

Task 1: Email Validation using TDD

Scenario

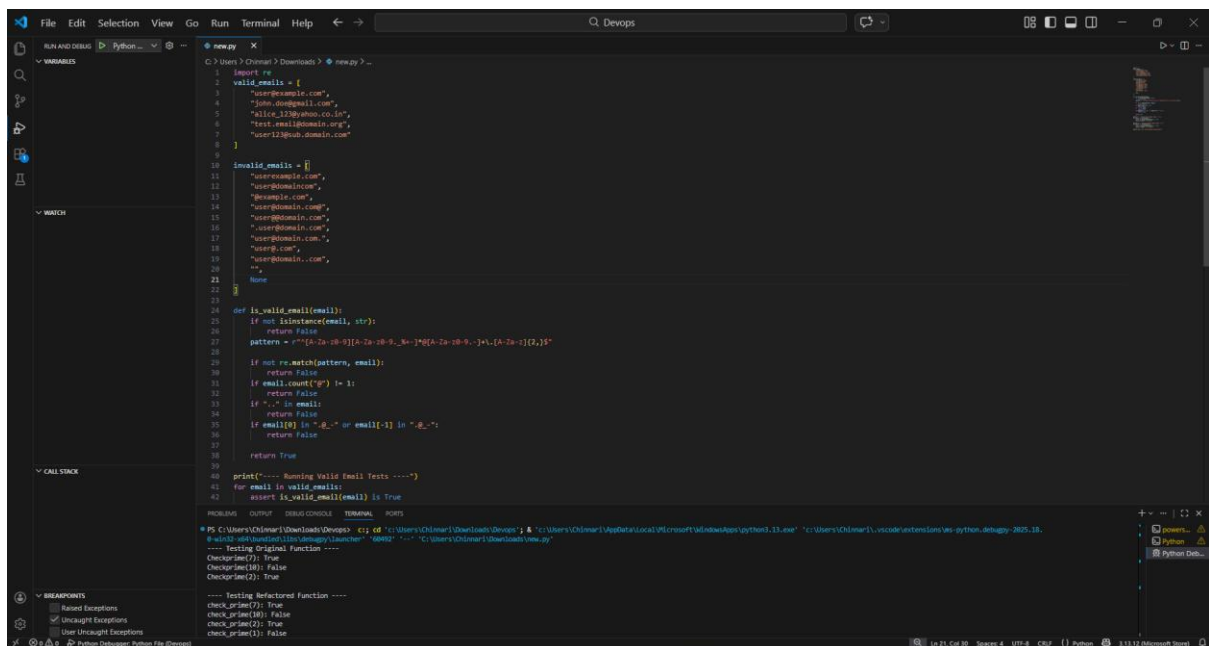
You are developing a user registration system that requires reliable email input validation.

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



The screenshot shows a Python IDE with a file named `new.py`. The code defines two lists: `valid_emails` and `invalid_emails`, and a function `is_valid_email(email)`. The function uses a regular expression to validate email formats. Below the function, there are test cases for both valid and invalid emails. The terminal output shows the results of these tests, indicating that all tests passed.

```
1 import re
2
3 valid_emails = [
4     "user@example.com",
5     "john.doe@domain.com",
6     "alice_123@yahoo.co.in",
7     "test_email@domain.org",
8     "user123@sub.domain.co.uk"
9 ]
10
11 invalid_emails = [
12     "userexample.com",
13     "user@domaincom",
14     "@domain.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com.",
18     "userp.com",
19     "user@domain.com",
20     ""
21 ]
22
23 def is_valid_email(email):
24     if not isinstance(email, str):
25         return False
26     pattern = r"^[a-zA-Z0-9_!#$%&'*~]{1,64}@([a-zA-Z0-9-]{1,63}\.){1,6}([a-zA-Z0-9-]{2,6})$"
27     if not re.match(pattern, email):
28         return False
29     if email.count("@") != 1:
30         return False
31     if "." in email:
32         return False
33     if email[-1] in ".-_" or email[0] in ".-_" or email[0] in "0-9":
34         return False
35     return True
36
37 print("Running Valid Email Tests ----")
38 for email in valid_emails:
39     assert is_valid_email(email) is True
40
41 print("Running Invalid Email Tests ----")
42 for email in invalid_emails:
43     assert is_valid_email(email) is False
```

