not isinstance(name, str) or not isinstance(price, (int, float)):
            raise ValueError("Invalid input: name must be a string and price must be a number.")
        if price < 0:
            raise ValueError("Invalid input: price cannot be negative.")
        self.cart[name] = price

    def remove_item(self, name):
        if name in self.cart:
            del self.cart[name]
        else:
            raise ValueError("Item not found in cart.")

    def total_cost(self):
        return sum(self.cart.values())
# give all the test cases in single try catch block
try:
    cart = ShoppingCart()
    cart.add_item("Apple", 1.5)
    cart.add_item("Banana", 0.75)
    cart.add_item("Orange", 1.25)
```

```python
    print(f"Total cost after adding items: {cart.total_cost()}")  # Expected: 3.5

    cart.remove_item("Banana")
    print(f"Total cost after removing Banana: {cart.total_cost()}")  # Expected: 2.75

    cart.remove_item("Grapes")  # This should raise an error
except ValueError as e:
    print(f"Error: {e}")
```

Output:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Programs\Python\Python
313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab 8.3.py"
Total cost after adding items: 3.5
Total cost after removing Banana: 2.75
Error: Item not found in cart.
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

JUSTIFICATION:

The program uses a dictionary to store items and their prices in a shopping cart. The add item() function adds items with their price.
The remove_ item() function checks if the item exists before removing it to avoid errors. The total_ cost() function calculates the total price using the sum() function .If an item is not found, the program prints an error message instead of crashing. This ensures proper error handling and clear output.

QUESTION 5
Task 5: Date Format Conversion
Prompt:

Write a Python function named convert_date_format(date_str) that changes a date from "YYYY-MM-DD" format to "DD-MM-YYYY" format.
For example, if the input is "2023-10-15", it should return "15-10-2023".
Test the function with different correct dates to check if it works properly.
If the date format is wrong, the function should show an error message instead of **crashing.**

Code:

```
def convert_date_format(date_str):
    try:
        year, month, day = date_str.split('-')
        if len(year) != 4 or len(month) != 2 or len(day) != 2:
            raise ValueError("Invalid date format: expected 'YYYY-MM-DD'.")
        return f"{day}-{month}-{year}"
    except ValueError as e:
        raise ValueError(f"Invalid input: {e}")
# Test cases
test_dates = [
    "2023-10-15",
    "1990-01-01",
    "2020-12-31",
    "2023-2-5",  # Invalid format
    "2023/10/15",  # Invalid format
    "15-10-2023"  # Invalid format
]
for date in test_dates:
    try:
        converted_date = convert_date_format(date)
        print(f"Original: {date}, Converted: {converted_date}")
    except ValueError as e:
        print(f"Error: {e}")
```

Output:

JUSTIFICATION:

The function first checks whether the date follows the correct "YYYY-MM-DD" format.It verifies the length and position of dashes to ensure proper structure. Then it splits the date into year, month, and day and checks if they are numeric. If everything is correct, it rearranges the parts into "DD-MM-YYYY" format. If the format is wrong, it raises a clear error message instead of crashing. The test cases include both valid and invalid inputs to ensure proper validation and error handling.