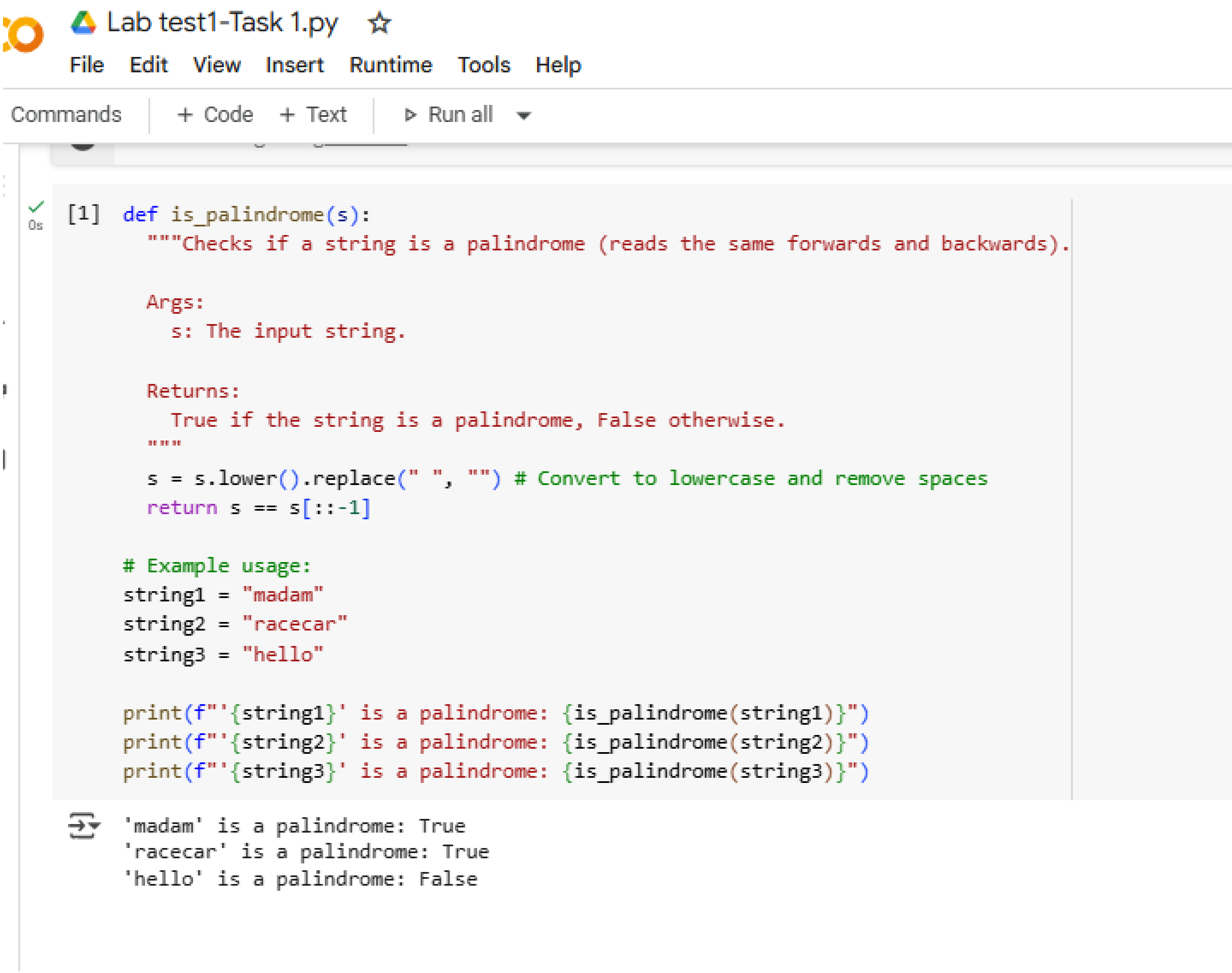
Lab Test-1

Question1:

Task1:

#Write a python program to check weather a string is palindrome or not.

