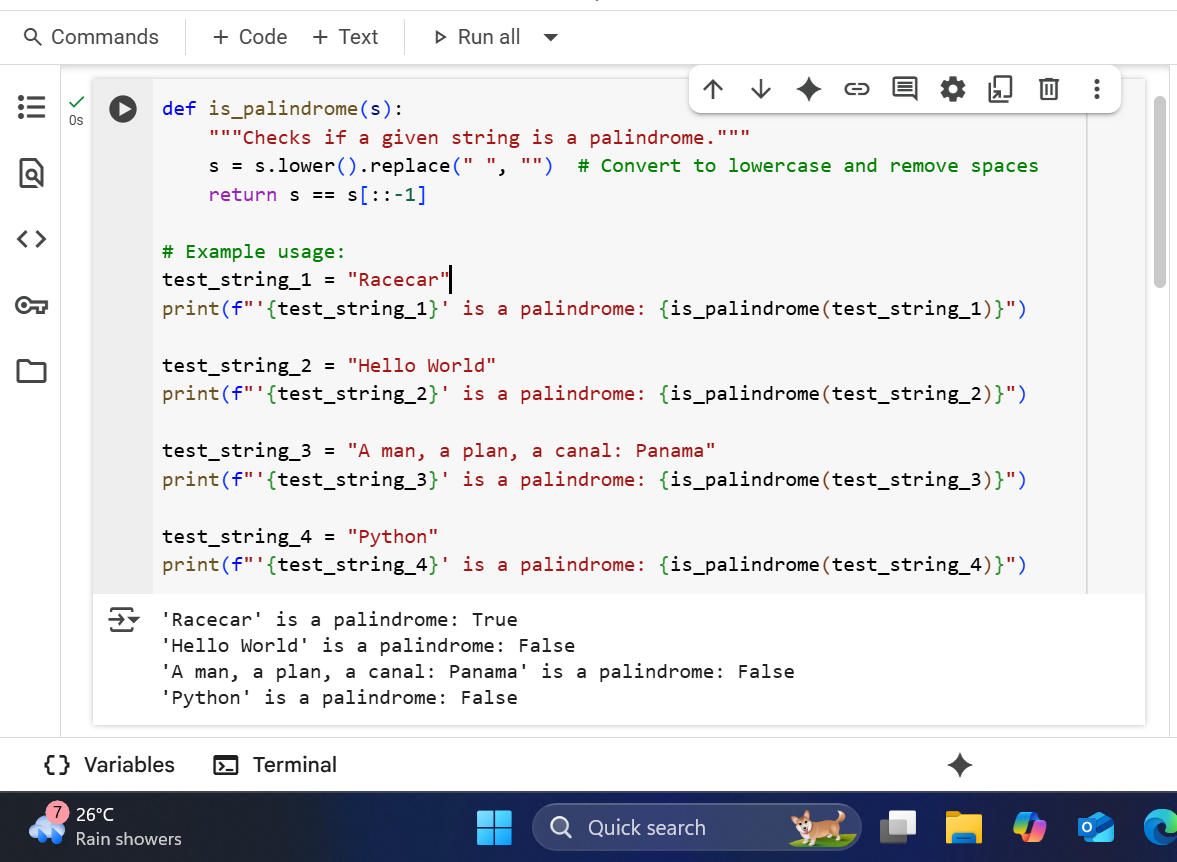
**LAB TEST -1**

**H.no: 2403A52139**

