AI Assisted Coding

Name : Vineeth-Chidurala **Date** : 16-01-2026

Ht.No. : 2303A52447

Task - 1 : Email Validation using TDD
You are developing a user registration system that requires reliable email input validation.

Prompt : Generate test cases for an email validation function called is_valid_email(email).

Validation Rules:

- Must contain exactly one '@'

- Must contain at least one '.'

- Must not start or end with special characters like '@' or '.'

- Must not allow multiple '@' symbols

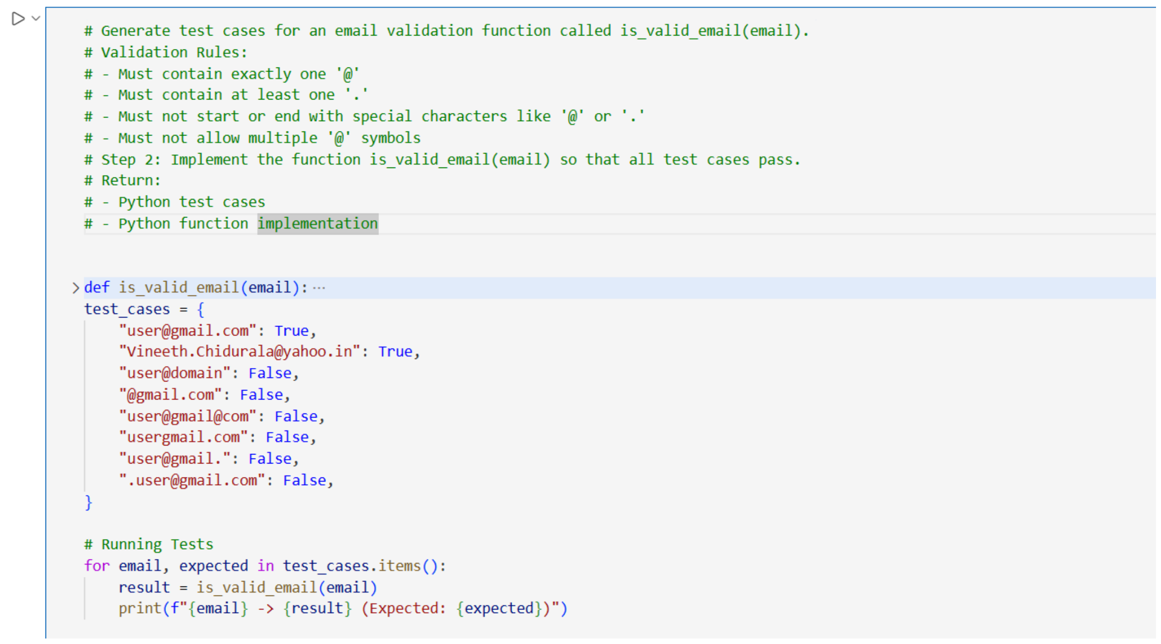
Step 2: Implement the function is_valid_email(email) so that all test cases pass.

Return:

