1. write a code to reverse a string

In [1]:
```
str1 = 'my name is sahil'
str1[::-1]
```

Out[1]:
```
'lihas si eman ym'
```

In [11]:
```
"".join(reversed(str1))
```

Out[11]:
```
'lihas si eman ym'
```

2. write a code to count number of vowels in a string.

In [14]:
```
vowels = ['a','e','i','o','u']
count=0
for i in str1.lower():
    if i in vowels:
        count+=1
count
```

Out[14]:
```
5
```

In [18]:
```
def count_vowels(string):
    vowels='aeiou'
    count=0
    for i in string:
        if i in vowels:
            count+=1
    return count

count_vowels(str1)
```

Out[18]:
```
5
```

3. write a code to check if a given string is palindrome or not.

A palindrome is a word, phrase, number, or any sequence of characters that reads the same forward and backward, ignoring spaces, punctuation, and capitalization.

In [20]:
```
li = ['level', 'madam','sir','121','sahil']
for i in li:
    if i==i[::-1]:
        print(i)
```

```
level
madam
121
```

In [22]:
```
for i in li:
    if i==''.join(reversed(i)):
        print(i)
```

```
level
madam
121
```

4. write a code to check if two given strings are anagrams of eack other

An anagram is a word or phrase formed by rearranging the letters of another word or phrase, using all the original letters exactly once.

In [24]:
```
str1 = 'earth'
str2 = 'heart'
sorted_str1 = ''.join(sorted(str1.lower()))
sorted_str2 = ''.join(sorted(str2.lower()))
if sorted_str1 ==sorted_str2:
    print('those two strings are anagram to each other')
else:
    print('those two strings are not anagram to each other')
```

```
those two strings are anagram to each other
```

In [25]:
```
def are_anagram(string1,string2):
    sorted_string1 = ''.join(string1.lower().split())
    sorted_string2 = ''.join(string2.lower().split())

    if sorted_string1==sorted_string2:
        print('those two strings are anagram to each other')
    else:
        print('those two strings are not anagram to each other')

are_anagram(str1,str2)
```

```
those two strings are not anagram to each other
```

In [26]:
```
str3='madam'
str4='sir'
are_anagram(str3,str4)
```

```
those two strings are not anagram to each other
```

5. Write a code to find all occurences of a given substring within another string

In [27]:
```
str1 = '''hello everyone my name is sahil, my favourite color is blue,
        my favourite hobby is playing cricket, my favourite dish is samosa'''

str1.count('my')
```

Out[27]:
```
4
```

In [28]:
```
str1.count('is')
```

Out[28]: 5

In [29]:
```
str1.count('sahil')
```

Out[29]: 1

6. write a code to perform basic string compression using the count of repeated characters

In [35]:
```
def compress_string(s):
    if not s:
        return ""

    compressed = []
    count = 1
    prev_char = s[0]

    for char in s[1:]:
        if char == prev_char:
            count += 1
        else:
            compressed.append(prev_char + str(count))
            prev_char = char
            count = 1

    # Add the last set of characters
    compressed.append(prev_char + str(count))

    # Join list into a string
    compressed_string = ''.join(compressed)

    # Return the compressed string only if it's shorter than the original
    return compressed_string if len(compressed_string) < len(s) else s

# Example usage:
original_string = "aaabbcaa"
compressed_string = compress_string(original_string)
print(f"Original string: {original_string}")
print(f"Compressed string: {compressed_string}")
```

```
Original string: aaabbcaa
Compressed string: aaabbcaa
```

In [ ]:

7. write a code to determine if a string has all unique characters

In [38]:
```
str1 = 'abcdefg'
unique=''
for i in str1:
    if i not in unique:
        unique+=i
print(unique)
```

```
if unique==str1:
    print('string has all unique characters')
else:
    print('string does not have all unique characters')
```

```
abcdefg
string has all unique characters
```

In [40]:
```
str2 = 'abbccddeeffgj'
unique=''
for i in str1:
    if i not in unique:
        unique+=i
print(unique)

if unique==str2:
    print('string has all unique characters')
else:
    print('string does not have all unique characters')
```

```
abcdefg
string does not have all unique characters
```

8. write a code to convert a given string to uppercase or lowercase

In [43]:
```
str1.upper()
```

Out[43]:
```
'ABCDEFG'
```

9. write a code to count number of words in a string.