Terminal Output:

```
PS C:\Users\Chinmay\Downloads\Devs> cd "C:\Users\Chinmay\Downloads\Devs" & "C:\Users\Chinmay\AppData\Local\Microsoft\WindowsApps\python.13.exe" "C:\Users\Chinmay\Downloads\Devs\new.py"
Running Valid Email Tests ----
Running Invalid Email Tests ----
Testing Original Function ----
check_email(1): True
check_email(2): True
check_email(3): True
check_email(4): True
check_email(5): True
Testing Refactored Function ----
check_email(1): True
check_email(2): True
check_email(3): True
check_email(4): True
check_email(5): True
```

```

1  import re
2  valid_email = [
3      "user@example.com",
4      "john.doe@wall.com",
5      "alice_123@ehon.co.in",
6      "test_email@domain.org",
7      "user123@sub.domain.com"
8  ]
9
10 invalid_email = [
11     "user@exampl.com",
12     "user@domain.com",
13     "@example.com",
14     "user@domain.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com",
18     "user@com",
19     "user@domain.com",
20     ""
21 ]
22 None
23
24 def is_valid_email(email):
25     if not isinstance(email, str):
26         return False
27     pattern = r"^[a-zA-Z0-9]{1,30}@[a-zA-Z0-9]{1,30}\.[a-zA-Z]{2,10}$"
28     if not re.match(pattern, email):
29         return False
30     if email.count("@") != 1:
31         return False
32     if "." in email:
33         return False
34     if email[0] in ".@_" or email[-1] in ".@_":
35         return False
36     return True
37
38 print("---- Running Valid Email Tests ----")
39 for email in valid_email:
40     assert is_valid_email(email) is True
41
42 print("---- Running Invalid Email Tests ----")
43 for email in invalid_email:
44     assert is_valid_email(email) is False
45
46 print("All test cases passed successfully!")

```

```

1  import re
2  valid_email = [
3      "user@example.com",
4      "john.doe@wall.com",
5      "alice_123@ehon.co.in",
6      "test_email@domain.org",
7      "user123@sub.domain.com"
8  ]
9
10 invalid_email = [
11     "user@exampl.com",
12     "user@domain.com",
13     "@example.com",
14     "user@domain.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com",
18     "user@com",
19     "user@domain.com",
20     ""
21 ]
22 None
23
24 def is_valid_email(email):
25     if not isinstance(email, str):
26         return False
27     pattern = r"^[a-zA-Z0-9]{1,30}@[a-zA-Z0-9]{1,30}\.[a-zA-Z]{2,10}$"
28     if not re.match(pattern, email):
29         return False
30     if email.count("@") != 1:
31         return False
32     if "." in email:
33         return False
34     if email[0] in ".@_" or email[-1] in ".@_":
35         return False
36     return True
37
38 print("---- Running Valid Email Tests ----")
39 for email in valid_email:
40     assert is_valid_email(email) is True
41     print(f"{email} -> Valid")
42
43 print("---- Running Invalid Email Tests ----")
44 for email in invalid_email:
45     assert is_valid_email(email) is False
46     print(f"{email} -> Invalid")
47
48 print("All test cases passed successfully!")

