

AI ASSISTED CODING

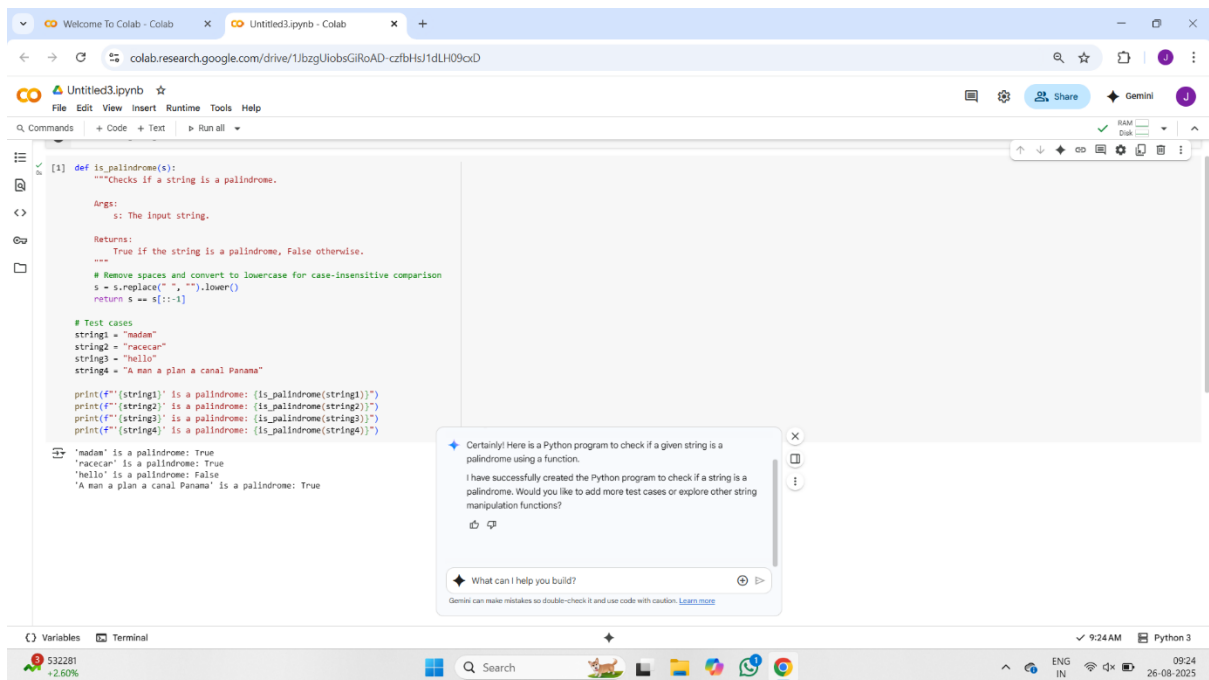
LAB EXAM -1

TASK-1:

TO FIND WHETHER THE GIVEN NUMBER IS PALINDROME OR NOT

CODE SCREENSHOTS

WITH OUTPUT:



```
[1] def is_palindrome(s):  
    """Checks if a string is a palindrome.  
  
    Args:  
        s: The input string.  
  
    Returns:  
        True if the string is a palindrome, False otherwise.  
    """  
    # Remove spaces and convert to lowercase for case-insensitive comparison  
    s = s.replace(" ", "").lower()  
    return s == s[::-1]  
  
# Test cases  
string1 = "nadan"  
string2 = "racecar"  
string3 = "hello"  
string4 = "A man a plan a canal Panama"  
  
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)}')  
  
'nadan' is a palindrome: True  
'racecar' is a palindrome: True  
'hello' is a palindrome: False  
'A man a plan a canal Panama' is a palindrome: True
```

Certainly! Here is a Python program to check if a given string is a palindrome using a function.

I have successfully created the Python program to check if a string is a palindrome. Would you like to add more test cases or explore other string manipulation functions?

What can I help you build?

Code explanation:

The screenshot shows a Google Colab notebook with a Python function `is_palindrome(s)` that checks if a string is a palindrome. The function removes spaces, converts to lowercase, and compares the string with its reverse. Test cases are provided for 'madam', 'racecar', 'hello', and 'A man a plan a canal Panama'.

```
[1] def is_palindrome(s):  
    """checks if a string is a palindrome.  
  
    Args:  
        s: The input string.  
  
    Returns:  
        True if the string is a palindrome, False otherwise.  
    """  
    # Remove spaces and convert to lowercase for case-insensitive comparison  
    s = s.replace(" ", "").lower()  
    return s == s[::-1]  
  
# Test cases  
string1 = "madam"  
string2 = "racecar"  
string3 = "hello"  
string4 = "A man a plan a canal Panama"  
  
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)}")  
  
'madam' is a palindrome: True  
'racecar' is a palindrome: True  
'hello' is a palindrome: False  
'A man a plan a canal Panama' is a palindrome: True
```