- Python test cases

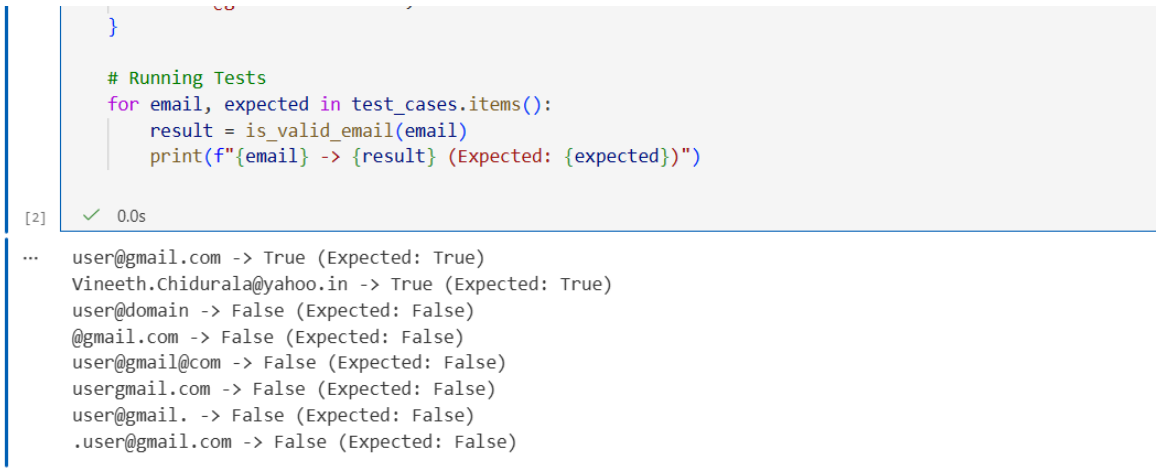
- Python function implementation

Screenshots :



```python
# Generate test cases for an email validation function called is_valid_email(email).
# Validation Rules:
# - Must contain exactly one '@'
# - Must contain at least one '.'
# - Must not start or end with special characters like '@' or '.'
# - Must not allow multiple '@' symbols
# Step 2: Implement the function is_valid_email(email) so that all test cases pass.
# Return:
# - Python test cases
# - Python function implementation


def is_valid_email(email):...
test_cases = {
    "user@gmail.com": True,
    "Vineeth.Chidurala@yahoo.in": True,
    "user@domain": False,
    "@gmail.com": False,
    "user@gmail@com": False,
    "usergmail.com": False,
    "user@gmail.": False,
    ".user@gmail.com": False,
}

# Running Tests
for email, expected in test_cases.items():
    result = is_valid_email(email)
    print(f"{email} -> {result} (Expected: {expected})")
```

Output :



```
}

# Running Tests
for email, expected in test_cases.items():
    result = is_valid_email(email)
    print(f"{email} -> {result} (Expected: {expected})")
```

[2] ✓ 0.0s

```
user@gmail.com -> True (Expected: True)
Vineeth.Chidurala@yahoo.in -> True (Expected: True)
user@domain -> False (Expected: False)
@gmail.com -> False (Expected: False)
user@gmail@com -> False (Expected: False)
usergmail.com -> False (Expected: False)
user@gmail. -> False (Expected: False)
.user@gmail.com -> False (Expected: False)
```

Task – 2 : Grade Assignment using Loops

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

Prompt : You are following Test-Driven Development (TDD).

Step 1:

Generate comprehensive test cases for a function called assign_grade(score).

Grading Rules:

90–100 → "A"

80–89 → "B"

70–79 → "C"

60–69 → "D"

Below 60 → "F"

Requirements:

Include boundary values (60, 70, 80, 90), Include edge values (0 and 100), Include invalid inputs such as:

Negative numbers (e.g., -5), Numbers greater than 100 (e.g., 105), Non-numeric values (e.g., "eighty", None), Invalid inputs must return "Invalid"

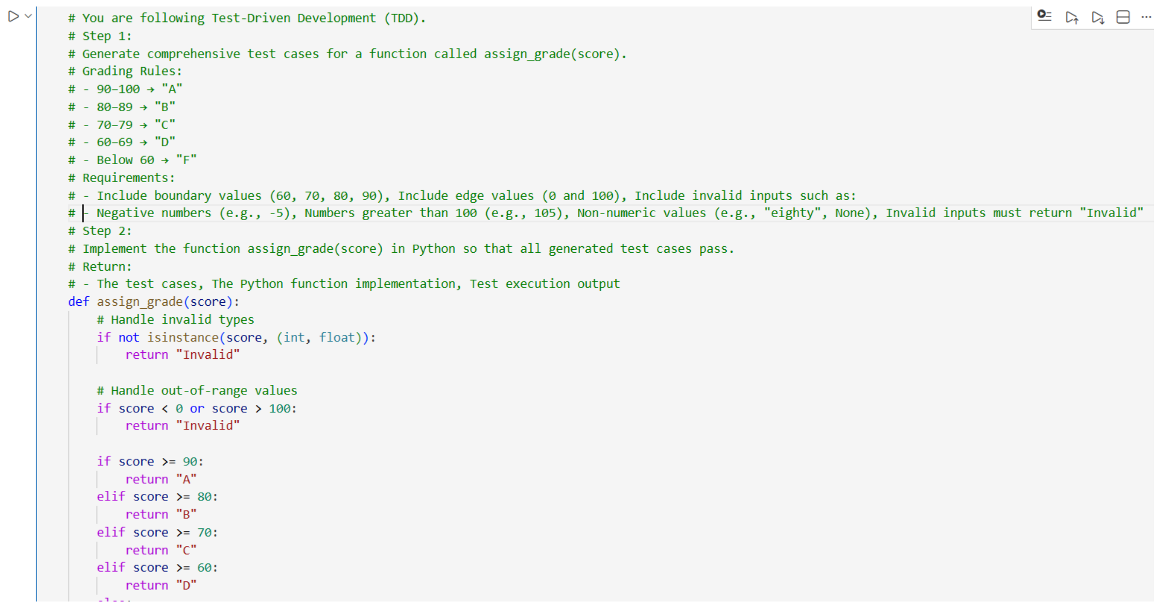
Step 2:

Implement the function assign_grade(score) in Python so that all generated test cases pass.

Return:

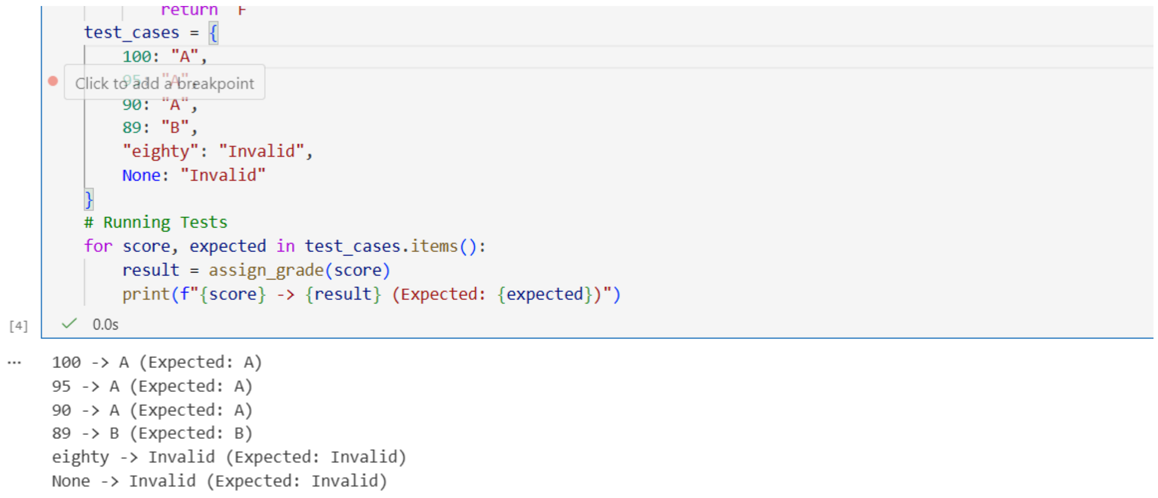
The test cases, The Python function implementation, Test execution output

ScreenShots :



```python
# You are following Test-Driven Development (TDD).
# Step 1:
# Generate comprehensive test cases for a function called assign_grade(score).
# Grading Rules:
# - 90–100 → "A"
# - 80–89 → "B"
# - 70–79 → "C"
# - 60–69 → "D"
# - Below 60 → "F"
# Requirements:
# - Include boundary values (60, 70, 80, 90), Include edge values (0 and 100), Include invalid inputs such as:
# Negative numbers (e.g., -5), Numbers greater than 100 (e.g., 105), Non-numeric values (e.g., "eighty", None), Invalid inputs must return "Invalid"
# Step 2:
# Implement the function assign_grade(score) in Python so that all generated test cases pass.
# Return:
# - The test cases, The Python function implementation, Test execution output
def assign_grade(score):
    # Handle invalid types
    if not isinstance(score, (int, float)):
        return "Invalid"

    # Handle out-of-range values
    if score < 0 or score > 100:
        return "Invalid"

    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
```

Output :



```python
        return "F"
test_cases = {
    100: "A",
    95: "A",
    90: "A",
    89: "B",
    "eighty": "Invalid",
    None: "Invalid"
}
# Running Tests
for score, expected in test_cases.items():
    result = assign_grade(score)
    print(f"{score} -> {result} (Expected: {expected})")
```

[4] ✓ 0.0s

```
100 -> A (Expected: A)
95 -> A (Expected: A)
90 -> A (Expected: A)
89 -> B (Expected: B)
eighty -> Invalid (Expected: Invalid)
None -> Invalid (Expected: Invalid)
```

Task-3 : Sentence Palindrome Checker

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

Prompt : Step 1:

Generate comprehensive test cases for a function called is_sentence_palindrome(sentence).