```

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

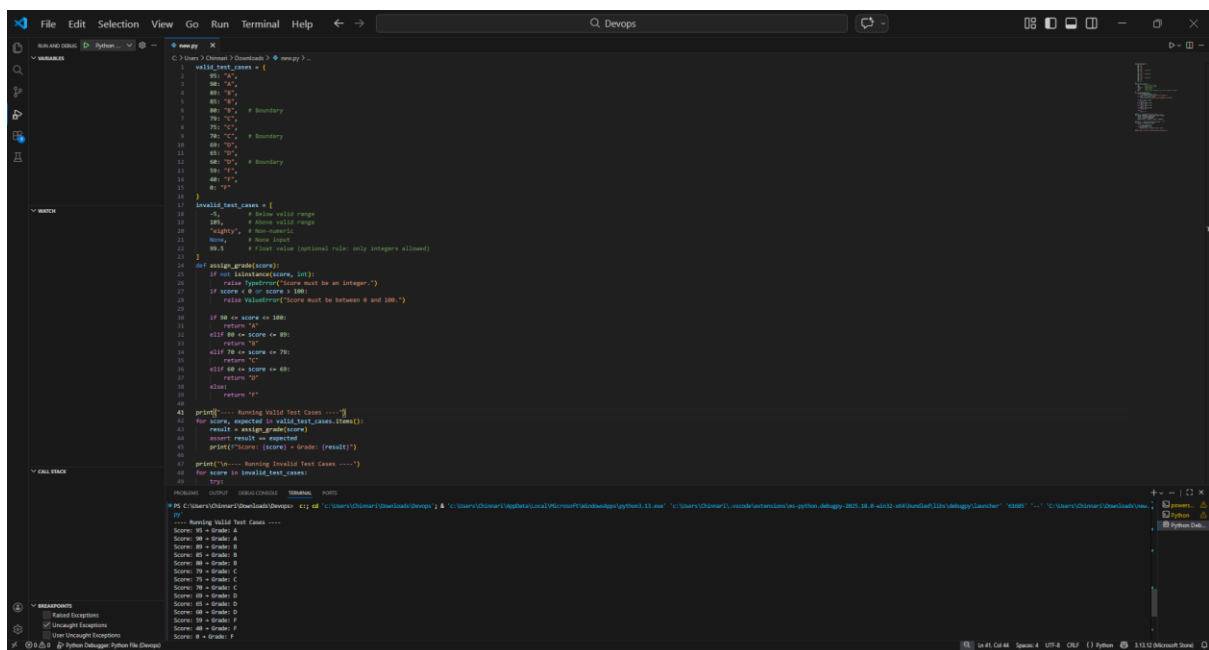
– 60–69 → D

- Below 60 \rightarrow 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



```
11 def assign_grade(score):
12     if not isinstance(score, int):
13         raise TypeError("Score must be an integer.")
14     if score < 0 or score > 100:
15         raise ValueError("Score must be between 0 and 100.")
16
17     if 90 <= score <= 100:
18         return "A"
19     elif 80 <= score <= 89:
20         return "B"
21     elif 70 <= score <= 79:
22         return "C"
23     elif 60 <= score <= 69:
24         return "D"
25     else:
26         return "F"
27
28 # Test cases
29 print("Running valid test cases ----")
30 for score, expected in valid_test_cases.items():
31     result = assign_grade(score)
32     assert result == expected, f"Score: {score} = Grade: {result}"
33     print(f"Score: {score} = Grade: {result}")
34
35 print("Running invalid test cases ----")
36 for score in invalid_test_cases:
37     try:
38         assign_grade(score)
39     except Exception as e:
40         print(f"Score: {score} = Error: {e}")
41
42 print("All valid test cases passed successfully!")
```

Running valid test cases ----
Score: 95 = Grade: A
Score: 85 = Grade: B
Score: 75 = Grade: C
Score: 65 = Grade: D
Score: 55 = Grade: F

Running invalid test cases ----
Score: 105 = Error: Score must be between 0 and 100.
Score: -5 = Error: Score must be between 0 and 100.
Score: 100.5 = Error: Score must be an integer.
Score: None = Error: Score must be an integer.

All valid test cases passed successfully!

