String

Important

- `''.join(lst)`
- `string.count(char)`
- `string.replace(char1, char2)`
- `string.find(sub)`
- `string.rfind(sub)`
- `string.index(sub)`
- `string.rindex(sub)`
- `string.split(sep)`
- `string.rsplit(sep)`
- `string.partition(sep)`
- `string.rpartition(sep)`

Methods

- `string.strip()`
- `string.lstrip()`
- `string.rstrip()`
- `string.lower()`
- `string.upper()`
- `string.capitalize()`
- `string.title()`
- `string.swapcase()`
- `string.startswith(prefix)`
- `string.endswith(suffix)`

Reverse a string

- **Problem**: Write a function to reverse a given string.

```
def reverse_string(string: str):
    return string[::-1]

string = "Hello, World"
print(reverse_string(string))
```

```
dlroW ,olleH
```

Check Palindrome

- **Problem:** Write a function to check if a given string is a palindrome (reads the same forward and backward, ignoring case and non-alphanumeric characters).

```
def is_palindrom(string: str):
    # Remove non-alphanumeric character
    cleaned = ''.join(char.lower() for char in string if char.isalnum())
    print(cleaned, cleaned[::-1])
    return cleaned == cleaned[::-1]

string = is_palindrom("A man, a plan, a canal: Panama")
print(string)
```

```
amanaplanacanalpanama amanaplanacanalpanama
True
```

Find the First Non-Repeating Character

- **Problem**: Write a function to find the first non-repeating character in a string and return its index. If none exists, return -1.

```
def first_non_repeating_char(string):
    char_count = dict()
    for char in string:
        char_count[char] = char_count.get(char, 0) + 1
    print(char_count)

    for i, char in enumerate(string):
        if char_count[char] == 1:
            return i, char
    return -1



print(first_non_repeating_char("leetcode"))
print(first_non_repeating_char("loveleetcode"))
```

```
{'l': 1, 'e': 3, 't': 1, 'c': 1, 'o': 1, 'd': 1}
(0, 'l')
{'l': 2, 'o': 2, 'v': 1, 'e': 4, 't': 1, 'c': 1, 'd': 1}
(2, 'v')
```

```
def first_non_repeating_char(string):
    char_count = dict()
    for char in string:
        char_count[char] = string.count(char)
    print(char_count)

    for i, char in enumerate(string):
        if char_count[char] == 1:
            return i, char
    return -1



print(first_non_repeating_char("leetcode"))
print(first_non_repeating_char("loveleetcode"))
```

```
{'l': 1, 'e': 3, 't': 1, 'c': 1, 'o': 1, 'd': 1}
(0, 'l')
{'l': 2, 'o': 2, 'v': 1, 'e': 4, 't': 1, 'c': 1, 'd': 1}
(2, 'v')
```

Check if Two Strings are Anagrams

- **Problem:** Write a function to check if two strings are anagrams (contain the same characters with the same frequency, ignoring spaces and case).

```
def sort(string):
    string = list(string.lower())
    for i in range(len(string)):
        for j in range(0, len(string)-i-1):
            if string[j] > string[j+1]:
                string[j], string[j+1] = string[j+1], string[j]
    return ''.join(string)

def check_anagrams(string1, string2):
    string1 = string1.replace(" ", "").lower()
    string2 = string2.replace(" ", "").lower()

    if len(string1) != len(string2):
        return False

    return sort(string1) == sort(string2)


print(check_anagrams("Listen", "Silent"))
```

```
True
```

Find All Substrings of a String

- **Problem:** Write a function to find all possible substrings of a given string.

```
def find_all_strings(string):
    result = []
    for i in range(len(string)):
        for j in range(i+1, len(string)+1):
            result.append(string[i:j])
    return result

print(find_all_strings("abc"))
print(find_all_strings("Zebronic"))
```

```
['a', 'ab', 'abc', 'b', 'bc', 'c']
['Z', 'Ze', 'Zeb', 'Zebr', 'Zebro', 'Zebron', 'Zebroni', 'Zebronic', 'e', 'eb', 'ebr', 'ebro', 'ebron', 'ebroni', 'ebronic', 'b', 'b
```

```
def get_substring_list(string:str):
    sub_string_list = list()
    for i in range(len(string)):
        for j in range(i, len(string)):
            sub_string_list.append(string[i:j+1])
    return sub_string_list

string = "abcd"
substring_list = get_substring_list(string)
print(substring_list)
```

```
['a', 'ab', 'abc', 'abcd', 'b', 'bc', 'bcd', 'c', 'cd', 'd']
```