Requirements:

Ignore case sensitivity, Ignore spaces, Ignore punctuation characters

Test both palindromic and non-palindromic sentences,Include edge cases like:

Empty string

Single character

Sentences with only punctuation

Mixed case sentences

Example:

"A man a plan a canal Panama" → True

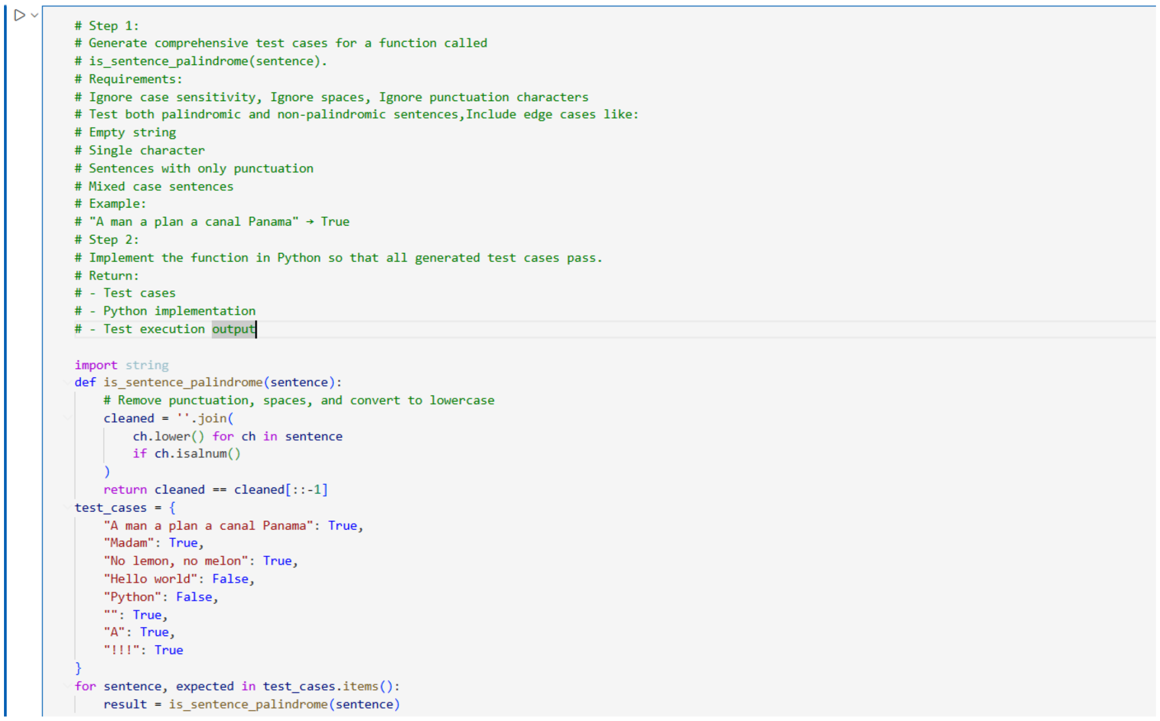
Step 2:

Implement the function in Python so that all generated test cases pass.

Return:

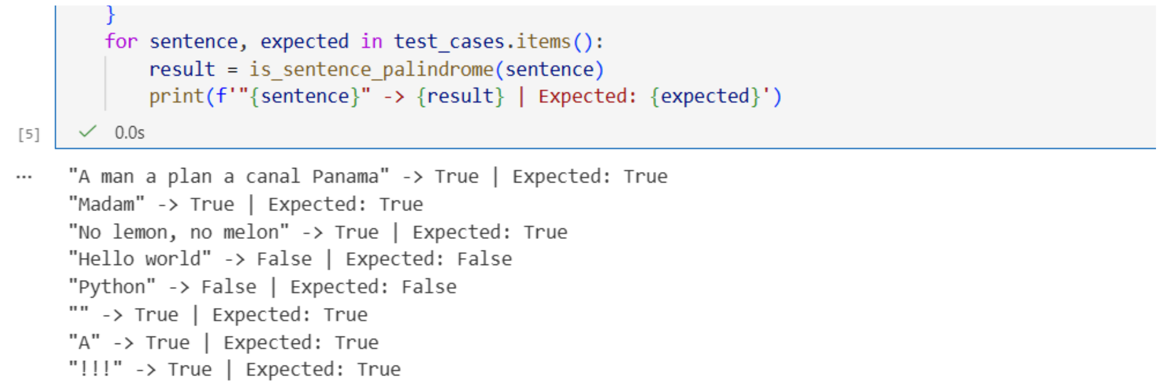
- Test cases
- Python implementation
- Test execution output

