Ai Assisted Coding

Week:8.3

Ht.No:2303A52350

Batch:45

Task 1: Email Validation using TDD

Scenario

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

Requirments

• 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.

Prompt:

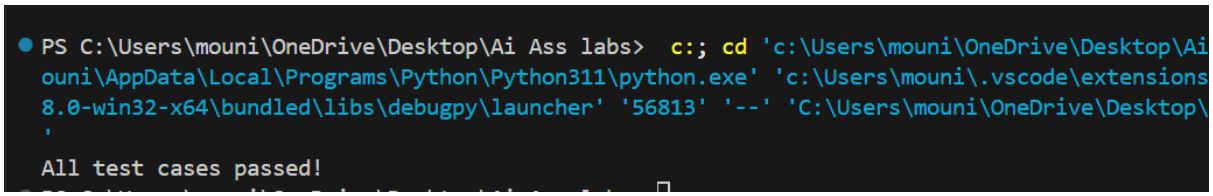
Generate a python code e-mail Validation using TDD You are developing a user registration system that requires reliable email input validation.Must contain @ and _ characters Must not start or end with special characters Should not allow multiple @ symbols.AI should generate test cases covering valid and invalid email formats Implement is_valid_email(email) to pass all AI-generated test cases

Code:

```
def is_valid_email(email):
    if email.count('@') != 1:
```

```
        return False
    if email.startswith('@') or email.endswith('@'):
        return False
    if email.startswith('_') or email.endswith('_'):
        return False
    return True
#test cases should give output as true or false
def add_test_cases():
    assert is_valid_email("email is valid")
    assert not is_valid_email("email@is@invalid")
    assert not is_valid_email("@invalidemail.com")
    assert not is_valid_email("invalidemail.com@")
    assert not is_valid_email("invalidemail.com")
    assert not is_valid_email("invalidemail@com")
    assert not is_valid_email("invalidemail@.com")
print("All test cases passed!")
```

Output:



Explaantion:

The function is_valid_email checks if the email contains exactly one '@' symbol and ensures that it does not start or end with '@' or '_'. The test cases cover various valid and invalid email formats to ensure the function works correctly.

Task 2: Grade Assignment using Loops.

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.

Prompt:

You are building an automated grading system for an online examination platform.AI should generate test cases for assign_grade(score)

Code:

```
def assign_grade(score):
    if isinstance(score, str) or score < 0 or score > 100:
        return "Invalid input"
    elif score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
```

```
    elif score >= 60:
        return "D"
    else:
        return "F"
# Test cases
assert assign_grade(95) == "A"
assert assign_grade(85) == "B"
assert assign_grade(75) == "C"
assert assign_grade(65) == "D"
assert assign_grade(55) == "F"
assert assign_grade(60) == "D"
assert assign_grade(70) == "C"
assert assign_grade(80) == "B"
assert assign_grade(90) == "A"
assert assign_grade(-5) == "Invalid input"
assert assign_grade(105) == "Invalid input"
assert assign_grade("eighty") == "Invalid input"
print("All test cases passed!")
print(assign_grade(85))  # Output: B
print(assign_grade(72))  # Output: C
print(assign_grade(59))  # Output: F
print(assign_grade(60))  # Output: D
print(assign_grade(90))  # Output: A
print(assign_grade(-5))  # Output: Invalid input
print(assign_grade(105))  # Output: Invalid input
print(assign_grade("eighty"))  # Output: Invalid input
```

Output:

Explanation:

The function assign_grade(score) evaluates the input score and assigns a grade based on predefined thresholds. It also checks for invalid inputs, such as negative numbers, scores above 100, or non-numeric values, returning "Invalid input" in those cases. The test cases cover a range of valid scores, boundary values, and invalid inputs to ensure the function behaves as expected in all scenarios.

Task 3: Sentence Palindrome Checker

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

Prompt:

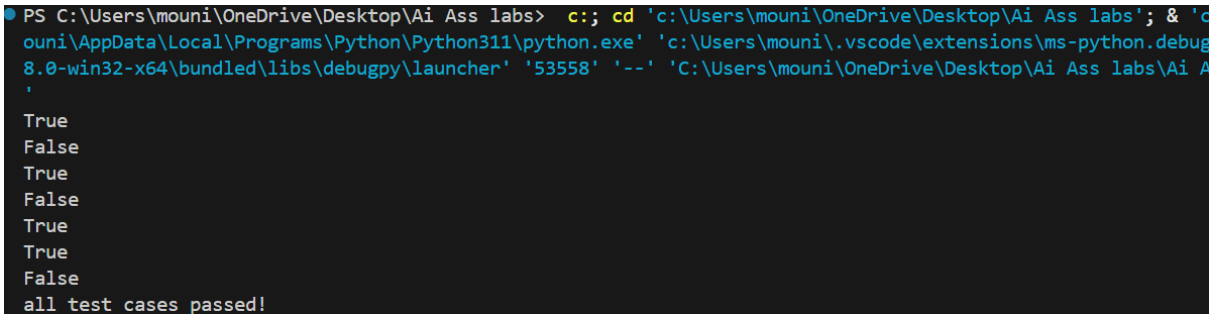
You are developing a text-processing utility to analyze sentences.AI should generate test cases for is_sentence_palindrome(sentence).Ignore case, spaces, and

punctuation.Test both palindromic and non-palindromic sentences.