Gemini explains the code as follows:

- def is_palindrome(s):** This line defines a function named `is_palindrome` that takes one argument, `s`, which represents the input string.
- s = s.replace(" ", "").lower()** Inside the function, this line preprocesses the input string `s`.
 - `s.replace(" ", "")` removes any spaces from the string.
 - `.lower()` converts the entire string to lowercase. This step ensures that the palindrome check is case-insensitive and ignores spaces, allowing phrases like "A man a plan a canal Panama" to be correctly identified as palindromes.
- return s == s[::-1]** This is the core of the palindrome check.
 - `s[::-1]` creates a reversed version of the preprocessed string `s`. The `[::-1]` slicing syntax is a common Python idiom for reversing sequences.
 - `s == s[::-1]` compares the preprocessed string with its reversed version.
 - The function returns `True` if the string is equal to its reverse (meaning it's a palindrome), and `False` otherwise.
- Test Cases:** The lines after the function definition demonstrate how to use the `is_palindrome` function with different strings (`string1`, `string2`, `string3`, `string4`).
- print(...)** These lines print the original string and the result of the `is_palindrome` function call for each test case, clearly indicating whether the string is a palindrome or not.

In essence, the code checks if a string is the same forwards and backward, ignoring spaces and capitalization.

2. WRITE A PYTHON PROGRAM TO FIND THE LARGEST AND THE SMALLEST NUMBERS BY USER PROVIDED LIST AND ASK GEMINI TO GIVE STEP BY STEP EXPLANATION OF CODE

CODE:

The screenshot shows a Google Colab notebook with a Python function `find_min_max(numbers)` that finds the smallest and largest numbers in a list. The function handles empty lists and user input.

```
def find_min_max(numbers):  
    """Finds the smallest and largest numbers in a list.  
  
    Args:  
        numbers: A list of numbers.  
  
    Returns:  
        A tuple containing the smallest and largest numbers (min_num, max_num).  
        Returns (None, None) if the list is empty.  
    """  
    if not numbers:  
        return None, None  
    min_num = numbers[0]  
    max_num = numbers[0]  
    for num in numbers:  
        if num < min_num:  
            min_num = num  
        if num > max_num:  
            max_num = num  
    return min_num, max_num  
  
# Get input from the user  
user_input = 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 user_input.split()]  
except ValueError:  
    print("Invalid input. Please enter numbers separated by spaces.")  
    numbers_list = []  
  
# Find the smallest and largest numbers  
smallest, largest = find_min_max(numbers_list)  
  
# Print the results  
if smallest is not None and largest is not None:  
    print(f"The smallest number is: {smallest}")
```

Gemini explains the code as follows:

- def find_min_max(numbers):** This line defines a function named `find_min_max` that takes one argument, `numbers`, which represents a list of numbers.
- if not numbers:** This line checks if the list is empty. If it is, the function returns `None, None`.
- min_num = numbers[0]** and **max_num = numbers[0]** These lines initialize the minimum and maximum values to the first element of the list.
- for num in numbers:** This line starts a loop that iterates over each number in the list.
- if num < min_num:** and **if num > max_num:** These lines check if the current number is smaller than the current minimum or larger than the current maximum. If so, they update `min_num` or `max_num` accordingly.
- return min_num, max_num** This line returns the smallest and largest numbers found in the list as a tuple.
- # Get input from the user** and **user_input = input(...)** These lines prompt the user to enter a list of numbers separated by spaces.
- # Convert the input string to a list of numbers** and the `try/except` block: These lines attempt to convert the user input into a list of floating-point numbers. If the input is invalid, a message is printed and an empty list is returned.
- # Find the smallest and largest numbers** and **smallest, largest = find_min_max(numbers_list)** These lines call the `find_min_max` function with the user-provided list.
- # Print the results** and the `if` statement: These lines check if the function returned valid values and print the smallest and largest numbers.

colab.research.google.com/drive/1jbzgUioBsGiRoAD-czfbHs1dLH09cx0D#scrollTo=9ba51cb9

Untitled3.ipynb

```
max_num = numbers[0]

for num in numbers:
    if num < min_num:
        min_num = num
    if num > max_num:
        max_num = num

return min_num, max_num

# Get input from the user
user_input = 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 user_input.split()]
except ValueError:
    print("Invalid input. Please enter numbers separated by spaces.")
    numbers_list = []

# Find the smallest and largest numbers
smallest, largest = find_min_max(numbers_list)

# Print the results
if smallest is not None and largest is not None:
    print(f"The smallest number is: {smallest}")
    print(f"The largest number is: {largest}")
else:
    print("No numbers were entered.")
```

Enter a list of numbers separated by spaces: 4 12 6 7 23 14 15 10
The smallest number is: 4.0
The largest number is: 23.0

Gemini

write a python program that finds the largest and smallest numbers in a user-provided list

Sure! Here is a Python program that takes a list of numbers from the user and finds the largest and smallest numbers in the list.

I have successfully created the Python program to find the largest and smallest numbers in a user-provided list. Would you like to explore other list manipulation tasks, such as sorting the list or finding the average? Or perhaps you'd like to refine the current code?

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Markup and share

CODE EXPLANATION:

colab.research.google.com/drive/1jbzgUioBsGiRoAD-czfbHs1dLH09cx0D#scrollTo=9ba51cb9

Untitled3.ipynb

```
return min_num, max_num

# Get input from the user
user_input = 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 user_input.split()]
except ValueError:
    print("Invalid input. Please enter numbers separated by spaces.")
    numbers_list = []

# Find the smallest and largest numbers
smallest, largest = find_min_max(numbers_list)

# Print the results
if smallest is not None and largest is not None:
    print(f"The smallest number is: {smallest}")
    print(f"The largest number is: {largest}")
else:
    print("No numbers were entered.")
```

Enter a list of numbers separated by spaces: 4 12 6 7 23 14 15 10
The smallest number is: 4.0
The largest number is: 23.0

Gemini

explain how the above code works

Certainly! Let me explain how the code above works to find the smallest and largest numbers in a list provided by the user.

Here's a breakdown:

- def find_min_max(numbers):** This line defines a function named `find_min_max` that takes one argument, `numbers`, which is expected to be a list of numbers.
- if not numbers:** This checks if the input list `numbers` is empty. If it is, it means there are no numbers to compare, so the function returns `(None, None)`.
- min_num = numbers[0]** and **max_num = numbers[0]**: If the list is not empty, the code initializes two variables, `min_num` and `max_num`, with the first element of the `numbers` list. These variables will hold the smallest and largest numbers found so far.
- for num in numbers:** This starts a loop that iterates through each `num` in the `numbers` list.
- if num < min_num:** Inside the loop, this condition checks if the current number `num` is smaller than the current `min_num`. If it is, `min_num` is updated to the value of `num`.
- if num > max_num:** This condition checks if the current number `num` is larger than the current `max_num`. If it is, `max_num` is updated to the value of `num`.
- return min_num, max_num:** After the loop finishes iterating through all the numbers in the list, the function returns a tuple containing the final `min_num` and `max_num`.
- user_input = input(...)**: This line prompts the user to enter a list of numbers separated by spaces and stores the input in the `user_input` variable.
- try...except ValueError:** This block attempts to convert the user's input string into a list of floating-point numbers.
 - `user_input.split()` splits the input string into a list of substrings based on spaces.
 - `[float(x) for x in ...]` is a list comprehension that attempts to convert each substring `x` into a floating-point number.

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)



RAM 
Disk 

Gemini X




the list, the function returns a tuple containing the final `min_num` and `max_num`.

8. `user_input = input(...)`: This line prompts the user to enter a list of numbers separated by spaces and stores the input in the `user_input` variable.
9. `try...except ValueError`: This block attempts to convert the user's input string into a list of floating-point numbers.
 - `user_input.split()` splits the input string into a list of substrings based on spaces.
 - `[float(x) for x in ...]` is a list comprehension that attempts to convert each substring `x` into a floating-point number.
 - If the conversion fails for any substring (e.g., the user enters text that is not a number), a `ValueError` occurs, and the code inside the `except` block is executed.
 - The `except` block prints an error message and sets `numbers_list` to an empty list.
10. `smallest, largest = find_min_max(numbers_list)`: This line calls the `find_min_max` function with the list of numbers obtained from the user input and assigns the returned smallest and largest values to the `smallest` and `largest` variables.
11. `if smallest is not None and largest is not None`: This checks if the `find_min_max` function returned actual numbers (i.e., the input list was not empty).
12. `print(...)`: If `smallest` and `largest` are not `None`, these lines print the smallest and largest numbers found. Otherwise, it prints a message indicating that no numbers were entered.

In summary, the code gets a string of space-separated numbers from the user, converts it into a list of numbers, and then iterates through the list to find and report the smallest and largest values.



 explain it in a simple way