<u>ScreenShots</u> :



```python
# Step 1:
# Generate comprehensive test cases for a function called
# is_sentence_palindrome(sentence).
# Requirements:
# Ignore case sensitivity, Ignore spaces, Ignore punctuation characters
# Test both palindromic and non-palindromic sentences,Include edge cases like:
# Empty string
# Single character
# Sentences with only punctuation
# Mixed case sentences
# Example:
# "A man a plan a canal Panama" → True
# Step 2:
# Implement the function in Python so that all generated test cases pass.
# Return:
# - Test cases
# - Python implementation
# - Test execution output

import string
def is_sentence_palindrome(sentence):
    # Remove punctuation, spaces, and convert to lowercase
    cleaned = ''.join(
        ch.lower() for ch in sentence
        if ch.isalnum()
    )
    return cleaned == cleaned[::-1]
test_cases = {
    "A man a plan a canal Panama": True,
    "Madam": True,
    "No lemon, no melon": True,
    "Hello world": False,
    "Python": False,
    "": True,
    "A": True,
    "!!!": True
}
for sentence, expected in test_cases.items():
    result = is_sentence_palindrome(sentence)
```

<u>Output</u> :



```python
}
for sentence, expected in test_cases.items():
    result = is_sentence_palindrome(sentence)
    print(f'"{sentence}" -> {result} | Expected: {expected}')
```

[5] ✓ 0.0s

```
... "A man a plan a canal Panama" -> True | Expected: True
    "Madam" -> True | Expected: True
    "No lemon, no melon" -> True | Expected: True
    "Hello world" -> False | Expected: False
    "Python" -> False | Expected: False
    "" -> True | Expected: True
    "A" -> True | Expected: True
    "!!!" -> True | Expected: True
```

<u>Task-4</u> : ShoppingCart Class

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

<u>Prompt</u> : Step 1:

Generate comprehensive test cases for a class named ShoppingCart.

Requirements:

The class must implement:

- add_item(name, price), remove_item(name), total_cost()

Test Cases Must Cover:

Adding single and multiple items, Correct total cost calculation, Removing existing items, Removing non-existing items, Empty cart total cost (should be 0)

Cart behavior after all items are removed

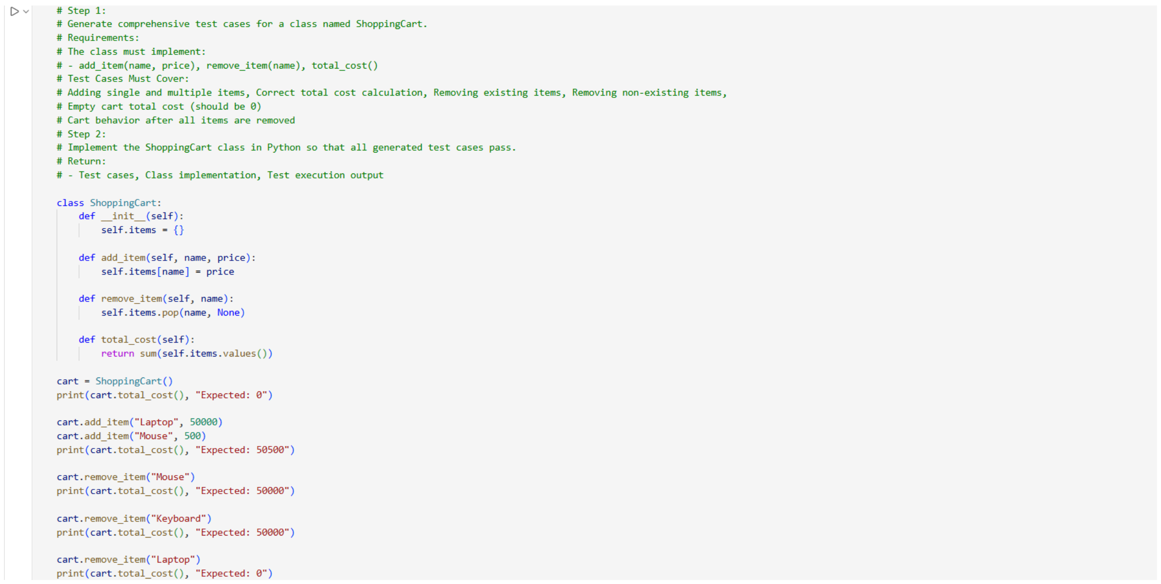
Step 2:

Implement the ShoppingCart class in Python so that all generated test cases pass.

