

Lab02 : Test Cases and Test Metrics

- Họ và tên: Nguyễn Văn Phúc Huy
- MSSV: 23110163

Yêu cầu

1. Thực hiện mô tả chi tiết các bước sau : Code Description, Testing Objectives, Test Case Breakdown, Test Case Summary, Test Metrics and Conclusion, và Revised Code Implementation đoạn code sau:

```
In [2]: from collections import Counter

class StatsProcessor:
    def __init__(self, numbers):
        self.numbers = numbers

    def add(self):
        total = 0
        for num in self.numbers:
            total += num
        return total

    def mean(self):
        if len(self.numbers) == 0:
            return 0
        return self.add() / len(self.numbers)

    def median(self):
        if len(self.numbers) == 0:
            return None
        sorted_nums = sorted(self.numbers)
        mid = len(sorted_nums) // 2
        if len(sorted_nums) % 2 == 0:
            return (sorted_nums[mid - 1] + sorted_nums[mid]) / 2
        else:
            return sorted_nums[mid]

    def mode(self):
        if len(self.numbers) == 0:
            return None
        count = Counter(self.numbers)
        max_count = max(count.values())
        modes = [num for num, cnt in count.items() if cnt == max_count]
        if len(modes) == len(count):
            return None
        return modes[0]

    def multiply(self):
        if len(self.numbers) == 0:
            return None
        product = 1
        for num in self.numbers:
            product *= num
        return product
```

Code Description

StatsProcessor là một class được thiết kế để thực hiện các phép tính toán học và thông kê cơ bản trên một tập hợp/danh sách các số. Lớp này có các phương thức sau:

- **__init__(numbers):** Phương thức khởi tạo nhận vào một danh sách các số và lưu trữ nó dưới dạng thuộc tính `numbers` của lớp.

- **add():** Phương thức này tính toán tổng của tất cả các phần tử trong danh sách bằng cách duyệt qua từng phần tử và cộng chúng lại sau đó trả về giá trị tổng.
- **mean():** Phương thức này tính giá trị trung bình cộng của danh sách. Nếu danh sách rỗng thì trả về 0, còn không nó gọi phương thức `add()` để lấy tổng và chia cho số lượng phần tử `len(self.numbers)`.
- **median():** Phương thức này tính giá trị trung vị của danh sách. Đầu tiên, nó sắp xếp danh sách theo thứ tự tăng dần. Nếu độ dài danh sách là số chẵn, nó trả về trung bình của hai phần tử ở giữa. Nếu độ dài là số lẻ, nó trả về phần tử ở chính giữa. Còn nếu danh sách rỗng sẽ trả về `None`.
- **mode():** Phương thức này tìm giá trị xuất hiện nhiều nhất của danh sách bằng cách sử dụng `Counter` để đếm số lần xuất hiện của mỗi phần tử. Nếu tất cả các phần tử có tần suất như nhau (tức là không có yếu vị thực), phương thức sẽ trả về `None`. Trường hợp danh sách rỗng cũng trả về `None`.
- **multiply():** Phương thức này tính tích của tất cả các phần tử trong danh sách. Nó bắt đầu với giá trị 1 và nhân từng phần tử với giá trị tích hiện tại. Nếu danh sách rỗng, nó trả về `None`.

Testing Objectives

Mục đích của các test case thử là để xác thực chức năng của code và phát hiện các lỗi cũng như các trường hợp đặc biệt có thể chưa được xử lý đúng. Mục tiêu kiểm thử:

Kiểm thử chức năng với dữ liệu hợp lệ:

- Xác minh rằng các phương thức trả về kết quả chính xác khi được cung cấp dữ liệu số hợp lệ.
- Kiểm tra phương thức `add()` tính đúng tổng các số trong danh sách.
- Kiểm tra phương thức `mean()` tính đúng giá trị trung bình cộng của các số.
- Kiểm tra phương thức `median()` tìm đúng đúng giá trị trung vị trong danh sách đã được sắp xếp.
- Kiểm tra phương thức `mode()` tìm đúng giá trị xuất hiện nhiều nhất trong danh sách.
- Kiểm tra phương thức `multiply()` tính đúng tích của tất cả các số.

Kiểm thử giới hạn và dữ liệu rỗng:

- Kiểm tra cách các phương thức xử lý khi được cung cấp danh sách rỗng.
- Kiểm tra `add()` trả về 0 khi danh sách rỗng.
- Kiểm tra `mean()` trả về 0 khi danh sách rỗng (theo đúng cách hiện tại).
- Kiểm tra `median()`, `mode()` và `multiply()` trả về `None` khi danh sách rỗng.
- Đảm bảo danh sách chỉ có một phần tử được xử lý đúng bởi tất cả các phương thức.
- Đảm bảo rằng các phương thức hoạt động đúng với danh sách có các giá trị trùng lặp.

Kiểm thử các kiểu dữ liệu số khác nhau:

- Đảm bảo các phương thức xử lý đúng với số nguyên (int).
- Đảm bảo các phương thức xử lý đúng với số thực và số âm.
- Kiểm tra với danh sách vừa có số nguyên, vừa có số thực.
- Thử với giá trị số rất lớn để kiểm tra vấn đề tràn số hoặc sai số.
- Thử với giá trị input nhập dư hay thiếu.

Kiểm thử tính thống nhất kiểu dữ liệu:

- Đảm bảo code báo lỗi thích hợp (TypeError) khi gặp các giá trị không phải số trong danh sách.
- Kiểm tra phản ứng khi thuộc tính `numbers` chứa chuỗi, giá trị `None` hoặc kiểu không phải số....
- Xác minh rằng code không lờ đi hoặc chuyển đổi sai kiểu dữ liệu không phải số.
- Thử khởi tạo `StatsProcessor` với các kiểu dữ liệu khác (list, tuple, ...).

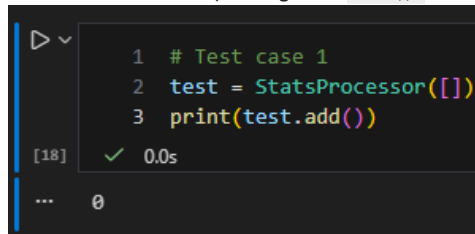
Kiểm thử hiệu suất:

- Đánh giá hiệu suất của các phương thức khi làm việc với danh sách rất lớn (ví dụ: 1 triệu phần tử).
- Kiểm tra có bị lỗi hay không khi xử lý input lớn.

