Started on	Friday, 3 May 2024, 12:34 PM
State	Finished
Completed on	Monday, 20 May 2024, 10:28 AM
Time taken	16 days 21 hours
Marks	5.00/5.00
Grade	**50.00** out of 50.00 (**100**%)
Name	DWIJESH SREERAM S 2022-CSD-A

Question **1**

Correct

Mark 1.00 out of 1.00

Mr.Harish is maintaining a phone directory which stores phone numbers. He will update the directory with phone numbers every week. While entering the input the number should not be stored inside if the phone number already exists. Finally he want his phone number to be printed in ascending order

Input: n – A1 array size and m – A2 arraysize

Array A1 containing phone numbers already existing and Array A2 containing numbers to be inserted

Ouput : Phone numbers printed in ascending order

Sample Test Case

Input

5

6

9840403212 9890909012 98123455 90123456 99123456

90909090 99999999 9840403212 12345678 12347890 99123456

Output

12345678 12347890 90123456 90909090 98123455 99123456 99999999 9840403212 9890909012

Answer: (penalty regime: 0 %)

```python
def insert_and_sort(existing_numbers, new_numbers):
    # Initialize a set to store unique phone numbers
    phone_numbers = set(existing_numbers)

    # Insert new numbers if they don't already exist
    for number in new_numbers:
        if number not in phone_numbers:
            phone_numbers.add(number)

    # Sort the phone numbers and convert them to a list
    sorted_numbers = sorted(phone_numbers)

    # Print the sorted phone numbers
    for number in sorted_numbers:
        print(number, end=' ')

# Runtime initialization
if __name__ == "__main__":
    # Input size of arrays A1 and A2
    n = int(input())
    m = int(input())

    # Input existing phone numbers in A1
    existing_numbers = list(map(int, input().split()[:n]))

    # Input new phone numbers in A2
    new_numbers = list(map(int, input().split()[:m]))

    # Call the function with input arrays
    insert_and_sort(existing_numbers, new_numbers)

```

	Input	Expected	Got	
✓	3 3 9876543211 1122334455 6677889911 6677889911 9876543211 4455667788	1122334455 4455667788 6677889911 9876543211	1122334455 4455667788 6677889911 9876543211	✓
✓	5 6 9840403212 9890909012 98123455 90123456 99123456 90909090 99999999 9840403212 12345678 12347890 99123456	12345678 12347890 90123456 90909090 98123455 99123456 99999999 9840403212 9890909012	12345678 12347890 90123456 90909090 98123455 99123456 99999999 9840403212 9890909012	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Given two lists, print all the common element of two lists.

Note: Sort the list before printing.

Examples:

```
Input :
1 2 3 4 5
5 6 7 8 9
Output :
5

Input :
1 2 3 4 5
6 7 8 9
Output :
No common elements

Input :
1 2 3 4 5 6
5 6 7 8 9
Output :
5 6
```

Answer: (penalty regime: 0 %)

```python
# Get the input for the two lists
list1 = list(map(int, input().split()))
list2 = list(map(int, input().split()))

# Find the common elements
common_elements = sorted(list(set(list1) & set(list2)))

# Print the common elements
if common_elements:
    print(' '.join(map(str, common_elements)))
else:
    print("No common elements")
```

	Input	Expected	Got	
✓	1 2 3 4 5 5 6 7 8 9	5	5	✓
✓	1 2 3 4 5 6 7 8 9	No common elements	No common elements	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

write a program to identify the common item present in three different set but not on the other set and display the items in the sorted order.

input:

10 50 40 60 30

40 30 70 60 30

20 50 10 75 80

output:

20 70 75 80

Answer: (penalty regime: 0 %)

```
import re

# Function to find common items present in one set but not in the other two sets
def find_unique_items(set1, set2, set3):
    unique_items = []
    for item in set1:
        if item not in set2 and item not in set3:
            unique_items.append(item)
    return unique_items

# Read input sets from user
set1_str = input()
set2_str = input()
set3_str = input()

# Extract integers from input strings
set1 = set(map(int, re.findall(r'\d+', set1_str)))
set2 = set(map(int, re.findall(r'\d+', set2_str)))
set3 = set(map(int, re.findall(r'\d+', set3_str)))

# Find common items present in one set but not in the other two sets
unique_items1 = find_unique_items(set1, set2, set3)
unique_items2 = find_unique_items(set2, set1, set3)
unique_items3 = find_unique_items(set3, set1, set2)

# Combine the unique items from each set
unique_items = unique_items1 + unique_items2 + unique_items3

# Sort the combined result
sorted_unique_items = sorted(unique_items)

# Print the sorted list of unique items as a set with curly braces
print("{" + ",".join(map(str, sorted_unique_items)) + "}")


