SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment 2.2

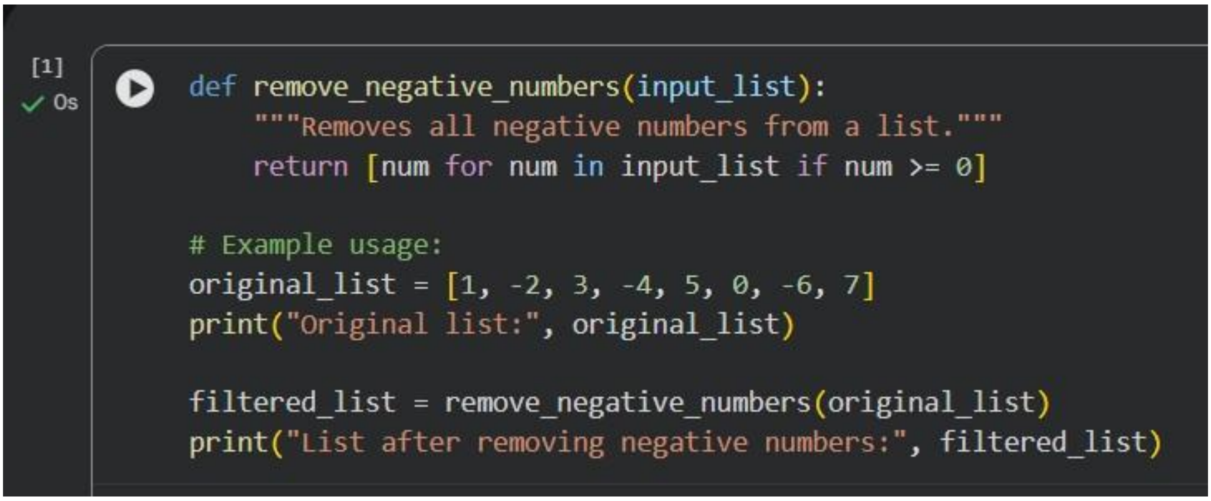
Program	**: B. Tech (CSE)**
Course Title	**: AI Assisted Coding**
Course Code	**: 23CS002PC304**
Semester	**: III**
Academic Session	**: 2025-2026**
Name of Student	**: Akhila G**
Enrollment No.	**: 2403A51L40**
Batch No.	**: 52**
Date	**: 13-01-2026**

Task 1:

Prompt: Generate a python function that removes all negative numbers from a list and show before and after output.

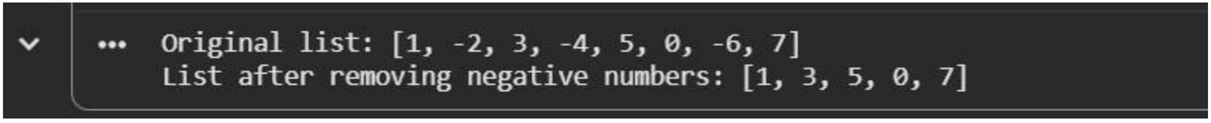
Code:

Gemini:



```python
def remove_negative_numbers(input_list):
    """Removes all negative numbers from a list."""
    return [num for num in input_list if num >= 0]

# Example usage:
original_list = [1, -2, 3, -4, 5, 0, -6, 7]
print("Original list:", original_list)

filtered_list = remove_negative_numbers(original_list)
print("List after removing negative numbers:", filtered_list)
```

Output:



```
Original list: [1, -2, 3, -4, 5, 0, -6, 7]
List after removing negative numbers: [1, 3, 5, 0, 7]
```

Explaination:

This program removes all negative numbers from a list.

- The function **remove_negative_numbers()** takes a list as input.
- It checks each number in the list.
- If the number is **0 or positive**, it is kept.
- If the number is **negative**, it is removed.