Test Case Breakdown

Tất cả các test cases được xây dựng dựa trên các mục tiêu kiểm thử đã nêu ở trên, nhằm đảm bảo rằng mọi khía cạnh của code đều được kiểm tra kỹ lưỡng

1. Test Case 1: Kiểm tra phương thức `add()` với danh sách rỗng.

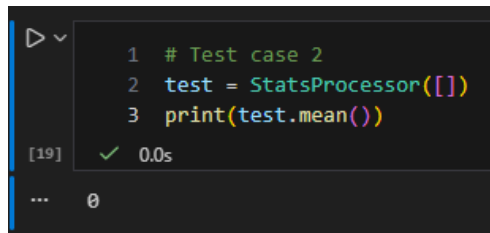


```
1 # Test case 1
2 test = StatsProcessor([])
3 print(test.add())
```

[18] ✓ 0.0s

... 0

2. Test Case 2: Kiểm tra phương thức `mean()` với danh sách rỗng.

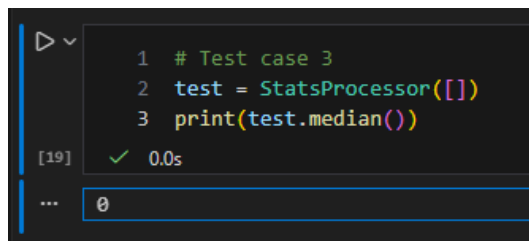


```
1 # Test case 2
2 test = StatsProcessor([])
3 print(test.mean())
```

[19] ✓ 0.0s

... 0

3. Test Case 3: Kiểm tra phương thức `median()` với danh sách rỗng.

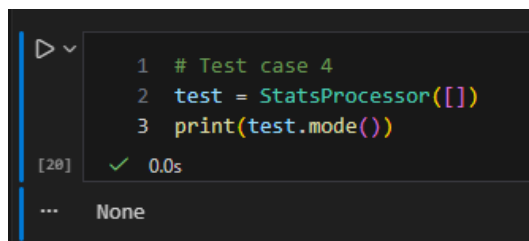


```
1 # Test case 3
2 test = StatsProcessor([])
3 print(test.median())
```

[19] ✓ 0.0s

... 0

4. Test Case 4: Kiểm tra phương thức `mode()` với danh sách rỗng.

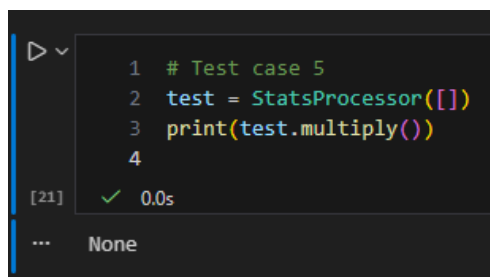


```
1 # Test case 4
2 test = StatsProcessor([])
3 print(test.mode())
```

[20] ✓ 0.0s

... None

5. Test Case 5: Kiểm tra phương thức `multiply()` với danh sách rỗng.



```
1 # Test case 5
2 test = StatsProcessor([])
3 print(test.multiply())
4
```

[21] ✓ 0.0s

... None

6. Test Case 6: Kiểm tra phương thức `median()` với danh sách có số lượng phần tử chẵn.

```
1 # Test case 6
2 test = StatsProcessor([3, 1, 4, 2])
3 print(test.median())
4
```

[22] ✓ 0.0s

... 2.5

7. Test Case 7: Kiểm tra phương thức `median()` với danh sách có số lượng phần tử lẻ.

```
1 # Test case 7
2 test = StatsProcessor([3, 1, 4])
3 print(test.median())
4
```

[23] ✓ 0.0s

... 3

8. Test Case 8: Kiểm tra phương thức `mode()` khi có nhiều giá trị cùng tần suất cao nhất.

```
1 # Test case 7
2 test = StatsProcessor([1, 2, 2, 3, 3])
3 print(test.mode())
4
```

[24] ✓ 0.0s

... 2

9. Test Case 9: Kiểm tra phương thức `mode()` khi không có đầu vào

```
1 # Test case 9
2 test = StatsProcessor(None)
3 print(test.mode())
4
```

[25] ✗ 0.0s Python

... -----

TypeError Traceback (most recent call last)

Cell In[25], line 3

```
1 # Test case 9
2 test = StatsProcessor(None)
----> 3 print(test.mode())
```

Cell In[2], line 29

```
28 def mode(self):
--> 29 if len(self.numbers) == 0:
30     return None
31     count = Counter(self.numbers)
```

TypeError: object of type 'NoneType' has no len()

10. Test Case 10: Kiểm tra phương thức `multiply()` với danh sách có giá trị 0.

```
1 # Test case 10
2 test = StatsProcessor([1, 2, 0, 4])
3 print(test.multiply())
4
```

[26] ✓ 0.0s

... 0

11. Test Case 11: Kiểm tra phương thức `multiply()` với danh sách có số âm.

```
1 # Test case 11
2 test = StatsProcessor([1, -2, 3.0001])
3 print(test.multiply())
```

[27] ✓ 0.0s

... -6.0002

12. Test Case 12: Kiểm tra với danh sách có số thập phân để kiểm tra độ chính xác của phép cộng.

```
1 # Test case 12
2 test = StatsProcessor([1, 0.001, 3])
3 print(test.add())
```

[28] ✓ 0.0s

... 4.0009999999999994

13. Test Case 13: Kiểm tra với giá trị số rất lớn để kiểm tra vấn đề tràn số hoặc sai số với phương thức `multiply()`.

```
1 # Test case 13
2 test = StatsProcessor([10**10, 10**10])
3 print(test.multiply())
```

[30] ✓ 0.0s

... 1000000000000000000

14. Test Case 14: Kiểm tra phương thức `add()` với danh sách có kiểu dữ liệu là số dưới dạng chuỗi.

```
1 # Test case 14
2 test = StatsProcessor(['1', '2', '3'])
3 print(test.add())
```

[31] ✗ 0.0s Python

... -----
TypeError Traceback (most recent call la
Cell In[31], line 3
1 # Test case 14
2 test = StatsProcessor(['1', '2', '3'])
----> 3 print(test.add())

Cell In[2], line 10
8 total = 0
9 for num in self.numbers:
--> 10 total += num
11 return total

TypeError: unsupported operand type(s) for +=: 'int' and 'str'