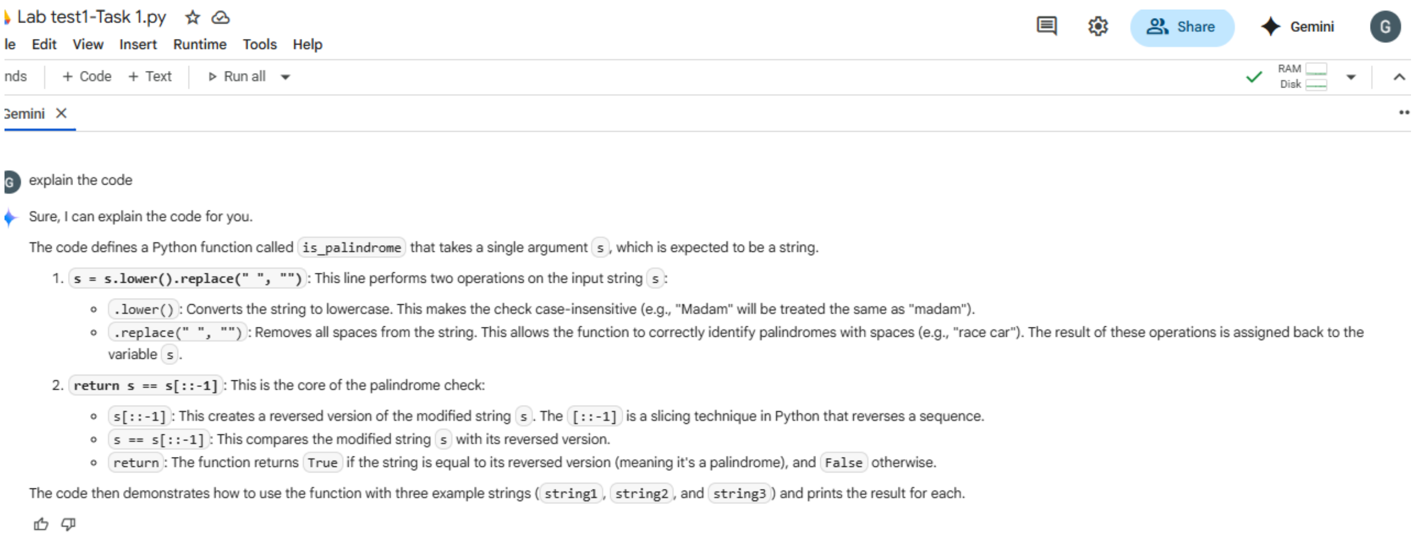
Code with output:



```python
def is_palindrome(s):
  """Checks if a string is a palindrome (reads the same forwards and backwards).

  Args:
    s: The input string.

  Returns:
    True if the string is a palindrome, False otherwise.
  """
  s = s.lower().replace(" ", "") # Convert to lowercase and remove spaces
  return s == s[::-1]

# Example usage:
string1 = "madam"
string2 = "racecar"
string3 = "hello"

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)}")
```

```
'madam' is a palindrome: True
'racecar' is a palindrome: True
'hello' is a palindrome: False
```

Explanation:


Lab test1-Task 1.py
le Edit View Insert Runtime Tools Help
Share
Gemini
G
nds + Code + Text ▷ Run all
RAM
Disk
Gemini ×

explain the code

Sure, I can explain the code for you.

The code defines a Python function called `is_palindrome` that takes a single argument `s`, which is expected to be a string.

1. `s = s.lower().replace(" ", "")`: This line performs two operations on the input string `s`:
    - `.lower()`: Converts the string to lowercase. This makes the check case-insensitive (e.g., "Madam" will be treated the same as "madam").
    - `.replace(" ", "")`: Removes all spaces from the string. This allows the function to correctly identify palindromes with spaces (e.g., "race car"). The result of these operations is assigned back to the variable `s`.
2. `return s == s[::-1]`: This is the core of the palindrome check:
    - `s[::-1]`: This creates a reversed version of the modified string `s`. The `[::-1]` is a slicing technique in Python that reverses a sequence.
    - `s == s[::-1]`: This compares the modified string `s` with its reversed version.
    - `return`: The function returns `True` if the string is equal to its reversed version (meaning it's a palindrome), and `False` otherwise.

