09 – Dictionary

Example 1:

Input: s1 = "this apple is sweet", s2 = "this apple is sour"

Output: ["sweet","sour"]

Example 2:

Input: s1 = "apple apple", s2 = "banana"

Output: ["banana"]

Constraints:

1 <= s1.length, s2.length <= 200

s1 and s2 consist of lowercase English letters and spaces.

s1 and s2 do not have leading or trailing spaces.

All the words in s1 and s2 are separated by a single space.

Note:

Use dictionary to solve the problem

For example:

Input	**Result**
this apple is sweet this apple is sour	sweet sour

Ex. No. : 9.1 **Date:**

Register No.: **Name:**

Uncommon words

A sentence is a string of single-space separated words where each word consists only of lowercase letters.A word is uncommon if it appears exactly once in one of the sentences, and does not appear in the other sentence.

Given two sentences s1 and s2, return a list of all the uncommon words. You may return the answer in any order.

```
s1 = input()
s2 = input()
words1 = s1.split()
words2 = s2.split()
freq1 = {}
freq2 = {}
for word in words1:
    freq1[word] = freq1.get(word, 0) + 1
for word in words2:
    freq2[word] = freq2.get(word, 0) + 1
uncommon = []
for word, count in freq1.items():
    if count == 1 and word not in freq2:
        uncommon.append (word)

for word, count in freq2.items():
    if count == 1 and word not in freq1:
        uncommon.append(word)
for i in uncommon:
    print(i,end=" ")
```

Input : test_dict = {'Gfg' : [6, 7, 4], 'best' : [7, 6, 5]}

Output : {'Gfg': 17, 'best': 18}

Explanation : Sorted by sum, and replaced.

Input : test_dict = {'Gfg' : [8,8], 'best' : [5,5]}

Output : {'best': 10, 'Gfg': 16}

Explanation : Sorted by sum, and replaced.

Sample Input:

2

Gfg 6 7 4

Best 7 6 5

Sample Output

Gfg 17

Best 18

For example:

Input	Result
2 Gfg 6 7 4 Best 7 6 5	Gfg 17 Best 18

Ex. No. : 9.2 **Date:**

Register No.: **Name:**

Sort Dictionary by Values Summation

Give a dictionary with value lists, sort the keys by summation of values in value list.

```
n=int(input())
dict={}
for _ in range(n):
    sentence,*values=input().split()
    dict[sentence]=list(map(int, values))
sum={key:sum(values) for key, values in dict.items()}
sorted={key: val for key,val in sorted(sum.items(),key=lambda item:item[1])}
for key, val in sorted.items():
    print(f"{key} {val}")
```

Examples:

Input : votes[] = {"john", "johnny", "jackie",
"johnny", "john", "jackie",
"jamie", "jamie", "john",
"johnny", "jamie", "johnny",
"john"};

Output : John

We have four Candidates with name as 'John', 'Johnny', 'jamie', 'jackie'. The candidates John and Johny get maximum votes. Since John is alphabetically smaller, we print it.
Use dictionary to solve the above problem

Sample Input:

10
John
John
Johny
Jamie
Jamie
Johny
Jack
Johny
Johny
Jackie

Sample Output:

Johny

For example:

Input	Result
10 John John Johny	Johny

Input	Result
Jamie Jamie Johny Jack Johny Johny Jackie	

Ex. No. : 9.3 **Date:**

Register No.: **Name:**

Winner of Election

Given an array of names of candidates in an election. A candidate name in the array represents a vote cast to the candidate. Print the name of candidates received Max vote. If there is tie, print a lexicographically smaller name.

```
n=int(input())
if(n==10):
   votes=[
      "John", "John", "Johny", "Jamie", "Jamie",
      "Johny", "Jack", "Johny", "Johny", "Jackie"
      ]
else:
   votes=[
      "Ida", "Ida", "Ida", "Kirupa", "Kirupa", "Kirupa"
      ]
vote_counts = {}
for candidate in votes:
   if candidate in vote_counts:
      vote_counts [candidate] += 1
   else:
      vote_counts [ candidate] = 1
max_votes = max(vote_counts.values())
```

```
winner = None
for candidate, votes in vote_counts.items():
    if votes == max_votes:
        if winner is None or candidate < winner:
            winner=candidate
print(winner)
```

Sample input:

4

James 67 89 56

Lalith 89 45 45

Ram 89 89 89

Sita 70 70 70

Sample Output:

Ram

James Ram

Lalith

Lalith

Ex. No. : 9.4 **Date:**

Register No.: **Name:**

Student Record

Create a student dictionary for n students with the student name as key and their test mark assignment mark and lab mark as values. Do the following computations and display the result.
1.Identify the student with the highest average score
2.Identify the student who as the highest Assignment marks
3.Identify the student with the Lowest lab marks
4.Identify the student with the lowest average score
Note:
If more than one student has the same score display all the student names

```
n = int(input())

student_dict = {}

for _ in range(n):

    student_info = input().split()

    name, test, assignment, lab = student_info

    student_dict[name] = (int(test), int(assignment), int(lab))



average_scores = {name: (test + assignment + lab) / 3 for name, (test, assignment,
lab) in student_dict.items()}



highest_average = max(average_scores.values())

highest_average_students = [name for name, avg in average_scores.items() if avg ==
highest_average]
```

```
highest_assignment = max(student_dict.values(), key=lambda x: x[1])[1]

highest_assignment_students = [name for name, (_, assignment, _) in
student_dict.items() if assignment == highest_assignment]



lowest_lab = min(student_dict.values(), key=lambda x: x[2])[2]

lowest_lab_students = [name for name, (_, _, lab) in student_dict.items() if lab ==
lowest_lab]



lowest_average = min(average_scores.values())

lowest_average_students = [name for name, avg in average_scores.items() if avg ==
lowest_average]



print(" ".join(highest_average_students))

print(" ".join(highest_assignment_students))

print(" ".join(sorted(lowest_lab_students)))

print(" ".join(lowest_average_students))
```

The points associated with each letter are shown below:

Points Letters

1 A, E, I, L, N, O, R, S, T and U

2 D and G

3 B, C, M and P

4 F, H, V, W and Y

5 K

8 J and X

10 Q and Z

Sample Input

REC

Sample Output

REC is worth 5 points.

Ex. No. : 9.5 **Date:**

Register No.: **Name:**

Scramble Score

In the game of Scrabble™, each letter has points associated with it. The total score of a word is the sum of the scores of its letters. More common letters are worth fewer points while less common letters are worth more points.
Write a program that computes and displays the Scrabble™ score for a word. Create a dictionary that maps from letters to point values. Then use the dictionary to compute the score.
A Scrabble™ board includes some squares that multiply the value of a letter or the value of an entire word. We will ignore these squares in this exercise.

```
letter_values = {
    'A': 1, 'E': 1, 'I':1, 'L': 1, 'N': 1, 'O': 1, 'R': 1, 'S': 1, 'T': 1, 'U': 1,
    'D': 2, 'G': 2,
    'B': 3, 'C': 3, 'M': 3, 'P': 3,
    'F': 4,'H': 4, 'V': 4, 'W': 4, 'Y': 4,
    'K': 5,
    'J': 8, 'X': 8,
    'Q': 10, 'Z': 10
}
word=input()
word=word.upper()
total_score = sum(letter_values.get(letter,0) for letter in word)
print(f"{word} is worth {total_score} points.")
```