In [52]:
```
str1 = 'my contact no is 9852860744'
count=0
for i in str1:
    if i.isalpha():
        count+=1
count
```

Out[52]:
```
13
```

10. write a code to concatenate two string without using '+' operator

In [58]:
```
str1 = 'sahil'
str2 = 'shende'
"".join([str1,str2])
```

Out[58]:
```
'sahilshende'
```

In [59]:
```
"{}{}".format(str1,str2)
```

Out[59]:
```
'sahilshende'
```

11. write a code to remove all the occurences of a specific element from the list

In [67]:
```
# 1st approach
li = [1,2,4,7,8,7,5,3,2,9,10,4]
li = list(set(li))
li
```

Out[67]:
```
[1, 2, 3, 4, 5, 7, 8, 9, 10]
```

In [64]:
```
# 2nd approach
li = [1,2,4,7,8,7,5,3,2,9,10,4]
li2 = []
for i in li:
    if i not in li2:
        li2.append(i)
print(li2)
```

```
[1, 2, 4, 7, 8, 5, 3, 9, 10]
```

12. implement a code to find the second largest number in a given list of integer

In [80]:
```
li = [2,5,8,0,6,3,23,67,22]
li.sort(reverse=True)
print('the second largest number is',li[1])
```

```
the second largest number is 23
```

In [84]:
```
li = [2,5,8,0,6,3,23,67,22]
new = sorted(li,reverse=True)
new[1]
```

Out[84]:
```
23
```

13. create a code to count occurrences of each element in a list and return a dictionary with elements as key and their counts as values.

In [86]:
```
li = [1,2,4,7,8,7,5,3,2,9,10,4,2,6,9,5,2]
dict1={}
for i in li:
    if i not in dict1:
        dict1[i]=1
    else:
        dict1[i]+=1
dict1
```

Out[86]:
```
{1: 1, 2: 4, 4: 2, 7: 2, 8: 1, 5: 2, 3: 1, 9: 2, 10: 1, 6: 1}
```

In [87]:
```
from collections import Counter
```

In [88]:
```
Counter(li)
```

Out[88]:
```
Counter({2: 4, 4: 2, 7: 2, 5: 2, 9: 2, 1: 1, 8: 1, 3: 1, 10: 1, 6: 1})
```

14. write a code to reverse a list in-place without using any reverse built in function

In [96]:
```
def reverse_list_in_place(lst):
    left = 0
    right = len(lst) - 1

    while left < right:
        # Swap the elements at the left and right pointers
        lst[left], lst[right] = lst[right], lst[left]

        # Move the pointers towards the center
        left += 1
        right -= 1

# Example usage:
my_list = [1, 2, 3, 4, 5]
reverse_list_in_place(my_list)
print(my_list)  # Output: [5, 4, 3, 2, 1]
```

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

15. implement a code to find and remove duplicates from a list while preserving the original order of elements.

In [98]:
```
# if we want to preserve the original order of items then we cant convert the list int
```

In [99]:
```
li = [1,2,4,7,8,7,5,3,2,9,10,4,2,6,9,5,2]
li1 = []
for i in li:
    if i not in li1:
        li1.append(i)
print(li1)
```

```
[1, 2, 4, 7, 8, 5, 3, 9, 10, 6]
```

16. create a code to check if a given list is sorted(either in ascending or descending order) or not

In [102…
```
li = [1,3,5,7,2,4,0]
li2 = [1,2,3,4,5]
def sorting(list1):
    if list1==sorted(list1):
```

```
        return 'the list is sorted'
    else:
        return 'list is not sorted'

print(sorting(li))
print(sorting(li2))
```

```
list is not sorted
the list is sorted
```

17. write a code to merge two sorted list into a single sorted list

In [104…

```
li1 = [7,4,2,6,8,9]
li2 = [22,87,35,71]
sorted_li1 = sorted(li1)
sorted_li2 = sorted(li2)
li1.extend(li2)
li1.sort()
li1
```