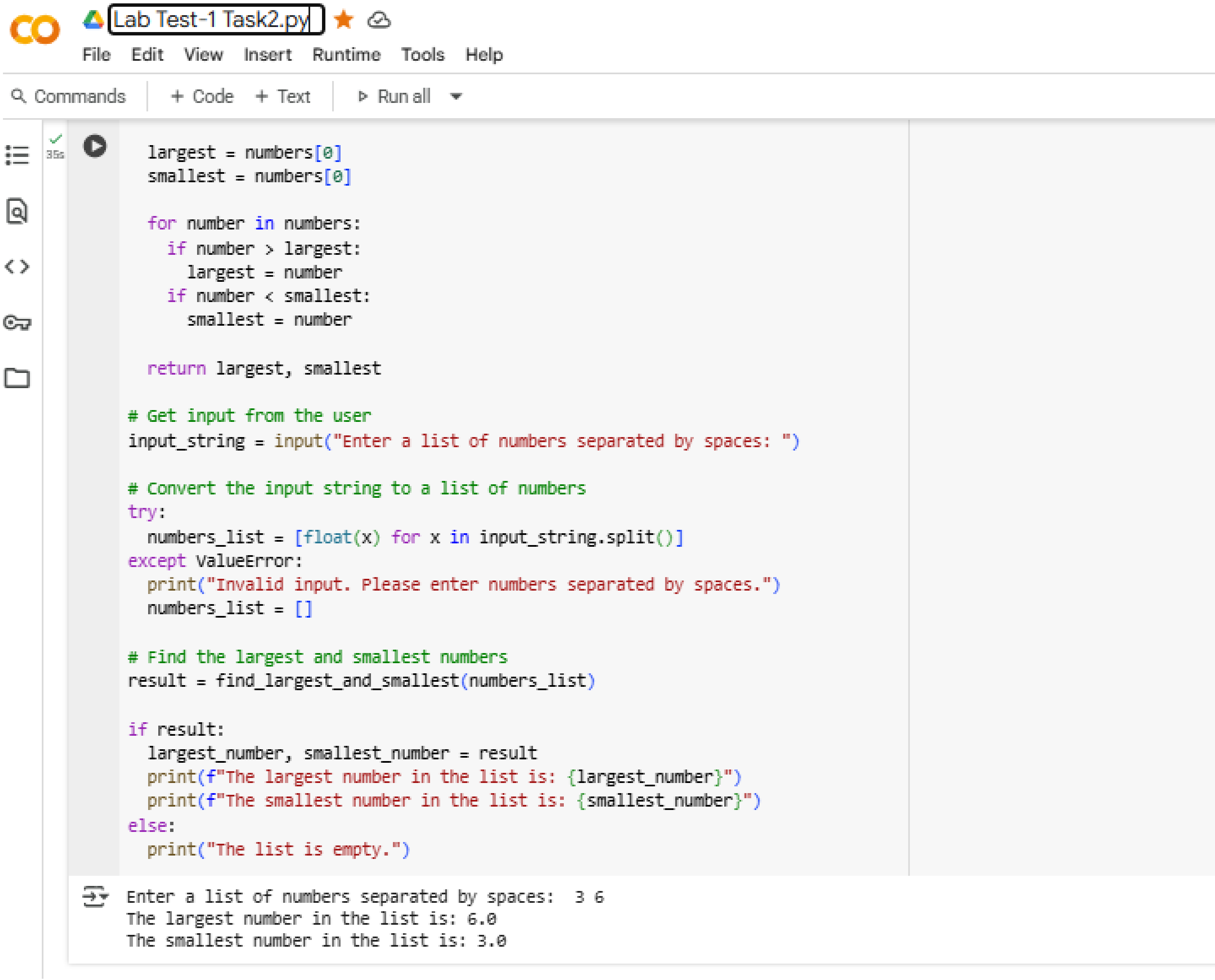
The code then demonstrates how to use the function with three example strings (`string1`, `string2`, and `string3`) and prints the result for each.