Return:

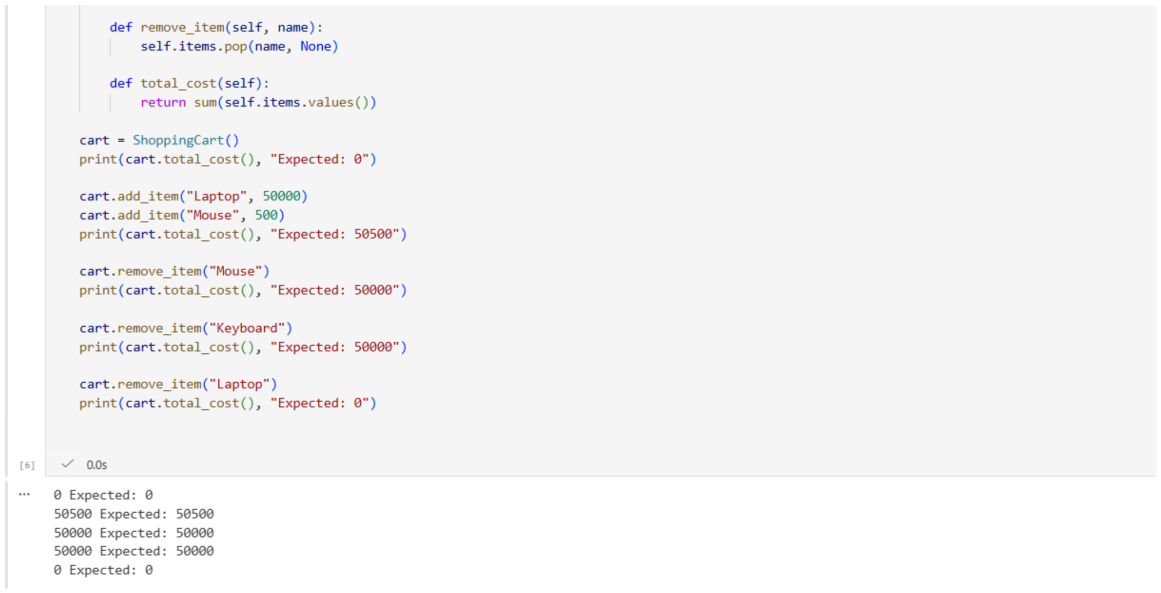
- Test cases, Class implementation, Test execution output

<u>Screenshots</u> :



```python
# Step 1:
# Generate comprehensive test cases for a class named ShoppingCart.
# Requirements:
# The class must implement:
# - add_item(name, price), remove_item(name), total_cost()
# Test Cases Must Cover:
# Adding single and multiple items, Correct total cost calculation, Removing existing items, Removing non-existing items,
# Empty cart total cost (should be 0)
# Cart behavior after all items are removed
# Step 2:
# Implement the ShoppingCart class in Python so that all generated test cases pass.
# Return:
# - Test cases, Class implementation, Test execution output

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

    def add_item(self, name, price):
        self.items[name] = price

    def remove_item(self, name):
        self.items.pop(name, None)

    def total_cost(self):
        return sum(self.items.values())

cart = ShoppingCart()
print(cart.total_cost(), "Expected: 0")

cart.add_item("Laptop", 50000)
cart.add_item("Mouse", 500)
print(cart.total_cost(), "Expected: 50500")

cart.remove_item("Mouse")
print(cart.total_cost(), "Expected: 50000")

cart.remove_item("Keyboard")
print(cart.total_cost(), "Expected: 50000")

cart.remove_item("Laptop")
print(cart.total_cost(), "Expected: 0")
```