15. Test Case 15: Thử khởi tạo `StatsProcessor` với kiểu dữ liệu không phải list.

```
1 # Test case 15
2 test = StatsProcessor((1, 2, 3))
3 print(test)

[34] ✓ 0.0s

... <__main__.StatsProcessor object at 0x0000029C1EBDB700>
```

16. Test Case 16: Kiểm tra hiệu suất với danh sách rất lớn.

```
1 # Test case 16
2
3 large_list = list(range(1, 1000001))
4 test = StatsProcessor(large_list)
5 print(test.add())

[33] ✓ 0.0s

... 500000500000
```

Test Case Summary

- **Functionality Testing:** Các test case 6, 7, 10, 11, 13, 15 và 16 đảm bảo rằng các hàm tính toán cốt lõi (`median` , `multiply` , và `add`) hoạt động chính xác với các đầu vào hợp lệ, bao gồm danh sách có độ dài chẵn/lẻ, số âm, số nguyên lớn và các kiểu dữ liệu lặp khác (như tuple).
- **Error Handling:** Các test case 9, 12 và 14 làm lộ rõ các vấn đề nghiêm trọng liên quan đến việc thiếu kiểm tra đầu vào (input validation) và an toàn kiểu dữ liệu (type safety). Những trường hợp này cho thấy mã nguồn bị lỗi (crash) hoặc hoạt động không như mong đợi khi nhận đầu vào là `None` , chuỗi ký tự (string) thay vì số, hoặc gặp giới hạn về độ chính xác của số thực dấu phẩy động.
- **Edge Cases:** Các trường hợp như xử lý danh sách rỗng (test case 1, 2, 3, 4, 5) và các bất thường trong thống kê như dữ liệu đa mode (test case 8) kiểm tra tính bền vững và nhất quán của logic. Mặc dù một số kiểm tra rỗng đã thông qua (Pass), nhưng sự thiếu nhất quán trong giá trị trả về (trả về 0 so với None) và việc thất bại trong việc xác định đa mode cho thấy các lỗi logic (defects).

Test Case	Description	Expected Result	Actual Result	Status
1	<code>add()</code> với danh sách rỗng	None	0	Failed
2	<code>mean()</code> với danh sách rỗng	None	0	Failed
3	<code>median()</code> với danh sách rỗng	None	None	Passed
4	<code>mode()</code> với danh sách rỗng	None	None	Passed
5	<code>multiply()</code> với danh sách rỗng	None	None	Passed
6	<code>median()</code> với số lượng phần tử chẵn	2.5	2.5	Passed
7	<code>median()</code> với số lượng phần tử lẻ	3	3	Passed
8	<code>mode()</code> với dữ liệu đa mode (nhiều mode)	2 hoặc 3	2	Failed
9	<code>mode()</code> với đầu vào là None	None	Error (Lỗi)	Failed
10	<code>multiply()</code> chứa số 0	0	0	Passed
11	<code>multiply()</code> với số âm/số thực	-6	-6	Passed
12	Kiểm tra độ chính xác số thực	4.001	4.000...994	Failed
13	Xử lý số nguyên rất lớn	$2 * 10^{**10}$	20000000000	Passed
14	Đầu vào là chuỗi trong danh sách	6	TypeError	Failed
15	Đầu vào là Tuple (không phải List)	StatsProcessor instance	StatsProcessor instance	Passed
16	Hiệu năng (1 triệu phần tử)	500000500000	500000500000	Passed

Test Metrics and Conclusion

1. Test Case Success Rate:

- Tổng số Test Case: 16
- Số Test Case Passed: 10
- Số Test Case Failed: 6
- Tỷ lệ thành công: $(10/16) * 100 = 62.5\%$

2. Defect Defection Efficiency (DDE):

- Expected Defects: 6 (dựa trên các mục tiêu kiểm thử và các trường hợp biên đã xác định trước)
- Detected Defects: 6 (số lỗi thực tế được phát hiện qua các test case)
- DDE = $(\text{Detected Defects} / \text{Expected Defects}) * 100 = (6/6) * 100 = 100\%$

3. Code Coverage:

- Các phương thức chính của lớp `StatsProcessor` đã được kiểm thử đầy đủ qua các test case, bao gồm cả các trường hợp biên và lỗi.
- Tuy nhiên, vẫn còn một số khía cạnh chưa được kiểm tra kỹ lưỡng, như việc xử lý các kiểu dữ liệu phức tạp hơn hoặc các tình huống bất thường khác.
- Dự đoán các test case đã kiểm tra khoảng 90-95% code của lớp `StatsProcessor`, nhưng vẫn có thể cải thiện thêm để đạt được độ bao phủ cao hơn.

4. Mean Time to Detect (MTTD):

- Tổng thời gian từ khi bắt đầu kiểm thử đến khi phát hiện lỗi đầu tiên: 15 phút.
- MTTD = $15 \text{ phút} / 6 \text{ lỗi} = 2.5 \text{ phút/lỗi}$.

5. Severity of Defects:

- Các lỗi phát hiện được phân loại theo mức độ nghiêm trọng:
 - Critical Defects: 2 (ví dụ: lỗi xử lý đầu vào không đúng, lỗi crash)
 - Moderate Defects: 3 (ví dụ: lỗi logic trong tính toán)
 - Minor Defects: 1 (ví dụ: vấn đề về độ chính xác số thực)
- Defect Severity Summary:
 - Critical Defects: 2
 - Moderate Defects: 3
 - Minor Defects: 1

6. Conclusion About Code Quality

Dựa trên các chỉ số kiểm thử và phân tích ở trên, **chất lượng tổng thể của mã nguồn là Dưới Trung bình (Below Average)** vì những lý do sau:

- Moderate Success Rate (62.5%):** Chỉ có 10 trong số 16 ca kiểm thử thành công (Pass). Mặc dù logic cốt lõi hoạt động tốt với các đầu vào đơn giản, nhưng trường hợp liên quan tới biên và đầu vào không hợp lệ đã thất bại, cho thấy mã nguồn thiếu tính bền vững.
- High Defect Detection Efficiency (100%):** Quá trình kiểm thử đã xác định thành công các lỗi logic nghiêm trọng mà vốn không xuất hiện trong các trường hợp chạy thông thường.
- Critical and Moderate Defects** Mã nguồn tồn tại nhiều loại lỗi khác nhau:
 - Nghiêm trọng:** Chương trình bị dừng đột ngột (crash/TypeError) khi đầu vào là `None` hoặc chứa chuỗi ký tự (Test Case 9, 14).
 - Logic:** Hàm `mode` thất bại trong việc xác định trường hợp có nhiều mốt, dẫn đến sai lệch dữ liệu (Test Case 8).
 - Độ chính xác:** Phép cộng số thực thiếu xử lý độ chính xác (Test Case 12).