Task 3: Sentence Palindrome Checker

Scenario

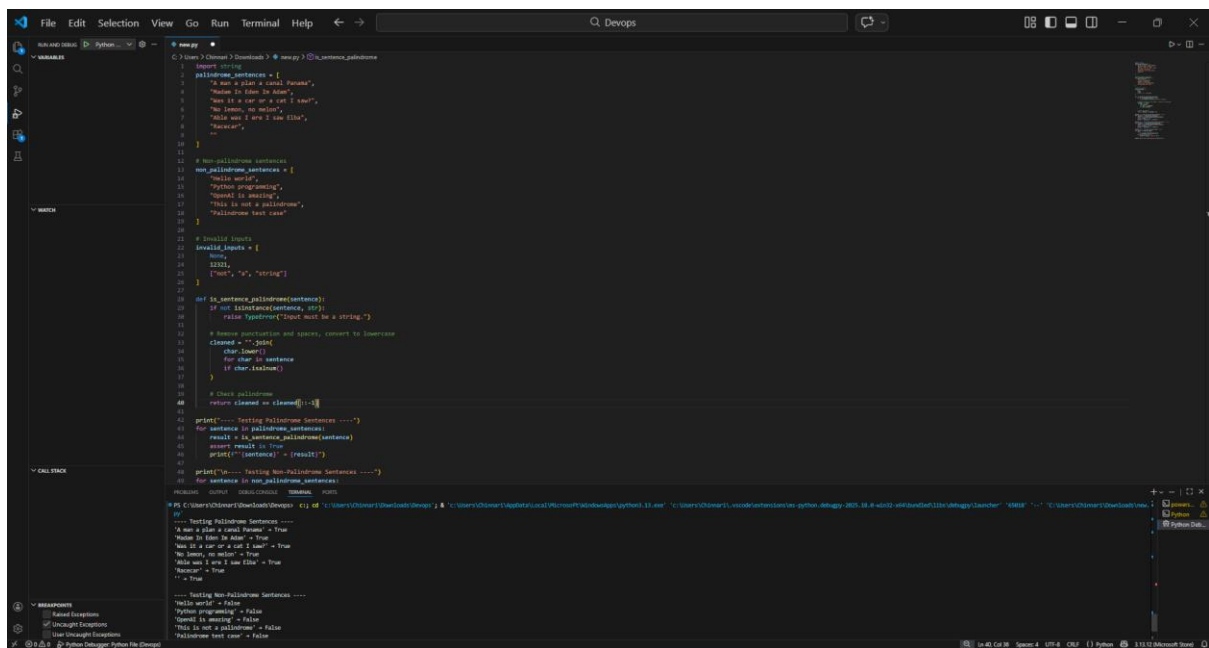
You are developing a text-processing utility to analyze sentences.

Requirements

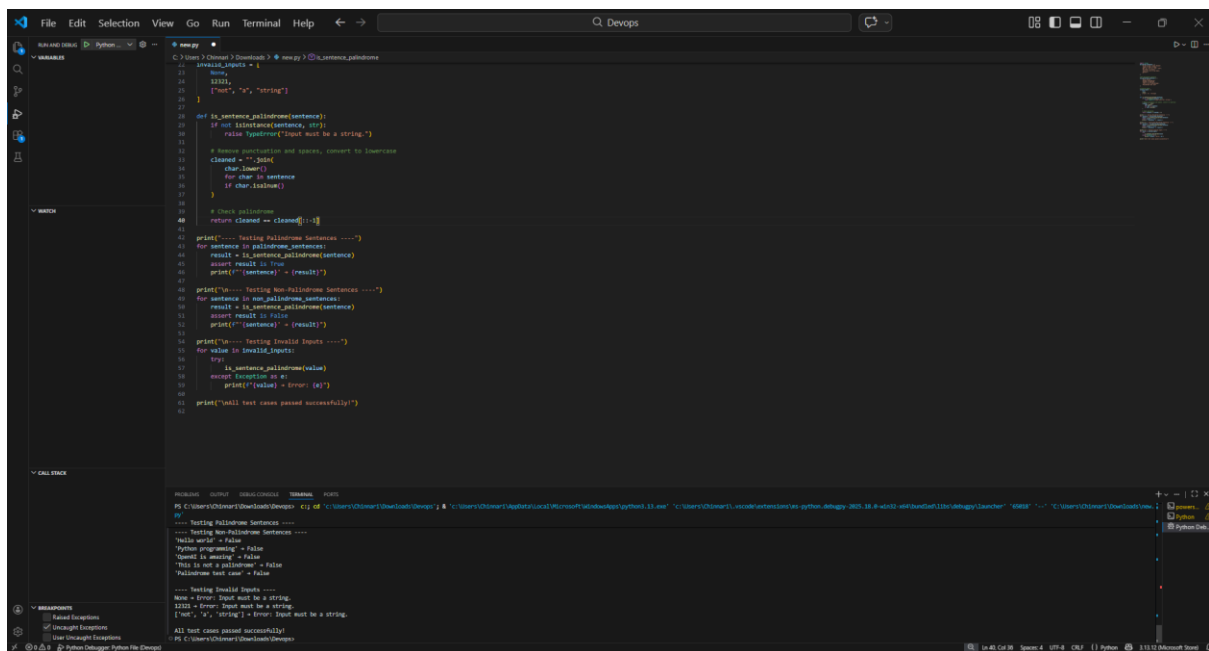
- AI should generate test cases for `is_sentence_palindrome(sentence)`
- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example:
– "A man a plan a canal Panama" → True