Output :



```
    def remove_item(self, name):
        self.items.pop(name, None)

    def total_cost(self):
        return sum(self.items.values())

cart = ShoppingCart()
print(cart.total_cost(), "Expected: 0")

cart.add_item("Laptop", 50000)
cart.add_item("Mouse", 500)
print(cart.total_cost(), "Expected: 50500")

cart.remove_item("Mouse")
print(cart.total_cost(), "Expected: 50000")

cart.remove_item("Keyboard")
print(cart.total_cost(), "Expected: 50000")

cart.remove_item("Laptop")
print(cart.total_cost(), "Expected: 0")
```

[6] ✓ 0.0s

```
0 Expected: 0
50500 Expected: 50500
50000 Expected: 50000
50000 Expected: 50000
0 Expected: 0
```

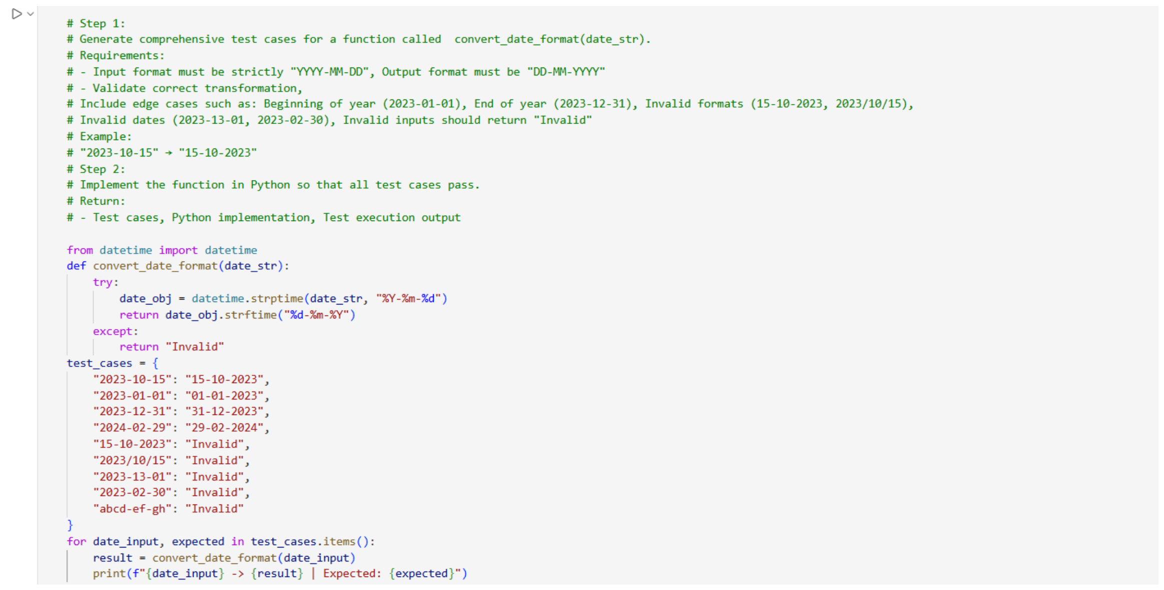
Task-5 : Date Format Conversion
You are creating a utility function to convert date formats for reports.

Prompt : Step 1:
Generate comprehensive test cases for a function called convert_date_format(date_str).
Requirements:
- Input format must be strictly "YYYY-MM-DD", Output format must be "DD-MM-YYYY"
- Validate correct transformation,
Include edge cases such as: Beginning of year (2023-01-01), End of year (2023-12-31), Invalid formats (15-10-2023, 2023/10/15), Invalid dates (2023-13-01, 2023-02-30), Invalid inputs should return "Invalid"

Example:
"2023-10-15" → "15-10-2023"
Step 2:
Implement the function in Python so that all test cases pass.