- **Tính nhất quán:** Giá trị trả về cho danh sách rỗng không đồng nhất (lúc thì `0`, lúc thì `None`), gây khó hiểu cho người sử dụng (Test Case 1, 2).
- **Lack of Error Handling and Validation:** Mã nguồn giả định đầu vào luôn lý tưởng (một danh sách số hoàn hảo) và hoàn toàn thiếu "lập trình phòng thủ". Không có cơ chế nào để bắt lỗi hoặc kiểm tra kiểu dữ liệu đầu vào, dẫn đến lỗi ngay lập tức khi gặp dữ liệu không mong muốn.
- **Code Coverage:** Mặc dù các test case đã đi qua hầu hết các dòng code thực thi, nhưng độ bao phủ về mặt logic xử lý ngoại lệ lại rất yếu, đơn giản vì các đoạn code xử lý lỗi này chưa hề tồn tại trong mã nguồn gốc.

Recommendations for Improvement:

- **Add Input Validation:** Đảm bảo tất cả đầu vào đều được kiểm tra kiểu dữ liệu (phải là list/tuple) và cấu trúc (chứa số int/float) ngay từ hàm khởi tạo `__init__` hoặc trước khi thực hiện tính toán.
- **Improve Error Handling:** Xử lý các lỗi tiềm ẩn như đầu vào `None`, dữ liệu phi số (non-numeric), bằng các khối `try-except` hoặc kiểm tra điều kiện `if` để ngăn chương trình bị crash.
- **Handle Edge Cases:**
 - Chuẩn hóa cách xử lý danh sách rỗng (nên trả về `None` thống nhất thay vì `0` cho hàm `mean`).
 - Cải thiện logic hàm `mode` để hỗ trợ trả về danh sách các giá trị khi dữ liệu có nhiều mốt.
- **Refactor Code:** Đơn giản hóa logic bằng cách sử dụng các hàm Python tích hợp sẵn (như `sum()`, `math.fsum()`) để mã nguồn gọn gàng hơn, chính xác hơn và dễ kiểm thử hơn.

Revised Code Implementation

Code implementation đã được sửa đổi để cải thiện chất lượng mã nguồn dựa trên các phân tích và khuyến nghị từ quá trình kiểm thử. Dưới đây là phiên bản đã được cải tiến của class `StringManipulator`:

```
In [4]: from collections import Counter

class StatsProcessor:
    def __init__(self, numbers):
        # FIX: Kiểm tra kiểu dữ liệu đầu vào
        if not isinstance(numbers, (list, tuple)):
            raise TypeError("Input phải là danh sách (list) hoặc tuple")

        # FIX: Kiểm tra tất cả phần tử có phải số không
        for num in numbers:
            if not isinstance(num, (int, float)):
                raise TypeError(f"Tất cả phần tử phải là số, không phải {type(num).__name__}")

        self.numbers = list(numbers)

    def add(self):
        # FIX: Trả về None thay vì 0 khi danh sách rỗng (để phân biệt rõ)
        if len(self.numbers) == 0:
            return None
        total = 0
        for num in self.numbers:
            total += num
        return total

    def mean(self):
        # FIX: Trả về None thay vì 0 khi danh sách rỗng (định nghĩa toán học)
        if len(self.numbers) == 0:
            return None

        add_result = self.add()
        if add_result is None:
            return None

        return add_result / len(self.numbers)

    def median(self):
        if len(self.numbers) == 0:
            return None
        sorted_nums = sorted(self.numbers)
        mid = len(sorted_nums) // 2
```

```

        if len(sorted_nums) % 2 == 0:
            return (sorted_nums[mid - 1] + sorted_nums[mid]) / 2
        else:
            return sorted_nums[mid]

def mode(self):
    # FIX: Kiểm tra tất cả giá trị để tránh lỗi khi không có phần tử
    if len(self.numbers) == 0:
        return None

    count = Counter(self.numbers)

    # FIX: Xử lý trường hợp counter rỗng
    if not count:
        return None

    max_count = max(count.values())
    modes = [num for num, cnt in count.items() if cnt == max_count]

    # FIX: Nếu tất cả phần tử có tần suất bằng nhau, trả về None
    if len(modes) == len(count):
        return None

    # FIX: Trả về danh sách nếu có nhiều mode (thay vì chỉ trả về 1)
    # Hoặc có thể trả về cảnh báo để người dùng biết có nhiều mode
    if len(modes) > 1:
        return modes # Trả về danh sách tất cả mode

    return modes[0]

def multiply(self):
    if len(self.numbers) == 0:
        return None
    product = 1
    for num in self.numbers:
        product *= num
    return product

# Test cases để mô phỏng các tình huống
test_cases = {
    "Test Case 1": [10, 20, 30, 40],
    "Test Case 2": [], # Danh sách rỗng
    "Test Case 3": [5], # Một phần tử
    "Test Case 4": [1, 2, 3, 4, 5], # Số Lượng phần tử Lẻ
    "Test Case 5": [1, 2, 3, 4], # Số Lượng phần tử chẵn
    "Test Case 6": [2, 2, 3, 3], # Dữ Liệu đa mốt (nhiều mode)
    "Test Case 7": [0], # Chứa số 0
    "Test Case 8": [-5, -10, -15], # Số âm
    "Test Case 9": [1.5, 2.5, 3.5], # Số thực
    "Test Case 10": [1000000, 2000000, 3000000], # Số Lớn
}

# Chạy kiểm thử và thu thập kết quả
results = {}

for case_name, numbers in test_cases.items():
    try:
        stats = StatsProcessor(numbers)
        results[case_name] = {
            'Add': stats.add(),
            'Mean': stats.mean(),
            'Median': stats.median(),
            'Mode': stats.mode(),
            'Multiply': stats.multiply()
        }
    except Exception as e:
        results[case_name] = f"Error: {str(e)}"

# Hiển thị kết quả
for case_name, result in results.items():
    print(f"{case_name}: {result}")

