Chapter 7

Collections

Chapter 7

Collections

In This Chapter:

1. collections.Counter()
2. DefaultDict
3. Collections.namedtuple()
4. Collections.OrderedDict()
5. Word Order
6. Collections.deque()

7.1 collections.Counter()

A counter is a container that stores elements as dictionary keys, and their counts are stored as dictionary values.

Sample Code

```
>>> from collections import Counter
>>>
>>> myList = [1,1,2,3,4,5,3,2,3,4,2,1,2,3]
>>> print Counter(myList)
Counter({2: 4, 3: 4, 1: 3, 4: 2, 5: 1})
>>>
>>> print Counter(myList).items()
[(1, 3), (2, 4), (3, 4), (4, 2), (5, 1)]
>>>
>>> print Counter(myList).keys()
[1, 2, 3, 4, 5]
>>>
>>> print Counter(myList).values()
[3, 4, 4, 2, 1]
```

Ragu is a shoe shop owner. His shop has X number of shoes.
He has a list containing the size of each shoe he has in his shop.
There are N number of customers who are willing to pay x_i amount of money only if they get the shoe of their desired size.
Your task is to compute how much money Ragu earned.

Input Format

The first line contains X, the number of shoes.
The second line contains the space separated list of all the shoe sizes in the shop. The third line contains N, the number of customers. The next N lines contain the space separated values of the shoe-size desired by the customer and x_i, the price of the shoe.

Constraints

$0 < X < 10^3$
$0 < N \leq 10^3$
$20 < x_i < 100$
$2 < shoe\ size < 20$

Output Format

Print the amount of money earned by Ragu.

Sample Input

```
10
2 3 4 5 6 8 7 6 5 18
6
6 55
6 45
6 55
4 40
18 60
10 50
```

Sample Output

```
200
```

Explanation

Customer 1: Purchased size 6 shoe for **$55**.
Customer 2: Purchased size 6 shoe for **$45**.
Customer 3: Size 6 no longer available, so no purchase.

Customer 4: Purchased size 4 shoe for **$40**.
Customer 5: Purchased size 18 shoe for **$60**.
Customer 6: Size 10 not available, so no purchase.
Total money earned = **55+45+40+60=$200**

Code

```
import collections

X = int(raw_input())
shoes = collections.Counter(map(int, raw_input().split()))
N = int(raw_input())

money = 0

for i in range(N):
    (size, price) = map(int, raw_input().split())

    if shoes[size] > 0:
        shoes[size] -= 1
        money += price

print money
```

7.2 DefaultDict

The *defaultdict* tool is a container in the collections class of Python. It's similar to the usual dictionary (*dict*) container, but it has one difference: The value fields' data type is specified upon initialization.

For example:

```
from collections import defaultdict
d = defaultdict(list)
d['python'].append("awesome")
d['something-else'].append("not relevant")
d['python'].append("language")
for i in d.items():
   print i
```

This prints:

```
('python', ['awesome', 'language'])
('something-else', ['not relevant'])
```

In this challenge, you will be given integers, n and m. There are n words, which might repeat, in word group A. There are m words belonging to word group B. For each m words, check whether the word has appeared in group A or not. Print the indices of each occurrence of m in group A. If it does not appear, print -1.

Constraints

$1 \leq n \leq 10000$

$1 \leq m \leq 100$

$1 \leq \ length\ of\ each\ word\ in\ the\ input \leq 100$

Input Format

The first line contains integers, n and m separated by a space.
The next n lines contains the words belonging to group A.
The next m lines contains the words belonging to group B.

Output Format

Output m lines.
The i^{th} line should contain the 1-indexed positions of the occurrences of the i^{th} word separated by spaces.

Sample Input

```
5 2
a
a
b
a
b
a
b
```

Sample Output

```
1 2 4
3 5
```

Explanation

'**a**' appeared 3 times in positions 1, 2 and 4.

'**b**' appeared 2 times in positions 3 and 5.

If '**c**' also appeared in word group B, you would print -1.