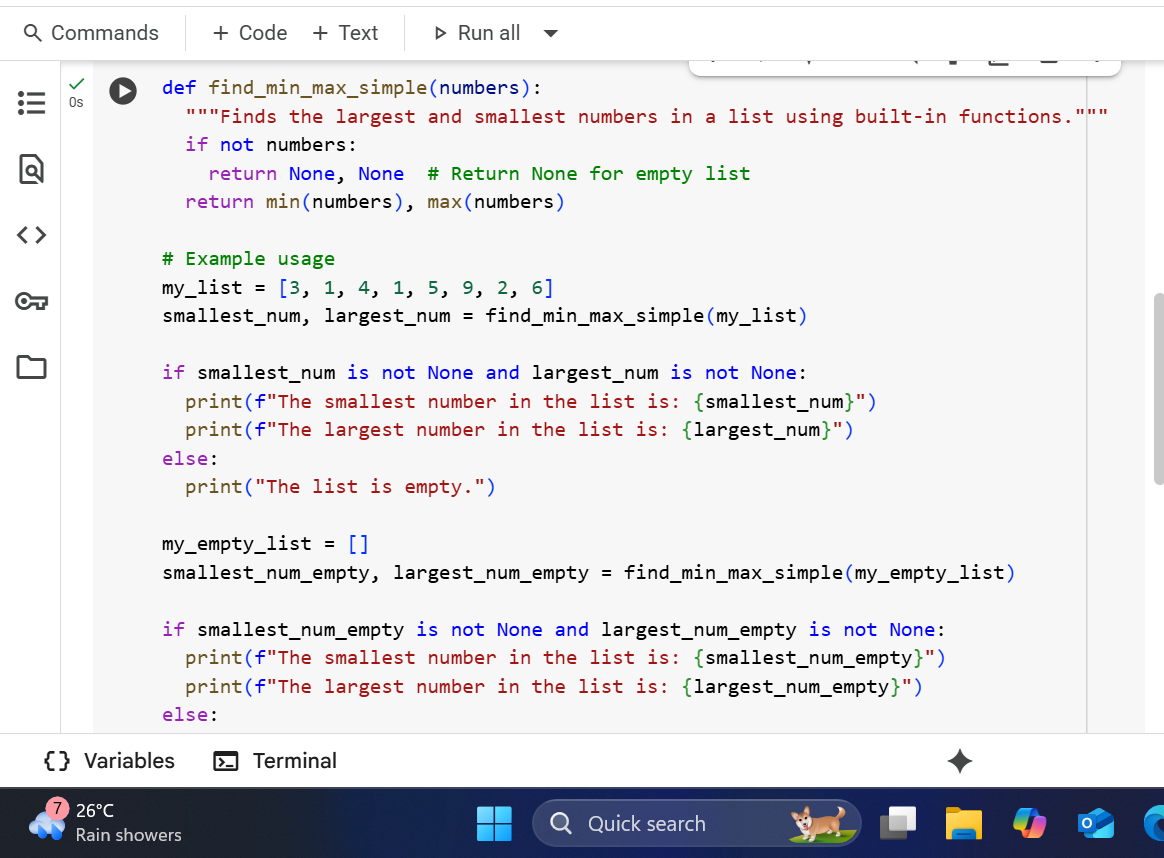
**Q1. To check whether the given string is palindrome or not**