Sort

```
def insersion_sort(string:str):
    string = list(string)
    for i in range(len(string)):
        for j in range(len(string)):
            if string[i] < string[j]:
                string[i], string[j] = string[j], string[i]
    string = ''.join(string)
    return string

string = "24589652"
print(insersion_sort(string))
```

```
22455689
```

```
def bubble_sort(string: str):
    string = list(string)
    for i in range(len(string)):
        for j in range(len(string)-i-1):
            if string[j] > string[j+1]:
                string[j], string[j+1] = string[j+1], string[j]
    string = ''.join(string)
    return string

string = "2569865423"
print(bubble_sort(string))
```

```
2234556689
```

Longest Common Prefix

- **Problem:** Write a function to find the longest common prefix among an array of strings. If there is no common prefix, return an empty string.

```
def longest_common_prefix(string_list):
    if not string_list:
        return ""

    prefix = []
    idx = 0

    while True:
        try:
            char = string_list[0][idx]
            for string in string_list:
                if string[idx] != char:
```

```
                return "".join(prefix)
        except IndexError:
            return "".join(prefix)

        prefix.append(char)
        idx += 1


print(longest_common_prefix(["flower", "flow", "flight"]))
print(longest_common_prefix(["dog", "racecar", "car"]))
```

```
fl
```

Count Vowels in a String

- **Problem:** Write a function to count the number of vowels (a, e, i, o, u) in a given string.

```
def count_vowels(string):
    vowels = list("aeiouAEIOU")
    return sum(1 for char in string if char in vowels)

print(count_vowels("Hello, World!"))
print(count_vowels("Python"))
print(count_vowels("Allaumora"))
```

```
3
1
5
```

Remove Duplicates from a String

- **Problem:** Write a function to remove duplicate characters from a string while maintaining the original order.

```
def filter_string(string):
    return "".join(list(set(string)))

def remove_duplicates(string):
    result = []
    for char in string:
        if char not in result:
            result.append(char)
    return ''.join(result)

print(filter_string("hello"))
print(filter_string("python"))
print()

print(remove_duplicates("hello"))
print(remove_duplicates("python"))
```

```
lheo
ptnyoh

helo
python
```

List

Remove Duplicates from a List

- **Problem:** Write a function to remove duplicate elements from a list while preserving the original order.

```
from random import randint

def remove_duplicates(lst):
    return list(set(lst))

lst = [randint(0, 9) for _ in range(10)]
```

```
print(lst)
print(remove_duplicates(lst))
```

```
[0, 3, 7, 9, 1, 2, 9, 2, 0, 9]
[0, 1, 2, 3, 7, 9]
```

Find the Second Largest Element in a List

- **Problem:** Write a function to find the second largest element in a list of integers.

```
from random import randint

def get_unique(lst):
    temp_lst = []
    for ele in lst:
        if ele not in temp_lst:
            temp_lst.append(ele)
    return temp_lst

def get_sort(lst):
    for i in range(len(lst)):
        for j in range(len(lst)-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst

def nth_largest(lst, n):
    lst = get_unique(lst)
    lst = get_sort(lst)
    n = len(lst) - n
    return lst[n]

lst = [randint(11, 99) for _ in range(20)]
print(lst)

print(nth_largest(lst, 5))
```

```
[38, 83, 60, 90, 43, 52, 26, 76, 37, 61, 85, 36, 66, 75, 76, 78, 16, 67, 61, 68]
76
```

Rotate a List by K Positions

- **Problem:** Write a function to rotate a list to the right by k positions.

```
from random import randint

def n_rotation(lst:list, n:int):
    n = n % len(lst)
    n = len(lst)-n
    return lst[n:] + lst[:n]

lst = [x for x in range(1, 7)]
print(lst); print()
print(n_rotation(lst, 1))
print(n_rotation(lst, 2))
print(n_rotation(lst, 3))
print(n_rotation(lst, 5))
```

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

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

Tuple

Start coding or generate with AI.