Question2:

Task2:

#give the prompt to gemini to generate a program that finds the largest and smallest number in the given list which provided by the user and ask gemini to explain the code.

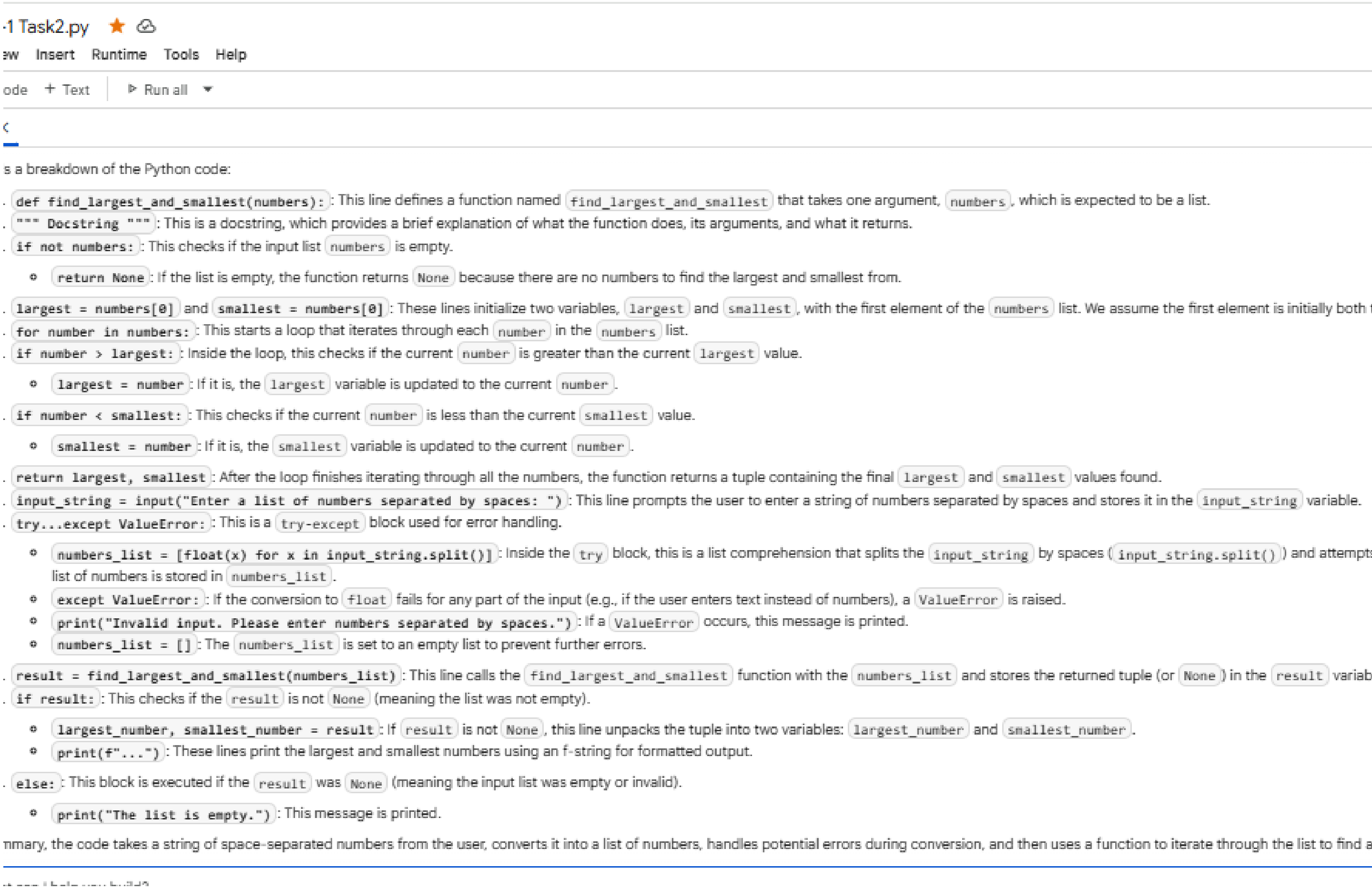
Code with output:



```
  largest = numbers[0]
  smallest = numbers[0]

  for number in numbers:
    if number > largest:
      largest = number
    if number < smallest:
      smallest = number

  return largest, smallest

# Get input from the user
input_string = input("Enter a list of numbers separated by spaces: ")

# Convert the input string to a list of numbers
try:
  numbers_list = [float(x) for x in input_string.split()]
except ValueError:
  print("Invalid input. Please enter numbers separated by spaces.")
  numbers_list = []

# Find the largest and smallest numbers
result = find_largest_and_smallest(numbers_list)

if result:
  largest_number, smallest_number = result
  print(f"The largest number in the list is: {largest_number}")
  print(f"The smallest number in the list is: {smallest_number}")
else:
  print("The list is empty.")
```

```
Enter a list of numbers separated by spaces:  3 6
The largest number in the list is: 6.0
The smallest number in the list is: 3.0
```