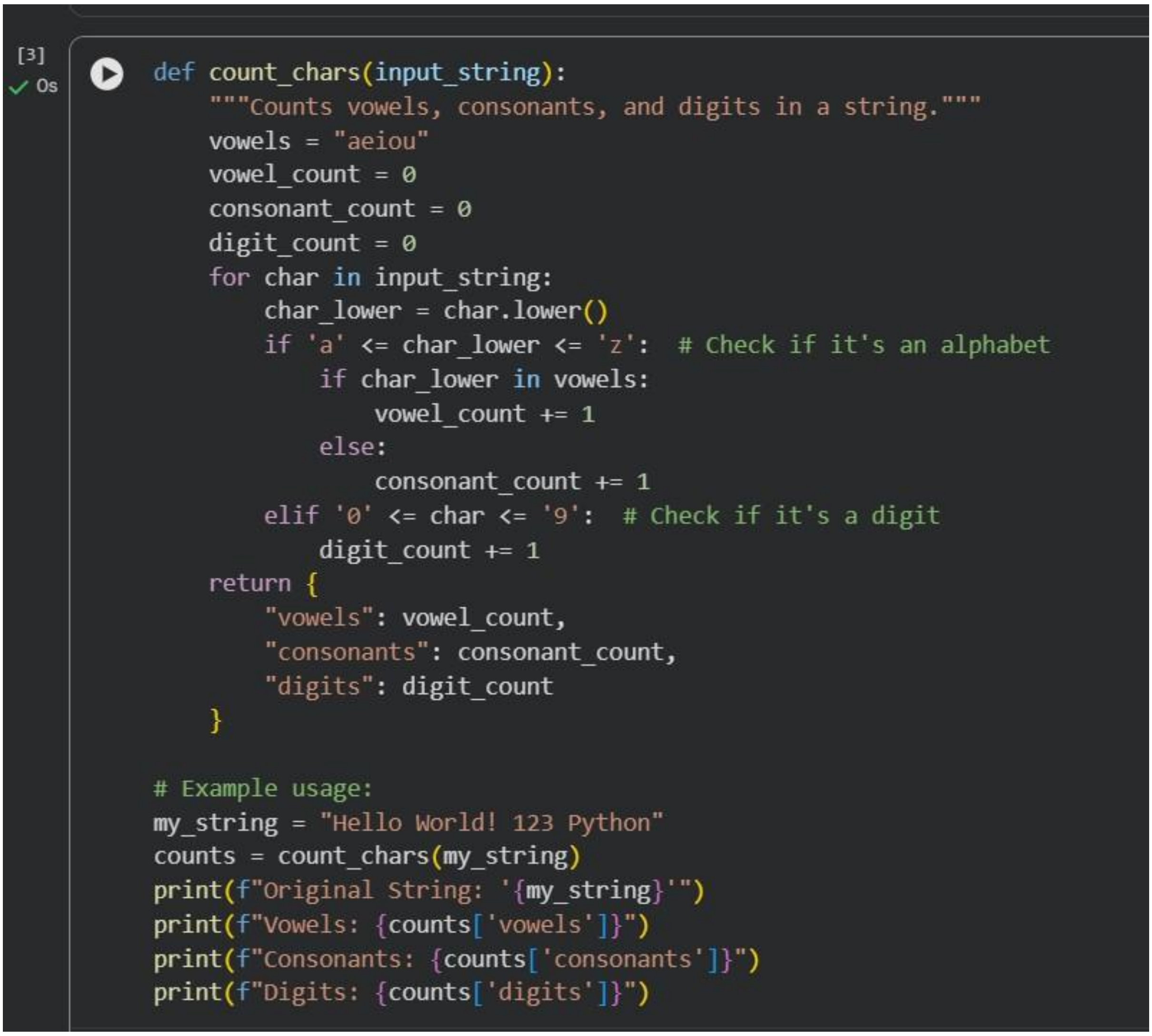
- A new list is returned without negative numbers.
- The original list remains unchanged.

Task 2:

Prompt: write a python function to count vowels,consonants,digits in a string.