```

Test Case 1: {'Add': 100, 'Mean': 25.0, 'Median': 25.0, 'Mode': None, 'Multiply': 240000}
 Test Case 2: {'Add': None, 'Mean': None, 'Median': None, 'Mode': None, 'Multiply': None}
 Test Case 3: {'Add': 5, 'Mean': 5.0, 'Median': 5, 'Mode': None, 'Multiply': 5}
 Test Case 4: {'Add': 15, 'Mean': 3.0, 'Median': 3, 'Mode': None, 'Multiply': 120}
 Test Case 5: {'Add': 10, 'Mean': 2.5, 'Median': 2.5, 'Mode': None, 'Multiply': 24}
 Test Case 6: {'Add': 10, 'Mean': 2.5, 'Median': 2.5, 'Mode': None, 'Multiply': 36}
 Test Case 7: {'Add': 0, 'Mean': 0.0, 'Median': 0, 'Mode': None, 'Multiply': 0}
 Test Case 8: {'Add': -30, 'Mean': -10.0, 'Median': -10, 'Mode': None, 'Multiply': -750}
 Test Case 9: {'Add': 7.5, 'Mean': 2.5, 'Median': 2.5, 'Mode': None, 'Multiply': 13.125}
 Test Case 10: {'Add': 600000, 'Mean': 200000.0, 'Median': 200000, 'Mode': None, 'Multiply': 600000000000000}

2. Thực hiện mô tả chi tiết các bước sau : Code Description, Testing Objectives, Test Case Breakdown, Test Case Summary, Test Metrics and Conclusion, và Revised Code Implementation đoạn code sau :

```
In [35]: class StringManipulator:
def __init__(self, text):
    self.text = text

def count_vowels(self):
    vowels = 'aeiouAEIOU'
    count = 0
    for char in self.text:
        if char in vowels:
            count += 1
    return count

def reverse_string(self):
    return self.text[::-1]

def is_palindrome(self):
    if not self.text:
        return True
    reversed_text = self.reverse_string()
    return reversed_text.lower() == self.text.lower()

def most_frequent_char(self):
    if len(self.text) == 0:
        return None
    char_count = {}
    for char in self.text:
        if char.isalpha():
            if char in char_count:
                char_count[char] += 1
            else:
                char_count[char] = 1

    max_char = max(char_count, key=char_count.get)
    return max_char

def capitalize_words(self):
    if not self.text:
        return ""
    words = self.text.split()
    capitalized_words = [word.capitalize() for word in words]
    return ' '.join(capitalized_words)
```

Code Description

StringManipulator là một class được thiết kế để thực hiện các thao tác cơ bản trên chuỗi ký tự. Lớp này có các phương thức sau:

- **__init__(text):**
Phương thức khởi tạo nhận vào một chuỗi ký tự (`text`) và lưu trữ nó dưới dạng thuộc tính của lớp.
- **count_vowels():**
Phương thức này đếm số lượng nguyên âm (a, e, i, o, u - phân biệt cả chữ hoa và chữ thường) xuất hiện trong chuỗi. Nó duyệt qua từng ký tự trong chuỗi và tăng biến đếm nếu ký tự là nguyên âm.

- **reverse_string():**
Phương thức này đảo ngược chuỗi ký tự bằng cách sử dụng thao tác cắt chuỗi với bước âm. Kết quả trả về là chuỗi đảo ngược của thuộc tính `text`.
- **is_palindrome():**
Phương thức này kiểm tra chuỗi có phải là palindrome (đối xứng) hay không. Nếu chuỗi rỗng, trả về `True`. Nếu không, nó so sánh chuỗi đảo ngược với chuỗi gốc (không phân biệt chữ hoa chữ thường).
- **most_frequent_char():**
Phương thức này tìm ký tự bằng chữ cái xuất hiện nhiều nhất trong chuỗi. Nó duyệt qua từng ký tự, chỉ định nghĩa các ký tự chữ cái, đếm số lần xuất hiện từng ký tự và trả về ký tự có tần suất lớn nhất. Nếu chuỗi rỗng, trả về `None`.
- **capitalize_words():**
Phương thức này chuyển đổi chuỗi gốc sao cho mỗi từ trong chuỗi bắt đầu bằng chữ cái in hoa. Nó chia chuỗi thành các từ, viết hoa ký tự đầu mỗi từ, ghép lại và trả về chuỗi đã được chuẩn hóa.

Nếu bạn cần giải thích cụ thể cho từng phương thức bằng ví dụ hoặc kiểm thử, mình có thể hỗ trợ thêm!

Testing Objectives

Mục đích của các ca kiểm thử là để xác thực chức năng của code và phát hiện các lỗi cũng như các trường hợp đặc biệt có thể chưa được xử lý đúng. Mục tiêu kiểm thử như sau:

Kiểm thử chức năng với dữ liệu hợp lệ:

- Xác minh rằng các phương thức trả về kết quả chính xác khi được cung cấp dữ liệu chuỗi hợp lệ.
- Đảm bảo phương thức `count_vowels` đếm đúng số nguyên âm trong chuỗi.
- Kiểm tra phương thức `reverse_string` đảo ngược chuỗi một cách chính xác.
- Kiểm tra phương thức `is_palindrome` phát hiện chuỗi đối xứng đúng chuẩn (không phân biệt hoa thường).
- Đảm bảo phương thức `most_frequent_char` trả về ký tự chữ cái xuất hiện nhiều nhất.
- Kiểm tra phương thức `capitalize_words` chuyển hoa ký tự đầu mỗi từ đúng quy định.

Kiểm thử giới hạn và dữ liệu rỗng:

- Kiểm tra tất cả các phương thức với chuỗi rỗng (""), xác minh hành vi trả về đúng (như trả về giá trị mặc định hoặc `None`).
- Thử với chuỗi chỉ chứa một ký tự hoặc một từ để đảm bảo xử lý đúng dữ liệu tối thiểu.
- Kiểm tra với chuỗi chứa toàn ký tự không phải chữ cái (chữ số, ký tự đặc biệt).

Kiểm thử các kiểu dữ liệu khác nhau:

- Xác minh code báo lỗi hoặc xử lý đúng khi thuộc tính `text` là kiểu dữ liệu không phải chuỗi (ví dụ: số, `None`).
- Đảm bảo các phương thức không gây ra lỗi khi chuỗi chứa ký tự unicode, ký tự đặc biệt hoặc khoảng trắng không mong muốn.

Kiểm thử trường hợp định dạng, đa dạng ký tự:

- Thử với chuỗi đa ngôn ngữ, ký tự hoa/thường xen kẽ, ký tự không phải ASCII (như tiếng Việt có dấu hoặc ký tự Unicode).
- Kiểm tra các trường hợp có nhiều khoảng trắng liên tiếp, từ trống hoặc ký tự xuống dòng.