```

	Test	Input	Expected	Got	

✓	1	{10,50,40,60,30}
		{40,30,70,60,65}
		{20,50,10,75,80}
			{20,65,70,75,80}	{20,65,70,75,80}	✓

	Test	Input	Expected	Got	

✓	2	`{10,15,20,40,50}`
		`{30,20,40,10,25}`
		`{40,50,10,45,55}`
			`{15,25,30,45,55}`	`{15,25,30,45,55}`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

A number is stable if each digit occur the same number of times.i.e, the frequency of each digit in the number is the same. For e.g. 2277,4004,11,23,583835,1010 are examples for stable numbers.
Similarly, a number is unstable if the frequency of each digit in the number is NOT same.

Sample Input:

2277

Sample Output:

Stable Number

Sample Input 2:

121

Sample Output 2:

Unstable Number

Answer: (penalty regime: 0 %)

```python
def is_stable_number(number):
    # Convert the number to a string to iterate through each digit
    num_str = str(number)

    # Create a dictionary to store the frequency of each digit
    digit_count = {}

    # Count the frequency of each digit
    for digit in num_str:
        if digit in digit_count:
            digit_count[digit] += 1
        else:
            digit_count[digit] = 1

    # Get the frequency of the first digit to compare with others
    frequencies = list(digit_count.values())
    first_frequency = frequencies[0]

    # Check if all frequencies are the same
    for frequency in frequencies:
        if frequency != first_frequency:
            return "Unstable Number"

    return "Stable Number"

# Read input number from user
number = input()

# Call the function and print the result
print(is_stable_number(number))

```

	Input	Expected	Got	
✓	9988	Stable Number	Stable Number	✓
✓	12	Stable Number	Stable Number	✓
✓	455	Unstable Number	Unstable Number	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Two strings, *a* and *b*, are called anagrams if they contain all the same characters in the same frequencies. For example, the anagrams of CAT are CAT, ACT, TAC, TCA, ATC, and CTA.

Complete the function in the editor. If *a* and *b* are case-insensitive anagrams, print "Anagrams"; otherwise, print "Not Anagrams" instead.

Input Format

The first line contains a string denoting *a*.
The second line contains a string denoting *b*.

Constraints

- *1 ≤ length(a), length(b) ≤ 50*
- Strings *a* and *b* consist of English alphabetic characters.
- The comparison should NOT be case sensitive.

Output Format

Print "Anagrams" if *a* and *b* are case-insensitive anagrams of each other; otherwise, print "Not Anagrams" instead.

Sample Input 0

anagram

margana

Sample Output 0

Anagrams

Explanation 0

Character	**Frequency:** anagram	**Frequency:** margana
A or a	3	3
G or g	1	1
N or n	1	1
M or m	1	1
R or r	1	1

The two strings contain all the same letters in the same frequencies, so we print "Anagrams".

Answer: (penalty regime: 0 %)

```
def are_anagrams(a, b):
    # Convert both strings to lower case
    a = a.lower()
    b = b.lower()

    # If the lengths are not equal, they cannot be anagrams
    if len(a) != len(b):
        return "Not Anagrams"

    # Create dictionaries to count the frequency of each character
    char_count_a = {}
    char_count_b = {}

    # Count the frequency of each character in string a
    for char in a:
        if char in char_count_a:
            char_count_a[char] += 1
        else:
            char_count_a[char] = 1
```

```python

    # Count the frequency of each character in string b
    for char in b:
        if char in char_count_b:
            char_count_b[char] += 1
        else:
            char_count_b[char] = 1

    # Compare the character frequency counts
    if char_count_a == char_count_b:
        return "Anagrams"
    else:
        return "Not Anagrams"

# Read input strings from user
a = input()
b = input()

# Call the function and print the result
print(are_anagrams(a, b))

```

	Input	Expected	Got	
✓	madam maDaM	Anagrams	Anagrams	✓
✓	DAD DAD	Anagrams	Anagrams	✓
✓	MAN MAM	Not Anagrams	Not Anagrams	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

◄ Week-09_MCQ

Jump to...

WEEK-09-Extra ►