Out[104]:

```
[2, 4, 6, 7, 8, 9, 22, 35, 71, 87]
```

18. implement a code to find the intersections of two given lists

In [119…

```
li1 = [1,3,4,6,9]
li2 = [0,2,3,9,10]
res=[]
for i in li1:
    if i in li2:
        res.append(i)
res
```

Out[119]:

```
[3, 9]
```

In [120…

```
res = [i for i in li1 if i in li2]
res
```

Out[120]:

```
[3, 9]
```

In [127…

```
li1 = set(li1)
li2 = set(li2)
res = li1.intersection(li2)
list(res)
```

Out[127]:

```
[9, 3]
```

19. create a code to find union of two lists without duplicates

In [133…
```
li1 = [1,3,4,6,9]
li2 = [0,2,3,9,10]
li1 = set(li1)
li2 = set(li2)
res = li1.union(li2)
list(res)
```

Out[133]:
```
[0, 1, 2, 3, 4, 6, 9, 10]
```

20. write a code to shuffle a given list randomly without using any built in shuffle function

In [134…
```
import random

def shuffle_list(lst):
    n = len(lst)
    for i in range(n - 1, 0, -1):
        # Generate a random index from 0 to i
        j = random.randint(0, i)
        # Swap the element at i with the element at j
        lst[i], lst[j] = lst[j], lst[i]

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
shuffle_list(my_list)
print(my_list)  # The output will be a randomly shuffled version of my_list
```

```
[1, 10, 6, 5, 7, 9, 2, 4, 8, 3]
```

21. write a code that takes two tuples as an input and return a new tuple containing elements that are common to both input tuples.

In [6]:
```
def common_tuple(tuple1,tuple2):
    common_elements = set(tuple1) & set(tuple2)
    return tuple(common_elements)

tu1 = (1,23,5,8,9,0)
tu2 = (34,6,8,1,0,4,9)
common_tuple(tu1,tu2)
```

Out[6]:
```
(0, 1, 8, 9)
```

22. create a code that prompts the user to enter to enter two sets of intengers seperated by commas, then print the intersection of these two sets.

In [39]:
```
set(map(int,input('enter the values seperated by commas: ').split(',')))
```

```
enter the values seperated by commas: 45,6,8,9
```

Out[39]:
```
{6, 8, 9, 45}
```

In [37]:
```
def get_int_set():
    return set(map(int,input().split(',')))# we r converting each string into int usir

def intersect():
    set1 = get_int_set()
    set2 = get_int_set()
    intersection = set1 & set2
    return intersection


intersect()
```

```
1,2,3,4,5
3,5,6,2,1,8
{1, 2, 3, 5}
```

Out[37]:

23. write a code to concatenate two tuples, the function should take two tuples as input and return a new tuple containing elements from both input tuples.

In [26]:
```
def concatenate_tuple():
    tuple1 = tuple(map(int,input().split(',')))
    tuple2 = tuple(map(int,input().split(',')))
    res = tuple1+tuple2
    return res

concatenate_tuple()
```

```
1,2,3,4
5,6,7,8
(1, 2, 3, 4, 5, 6, 7, 8)
```

Out[26]:

In [35]:
```
def concatenate_tuple(tuple1,tuple2):
    tuple1=list(tuple1)
    tuple2=list(tuple2)
    tuple1.extend(tuple2)
    return tuple(tuple1)

tu1 = (1,2,3,4)
tu2 = (5,6,7,8)
concatenate_tuple(tu1,tu2)
```

Out[35]:
```
(1, 2, 3, 4, 5, 6, 7, 8)
```

24. Develop the code that prompts the user to input two sets of strings. then, print the elements that are present in the first set but not in the second set.