Explanation:



s a breakdown of the Python code:

- `def find_largest_and_smallest(numbers):`: This line defines a function named `find_largest_and_smallest` that takes one argument, `numbers`, which is expected to be a list.
- `""" Docstring """`: This is a docstring, which provides a brief explanation of what the function does, its arguments, and what it returns.
- `if not numbers:`: This checks if the input list `numbers` is empty.
  - `return None`: If the list is empty, the function returns `None` because there are no numbers to find the largest and smallest from.
- `largest = numbers[0]` and `smallest = numbers[0]`: These lines initialize two variables, `largest` and `smallest`, with the first element of the `numbers` list. We assume the first element is initially both
- `for number in numbers:`: This starts a loop that iterates through each `number` in the `numbers` list.
- `if number > largest:`: Inside the loop, this checks if the current `number` is greater than the current `largest` value.
  - `largest = number`: If it is, the `largest` variable is updated to the current `number`.
- `if number < smallest:`: This checks if the current `number` is less than the current `smallest` value.
  - `smallest = number`: If it is, the `smallest` variable is updated to the current `number`.
- `return largest, smallest`: After the loop finishes iterating through all the numbers, the function returns a tuple containing the final `largest` and `smallest` values found.
- `input_string = input("Enter a list of numbers separated by spaces: ")`: This line prompts the user to enter a string of numbers separated by spaces and stores it in the `input_string` variable.
- `try...except ValueError:`: This is a `try-except` block used for error handling.
  - `numbers_list = [float(x) for x in input_string.split()]`: Inside the `try` block, this is a list comprehension that splits the `input_string` by spaces (`input_string.split()`) and attempts
    list of numbers is stored in `numbers_list`.
  - `except ValueError:`: If the conversion to `float` fails for any part of the input (e.g., if the user enters text instead of numbers), a `ValueError` is raised.
  - `print("Invalid input. Please enter numbers separated by spaces.")`: If a `ValueError` occurs, this message is printed.
  - `numbers_list = []`: The `numbers_list` is set to an empty list to prevent further errors.
- `result = find_largest_and_smallest(numbers_list)`: This line calls the `find_largest_and_smallest` function with the `numbers_list` and stores the returned tuple (or `None`) in the `result` variab
- `if result:`: This checks if the `result` is not `None` (meaning the list was not empty).
  - `largest_number, smallest_number = result`: If `result` is not `None`, this line unpacks the tuple into two variables: `largest_number` and `smallest_number`.
  - `print(f"...")`: These lines print the largest and smallest numbers using an f-string for formatted output.
- `else:`: This block is executed if the `result` was `None` (meaning the input list was empty or invalid).
  - `print("The list is empty.")`: This message is printed.

nmary, the code takes a string of space-separated numbers from the user, converts it into a list of numbers, handles potential errors during conversion, and then uses a function to iterate through the list to find a