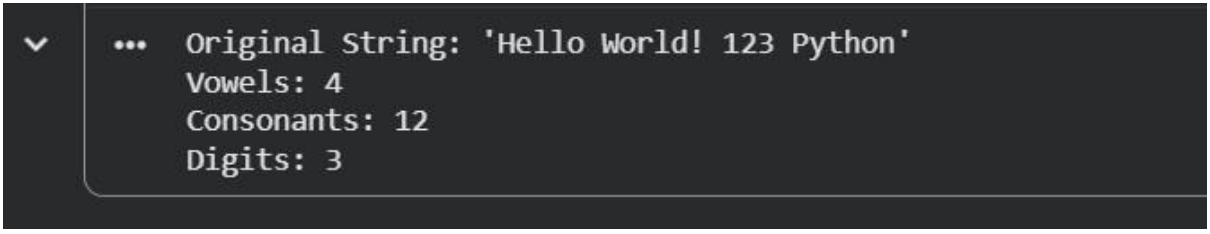
Code:

Gemini:



```python
def count_chars(input_string):
    """Counts vowels, consonants, and digits in a string."""
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    for char in input_string:
        char_lower = char.lower()
        if 'a' <= char_lower <= 'z':  # Check if it's an alphabet
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif '0' <= char <= '9':  # Check if it's a digit
            digit_count += 1
    return {
        "vowels": vowel_count,
        "consonants": consonant_count,
        "digits": digit_count
    }

# Example usage:
my_string = "Hello World! 123 Python"
counts = count_chars(my_string)
print(f"Original String: '{my_string}'")
print(f"Vowels: {counts['vowels']}")
print(f"Consonants: {counts['consonants']}")
print(f"Digits: {counts['digits']}")
```

Output:



```
Original String: 'Hello World! 123 Python'
Vowels: 4
Consonants: 12
Digits: 3
```

Explaination:

This program counts the number of vowels, consonants, and digits in a given string.

- The function **count_chars()** takes a string as input.
- It defines vowels as a, e, i, o, u.

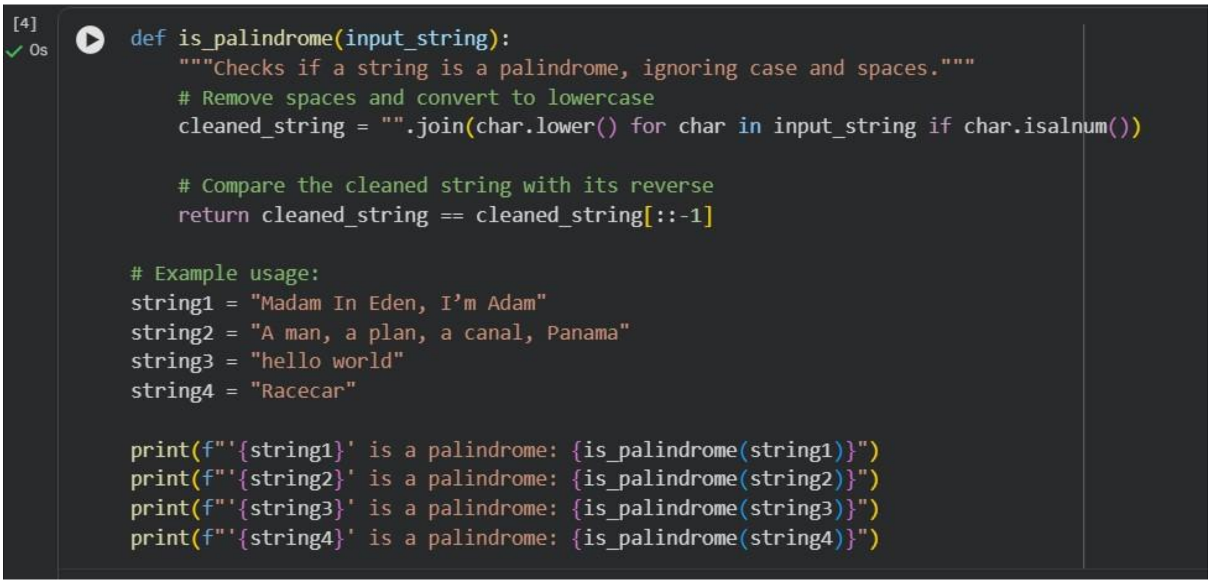
- Three counters are initialized:
    - **vowel_count** for vowels
    - **consonant_count** for consonants
    - **digit_count** for digits
- The program checks each character in the string:
    - If the character is an alphabet:
        - It is counted as a vowel if it is in the vowels list.
        - Otherwise, it is counted as a consonant.
    - If the character is a number (0–9), it is counted as a digit.
- Special characters and spaces are ignored.
- The function returns the total counts.
- The results are printed using the returned values.