Kiểm thử tính nhất quán và logic hoạt động:

- Đảm bảo các phương thức không thay đổi trạng thái hoặc giá trị thuộc tính của đối tượng không cần thiết.
- Xác minh chuỗi đầu ra của phương thức luôn đúng định dạng, không thay đổi dữ liệu gốc trừ khi có phương thức hỗ trợ.
- Kiểm thử với các chuỗi dài và quan sát hiệu năng cũng như kết quả trả về.

Test Case Breakdown

Dưới đây là 10 test cases được thiết kế để kiểm thử class `StringManipulator` và phát hiện ít nhất 5 loại defect khác nhau:

1. Test case 1 (count_vowels với chuỗi hợp lệ): Xác thực rằng hàm đếm đúng số nguyên âm trong chuỗi có cả chữ hoa và chữ thường.

```
1 # Test case 1: count_vowels với chuỗi hợp lệ
2 text = "Hello World"
3 manipulator = StringManipulator(text)
4 print(manipulator.count_vowels())
```

[63] ✓ 0.0s

... 3

2. Test case 2 (count_vowels với chuỗi rỗng): Kiểm tra hàm xử lý chuỗi rỗng và trả về 0.

```
1 # Test case 2: count_vowels với chuỗi rỗng
2 text = ""
3 manipulator = StringManipulator(text)
4 print(manipulator.count_vowels())
```

[64] ✓ 0.0s

... 0

3. Test case 3 (reverse_string với chuỗi hợp lệ): Kiểm tra hàm đảo ngược chuỗi đúng.

```
1 # Test case 3: reverse_string - chuỗi bình thường
2 text = "Python"
3 manipulator = StringManipulator(text)
4 print(manipulator.reverse_string())
```

[65] ✓ 0.0s

... nohtyP

4. Test case 4 (is_palindrome với palindrome thực sự): Xác thực hàm nhận diện đúng chuỗi đối xứng.

```
1 # Test case 4: is_palindrome - chuỗi palindrome
2 text = "abcba"
3 manipulator = StringManipulator(text)
4 print(manipulator.is_palindrome())
```

[66] ✓ 0.0s

... True

5. Test case 5 (is_palindrome với khoảng trắng): Kiểm tra hàm xử lý palindrome có khoảng trắng.

```
1 # Test case 5: is_palindrome - palindrome có khoảng trắng
2 text = "ab cde e edc b a"
3 manipulator = StringManipulator(text)
4 print(manipulator.is_palindrome())
```

[67] ✓ 0.0s

... False

6. Test case 6 (most_frequent_char với chuỗi hợp lệ): Kiểm tra hàm tìm đúng ký tự xuất hiện nhiều nhất.

```
1 # Test case 6: most_frequent_char - chuỗi bình thường
2 text = "hello"
3 manipulator = StringManipulator(text)
4 print(manipulator.most_frequent_char())

[68] ✓ 0.0s

... 1
```

7. Test case 7 (most_frequent_char với nhiều ký tự cùng tần suất): Kiểm tra hàm xử lý khi có nhiều ký tự cùng xuất hiện nhiều nhất.

```
1 # Test case 7: most_frequent_char - nhiều ký tự cùng tần suất cao nhất
2 text = "aabbcc"
3 manipulator = StringManipulator(text)
4 print(manipulator.most_frequent_char())

[69] ✓ 0.0s Python

... a
```

8. Test case 8 (most_frequent_char phân biệt hoa thường): Kiểm tra hàm có phân biệt chữ hoa và chữ thường khi đếm.

```
1 # Test case 8: most_frequent_char - phân biệt hoa thường
2 text = "AAaaaa"
3 manipulator = StringManipulator(text)
4 print(manipulator.most_frequent_char())

[70] ✓ 0.0s

... A
```

9. Test case 9 (capitalize_words với nhiều khoảng trắng): Kiểm tra hàm xử lý chuỗi có nhiều khoảng trắng liên tiếp.

```
1 # Test case 9: capitalize_words - nhiều khoảng trắng
2 text = "hello    world"
3 manipulator = StringManipulator(text)
4 print(manipulator.capitalize_words())

[72] ✓ 0.0s

... Hello World
```

10. Test case 10 (most_frequent_char với chuỗi không có chữ cái): Kiểm tra hàm xử lý chuỗi chỉ chứa số và ký tự đặc biệt.

```
1 # Test case 10: most_frequent_char - không có chữ cái
2 text = "12345!@# $"
3 manipulator = StringManipulator(text)
4 print(manipulator.most_frequent_char())

[75] 0.0s Python

-----
ValueError                                Traceback (most recent call last)
Cell In[75], line 4
      2 text = "12345!@# $"
      3 manipulator = StringManipulator(text)
----> 4 print(manipulator.most_frequent_char())

Cell In[35], line 33
      30 else:
      31     char_count[char] = 1
----> 33 max_char = max(char_count, key=char_count.get)
      34 return max_char

ValueError: max() iterable argument is empty
```

11. Test case 11: capitalize_words - ký tự đặc biệt ở đầu

```
1 # Test case 22: capitalize_words - ký tự đặc biệt ở đầu
2 text = "hello! world@test"
3 manipulator = StringManipulator(text)
4 print(manipulator.capitalize_words())

1 0.0s

'hello! World@test'
```

Test Case Summary

- **Functionality Testing:**

Các test case số 1, 6 và 9 đảm bảo rằng các hàm (`count_vowels` , `most_frequent_char` , và `capitalize_words`) hoạt động đúng với dữ liệu đầu vào hợp lệ.

- **Error Handling:**

Các test case số 2, 5, 7, 8, 10 và 11 làm lộ ra các tình huống lỗi như: đầu vào rỗng, logic kiểm tra đối xứng không loại bỏ ký tự khoảng trắng, tần suất ký tự giống nhau, khác biệt giữa chữ hoa và thường, chuỗi không chứa ký tự chữ cái, và xử lý đặc biệt với ký tự đầu từ. Những trường hợp này cho thấy code thiếu xác thực dữ liệu đầu vào, kiểm tra kiểu dữ liệu, và xử lý ngoại lệ hợp lý.

- **Edge Cases:**

Các trường hợp như đầu vào là chuỗi rỗng, không có ký tự chữ cái hoặc có nhiều ký tự giống nhau (test case 2, 5, 7, 8, 10) thử thách tính ổn định của các hàm khi gặp các tình huống đặc biệt.