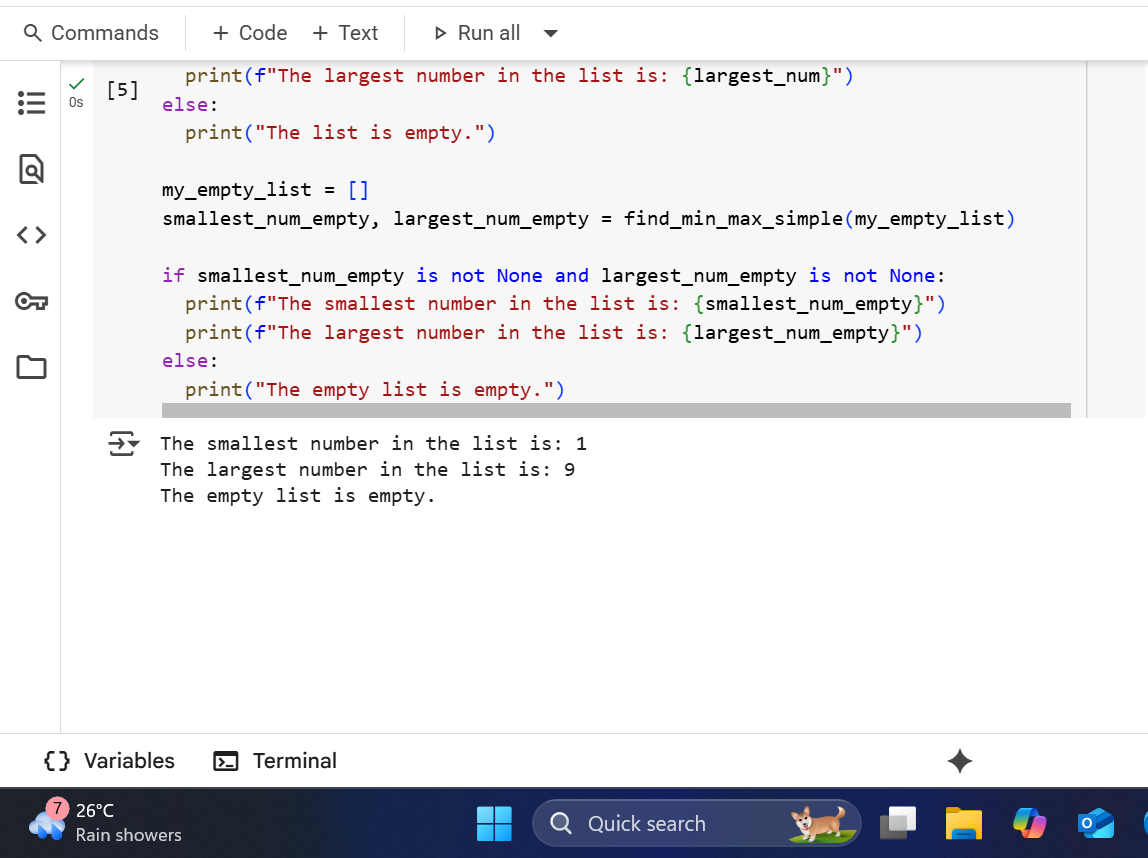


**EXPLANATION**

1. **def is\_palindrome(s):**: This line defines a function named is\_palindrome. It takes one argument, s, which is expected to be a string.
2. **"""Checks if a given string is a palindrome."""**: This is the docstring for the function, providing a brief description of what it does.
3. **s = s.lower().replace(" ", "") # Convert to lowercase and remove spaces**: This is where the input string is processed.
   * s.lower(): This method converts the entire string s to lowercase.
   * .replace(" ", ""): This method is then called on the lowercase string. It finds all occurrences of a space (" ") and replaces them with an empty string (""), effectively removing all spaces.
   * The result of these two operations (lowercase and no spaces) is then assigned back to the variable s, overwriting the original input string. The part after # is a comment explaining this step.
4. **return s == s[::-1]**: This is the core of the palindrome check.
   * s[::-1]: This creates a reversed copy of the processed string s. The [::-1] is a slicing trick where the first and second arguments of the slice are omitted (defaulting to the start and end of the string), and the step is -1, which means it steps backward through the string.
   * s == s[::-1]: This compares the processed string s with its reversed version. The == operator checks for equality. If the original processed string is equal to its reversed version, the expression evaluates to True (it's a palindrome). Otherwise, it evaluates to False.
   * return: The function returns the boolean result (True or False) of the comparison.
5. **# Example usage:**: This is a comment indicating that the lines below demonstrate how to use the is\_palindrome function.
6. **test\_string\_1 = "Racecar"**: This line creates a string variable named test\_string\_1 and assigns it the value "Racecar".
7. **print(f"'{test\_string\_1}' is a palindrome: {is\_palindrome(test\_string\_1)}")**: This line prints the result for the first test string.
   * f"...": This denotes an f-string, which allows embedding variables directly within the string using curly braces {}.
   * '{test\_string\_1}': This includes the value of test\_string\_1 in single quotes within the output string.
   * is a palindrome:: This is literal text printed as part of the output.
   * {is\_palindrome(test\_string\_1)}: This calls the is\_palindrome function with test\_string\_1 as the argument. The function returns either True or False, and that boolean value is embedded into the output string.
8. **test\_string\_2 = "Hello World"**: This line creates a string variable named test\_string\_2 and assigns it the value "Hello World".
9. **print(f"'{test\_string\_2}' is a palindrome: {is\_palindrome(test\_string\_2)}")**: This line is similar to line 7, but it calls is\_palindrome with test\_string\_2 and prints the result for that string.
10. **test\_string\_3 = "A man, a plan, a canal: Panama"**: This line creates a string variable named test\_string\_3 and assigns it the value "A man, a plan, a canal: Panama".
11. **print(f"'{test\_string\_3}' is a palindrome: {is\_palindrome(test\_string\_3)}")**: This line is similar to line 7, but it calls is\_palindrome with test\_string\_3 and prints the result for that string.
12. **test\_string\_4 = "Python"**: This line creates a string variable named test\_string\_4 and assigns it the value "Python".
13. **print(f"'{test\_string\_4}' is a palindrome: {is\_palindrome(test\_string\_4)}")**: This line is similar to line 7, but it calls is\_palindrome with test\_string\_4 and prints the result for that string.