Task 3:

Prompt: Generate a Python function to check whether a given string is a palindrome. The function should ignore case differences and spaces.

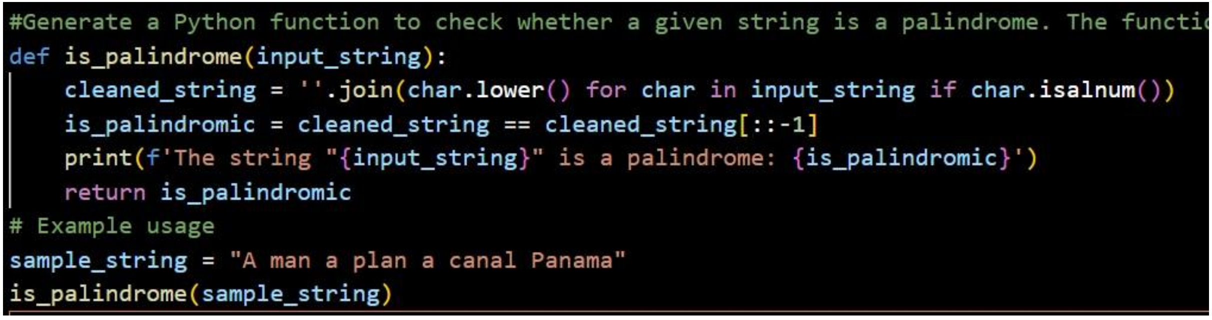
Code:

Gemini:



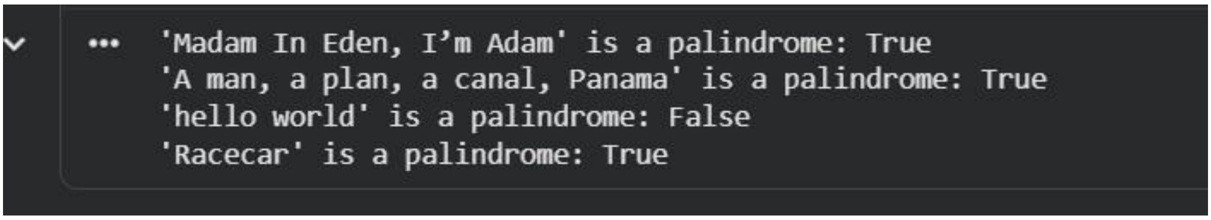
```
[4]
0s
def is_palindrome(input_string):
    """Checks if a string is a palindrome, ignoring case and spaces."""
    # Remove spaces and convert to lowercase
    cleaned_string = "".join(char.lower() for char in input_string if char.isalnum())

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

# Example usage:
string1 = "Madam In Eden, I’m Adam"
string2 = "A man, a plan, a canal, Panama"
string3 = "hello world"
string4 = "Racecar"

print(f"'{string1}' is a palindrome: {is_palindrome(string1)}")
print(f"'{string2}' is a palindrome: {is_palindrome(string2)}")
print(f"'{string3}' is a palindrome: {is_palindrome(string3)}")
print(f"'{string4}' is a palindrome: {is_palindrome(string4)}")
```

Copilot:

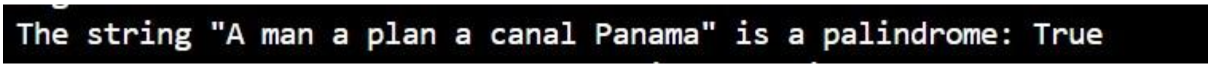


```
#Generate a Python function to check whether a given string is a palindrome. The functi
def is_palindrome(input_string):
    cleaned_string = ''.join(char.lower() for char in input_string if char.isalnum())
    is_palindromic = cleaned_string == cleaned_string[::-1]
    print(f'The string "{input_string}" is a palindrome: {is_palindromic}')
    return is_palindromic
# Example usage
sample_string = "A man a plan a canal Panama"
is_palindrome(sample_string)
```

Output:



```
'Madam In Eden, I’m Adam' is a palindrome: True
'A man, a plan, a canal, Panama' is a palindrome: True
'hello world' is a palindrome: False
'Racecar' is a palindrome: True
```



```
The string "A man a plan a canal Panama" is a palindrome: True
```

Explaination:
Gemini:

- Provides a clean and well-documented function with a docstring.
- Handles multiple test cases in the example.
- Separates logic and output clearly.
- Focuses on returning only the result (True or False) without printing inside the function.
- More suitable for academic assignments and lab records.
- Code is more readable and explanatory. **Copilot:**
- Provides a shorter and more compact code.
- Prints the result inside the function itself.
- Uses only one sample input for testing.
- Less explanation and documentation compared to Gemini.
- More suitable for quick coding and development purposes.