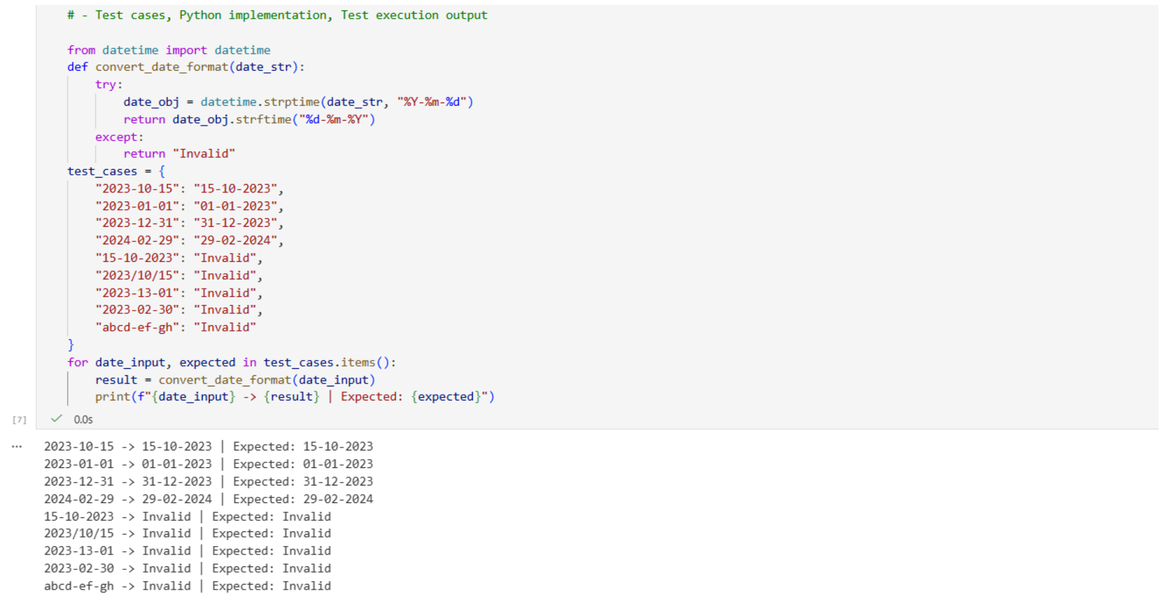
Return:
- Test cases, Python implementation, Test execution output

Screenshots :



```python
# Step 1:
# Generate comprehensive test cases for a function called  convert_date_format(date_str).
# Requirements:
# - Input format must be strictly "YYYY-MM-DD", Output format must be "DD-MM-YYYY"
# - Validate correct transformation,
# Include edge cases such as: Beginning of year (2023-01-01), End of year (2023-12-31), Invalid formats (15-10-2023, 2023/10/15),
# Invalid dates (2023-13-01, 2023-02-30), Invalid inputs should return "Invalid"
# Example:
# "2023-10-15" → "15-10-2023"
# Step 2:
# Implement the function in Python so that all test cases pass.
# Return:
# - Test cases, Python implementation, Test execution output

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:
        return "Invalid"
test_cases = {
    "2023-10-15": "15-10-2023",
    "2023-01-01": "01-01-2023",
    "2023-12-31": "31-12-2023",
    "2024-02-29": "29-02-2024",
    "15-10-2023": "Invalid",
    "2023/10/15": "Invalid",
    "2023-13-01": "Invalid",
    "2023-02-30": "Invalid",
    "abcd-ef-gh": "Invalid"
}
for date_input, expected in test_cases.items():
    result = convert_date_format(date_input)
    print(f"{date_input} -> {result} | Expected: {expected}")
```

Output :



```python
# - Test cases, Python implementation, Test execution output

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:
        return "Invalid"
test_cases = {
    "2023-10-15": "15-10-2023",
    "2023-01-01": "01-01-2023",
    "2023-12-31": "31-12-2023",
    "2024-02-29": "29-02-2024",
    "15-10-2023": "Invalid",
    "2023/10/15": "Invalid",
    "2023-13-01": "Invalid",
    "2023-02-30": "Invalid",
    "abcd-ef-gh": "Invalid"
}
for date_input, expected in test_cases.items():
    result = convert_date_format(date_input)
    print(f"{date_input} -> {result} | Expected: {expected}")
```

[7] ✓ 0.0s

```
2023-10-15 -> 15-10-2023 | Expected: 15-10-2023
2023-01-01 -> 01-01-2023 | Expected: 01-01-2023
2023-12-31 -> 31-12-2023 | Expected: 31-12-2023
2024-02-29 -> 29-02-2024 | Expected: 29-02-2024
15-10-2023 -> Invalid | Expected: Invalid
2023/10/15 -> Invalid | Expected: Invalid
2023-13-01 -> Invalid | Expected: Invalid
2023-02-30 -> Invalid | Expected: Invalid
abcd-ef-gh -> Invalid | Expected: Invalid
```