Output By Sonnet 4.5

```
Palindrome Checker - Problem Explanation and Solution

Problem Understanding
We need to create a function that determines if a string reads the same
forwards and backwards, while ignoring:
Spaces
Punctuation marks
Letter case (uppercase vs lowercase)

For example:
"A man, a plan, a canal: Panama" → True (palindrome)
"race a car" → False (not a palindrome)

Logic Breakdown (Step by Step)

Step 1: Clean the Input String
Remove all non-alphanumeric characters (keep only letters and numbers)
Convert all characters to the same case (either lowercase or uppercase)

Step 2: Compare the Cleaned String
Compare the cleaned string with its reverse
If they match, it's a palindrome; otherwise, it's not

Step 3: Return Result
Return True if palindrome, False otherwise

Alternative Approach (Two Pointers)
Instead of creating a reversed string, we can use two pointers:
One pointer starts at the beginning
One pointer starts at the end
Move both pointers toward the center, comparing characters
This is more memory-efficient


Python Implementation
python
def is_palindrome(s: str) -> bool:
    """
    Check if a string is a palindrome, ignoring spaces, punctuation, and
case.
```

```
    Args:
        s: Input string to check

    Returns:
        bool: True if the string is a palindrome, False otherwise
    """
    # Method 1: Using string cleaning and reversal
    # Clean the string: keep only alphanumeric characters and convert to
lowercase
    cleaned = ''.join(char.lower() for char in s if char.isalnum())

    # Compare with its reverse
    return cleaned == cleaned[::-1]


def is_palindrome_two_pointers(s: str) -> bool:
    """
    Check if a string is a palindrome using two-pointer approach.
    More space-efficient alternative.

    Args:
        s: Input string to check

    Returns:
        bool: True if the string is a palindrome, False otherwise
    """
    # Initialize two pointers
    left = 0
    right = len(s) - 1

    while left < right:
        # Skip non-alphanumeric characters from the left
        while left < right and not s[left].isalnum():
            left += 1

        # Skip non-alphanumeric characters from the right
        while left < right and not s[right].isalnum():
            right -= 1

        # Compare characters (case-insensitive)
        if s[left].lower() != s[right].lower():
            return False
```

```
        # Move pointers toward center
        left += 1
        right -= 1

    return True


# Test cases
if __name__ == "__main__":
    test_cases = [
        ("A man, a plan, a canal: Panama", True),
        ("race a car", False),
        ("Was it a car or a cat I saw?", True),
        ("Madam", True),
        ("hello", False),
        ("A Santa at NASA", True),
        ("", True),  # Empty string is considered a palindrome
        ("a", True),  # Single character is a palindrome
        ("ab", False),
        ("Able was I ere I saw Elba", True),  # Napoleon's famous
palindrome
    ]

    print("Testing Method 1 (String Reversal):")
    print("-" * 50)
    for test_string, expected in test_cases:
        result = is_palindrome(test_string)
        status = "✓" if result == expected else "✗"
        print(f"{status} '{test_string}' → {result}")

    print("\n\nTesting Method 2 (Two Pointers):")
    print("-" * 50)
    for test_string, expected in test_cases:
        result = is_palindrome_two_pointers(test_string)
        status = "✓" if result == expected else "✗"
        print(f"{status} '{test_string}' → {result}")
```

Output

```
Testing Method 1 (String Reversal):
--------------------------------------------------
✓ 'A man, a plan, a canal: Panama' → True
✓ 'race a car' → False
✓ 'Was it a car or a cat I saw?' → True
```

```
✓ 'Madam' → True
✓ 'hello' → False
✓ 'A Santa at NASA' → True
✓ '' → True
✓ 'a' → True
✓ 'ab' → False
✓ 'Able was I ere I saw Elba' → True


Testing Method 2 (Two Pointers):
--------------------------------------------------
✓ 'A man, a plan, a canal: Panama' → True
✓ 'race a car' → False
✓ 'Was it a car or a cat I saw?' → True
✓ 'Madam' → True
✓ 'hello' → False
✓ 'A Santa at NASA' → True
✓ '' → True
✓ 'a' → True
✓ 'ab' → False
✓ 'Able was I ere I saw Elba' → True

Time Complexity Analysis

Method 1 (String Reversal):
Time Complexity: O(n) where n is the length of the input string
Cleaning the string: O(n) - we iterate through each character once
Reversing the string: O(n) - creating the reversed string
Comparison: O(n) - comparing two strings character by character
Overall: O(n) + O(n) + O(n) = O(n)
Space Complexity: O(n) - we create a new cleaned string

Method 2 (Two Pointers):
Time Complexity: O(n) where n is the length of the input string
We iterate through the string once with two pointers
Each character is visited at most once
Space Complexity: O(1) - we only use a constant amount of extra space (two
pointers)
This is more space-efficient than Method 1
Conclusion: Both methods have O(n) time complexity, but the two-pointer
approach is more space-efficient with O(1) space vs O(n) space.
```

Output By Chatgpt 5.2

A palindrome is a string that reads the same forward and backward--after applying the comparison rules you want.
Here, the rules are:
Ignore spaces
Ignore punctuation
Ignore letter case
So "A man, a plan, a canal: Panama!" should count as a palindrome.

Step-by-step logic
Normalize the string (clean it):
Go through each character.
Keep only alphanumeric characters (letters and digits).
This automatically removes spaces and punctuation.
Convert kept characters to lowercase so case doesn't matter.
After this step, the string:
"A man, a plan, a canal: Panama!" becomes:
"amanaplanacanalpanama"
Check palindrome property:
Compare the cleaned string to its reverse.
If they are identical, it's a palindrome.
Return the result:
True if it matches its reverse, otherwise False.

Python implementation

```python
def is_palindrome(s: str) -> bool:
    # 1) Keep only letters/digits and make everything lowercase
    cleaned = ''.join(ch.lower() for ch in s if ch.isalnum())

    # 2) A palindrome equals its reverse
    return cleaned == cleaned[::-1]
```

Quick examples
python
Copy

```python
print(is_palindrome("A man, a plan, a canal: Panama!"))  # True
print(is_palindrome("No lemon, no melon"))               # True
print(is_palindrome("Hello, world!"))                    # False
```

Time complexity
Let
n

```
n be the length of the input string.
Cleaning the string scans it once:
O(n)
O(n)
Reversing and comparing is also
O(n)
O(n)
Total time complexity:
O(n)
O(n)
Space complexity (extra space):
O(n)
O(n) (for the cleaned string)
```