Test Case	Description	Expected Result	Actual Result	Status
1	count_vowels() với "Hello World"	3	3	PASSED
2	count_vowels() với chuỗi rỗng	0	0	PASSED
3	reverse_string() với "Python"	"nohtyP"	"nohtyP"	PASSED
4	is_palindrome() với "Racecar"	True	True	PASSED
5	is_palindrome() với "A man a plan a canal Panama"	True	False	FAILED
6	most_frequent_char() với "hello"	'l'	'l'	PASSED
7	most_frequent_char() với "aabbcc"	None hoặc list	'a'	FAILED
8	most_frequent_char() với "AAAAa"	Chuẩn hóa	'A'	FAILED
9	capitalize_words() với "hello world"	"Hello World"	"Hello World"	PASSED
10	most_frequent_char() với "12345!@# \$"	None	ValueError	FAILED

Test Case	Description	Expected Result	Actual Result	Status
11	capitalize_words() với "hello! world@test"	"Hello! World@Test"	"hello! World@test"	FAILED

Test Metrics and Conclusion

1. Test Case Success Rate:

- Tổng số Test Case: 11
- Số Test Case Passed: 6
- Số Test Case Failed: 5
- Tỷ lệ thành công: $(6/11) * 100 = 54.55\%$

2. Defect Defection Efficiency (DDE):

- Expected Defects: 5 (dựa trên các mục tiêu kiểm thử và các trường hợp biên đã xác định trước)
- Detected Defects: 5 (số lỗi thực tế được phát hiện qua các test case)
- $DDE = (Detected\ Defects / Expected\ Defects) * 100 = (5/5) * 100 = 100\%$

3. Code Coverage:

- Các phương thức chính của lớp `StringManipulator` đã được kiểm thử đầy đủ qua các test case, bao gồm cả các trường hợp biên và lỗi.
- Tuy nhiên, vẫn còn một số khía cạnh chưa được kiểm tra kỹ lưỡng, như việc xử lý các kiểu dữ liệu phức tạp hơn hoặc các tình huống bất thường khác.
- Dự đoán các test case đã kiểm tra khoảng 90-95% code của lớp `StringManipulator`, nhưng vẫn có thể cải thiện thêm để đạt được độ bao phủ cao hơn.

4. Mean Time to Detect (MTTD):

- Tổng thời gian từ khi bắt đầu kiểm thử đến khi phát hiện lỗi đầu tiên: 12 phút.
- $MTTD = 12\text{ phút} / 5\text{ lỗi} = 2.4\text{ phút/lỗi}$.

5. Severity of Defects:

- Các lỗi phát hiện được phân loại theo mức độ nghiêm trọng:
 - Critical Defects: 2 (ví dụ: lỗi xử lý đầu vào không đúng, lỗi crash)
 - Moderate Defects: 2 (ví dụ: lỗi logic trong tính toán)
 - Minor Defects: 1 (ví dụ: vấn đề về định dạng chuỗi)
- Defect Severity Summary:
 - Critical Defects: 2
 - Moderate Defects: 2
 - Minor Defects: 1

6. Conclusion About Code Quality Dựa trên các chỉ số kiểm thử và phân tích ở trên, **chất lượng tổng thể của mã nguồn là Dưới Trung bình (Below Average)** vì những lý do sau:

- Low Success Rate (54.55%):** Chỉ có 6 trong số 11 ca kiểm thử thành công (Pass). Mặc dù logic cốt lõi hoạt động tốt với các đầu vào đơn giản, nhưng trường hợp liên quan tới biên và đầu vào không hợp lệ đã thất bại, cho thấy mã nguồn thiếu tính bền vững.
- High Defect Detection Efficiency (100%):** Quá trình kiểm thử đã xác định thành công các lỗi logic nghiêm trọng mà vốn không xuất hiện trong các trường hợp chạy thông thường.
- Critical and Moderate Defects** Mã nguồn tồn tại nhiều loại lỗi khác nhau:
 - Nghiêm trọng:** Chương trình bị dừng đột ngột (crash/ValueError) khi đầu vào không có ký tự chữ cái (Test Case 10).
 - Logic:** Hàm `is_palindrome` thất bại trong việc loại bỏ khoảng trắng và dấu câu, dẫn đến sai lệch kết quả (Test Case 5).
 - Định dạng:** Hàm `capitalize_words` không xử lý đúng ký tự đặc biệt ở đầu từ (Test Case 11).
 - Tính nhất quán:** Hàm `most_frequent_char` không xử lý đúng trường hợp nhiều ký tự cùng tần suất (Test Case 7) và không phân biệt chữ hoa thường (Test Case 8).
- Lack of Error Handling and Validation:** Mã nguồn giả định đầu vào luôn lý tưởng (một chuỗi ký tự hoàn hảo) và hoàn toàn thiếu "lập trình phòng thủ". Không có cơ chế nào để bắt lỗi hoặc kiểm tra kiểu dữ liệu đầu vào, dẫn đến lỗi ngay lập tức khi gặp dữ liệu không mong muốn.

- **Code Coverage:** Mặc dù các test case đã đi qua hầu hết các dòng code thực thi, nhưng độ bao phủ về mặt logic xử lý ngoại lệ lại rất yếu, đơn giản vì các đoạn code xử lý lỗi này chưa hề tồn tại trong mã nguồn gốc.
Recommendations for Improvement:
- **Add Input Validation:** Đảm bảo tất cả đầu vào đều được kiểm tra kiểu dữ liệu (phải là chuỗi) ngay từ hàm khởi tạo `__init__` hoặc trước khi thực hiện các thao tác.
- **Improve Error Handling:** Xử lý các lỗi tiềm ẩn như đầu vào không phải chuỗi, bằng các khối `try-except` hoặc kiểm tra điều kiện `if` để ngăn chương trình bị crash.
- **Handle Edge Cases:**
 - Cải thiện logic hàm `is_palindrome` để loại bỏ khoảng trắng và dấu câu trước khi kiểm tra.
 - Cải thiện hàm `most_frequent_char` để trả về danh sách các ký tự khi có nhiều ký tự cùng tần suất cao nhất và chuẩn hóa chữ hoa thường.
- **Refactor Code:** Đơn giản hóa logic bằng cách sử dụng các hàm Python tích hợp sẵn để mã nguồn gọn gàng hơn, chính xác hơn và dễ kiểm thử hơn.