Conclusion:

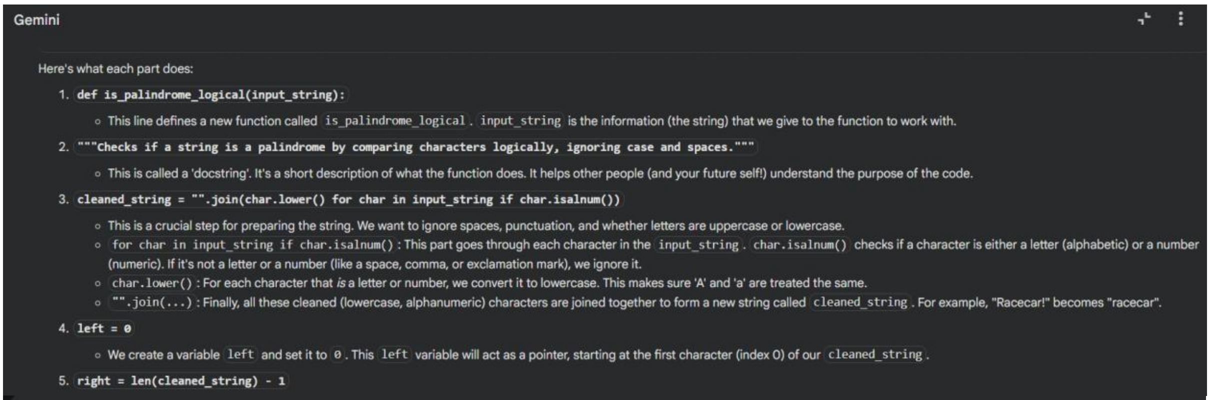
- Gemini is better for learning, clarity, and documentation.
- Copilot is better for fast coding and quick implementation.

Task 4:

Prompt:

Explain the following Python palindrome-checking function line by line in simple terms so that a beginner can understand it.

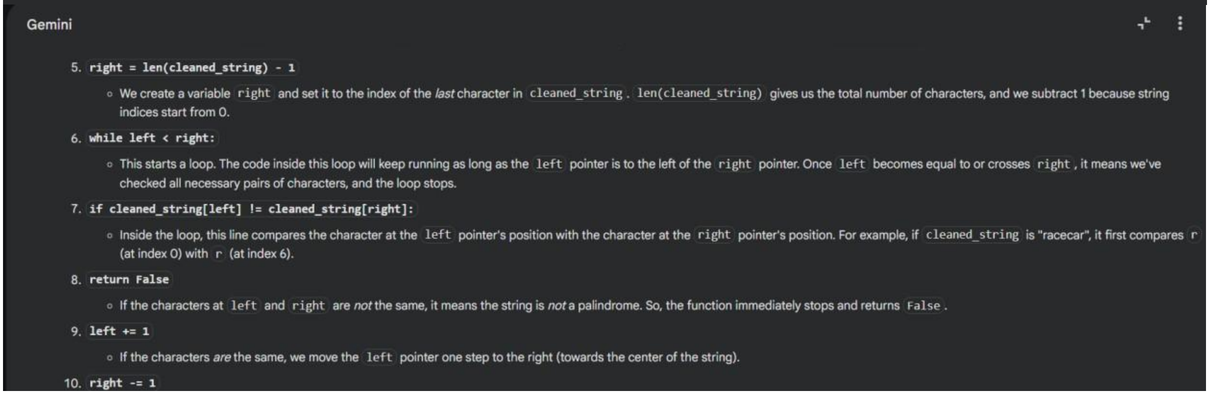
Explaination (Gemini)