In [43]:
```
set(input().split(','))
```

```
a,b,c
```

Out[43]:
```
{'a', 'b', 'c'}
```

In [42]:
```
def difference_set_of_string():
    set1 = set(input('enter the string seperated by commas: ').split(','))
    set2 = set(input('enter the string seperated by commas: ').split(','))
    diff = set1.difference(set2)
    print("Elements present in the first set but not in the second set:",diff)

difference_set_of_string()
```

```
enter the string seperated by commas: banana, mango, cherry, apple
enter the string seperated by commas: chicken, eggs, mango, apple
Elements present in the first set but not in the second set: {'banana', ' cherry'}
```

25. create a code that takes tuple and two integers as input. The function should returns a new tuple containing elements from the original tuple with the specifed range of indices

In [44]:
```
def slice_tuple(original_tuple, start_index, end_index):
    # Use slicing to create a new tuple with elements from start_index to end_index
    return original_tuple[start_index:end_index]

# Example usage:
user_tuple = (10, 20, 30, 40, 50, 60, 70, 80, 90)
start = 2  # Starting index (inclusive)
end = 5    # Ending index (exclusive)
result = slice_tuple(user_tuple, start, end)
print(result)  # Output: (30, 40, 50)
```

```
(30, 40, 50)
```

26. write a code that prompts the user to input two sets of characters, then print the union of these two sets

In [46]:
```
def union_of_string():
    set1 = set(input('enter the string seperated by commas: ').split(','))
    set2 = set(input('enter the string seperated by commas: ').split(','))
    union = set1.union(set2)
    return union

union_of_string()
```

```
enter the string seperated by commas: banana, mango, cherry, apple
enter the string seperated by commas: chicken, eggs, mango, apple
```

Out[46]:
```
{' apple', ' cherry', ' eggs', ' mango', 'banana', 'chicken'}
```

27. Develop a code that takes a tuple of integers as input . The function should return the maximum and minimum values of the tuple using tuple unpacking.

In [2]:
```python
def get_max_min_from_tuple(tup):
    # Ensure the tuple is not empty
    if not tup:
        raise ValueError("The tuple is empty and has no maximum or minimum values.")

    # Use built-in functions to get the maximum and minimum values
    max_val = max(tup)
    min_val = min(tup)

    # Return the results as a tuple
    return max_val, min_val

def input_to_tuple(prompt):
    # Prompt the user to input a tuple of integers
    user_input = input(prompt)
    # Convert the input string to a tuple of integers
    return tuple(map(int, user_input.split(',')))

def main():
    # Get the tuple of integers from the user
    user_tuple = input_to_tuple("Enter a tuple of integers separated by commas: ")

    # Get the maximum and minimum values using tuple unpacking
    max_val, min_val = get_max_min_from_tuple(user_tuple)

    # Print the results
    print(f"The maximum value is: {max_val}")
    print(f"The minimum value is: {min_val}")

# Run the program
main()
```

```
Enter a tuple of integers separated by commas: 2,3,4,6,7,9,90,6,3,12
The maximum value is: 90
The minimum value is: 2
```

28. create a code thhat defines two sets of integers. Then print the union, intersection, difference of these two sets.

In [4]:
```python
set1 = {1,2,34,5,6}
set2 = {7,9,0,2,1,5,87}
union = set1 | set2
print('Union of two sets: ',union)
print()
intersection = set1 & set2
print('intersection of two sets: ',intersection)
print()
diff1 = set1 - set2
print('Difference of set1: ',diff1)
print()
diff2 = set2-set1
print('Difference of set2: ',diff2)
```

```
Union of two sets:  {0, 1, 2, 34, 5, 6, 7, 9, 87}

intersection of two sets:  {1, 2, 5}

Difference of set1:  {34, 6}

Difference of set2:  {0, 9, 7, 87}
```

29. Write a code that takes a tuple and an element as input. The function should return the count of occurences of the given element in tuple

In [8]:
```
def count_of_elements(tuple1, value):
    res = tuple1.count(value)
    print('the count of {} is {}'.format(value,res))

tu1 = tuple(map(int,input().split(',')))
print(tu1)
val = int(input('enter the value: '))

count_of_elements(tu1,val)
```

```
1,2,4,6,7,9,0,8,6,4,3,2,1,3,5,67,8
(1, 2, 4, 6, 7, 9, 0, 8, 6, 4, 3, 2, 1, 3, 5, 67, 8)
enter the value: 4
the count of 4 is 2
```

30. Develop a code that prompts the user to input two sets of strings. Then print the symmetric difference of these two sets

In [9]:
```
def symmetric_diff():
    set1 = set(input().split(','))
    set2 = set(input().split(','))
    res = set1.symmetric_difference(set2)
    print('the symmetric difference of these two sets are: ',res)

symmetric_diff()
```

```
a,b,c,d,E,A,B
B,a,v,s,d
the symmetric difference of these two sets are:  {'s', 'A', 'c', 'b', 'E', 'v'}
```

31. Write a code that takes a list of words as input and returns dictionary where the keys are unique words and the values are frequencies of those words in the input lists.