Code

```
from __future__ import print_function
from collections import defaultdict

(n, m) = map(int, raw_input().split())

word2indices = defaultdict(list)

for i in range(n):
    word2indices[raw_input()].append(i + 1)

for i in range(m):
    word = raw_input()
    if word2indices[word]:
        print(*word2indices[word])
    else:
        print(-1)
```

7.3. Collections.namedtuple()

Basically, *namedtuples* are easy to create, lightweight object types. They turn tuples into convenient containers for simple tasks. With *namedtuples*, you don't have to use integer indices for accessing members of a tuple.

Example 01

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point','x,y')
>>> pt1 = Point(1,2)
>>> pt2 = Point(3,4)
>>> dot_product = ( pt1.x * pt2.x ) +( pt1.y * pt2.y )
>>> print dot_product
11
```

Example 02

```
>>> from collections import namedtuple
>>> Car = namedtuple('Car','Price Mileage Colour Class')
>>> xyz = Car(Price = 100000, Mileage = 30, Colour = 'Cyan',
Class = 'Y')
>>> print xyz
Car(Price=100000, Mileage=30, Colour='Cyan', Class='Y')
>>> print xyz.Class
Y
```

Dr. John Wesley has a spreadsheet containing a list of student's , ID, MARKS, CLASS and NAME. Help Dr. Wesley calculate the average marks of the students.
Columns can be in any order. IDs, marks, class and name can be written in any order in the spreadsheet. Column names are **ID**, **MARKS**, **CLASS** and **NAME**. (The spelling and case type of these names won't change.)

Input Format

The first line contains an integer N, the total number of students.
The second line contains the names of the columns in any order.
The next N lines contains **ID**, **MARKS**, **CLASS** and **NAME**, under their respective column names.

Constraints

$0 < N \leq 100$

Output Format

Print the average marks of the list corrected to 2 decimal places.

Sample Input, TESTCASE 01

```
5
ID          MARKS       NAME        CLASS
1           97          Raymond     7
2           50          Steven      4
3           91          Adrian      9
4           72          Stewart     5
5           80          Peter       6
```

TESTCASE 02

```
5
MARKS       CLASS       NAME        ID
92          2           Calum       1
82          5           Scott       2
94          2           Jason       3
55          8           Glenn       4
82          2           Fergus      5
```

Sample Output, TESTCASE 01

```
78.00
```

TESTCASE 02

```
81.00
```

```
from collections import namedtuple
N, COLS = int(raw_input()), namedtuple('Student', raw_input().split())

students = []

for i in range(N):
    data = raw_input().split()
    students.append(COLS(data[0], data[1], data[2], data[3]))
sum = 0

for iterator in students:
    sum += float(iterator.MARKS)
print (sum / float(N))
```

7.4. Collections.OrderedDict()

An *OrderedDict* is a dictionary that remembers the order of the keys that were inserted first. If a new entry overwrites an existing entry, the original insertion position is left unchanged.

Example

```
>>> from collections import OrderedDict
>>> ordinary_dictionary = {}
>>> ordinary_dictionary['a'] = 1
>>> ordinary_dictionary['b'] = 2
>>> ordinary_dictionary['c'] = 3
>>> ordinary_dictionary['d'] = 4
>>> ordinary_dictionary['e'] = 5
>>> print ordinary_dictionary
{'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}
```

```
>>> ordered_dictionary = OrderedDict()
>>> ordered_dictionary['a'] = 1
>>> ordered_dictionary['b'] = 2
>>> ordered_dictionary['c'] = 3
>>> ordered_dictionary['d'] = 4
>>> ordered_dictionary['e'] = 5
>>> print ordered_dictionary
OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)])
```

You are the manager of a supermarket. You have a list of N items together with their prices that consumers bought on a particular day. Your task is to print each item_name and net_price in order of its first occurrence.

item_name = Name of the item.

net_price = Quantity of the item sold multiplied by the price of each item.

Input Format

The first line contains the number of items,N .

The next N lines contains item's name &price, separated by a space.

Constraints

$0 < N \leq 100$

Output Format

Print the item_name and net_price in order of its first occurrence.

Sample Input

```
9
BANANA FRIES 12
POTATO CHIPS 30
APPLE JUICE 10
CANDY 5
APPLE JUICE 10
CANDY 5
CANDY 5
CANDY 5
POTATO CHIPS 30
```

Sample Output