Gemini

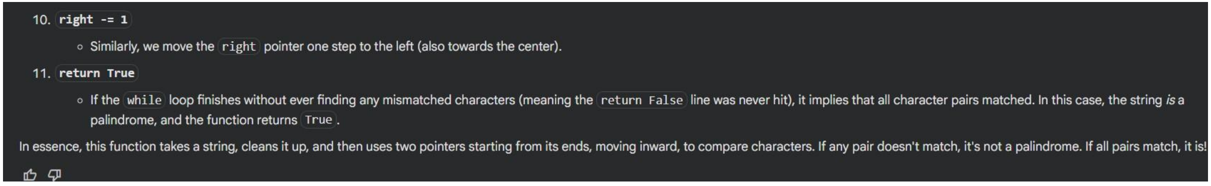
Here's what each part does:

1. **`def is_palindrome_logical(input_string):`**
    - This line defines a new function called `is_palindrome_logical`. `input_string` is the information (the string) that we give to the function to work with.
2. **`"""Checks if a string is a palindrome by comparing characters logically, ignoring case and spaces."""`**
    - This is called a 'docstring'. It's a short description of what the function does. It helps other people (and your future self!) understand the purpose of the code.
3. **`cleaned_string = "".join(char.lower() for char in input_string if char.isalnum())`**
    - This is a crucial step for preparing the string. We want to ignore spaces, punctuation, and whether letters are uppercase or lowercase.
    - `for char in input_string if char.isalnum()` : This part goes through each character in the `input_string`. `char.isalnum()` checks if a character is either a letter (alphabetic) or a number (numeric). If it's not a letter or a number (like a space, comma, or exclamation mark), we ignore it.
    - `char.lower()` : For each character that *is* a letter or number, we convert it to lowercase. This makes sure 'A' and 'a' are treated the same.
    - `"".join(...)` : Finally, all these cleaned (lowercase, alphanumeric) characters are joined together to form a new string called `cleaned_string`. For example, "Racecar!" becomes "racecar".
4. **`left = 0`**
    - We create a variable `left` and set it to `0`. This `left` variable will act as a pointer, starting at the first character (index 0) of our `cleaned_string`.
5. **`right = len(cleaned_string) - 1`**



Gemini

5. **`right = len(cleaned_string) - 1`**
    - We create a variable `right` and set it to the index of the *last* character in `cleaned_string`. `len(cleaned_string)` gives us the total number of characters, and we subtract 1 because string indices start from 0.
6. **`while left < right:`**
    - This starts a loop. The code inside this loop will keep running as long as the `left` pointer is to the left of the `right` pointer. Once `left` becomes equal to or crosses `right`, it means we've checked all necessary pairs of characters, and the loop stops.
7. **`if cleaned_string[left] != cleaned_string[right]:`**
    - Inside the loop, this line compares the character at the `left` pointer's position with the character at the `right` pointer's position. For example, if `cleaned_string` is "racecar", it first compares `r` (at index 0) with `r` (at index 6).
8. **`return False`**
    - If the characters at `left` and `right` are *not* the same, it means the string is *not* a palindrome. So, the function immediately stops and returns `False`.
9. **`left += 1`**
    - If the characters *are* the same, we move the `left` pointer one step to the right (towards the center of the string).
10. **`right -= 1`**



10. **`right -= 1`**
    - Similarly, we move the `right` pointer one step to the left (also towards the center).
11. **`return True`**
    - If the `while` loop finishes without ever finding any mismatched characters (meaning the `return False` line was never hit), it implies that all character pairs matched. In this case, the string *is* a palindrome, and the function returns `True`.

In essence, this function takes a string, cleans it up, and then uses two pointers starting from its ends, moving inward, to compare characters. If any pair doesn't match, it's not a palindrome. If all pairs match, it is!

Comment: The Gemini explanation is very detailed and well-structured. It explains each line of the code step by step, making the logic easy to understand even for beginners. The use of two-pointer technique (left and right) is clearly described, along with why characters are converted to lowercase and why nonalphanumeric characters are removed. The explanation also clearly states how and when the function returns True or False. Overall, this explanation is excellent for learning and understanding the algorithm, though it may feel slightly lengthy for quick reference.