Expected Output

- Function correctly identifies sentence palindromes
- Case and punctuation are ignored
- Returns True or False accurately
- All AI-generated test cases pass



```
1 #!/usr/bin/env python3
2
3 # Palindrome test cases
4
5 # Valid palindromes
6 valid_sentences = [
7     "A man a plan a canal Panama",
8     "Was it a car or a cat I saw?",
9     "No lemon, no melon",
10    "While was I saw I saw Ella",
11    "Racecar",
12 ]
13
14 # Non-palindromes
15 non_palindrome_sentences = [
16     "Hello world",
17     "Python programming",
18     "Spam! is missing",
19     "This is not a palindrome",
20     "Palindrome test case!"
21 ]
22
23 # Invalid inputs
24 invalid_inputs = [
25     None,
26     12345,
27     ["not", "a", "string"]
28 ]
29
30 def is_sentence_palindrome(sentence):
31     """Check if a sentence is a palindrome.
32     Returns True if the sentence is a palindrome, False otherwise.
33     """
34     if not isinstance(sentence, str):
35         raise TypeError("Input must be a string.")
36
37     # Remove punctuation and spaces, convert to lowercase
38     cleaned = ""
39     for char in sentence:
40         if char.isalnum():
41             cleaned += char.lower()
42
43     # Check palindrome
44     return cleaned == cleaned[::-1]
45
46 print("---- Testing Palindrome Sentences ----")
47 for sentence in valid_sentences:
48     result = is_sentence_palindrome(sentence)
49     assert result is True
50     print(f"({sentence}) = {result}")
51
52 print("---- Testing Non-Palindrome Sentences ----")
53 for sentence in non_palindrome_sentences:
54     result = is_sentence_palindrome(sentence)
55     assert result is False
56     print(f"({sentence}) = {result}")
57
58 print("---- Testing Invalid Inputs ----")
59 for value in invalid_inputs:
60     try:
61         is_sentence_palindrome(value)
62     except TypeError as e:
63         print(f"({value}) = Error: {e}")
64
65 print("All test cases passed successfully!")
```



```
Python 3.11.1 Shell
C:\Users\Chinor> python3 is_sentence_palindrome.py
---- Testing Palindrome Sentences ----
('A man a plan a canal Panama') = True
('Was it a car or a cat I saw?') = True
('No lemon, no melon') = True
('While was I saw I saw Ella') = True
('Racecar') = True
---- Testing Non-Palindrome Sentences ----
('Hello world') = False
('Python programming') = False
('Spam! is missing') = False
('This is not a palindrome') = False
('Palindrome test case!') = False
---- Testing Invalid Inputs ----
None = Error: Input must be a string.
12345 = Error: Input must be a string.
['not', 'a', 'string'] = Error: Input must be a string.
All test cases passed successfully!
Python Shell
```