**Q2. To find the largest and smallest numbers in the given list**





**EXPLANATION:**

1. def find\_min\_max\_simple(numbers):: This line defines a function named find\_min\_max\_simple. It takes one argument, which is expected to be a list of numbers, and it's named numbers within the function.
2. """Finds the largest and smallest numbers in a list using built-in functions.""": This is a docstring, which is a brief explanation of what the function does. It's good practice to include these for documentation.
3. if not numbers:: This line checks if the list numbers is empty. In Python, an empty list is considered "falsey", so not numbers will be True if the list is empty and False otherwise.
4. return None, None # Return None for empty list: If the list is empty (the if not numbers: condition is true), this line is executed. It returns two None values. This indicates that no smallest or largest number could be found because the list was empty. The part after # is a comment explaining this.
5. return min(numbers), max(numbers): If the list is not empty, this line is executed.
   * min(numbers): This is a built-in Python function that finds the smallest item in the iterable (in this case, the list numbers).
   * max(numbers): This is a built-in Python function that finds the largest item in the iterable (in this case, the list numbers).
   * The line returns the result of min(numbers) and max(numbers) as a tuple.
6. # Example usage: This is a comment indicating that the following lines are examples of how to use the function.
7. my\_list = [3, 1, 4, 1, 5, 9, 2, 6]: This line creates a list named my\_list and assigns it a set of integer values.
8. smallest\_num, largest\_num = find\_min\_max\_simple(my\_list): This line calls the find\_min\_max\_simple function with my\_list as the argument. The function returns a tuple containing the smallest and largest numbers. This line uses tuple unpacking to assign the first returned value (the smallest) to the variable smallest\_num and the second returned value (the largest) to the variable largest\_num.
9. if smallest\_num is not None and largest\_num is not None:: This line checks if both smallest\_num and largest\_num are not None. This is a way to check if the function returned actual numbers (meaning the list was not empty).
10. print(f"The smallest number in the list is: {smallest\_num}"): If the condition in the previous line is true, this line is executed. It prints a formatted string (indicated by the f before the opening quote). The {smallest\_num} inside the string is a placeholder that will be replaced by the value of the smallest\_num variable.
11. print(f"The largest number in the list is: {largest\_num}"): If the condition in line 9 is true, this line is also executed. It prints a formatted string, replacing {largest\_num} with the value of the largest\_num variable.
12. else:: This introduces the block of code to be executed if the condition in line 9 is false (meaning the list was empty and the function returned None).
13. print("The list is empty."): If the list was empty, this line is executed, printing a message indicating that.
14. my\_empty\_list = []: This line creates an empty list named my\_empty\_list.
15. smallest\_num\_empty, largest\_num\_empty = find\_min\_max\_simple(my\_empty\_list): This line calls the find\_min\_max\_simple function with the empty list my\_empty\_list. The function will return None, None. These None values are then assigned to smallest\_num\_empty and largest\_num\_empty.
16. if smallest\_num\_empty is not None and largest\_num\_empty is not None:: This line checks if the values returned for the empty list are not None. Since they are None, this condition will be false.
17. print(f"The smallest number in the list is: {smallest\_num\_empty}"): This line would be executed if the condition in the previous line were true (which it isn't for an empty list).
18. print(f"The largest number in the list is: {largest\_num\_empty}"): This line would also be executed if the condition in line 16 were true.
19. else:: This introduces the block of code to be executed if the condition in line 16 is false (meaning the function returned None for the empty list).
20. print("The empty list is empty."): Since the list was empty, this line is executed, printing a message specific to the empty list example.