1. 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.

**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

 Program:

def find\_winner(votes):

count = {}

for candidate in votes:

count[candidate] = count.get(candidate, 0) + 1

max\_votes = max(count.values())

winner = min([candidate for candidate, votes in count.items() if votes == max\_votes])

return winner

n = int(input().strip())

votes = []

for \_ in range(n):

vote = input().strip()

votes.append(vote)

print(find\_winner(votes))



2. 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

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

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4  James 67 89 56  Lalith 89 45 45  Ram 89 89 89  Sita 70 70 70 | Ram  James Ram  Lalith  Lalith |

Program:

def highest\_average\_score(students):

averages = {name: sum(scores) / len(scores) for name, scores in students.items()}

max\_average = max(averages.values())

top\_students = sorted([name for name, avg in averages.items() if avg == max\_average])

return top\_students

def highest\_assignment\_marks(students):

max\_assignment = max([scores[1] for scores in students.values()])

top\_students = sorted([name for name, scores in students.items() if scores[1] == max\_assignment])

return top\_students

def lowest\_lab\_marks(students):

min\_lab = min([scores[2] for scores in students.values()])

bottom\_students = sorted([name for name, scores in students.items() if scores[2] == min\_lab])

return bottom\_students

def lowest\_average\_score(students):

averages = {name: sum(scores) / len(scores) for name, scores in students.items()}

min\_average = min(averages.values())

bottom\_students = sorted([name for name, avg in averages.items() if avg == min\_average])

return bottom\_students

n = int(input())

students = {}

for \_ in range(n):

name, test, assignment, lab = input().split()

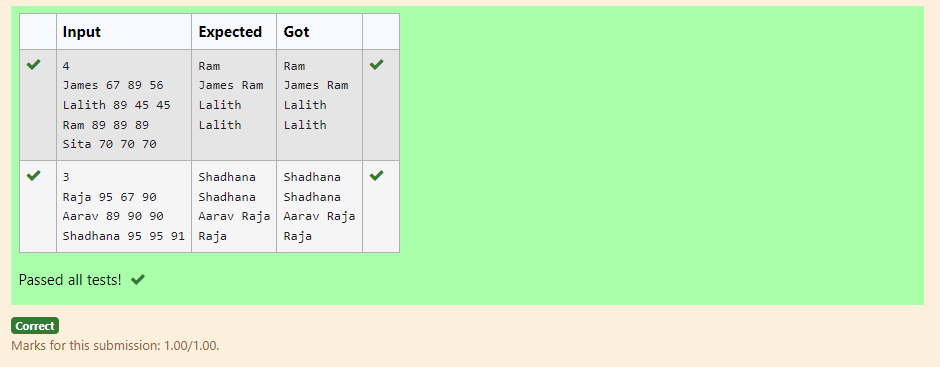
students[name] = [int(test), int(assignment), int(lab)]

print(\*highest\_average\_score(students))

print(\*highest\_assignment\_marks(students))

print(\*lowest\_lab\_marks(students))

print(\*lowest\_average\_score(students))



3. 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. 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

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.

[Sample](https://www.rajalakshmicolleges.net/moodle/mod/quiz/view.php?id=5127) Input

REC

[Sample](https://www.rajalakshmicolleges.net/moodle/mod/quiz/view.php?id=5127) Output

REC is worth 5 points.

**For example:**

| **Input** | **Result** |
| --- | --- |
| REC | REC is worth 5 points. |

Program:

def scrabble\_score(word):

letter\_scores = {

'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,

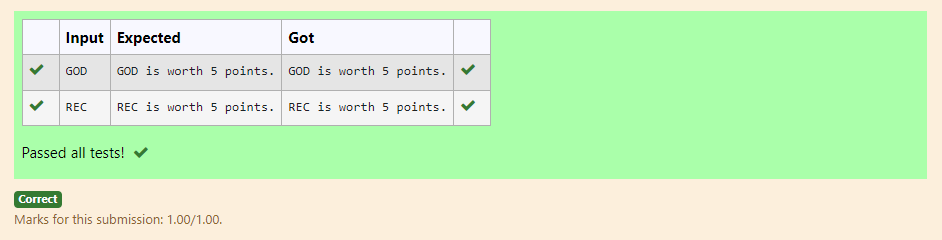
'Q': 10, 'Z': 10

}

total\_score = sum(letter\_scores.get(letter.upper(), 0) for letter in word)

return total\_score

word = input()

print(f"{word.upper()} is worth {scrabble\_score(word)} points.")

4.  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.

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 |

Program:

def uncommon\_words(s1, s2):

def word\_count(sentence):

words = sentence.split()

word\_dict = {}

for word in words:

word\_dict[word] = word\_dict.get(word, 0) + 1

return word\_dict

s1\_word\_count = word\_count(s1)

s2\_word\_count = word\_count(s2)

uncommon = set()

for word, count in s1\_word\_count.items():

if count == 1 and word not in s2\_word\_count:

uncommon.add(word)

for word, count in s2\_word\_count.items():

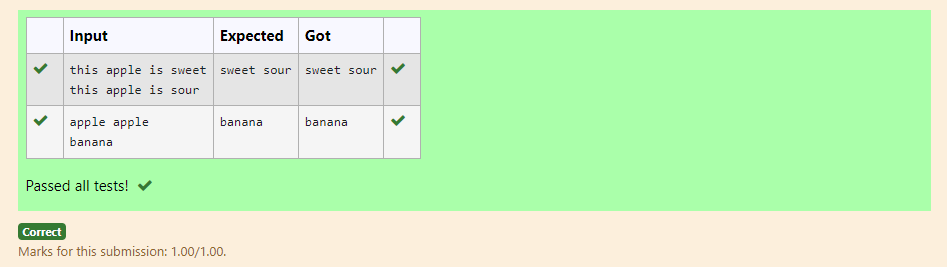
if count == 1 and word not in s1\_word\_count:

uncommon.add(word)

return ' '.join(uncommon)

s1 = input()

s2 = input()

print(uncommon\_words(s1, s2))

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

**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 |

Program:

num\_pairs = int(input())

test\_dict = {}

for \_ in range(num\_pairs):

key, \*values = input().split()

test\_dict[key] = list(map(int, values))

sorted\_keys = sorted(test\_dict, key=lambda k: sum(test\_dict[k]))

sorted\_dict = {key: sum(test\_dict[key]) for key in sorted\_keys}

for key, value in sorted\_dict.items():

print(f"{key} {value}")