In [10]:
```
l1 = list(input().split(','))
from collections import Counter
Counter(l1)
```

```
sahil,parrot,lion,tiger,parrot,sahil,apple,banana,mango,cherry,car,bike,building,car,
apple,apple,apple,banana
```

Out[10]:
```
Counter({'apple': 4,
         'sahil': 2,
         'parrot': 2,
         'banana': 2,
         'car': 2,
         'lion': 1,
         'tiger': 1,
         'mango': 1,
         'cherry': 1,
         'bike': 1,
         'building': 1})
```

32. write a code that takes two dictionaries as input and merge them into a single dictionary. if there are common keys, the values should be added together

In [22]:
```python
def merge_dicts(dict1, dict2):
    # Iterate through items in dict2
    for key, value in dict2.items():
        if key in dict1:
            # If the key exists, add the values
            dict1[key] += value
        else:
            # If the key does not exist, add it to the dictionary
            dict1[key] = value

    return dict1

dict1={'a':10,'b':24,'c':93}
dict2 = {'f':35,'a':40,'h':65}
merge_dicts(dict1,dict2)
```

Out[22]:
```
{'a': 50, 'b': 24, 'c': 93, 'f': 35, 'h': 65}
```

33. write a code to access a value in a nested dictionary. The function should take the dictionary and a list of keys as input, and return the corresponding value. If any of the keys do not exist in the dictionary, the function should return none.

In [33]:
```python
def get_nested_value(nested_dict, keys):
    # Start with the given nested dictionary
    current_level = nested_dict

    # Iterate through the list of keys
    for key in keys:
        # Check if the current key exists in the current level of the dictionary
        if key in current_level:
            # Move one level deeper
            current_level = current_level[key]
        else:
```

```
            # If the key does not exist, return None
            return None

    # Return the final value after traversing all keys
    return current_level

# Example usage
nested_dict = {
    'a': {
        'b': {
            'c': 10
        }
    }
}

keys = ['a', 'b', 'c']
value = get_nested_value(nested_dict, keys)
print(value)  # Output: 10

# Example with a non-existing key
keys = ['a', 'b', 'd']
value = get_nested_value(nested_dict, keys)
print(value)
```

```
10
None
```

34. write a code that takes a dictionary as input and returns a sorted version of it based on the values. You can choose whether to sort in ascending or descending order

In [45]:
```
def sorted_dic(dict1, reverse=True):
    res = sorted(dict1.items(),key=lambda x:x[1],reverse=reverse)
    return res

dict1 = {'a': 50, 'b': 24, 'c': 93, 'f': 35, 'h': 65}
print(sorted_dic(dict1,reverse=True)) # for descending orders
print(sorted_dic(dict1,reverse=False)) # for ascending orders
```

```
[('c', 93), ('h', 65), ('a', 50), ('f', 35), ('b', 24)]
[('b', 24), ('f', 35), ('a', 50), ('h', 65), ('c', 93)]
```

35. write a code that inverts a dictionary , swapping keys and values. Ensure that the inverted dictionary correctly handles cases where multiple keys have the same value by storing the keys as a list in the inverted dictionary.

In [46]:
```
def invert_dict(original_dict):
    inverted_dict = {}

    # Iterate through the items in the original dictionary
    for key, value in original_dict.items():
```

```
        # Check if the value already exists as a key in the inverted dictionary
        if value in inverted_dict:
            # If it does, append the current key to the list of keys
            inverted_dict[value].append(key)
        else:
            # If it doesn't, create a new list with the current key
            inverted_dict[value] = [key]

    return inverted_dict

# Example usage
original_dict = {
    'a': 1,
    'b': 2,
    'c': 1,
    'd': 3
}

inverted = invert_dict(original_dict)
print(inverted)
```

```
{1: ['a', 'c'], 2: ['b'], 3: ['d']}
```

In [ ]: