LAB TEST 1

RAMIDI . SAI CHARAN

2403A52124

WRITE A PYTHON PROGRAM TO CHECK WHETHER A STRING IS PALINDROME OR NOT

CODE:

```python
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 non-alphanumeric characters and convert to lowercase
  s = ''.join(filter(str.isalnum, s)).lower()
  return s == s[::-1]

# Example usage with function
print(is_palindrome("Racecar"))
print(is_palindrome("hello"))
print(is_palindrome("A man, a plan, a canal: Panama"))
print(is_palindrome("No 't is open"))
print(is_palindrome("Was it a car or a cat I saw?"))
```

```
True
False
True
False
True
```

Explanation :

Certainly! I can explain the code in cell `3050d550` which checks if a string is a palindrome.

Here's how the `is_palindrome` function works:

1. `def is_palindrome(s):`: This line defines a function named `is_palindrome` that takes one argument, `s`, which is the string you want to check.
2. `s = ''.join(filter(str.isalnum, s)).lower()`: This is the core of the preprocessing.
    - `filter(str.isalnum, s)`: This part goes through each character in the input string `s` and keeps only those that are alphanumeric (letters and numbers). It filters out spaces, punctuation, and other symbols.
    - `''.join(...)`: This joins the filtered characters back into a single string.
    - `.lower()`: This converts the resulting string to lowercase. This makes the check case-insensitive, so "Racecar" and "racecar" are both considered palindromes.
3. `return s == s[::-1]`: This line performs the actual palindrome check.
    - `s[::-1]`: This creates a reversed version of the processed string `s`. The slicing `[::-1]` is a Python idiom for reversing a sequence.
    - `s == s[::-1]`: This compares the processed string `s` with its reversed version. If they are the same, the function returns `True` (it's a palindrome); otherwise, it returns `False`.

In summary, the function cleans the input string by removing non-alphanumeric characters and converting it to lowercase, and then checks if the cleaned string is the same forwards and backward.

LAB TEST 1

GENERATE A PROGRAM THET FIND THE LARGEST AND SMALLEST NUMBERS IN A USER PROVIDED LIST

CODE:

```
# 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:
  user_numbers = [float(num) for num in input_string.split()]

  if user_numbers:
    # Find the smallest and largest numbers using built-in functions
    smallest_num = min(user_numbers)
    largest_num = max(user_numbers)

    print(f"The smallest number is: {smallest_num}")
    print(f"The largest number is: {largest_num}")
  else:
    print("The list is empty.")

except ValueError:
  print("Invalid input. Please enter numbers separated by spaces.")
```

```
Enter a list of numbers separated by spaces: 85 98  2 3 250
The smallest number is: 2.0
The largest number is: 250.0
```

LAB TEST 1

Explanation :

1. `input_string = input("Enter a list of numbers separated by spaces: ")`: This line prompts the user to enter a list of numbers, separated by spaces, and stores the input as a string in the `input_string` variable.
2. `try...except ValueError:`: This block is used for error handling. It attempts to execute the code within the `try` block. If a `ValueError` occurs (which would happen if the user enters something that cannot be converted to a number), the code in the `except` block is executed.
3. `user_numbers = [float(num) for num in input_string.split()]`: This is a list comprehension that converts the input string into a list of numbers.
    - `input_string.split()`: This splits the `input_string` into a list of substrings using spaces as the delimiter.
    - `for num in ...`: This iterates through each substring (`num`) in the split list.
    - `float(num)`: This attempts to convert each substring (`num`) into a floating-point number.
    - `[...]`: This collects the converted numbers into a new list called `user_numbers`.
4. `if user_numbers:`: This checks if the `user_numbers` list is not empty. If the user just presses Enter without typing anything, the list would be empty.
5. `smallest_num = min(user_numbers)`: If the list is not empty, this line uses the built-in Python function `min()` to find the smallest number in the `user_numbers` list and stores it in the `smallest_num` variable.
6. `largest_num = max(user_numbers)`: Similarly, this line uses the built-in Python function `max()` to find the largest number in the `user_numbers` list and stores it in the `largest_num` variable.
7. `print(f"The smallest number is: {smallest_num}")` and `print(f"The largest number is: {largest_num}")`: These lines print the smallest and largest numbers found using f-strings for formatted output.
8. `else: print("The list is empty.")`: If the `user_numbers` list was empty (because the user didn't enter any numbers), this message is printed.
9. `except ValueError: print("Invalid input. Please enter numbers separated by spaces.")`: If a `ValueError` occurred during the conversion of input to numbers, this message is printed, informing the user about the invalid input.