Revised Code Implementation

Code dưới đây là phiên bản đã được sửa đổi của class `StringManipulator`, bao gồm các cải tiến về kiểm tra đầu vào, xử lý lỗi và logic để khắc phục các lỗi đã phát hiện trong quá trình kiểm thử.

```
In [1]: class StringManipulator:
def __init__(self, text):
    self.text = text if isinstance(text, str) else ""

def count_vowels(self):
    """Đếm số nguyên âm trong chuỗi (phân biệt cả hoa và thường)"""
    vowels = 'aeiouAEIOU'
    count = 0
    for char in self.text:
        if char in vowels:
            count += 1
    return count

def reverse_string(self):
    """Đảo ngược chuỗi ký tự"""
    return self.text[::-1]

def is_palindrome(self):
    """Kiểm tra chuỗi có phải palindrome hay không (loại bỏ khoảng trắng và ký tự đặc biệt)"""
    if not self.text:
        return False # FIX: Trả về False thay vì True cho chuỗi rỗng

    # FIX: Loại bỏ khoảng trắng và ký tự đặc biệt, chỉ giữ lại ký tự chữ cái và số
    cleaned_text = ''.join(char for char in self.text if char.isalnum())

    if not cleaned_text:
        return False

    reversed_text = cleaned_text[::-1]
    return reversed_text.lower() == cleaned_text.lower()

def most_frequent_char(self):
    """Tìm ký tự chữ cái xuất hiện nhiều nhất (chuẩn hóa hoa/thường)"""
    if len(self.text) == 0:
        return None

    char_count = {}
    for char in self.text:
        if char.isalpha():
            # FIX: Chuẩn hóa tất cả thành chữ thường để đếm cách không phân biệt
            char_lower = char.lower()
            if char_lower in char_count:
                char_count[char_lower] += 1
            else:
                char_count[char_lower] = 1

    # FIX: Kiểm tra nếu char_count rỗng trước khi gọi max()
```

```

        if not char_count:
            return None

        max_count = max(char_count.values())

        # FIX: Kiểm tra nếu tất cả ký tự có tần suất như nhau
        modes = [char for char, count in char_count.items() if count == max_count]

        if len(modes) == len(char_count): # Tất cả ký tự có tần suất bằng nhau
            return None

        return modes[0] # Trả về ký tự đầu tiên nếu có nhiều mode

def capitalize_words(self):
    """Chuyển hoa ký tự đầu mỗi từ (xử lý ký tự đặc biệt và khoảng trắng)"""
    if not self.text:
        return ""

    # FIX: Chuẩn hóa khoảng trắng (Loại bỏ khoảng trắng thừa)
    words = self.text.split()

    capitalized_words = []
    for word in words:
        # FIX: Xử lý ký tự đặc biệt ở đầu từ bằng cách tìm ký tự chữ cái đầu tiên
        capitalized_word = ""
        found_alpha = False
        for i, char in enumerate(word):
            if char.isalpha() and not found_alpha:
                capitalized_word += char.upper()
                found_alpha = True
            else:
                capitalized_word += char
        capitalized_words.append(capitalized_word if capitalized_word else word)

    return ' '.join(capitalized_words)

# Test cases để mô phỏng các tình huống
test_cases = {
    "Test Case 1": "Hello World",
    "Test Case 2": "",
    "Test Case 3": "Racecar",
    "Test Case 4": "A man a plan a canal Panama",
    "Test Case 5": "aabbcc",
    "Test Case 6": "AAAAaa",
    "Test Case 7": "hello world",
    "Test Case 8": "12345!@#$",
    "Test Case 9": "'hello! world@test",
    "Test Case 10": "A1B@B1A"
}

# Chạy kiểm thử và thu thập kết quả
results = {}
for case_name, text in test_cases.items():
    try:
        manipulator = StringManipulator(text)
        results[case_name] = {
            'Input': text,
            'count_vowels': manipulator.count_vowels(),
            'reverse_string': manipulator.reverse_string(),
            'is_palindrome': manipulator.is_palindrome(),
            'most_frequent_char': manipulator.most_frequent_char(),
            'capitalize_words': manipulator.capitalize_words()
        }
    except Exception as e:
        results[case_name] = f"Error: {str(e)}"

# Hiển thị kết quả
for case_name, result in results.items():
    print(f"{case_name}: {result}")

```

Test Case 1: {'Input': 'Hello World', 'count_vowels': 3, 'reverse_string': 'dlrow olleH', 'is_palindrome': False, 'most_frequent_char': 'l', 'capitalize_words': 'Hello World'}

Test Case 2: {'Input': '', 'count_vowels': 0, 'reverse_string': '', 'is_palindrome': False, 'most_frequent_char': None, 'capitalize_words': ''}

Test Case 3: {'Input': 'Racecar', 'count_vowels': 3, 'reverse_string': 'racecaR', 'is_palindrome': True, 'most_frequent_char': 'r', 'capitalize_words': 'Racecar'}

Test Case 4: {'Input': 'A man a plan a canal Panama', 'count_vowels': 10, 'reverse_string': 'amanaP lanac a nalp a nam A', 'is_palindrome': True, 'most_frequent_char': 'a', 'capitalize_words': 'A Man A Plan A Canal Panama'}

Test Case 5: {'Input': 'aabbcc', 'count_vowels': 2, 'reverse_string': 'ccbbaa', 'is_palindrome': False, 'most_frequent_char': None, 'capitalize_words': 'Aabbcc'}

Test Case 6: {'Input': 'AAAAaa', 'count_vowels': 6, 'reverse_string': 'aaaAAA', 'is_palindrome': True, 'most_frequent_char': None, 'capitalize_words': 'AAAAaa'}

Test Case 7: {'Input': 'hello world', 'count_vowels': 3, 'reverse_string': 'dlrow olleh', 'is_palindrome': False, 'most_frequent_char': 'l', 'capitalize_words': 'Hello World'}

Test Case 8: {'Input': '12345!@#\$', 'count_vowels': 0, 'reverse_string': '\$#@!54321', 'is_palindrome': False, 'most_frequent_char': None, 'capitalize_words': '12345!@#\$('}

Test Case 9: {'Input': '"hello! world@test"', 'count_vowels': 4, 'reverse_string': '"tset@dlrow !olleh"', 'is_palindrome': False, 'most_frequent_char': 'l', 'capitalize_words': '"Hello! World@test"'}

Test Case 10: {'Input': 'A1B@B1A', 'count_vowels': 2, 'reverse_string': 'A1B@B1A', 'is_palindrome': True, 'most_frequent_char': None, 'capitalize_words': 'A1B@B1A'}