String_Methods

1. upper()

Description: Converts all lowercase letters in a string to uppercase. Usage:

```
text = "hello"
result = text.upper()
print(result)  # Output: "HELLO"
```

```
HELLO
```

Step by Step:

The `upper()` method takes the string text and iterates over each character.

If a character is lowercase, it converts it to uppercase.

It assembles the converted characters into a new string and returns it.

2. lower()

Description: Converts all uppercase letters in a string to lowercase.

```
text = "HELLO"
result = text.lower()
print(result)  # Output: "hello"
```

```
hello
```

```
"Hello".upper()
```

```
'HELLO'
```

3. strip()

Description: Removes leading and trailing whitespace from the string.

```
text = "  hello  "
result = text.strip()
print(result)  # Output: "hello"
```

```
hello
```

Step by Step:

The strip() method scans the string from both ends for any whitespace characters (spaces, tabs, newlines).

It stops at the first non-whitespace character from both ends.

Returns the substring that excludes the leading and trailing whitespace.

4. replace(old, new[, count])

Description: Replaces occurrences of a substring within the string.

```
text = "hello world world"
result = text.replace("world", "everyone")
print(result)  # Output: "hello everyone"
```

```
hello everyone everyone
```

Step by Step:

replace() method searches text for the substring old.

Each occurrence of old is replaced by new.

If count is specified, only the first count occurrences are replaced. The method constructs a new string with these replacements and returns it.

List Methods

1. append(element)

Description: Adds an element to the end of the list.

```
my_list = [1, 2, 3]
my_list.append([4, 5,6,7])
print(my_list)  # Output: [1, 2, 3, 4]
```

```
[1, 2, 3, [4, 5, 6, 7]]
```

2. extend(iterable)

Description: Extends the list by appending all elements from an iterable (like another list).

```
my_list = [1, 2, 3]
another_list = [4, 5,6,7]
my_list.extend(another_list)
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5, 6, 7]
```

3. remove(element)

Description: Removes the first occurrence of a specified value from the list.

```
my_list = [1, 2, 3, 2, 4]
my_list.remove(5)
print(my_list)  # Output: [1, 3, 2, 4]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-0df4b388e106> in <cell line: 2>()
      1 my_list = [1, 2, 3, 2, 4]
----> 2 my_list.remove(5)
      3 print(my_list)  # Output: [1, 3, 2, 4]

ValueError: list.remove(x): x not in list
```

Next steps: Explain error

The remove() method searches for the first occurrence of element in the list.

Once found, it removes that element.

If the element is not found, a ValueError is raised.

The list is updated in place without returning any value.

4. pop([index])

Description: Removes the element at the specified position in the list and returns it. If no index is specified, pop() removes and returns the last item in the list.

```
my_list = [1, 2, 3, 4]
popped_element = my_list.pop()  # No index specified
print(popped_element)  # Output: 4
print(my_list)  # Output: [1, 2, 3]
```

```
4
[1, 2, 3]
```

```
my_list.pop(0)
```

```
1
```

```
my_list
```

```
[2, 3]
```

If no index is specified, pop() removes the last element.

If an index is specified, it removes the element at that position.

The method returns the removed element.

The list size is reduced by one, and the method modifies the list in place.

Tuples

Tuples are immutable sequences, which means their contents cannot be changed after creation

Concatenation

Description: Tuples can be concatenated using the + operator to create a new tuple

```
tuple1 = (1, 2, 3)
tuple2 = ('a', 'b', 'c')
new_tuple = tuple1 + tuple2
print(new_tuple)  # Output: (1, 2, 3, 'a', 'b', 'c')
```

```
(1, 2, 3, 'a', 'b', 'c')
```

Step by Step:

The + operator merges two tuples into a new tuple.

The original tuples remain unchanged.

The new tuple contains elements from both original tuples.

Sets

Sets are unordered collections of unique elements.

1. Adding Elements (add method)

Description: Adds an element to the set.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

```
{1, 2, 3, 4}
```

The add() method adds the specified element to the set.

If the element already exists, the set doesn't change.

Sets are unordered, so the order of elements isn't guaranteed.

2. Removing Elements (remove method)

Description: Removes a specific element from the set.

```
my_set = {1, 2, 3}
my_set.remove(2)
print(my_set)  # Output: {1, 3}
```

```
{1, 3}
```

The remove() method deletes the specified element.

If the element is not found, it raises a `KeyError.`

The set is updated in place.

Dictionaries

Dictionaries are collections of key-value pairs.

Adding/Updating Entries

Description: Keys are unique, and values can be updated using indexing.

```
my_dict = {'apple': 1, 'banana': 2}
my_dict['cherry'] = 3  # Adding a new key
my_dict['apple'] = 4   # Updating an existing key
print(my_dict)  # Output: {'apple': 4, 'banana': 2, 'cherry': 3}
```

```
{'apple': 4, 'banana': 2, 'cherry': 3}
```

```
my_dict.get('apple')
```

```
4
```

```
name = "Sriharsha"
len(name)
name[9-1]
```

```
'a'
```

```
ls = [1,2,3,4,5]
ls
```