Find the First Repeating Element in a Tuple

- **Problem:** Write a function to find the first element that repeats in a tuple.

```
from random import randint

def first_repeating(tpl:tuple):
    lst = list(tpl)
    for idx in range(len(tpl)):
        lst.remove(tpl[idx])
        if tpl[idx] in lst:
            return tpl[idx]
    return None


tpl = tuple([randint(1, 9) for _ in range(10)])
print(tpl); print()
print(first_repeating(tpl))

tpl = tuple([x for x in range(10)])
print(first_repeating(tpl))
```

```
(5, 4, 5, 8, 2, 3, 8, 8, 5, 5)

5
None
```

Check if Two tuples have the same elements

- **Problem:** Write a function to check if two tuples contain the same elements, regardless of order.

```
from random import randint

def get_sort(lst:list):
    for i in range(len(lst)):
        for j in range(len(lst)-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst

def same_elements(tpl1, tpl2):
    tpl1 = tuple(get_sort(list(tpl1)))
    tpl2 = tuple(get_sort(list(tpl2)))
    return True if tpl1 == tpl2 else False

print(same_elements((1, 2, 3), (3, 1, 2)))  # Output: True
print(same_elements((1, 2, 3), (1, 2, 4)))  # Output: False

print(get_sort([randint(1, 9) for _ in range(10)]))
```

```
True
False
[1, 2, 4, 6, 7, 7, 7, 8, 8, 9]
```

Dictionary

Merge Two Dictionaries

- **Problem:** Write a function to merge two dictionaries. If a key exists in both, use the value from the second dictionary.

```
def merge_dicts(dict1, dict2):
    # You can use either of the lines below:
    # return {**dict1, **dict2}
    return dict1 | dict2

def split_dicts(dct: dict, k: int):
    items = list(dct.items())
    d1 = dict(items[:k])
    d2 = dict(items[k:])
    return d1, d2

# Create a dictionary with letters a-z and numbers 1-26
lst1 = [chr(x) for x in range(ord('a'), ord('z') + 1)]
lst2 = list(range(1, 27))
dct = {k: v for k, v in zip(lst1, lst2)}

print("Original Dictionary:\n", dct)
print()
```

```
# Now split the dictionary
dict1, dict2 = split_dicts(dct, 10)
print("Dict 1:\n", dict1)
print("Dict 2:\n", dict2)

# Merge them back
merged = merge_dicts(dict1, dict2)
print("\nMerged Dictionary:\n", merged)
```

```
Original Dictionary:
 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p'

Dict 1:
 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10}
Dict 2:
 {'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y'

Merged Dictionary:
 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p'
```

Find the Most Frequest Element in a List using Dictionary

- **Problem:** Write a function to find the element with the highest frequency in a list. If multiple elements have the same frequency, return any.

```
from random import randint

def get_frequency(lst:list):
    dct = dict()
    for ele in lst:
        dct[ele] = lst.count(ele)
    return dct

lst = [randint(1, 9) for _ in range(200)]
dct = get_frequency(lst)
print(lst); print()
print(dct)
```

```
[5, 5, 9, 8, 2, 2, 9, 5, 7, 5, 2, 8, 3, 2, 9, 1, 7, 4, 8, 6, 8, 1, 4, 8, 3, 9, 2, 8, 8, 4, 8, 6, 3, 3, 2, 6, 1, 1, 4, 4, 5, 6, 3, 3,

{5: 24, 9: 27, 8: 29, 2: 26, 7: 17, 3: 20, 1: 17, 4: 20, 6: 20}
```

Invert a dictionary

- **Problem:** Write a function to invert a dictionary, swapping keys and values. Assume values are unique and hashable.

```
def invert_dict(dct: dict):
    return {v: k for k, v in dct.items()}

# Create a dictionary with letters a-z and numbers 1-26
lst1 = [chr(x) for x in range(ord('a'), ord('z') + 1)]
lst2 = list(range(1, 27))
dct = {k: v for k, v in zip(lst1, lst2)}
print(dct); print()

inverted_dct = invert_dict(dct)
print(inverted_dct); print()
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p':

{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o', 16:
```

Mixed Questions (List, Tuple, Dictionary)

Group Anagrams from a List of Strings

- **Problem:** Given a list of strings, group anagrams together (words that have the same characters with the same frequency).

```
def group_anagrams(strings_list):
    angrams = dict()
    for string in strings_list:
        sorted_string = ''.join(sorted(string))
        if sorted_string not in angrams:
            angrams[sorted_string] = []
        angrams[sorted_string].append(string)
    return list(angrams.values())


ang_lst = group_anagrams(["eat","tea", "tan", "ate", "nat", "bat"])
print(ang_lst)
```

```
[['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]
```

Convert List of Tuples to Dictionary

- **Problem:** Write a function to convert a list of tuples (key-value pairs) into a dictionary.

```
tpl_lst = [(chr(x), x) for x in range(ord('a'), ord('z')+1)]
print(tpl_lst); print()

dct = dict(tpl_lst)
print(dct); print()

items = list(dct.items())
print(items); print()
```

```
[('a', 97), ('b', 98), ('c', 99), ('d', 100), ('e', 101), ('f', 102), ('g', 103), ('h', 104), ('i', 105), ('j', 106), ('k', 107), (

{'a': 97, 'b': 98, 'c': 99, 'd': 100, 'e': 101, 'f': 102, 'g': 103, 'h': 104, 'i': 105, 'j': 106, 'k': 107, 'l': 108, 'm': 109, 'n'

[('a', 97), ('b', 98), ('c', 99), ('d', 100), ('e', 101), ('f', 102), ('g', 103), ('h', 104), ('i', 105), ('j', 106), ('k', 107), (
```

Sorting

Sort a List of Strings by Length

- **Problem:** Write a function to sort a list of strings by their length in ascending order. If lengths are equal, maintain lexicographical order.

```
def sort_by_length(str_lst):
    return sorted(str_lst, key=lambda x: (len(x), x))

lst = sort_by_length(["alpha", "beta", "gum", "a", "alumini", "pow", "an"])
lst
```

```
['a', 'an', 'gum', 'pow', 'beta', 'alpha', 'alumini']
```

```
def sort_by_length(str_lst):
    dct = dict()
    for string in str_lst:
        dct[string] = len(string)
    lst_dct = list(dct.items())
    for i in range(len(lst_dct)):
        for j in range(len(lst_dct)-i-1):
            if lst_dct[j][1] > lst_dct[j+1][1]:
                lst_dct[j], lst_dct[j+1] = lst_dct[j+1], lst_dct[j]
    dct = dict(lst_dct)
    return list(dct.keys())

lst = sort_by_length(["alpha", "beta", "gum", "a", "alumini", "pow", "an"])
lst
```

```
['a', 'an', 'gum', 'pow', 'beta', 'alpha', 'alumini']
```

Matrix

Transpose a Matrix

- **Problem:** Write a function to transpose a matrix (swap rows and columns).

```
def transpose_matrix(matrix):
    rows = len(matrix)
    cols = len(matrix[0]) if rows > 0 else 0
    transpose = []

    for c in range(cols):
        new_row = []
        for r in range(rows):
            new_row.append(matrix[r][c])
        transpose.append(new_row)

    return transpose

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
transpose_matrix(matrix)
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

Rotate a Matrix by 90 Degrees(Clockwise)

- **Problem:** Write a function to rotate a square matrix by 90 degrees clockwise in-place.