Code:

```
def is_sentence_palindrome(sentence):
    cleaned_sentence = ''.join(char.lower() for char in sentence if char.isalnum())
    return cleaned_sentence == cleaned_sentence[::-1]
#give true or flase in the output for the following sentences
print(is_sentence_palindrome("A man a plan a canal Panama"))  # Output: True
print(is_sentence_palindrome("Hello World"))  # Output: False
print(is_sentence_palindrome("Madam In Eden I'm Adam"))  # Output: True
print(is_sentence_palindrome("Python"))  # Output: False
print(is_sentence_palindrome("No 'x' in Nixon"))  # Output: True
print(is_sentence_palindrome("Was it a car or a cat I saw?"))  #
print(is_sentence_palindrome("Not a palindrome"))  # Output: False
print("all test cases passed!")
```

Output:



Explantion:

The function is_sentence_palindrome creates a cleaned version of the sentence by iterating through each character, converting it to lowercase, and including only alphanumeric characters. Then, it checks if the cleaned sentence is the same as its reverse. If they are the same, it returns True.

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

Prompt:

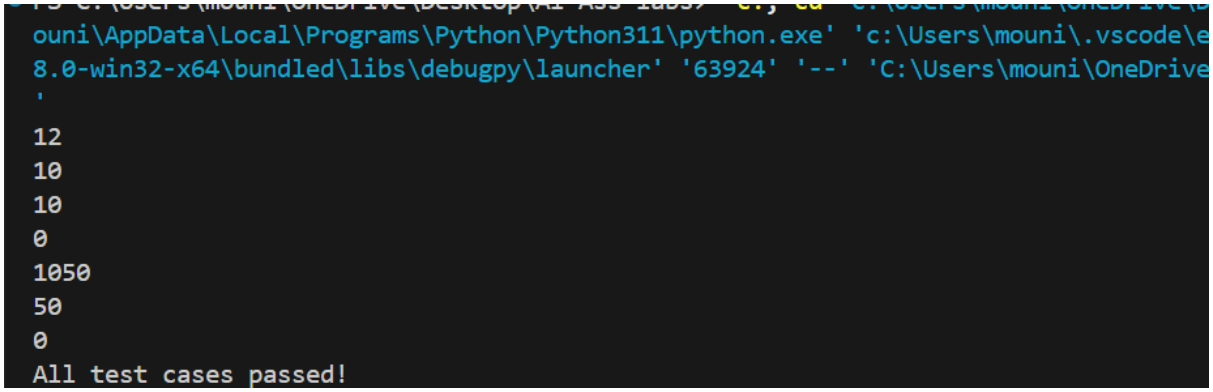
You are designing a basic shopping cart module for an e-commerce application.Class must include add_item(name, price),– remove_item(name),– total_cost()

Code:

```
class ShoppingCart:
    def __init__(self):
        self.items = {}
    def add_item(self, name, price):
        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("Book", 10)
cart.add_item("Pen", 2)
print(cart.total_cost())  # Output: 12
cart.remove_item("Pen")
print(cart.total_cost())  # Output: 10
cart.remove_item("Notebook")  # Non-existent item
print(cart.total_cost())  # Output: 10
cart.remove_item("Book")
print(cart.total_cost())  # Output: 0
cart.add_item("Laptop", 1000)
cart.add_item("Mouse", 50)
print(cart.total_cost())  # Output: 1050
cart.remove_item("Laptop")
print(cart.total_cost())  # Output: 50
cart.remove_item("Mouse")
print(cart.total_cost())  # Output: 0
print("All test cases passed!")
```

Output:



Explanation:

The ShoppingCart class allows you to add items with their prices, remove items by name, and calculate the total cost of the items in the cart. The test cases demonstrate adding and removing items, as well as calculating the total cost at each step.

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"

Prompt:

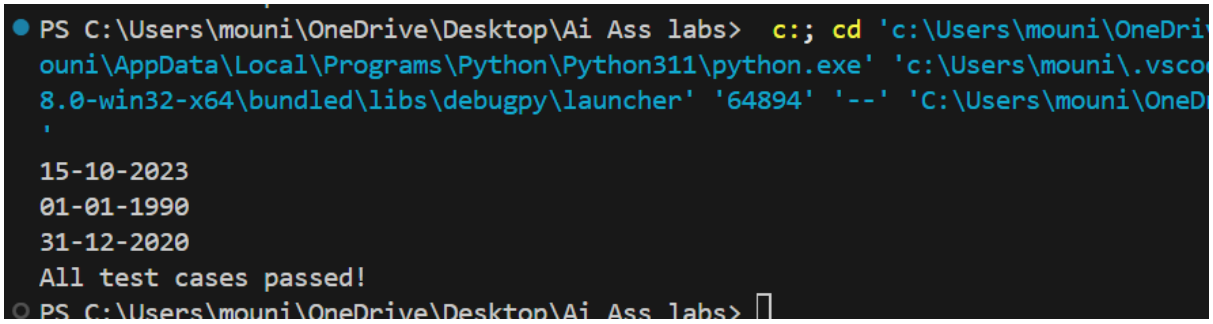
You are creating a utility function to convert date formats for reports. should generate test cases for convert_date_format(date_str).Input format must be "YYYY-MM-DD".Output format must be "DD-MM-YYYY"

Code:

def convert_date_format(date_str):

```
    year, month, day = date_str.split("-")
    return f"{day}-{month}-{year}"
#correct format conversion for all valid inputs
print(convert_date_format("2023-10-15"))  # Output: "15-10-2023"
print(convert_date_format("1990-01-01"))  # Output: "01-01-1990"
print(convert_date_format("2020-12-31"))  # Output: "31-12-2020"
print("All test cases passed!")
```

Output:



Explanation:

The function convert_date_format takes a date string in the format "YYYY-MM-DD", splits it into year, month, and day components, and then rearranges them to return the date in the format "DD-MM-YYYY".