Task 4: ShoppingCart Class

Scenario

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

Requirements

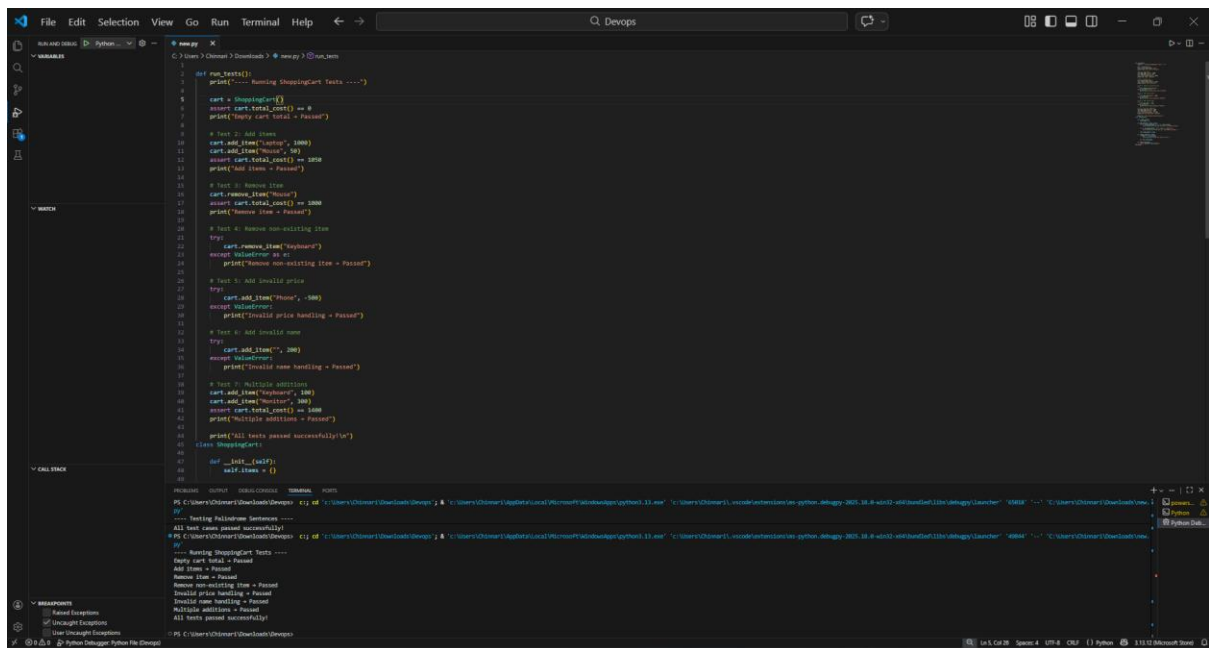
- AI should generate test cases for the ShoppingCart class
- Class must include the following methods:
 - add_item(name, price)
 - remove_item(name)

– 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 give code for this



```
def run_tests():
    print("Running ShoppingCart Tests ....")

    cart = ShoppingCart()
    assert cart.total_cost() == 0
    print("Empty cart total = Passed")

    # Test 1: Add items
    cart.add_item("apple", 1000)
    cart.add_item("banana", 50)
    assert cart.total_cost() == 1050
    print("Add items = Passed")

    # Test 2: Remove item
    cart.remove_item("banana")
    assert cart.total_cost() == 1000
    print("Remove item = Passed")

    # Test 3: Remove non-existing item
    try:
        cart.remove_item("watermelon")
    except ValueError as e:
        print("Remove non-existing item = Passed")

    # Test 4: Add invalid price
    try:
        cart.add_item("orange", -500)
    except ValueError:
        print("Invalid price handling = Passed")

    # Test 5: Add invalid name
    try:
        cart.add_item("", 100)
    except ValueError:
        print("Invalid name handling = Passed")

    # Test 6: Multiple additions
    cart.add_item("watermelon", 100)
    cart.add_item("watermelon", 100)
    assert cart.total_cost() == 1200
    print("Multiple additions = Passed")

    print("All tests passed successfully!")

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

    def add_item(self, item, price):
        if price < 0:
            raise ValueError("Invalid price")
        if not item:
            raise ValueError("Invalid name")
        self.items[item] = self.items.get(item, 0) + price

    def remove_item(self, item):
        if item not in self.items:
            raise ValueError("Item not in cart")
        del self.items[item]

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

Task 5: Date Format Conversion

Scenario

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

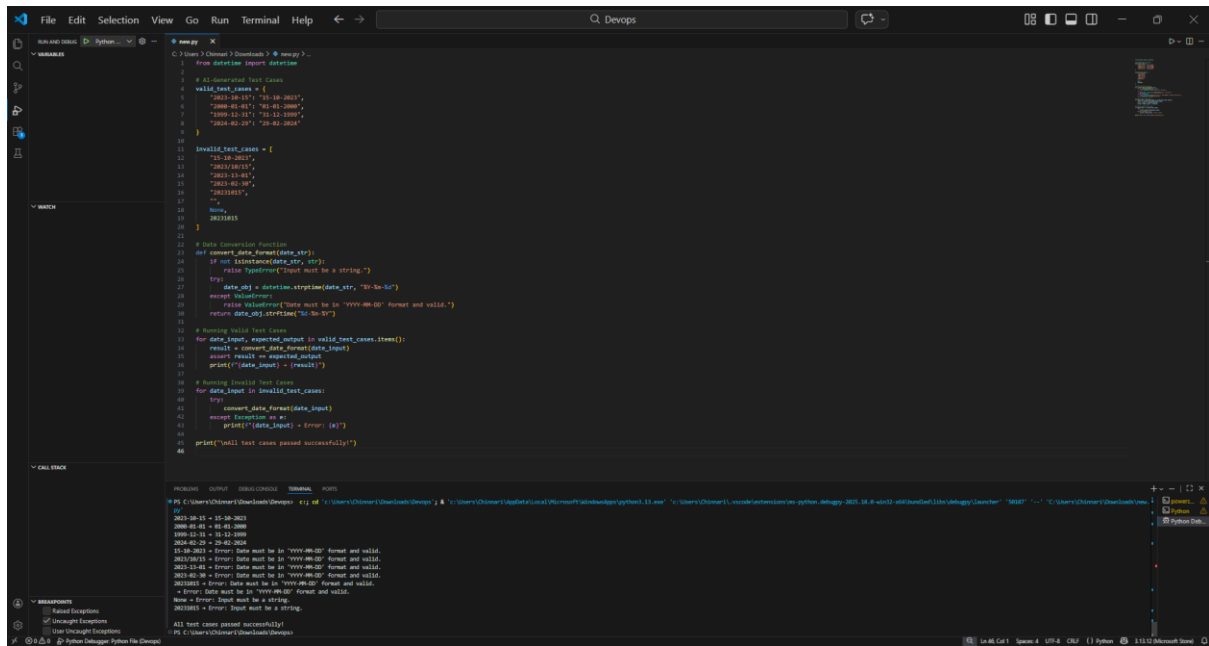
Requirements

- 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 give code for this