```
BANANA FRIES 12
POTATO CHIPS 60
APPLE JUICE 20
CANDY 20
```

Explanation

BANANA FRIES: Quantity bought: 1, Price: 12

`Net Price`: 12

POTATO CHIPS: Quantity bought: 2, Price: 30

`Net Price`: 60

APPLE JUICE: Quantity bought: 2, Price: 10

`Net Price`: 20

CANDY: Quantity bought: 4, Price: 5

`Net Price`: 20

Code

```
from collections import OrderedDict
d = OrderedDict()

N = int(raw_input())

for i in range(N):
    itemName, itemPrice = raw_input().rsplit(' ', 1)
    if itemName not in d:
        d[itemName] = int(itemPrice)
    else:
        d[itemName] += int(itemPrice)

for name, price in d.items():
    print str(name), str(price)
```

7.5. Word Order

You are given n words. Some words may repeat. For each word, output its number of occurrences. The output order should correspond with the input order of appearance of the word.

Note: Each input line ends with a "**\n**" character.

Constraints:

$1 \leq n \leq 10^5$

The sum of the lengths of all the words do not exceed 10^6
All the words are composed of lowercase English letters only.

Input Format

The first line contains the integer, n.
The next n lines each contain a word.

Output Format

Output 2 lines.
On the first line, output the number of distinct words from the input.
On the second line, output the number of occurrences for each distinct word according to their appearance in the input.

Sample Input

```
4
bcdef
abcdefg
bcde
bcdef
```

Sample Output

```
3
2 1 1
```

Explanation

There are 3 distinct words. Here, "**bcdef**" appears twice in the input at the first and last positions. The other words appear once each. The order of the first appearances
are "**bcdef**", "**abcdefg**" and "**bcde**" which corresponds to the output.

Code

```
import collections

n = int(raw_input())

d = collections.OrderedDict()

for i in range(n):
    word = raw_input()
    if word in d:
        d[word] += 1
    else:
        d[word] = 1

print len(d)

for key, value in d.iteritems():
    print value,
```

7.6. Collections.deque()

A *deque* is a double-ended queue. It can be used to add or remove elements from both ends.

Deques support thread safe, memory efficient appends and pops from either side of the deque with approximately the same O(1) performance in either direction.

Example

```
>>> from collections import deque
>>> d = deque()
>>> d.append(1)
>>> print d
deque([1])
>>> d.appendleft(2)
>>> print d
```

```
deque([2, 1])
>>> d.clear()
>>> print d
deque([])
>>> d.extend('1')
>>> print d
deque(['1'])
>>> d.extendleft('234')
>>> print d
deque(['4', '3', '2', '1'])
>>> d.count('1')
1
>>> d.pop()
'1'
>>> print d
deque(['4', '3', '2'])
>>> d.popleft()
'4'
>>> print d
deque(['3', '2'])
>>> d.extend('7896')
>>> print d
deque(['3', '2', '7', '8', '9', '6'])
>>> d.remove('2')
>>> print d
deque(['3', '7', '8', '9', '6'])
>>> d.reverse()
>>> print d
deque(['6', '9', '8', '7', '3'])
>>> d.rotate(3)
>>> print d
deque(['8', '7', '3', '6', '9'])
```

Task

Perform *append*, *pop*, *popleft* and *appendleft* methods on an empty deque d.

Input Format

The first line contains an integer N, the number of operations.
The next N lines contains the space separated names of methods and their values.

Constraints

$0 < N \leq 100$

Output Format

Print the space separated elements of deque d.

Sample Input

```
6
append 1
append 2
append 3
appendleft 4
pop
popleft
```

Sample Output

```
1 2
```

Code

```
from collections import deque
d = deque()
N = int(raw_input())

for i in range(N):
    command = raw_input().split()
    if command[0] == 'pop':
        d.pop()
    elif command[0] == 'append':
        d.append(command[1])
    elif command[0] == 'appendleft':
        d.appendleft(command[1])
    elif command[0] == 'popleft':
        d.popleft()
print ' '.join(d)
```