```
def rotate_matrix(matrix):
    rows = len(matrix)
    cols = len(matrix[0]) if rows>0 else 0

    rotate = []

    for c in range(cols):
        temp = []
        for r in range(rows, 0, -1):
            temp.append(matrix[r-1][c])
        rotate.append(temp)
    return rotate


matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(rotate_matrix(matrix))
```

```
[[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

Inverse Matrix

```
def get_matrix_minor(matrix, i, j):
    return [row[:j] + row[j+1:] for r, row in enumerate(matrix) if r != i]

def determinant(matrix):
    if len(matrix) == 1:
        return matrix[0][0]
    if len(matrix) == 2:  # 2x2 base case
        return matrix[0][0]*matrix[1][1] - matrix[0][1]*matrix[1][0]

    det = 0
    for c in range(len(matrix)):
        det += ((-1) ** c) * matrix[0][c] * determinant(get_matrix_minor(matrix, 0, c))
    return det

def transpose(matrix):
    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix))]

def inverse_matrix(matrix):
    size = len(matrix)
    det = determinant(matrix)
    if det == 0:
        return "Matrix is singular and cannot be inverted."

    # Create matrix of cofactors
    cofactors = []
    for r in range(size):
        cofactor_row = []
        for c in range(size):
            minor = get_matrix_minor(matrix, r, c)
            cofactor = ((-1) ** (r + c)) * determinant(minor)
            cofactor_row.append(cofactor)
        cofactors.append(cofactor_row)

    # Adjugate (transpose of cofactor matrix)
    adjugate = transpose(cofactors)

    # Multiply by 1/det
    inverse = [[adjugate[r][c] / det for c in range(size)] for r in range(size)]

    return inverse


matrix = [
    [2, 5, 7],
    [6, 3, 4],
    [5, -2, -3]
]

inv = inverse_matrix(matrix)
for row in inv:
    print(row)
```

```
[1.0, -1.0, 1.0]
[-38.0, 41.0, -34.0]
[27.0, -29.0, 24.0]
```

Dynamic Programming

1. Fibonacci Number

- **Problem:** Write a function to compute the nth Fibonacci number, where the sequence is defined as F(n) = F(n-1) + F(n-2), with F(0) = 0 and F(1) = 1.

```
def fibonacci(number: int):
    if number <= 1:
        return number

    lst = [0, 1]
    for _ in range(number-1):
        lst.append(lst[-1] + lst[-2])
    return lst

result = fibonacci(10)
print(result)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

```
def multiplicative_fibonacci(number: int):
    if number <= 1:
        return number
    lst = [1, 2]
    for _ in range(number-1):
        lst.append(lst[-1] * lst[-2])
    return lst

result = multiplicative_fibonacci(10)
print(result)
```

```
[1, 2, 2, 4, 8, 32, 256, 8192, 2097152, 17179869184, 36028797018963968]
```

```
def power_fibonacci(number: int):
    if number <= 1:
        return number
    lst = [1, 2]
    for _ in range(number-1):
        lst.append(lst[-1] ** lst[-2])
    return lst

result = power_fibonacci(4)
print(result)


def power_fibonacci_gen(n):
    a, b = 1, 2
    yield a
    if n > 1:
        yield b
    for _ in range(n - 2):
        a, b = b, b ** a
        yield b

result = list(power_fibonacci_gen(10))
print(result)
```

Climbing Stairs

- **Problem:** You are climbing a staircase with n steps. You can climb 1 or 2 steps at a time. Write a function to return the number of distinct ways to reach the top.

```
def climb_stairs(n):
    if n <= 1:
        return 1
    dp = [0] * (n + 1)
    dp[0] = 1
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]

# Example usage
print(climb_stairs(4))  # Output: 5 (Ways: [1,1,1,1], [1,1,2], [1,2,1], [2,1,1], [2,2])
```

```
5
```

Longest Common Subsequence(LCS)

- **Problem:** Given two strings, find the length of their longest common subsequence (a subsequence is a sequence of characters that can be derived by deleting some characters without changing the order).

```
def longest_common_subsequence(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i - 1] == text2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]

# Example usage
print(longest_common_subsequence("ABCDGH", "AEDFHR"))  # Output: 3 (Subsequence: "ADH")
```

```
3
```

Start coding or generate with AI.