'''

This function takes two arguments,

data1 and data2, which contain

key-value pairs. All key-value

pairs within data1 are unique.

Similarly, all key-value pairs

within data2 are unique. However,

there may be key-value pairs (k, v1)

in data1 and (k, v2) in data2 with a

common key k. In this case, v1 and

v2 may be the same, or v1 and v2 may

be different.

This function should modify only

data1 and return a (possibly empty)

dictionary as follows:

For every key-value pair (k, v2) in

data2, if no key-value pair with key

k exists in data1, then the pair

(k, v2) should be added to data1.

Otherwise, there is a unique pair

(k, v1) already in data1. If v1 and

v2 are different, the pair (k, v1)

should be removed from data1 and the

key-value pair (k, [v1, v2]) should

be added to the (initially empty)

dictionary to be returned.

In this implementation, data1 is a

dictionary and data2 is a list where

each key-value pair in data2 is also

a list [key, value] of length 2.

'''

Answer:

a)Input

3

1 2

2 2

8 7

3 3

4 4

when key does not exist in data 1, the key value pair is not added to it

b)

def uniqueUpdate(data1, data2):

# Initially empty dictionary

dupKeys = {}

# Examine every (k, v2) pair in data2

for [k, v2] in data2:

# Check if there is a key-value

# pair with key = k in data1

if k in data1:

v1 = data1[k]

# (k, v1) in dict1

# Check if v1 != v2

if v1 != v2:

# Add (k, [v1, v2])

# to dictionary

dupKeys[k] = [v1, v2]

# Remove (k, v1) from data1

del data1[k]

else:

# Add (k, v2) to data1

data1[k] = v2

# After processing all (k, v2) in

# data2, return the dictionary

return dupKeys

f k in data1:

v1 = data1[k]

if v1 != v2:

dupKeys[k] = [v1, v2]

del data1[k]

else:

data1[k] = v2

return dupKeys

Test cases : 5c.

Test case 1:

**4**

1 2

3 3

3 8

4 9

**2**

3 3

4 4

Test case 2:

**4**

1 2

2 2

3 3

4 19

**2**

3 3

4 19

Test case 3;

The test case written in 5a which breaks the initially written code can be written.