```
File Edit Selection View Go Run Terminal Help
C:\Users\Chinmay> Download > energy > ..
1 from datetime import datetime
2
3 # All generated test cases
4 valid_test_cases = [
5     "2023-10-20", "15-10-2023",
6     "2000-01-01", "31-12-2000",
7     "1999-12-31", "11-11-1999",
8     "2024-01-20", "20-01-2024"
9 ]
10
11 invalid_test_cases = [
12     "15-10-2022",
13     "2023/10/20",
14     "2023-10-01",
15     "2023-02-30",
16     "20231015",
17     ""
18 ]
19 Name:
20 20231015
21
22 # Date Conversion Function
23 def convert_date_format(date_str):
24     if not isinstance(date_str, str):
25         raise TypeError("Input must be a string.")
26     try:
27         date_obj = datetime.strptime(date_str, "%Y-%m-%d")
28         return date_obj.strftime("%Y-%m-%d")
29     except ValueError:
30         raise ValueError("Date must be in 'YYYY-MM-DD' format and valid.")
31
32 # Running Valid Test Cases
33 for data_input, expected_output in valid_test_cases.items():
34     result = convert_date_format(data_input)
35     assert result == expected_output
36     print(f"[data_input] = {result}")
37
38 # Running Invalid Test Cases
39 for data_input in invalid_test_cases:
40     try:
41         convert_date_format(data_input)
42     except Exception as e:
43         print(f"[data_input] = Error: {e}")
44
45 print("All test cases passed successfully!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Chinmay\Downloads> cd "C:\Users\Chinmay\Downloads" & "C:\Users\Chinmay\AppData\Local\Microsoft\WindowsApps\python.11.exe" "C:\Users\Chinmay\Downloads\energy.py"
2023-10-15 = 15-10-2023
2000-01-01 = 01-01-2000
1999-12-31 = 31-12-1999
2024-01-20 = 20-01-2024
15-10-2023 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023/10/20 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023-10-01 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023-02-30 = Error: Date must be in 'YYYY-MM-DD' format and valid.
20231015 = Error: Date must be in 'YYYY-MM-DD' format and valid.
= Error: Date must be in 'YYYY-MM-DD' format and valid.
= Error: Input must be a string.
20231015 = Error: Input must be a string.

All test cases passed successfully!
PS C:\Users\Chinmay\Downloads>
```