Week 6 - Practice Programming

Problem 1

Suppose a crowd is waiting outside a shop counter forming a waiting line for collecting valuables. Each person holds a piece of paper with a order number written on it. One of the person at a given position leaves the queue for some reason and joins at the end of the queue.

Please write a python program `shift_a_person` to imitate this scenario.

1. The function has two parameters :
    - A list named `order_list` of datatype `list` consists of positive integers representing the order number of each person in the list and
    - A variable `position` of type `int` indicating the initial position of a person to be shifted at the end of the list. The position is counted from left to right of the list starting from `1`.
2. The person located at the `position` is moved to the end of the list. This means the corresponding order number becomes the rightmost element of the updated list. If the `position` is an invalid number (less than 1 or more than the length of the list), it should be considered the first person standing in the list from the left.
3. If two or more persons have the same order number, a person holding duplicate order number is removed from the beginning of the list until every person is left with a unique order number.
4. In the function point-2 is followed by point-3, this means they are executed in the same order as they are listed. The updated list is returned by the function.

```
def shift_a_person(order_list, position = 1):
    # write function body
    #
    #
```

Test Cases

Type	Input	Output
Public	`1 2 3 4 5 6 7 8 9` `6`	`[1, 2, 3, 4, 5, 7, 8, 9, 6]`
Public	`1 2 3 4 5 6 7 2 9` `6`	`[1, 3, 4, 5, 7, 2, 9, 6]`
Private	`1 3 4 5 6 7 7 8` `3`	`[1, 3, 5, 6, 7, 8, 4]`
Private	`27 73 13 98 56 37` `9`	`[73, 13, 98, 56, 37, 27]`
Private	`27 73 13 98 56 37 13 13` `2`	`[27, 98, 56, 37, 13, 73]`
Private	`27 27 27 27` `2`	`[27]`

Suffix Code Block - Hidden

```
order_list = list(map(int, input().split(" ")))
position = int(input())
print(shift_a_person(order_list, position))
```

Answer

```
def shift_a_person(order_list, position = 1):
    # position set to 1 if it is out of bound [1, len(order_list)]
    if position not in range(1, len(order_list) + 1):
        position = 1
    # create a new list by slicing original list
    # and append element indicated by position to the end of this list
    new_order_list = order_list[:position-1] + order_list[position:] + order_list[position-1:position]
    # remove an element if it occurs multiple times till only first occurance are left
    for num in new_order_list[:]: # new_order_list[:] is creating a copy of new_order_list
        if new_order_list.count(num) > 1:
            new_order_list.remove(num)
    return new_order_list
```

Problem 2

Question

You are given data in the form described in the first column of the below table. The data is stored as a list of dictionaries. Each dictionary item is a list of values that represents a data column. This is called a `tablular form (or column measure)` of data.

The second row of the table shows one sample test case input and output.

Input form (tabular form)	Output form (record form)

`[`
`{col_A: [A_val_1, A_val_2, A_val_3,...]},`
`{col_B: [B_val_1, B_val_2, B_val_3,...]},`
`....`
`]`
	`[`
	`{col_A: A_val_1, col_B: B_val_1, col_C: C_val_1, ....},`
	`{col_A: A_val_2, col_B: B_val_2, col_C: C_val_3, ....},`
	`....`
	`]`

`[{'Roll' : [1, 2, 3]},`
`{'Name' : ['A', 'B', 'C']},`
`{'Course' : ['X', 'Y', 'Z']}]`
	`[{'Roll': 1, 'Name': 'A', 'Course': 'X'},`
	`{'Roll': 2, 'Name': 'B', 'Course': 'Y'},`
	`{'Roll': 3, 'Name': 'C', 'Course': 'Z'}]`

You are asked to convert this data into another form called `record form (or row measure)`. This is also a list of dictionaries. Here, a dictionary element is composed of data values from each data columns of `tabular form`.

Write a generic python function `table_to_record` that can take data in the `tablular` form and return the data in `record form`. Please see a public test case for an example input and output.

The program validates the function `table_to_record` with different data inputs based on the test case number.

```
def table_to_record(table_db):
    # write function body
    #
    #
```

Suffix Code Block - Hidden

```
# test cases(key) and input test data
table_db = {
1: [{'Roll' : [1, 2, 3]}, {'Name' : ['A', 'B', 'C']}, {'Course' : ['X', 'Y', 'Z']}],
2: [{"A" : [1, 2, 3]}, {"B" : ["1", "2", "3"]}, {"C" : [1, 2.0, "abc"]}],
3: [{1 : [1, 2, 3]}, {2 : ["A", "B", "C"]}, {3 : ["X", "Y", "Z"]}],
4: [{"roll": [100, 200, 300]}, {"name": ["Joy", "Anand", "Sunita"]}, {"city": ["Delhi", "Kochi", "Pune"]}]
}

print(table_to_record(table_db[int(input())]))
```

Test Cases

Type	Input	`data`	Output
Public	`1`	`[{'Roll' : [1, 2, 3]}, {'Name' : ['A', 'B', 'C']}, {'Course' : ['X', 'Y', 'Z']}]`	`[{'Roll': 1, 'Name': 'A', 'Course': 'X'}, {'Roll': 2, 'Name': 'B', 'Course': 'Y'}, {'Roll': 3, 'Name': 'C', 'Course': 'Z'}]`
Private	2	`[{"A" : [1, 2, 3]}, {"B" : ["1", "2", "3"]}, {"C" : [1, 2.0, "abc"]}]`	`[{'A': 1, 'B': '1', 'C': 1}, {'A': 2, 'B': '2', 'C': 2.0}, {'A': 3, 'B': '3', 'C': 'abc'}]`
Private	3	`[{1 : [1, 2, 3]}, {2 : ['A', 'B', 'C']}, {3 : ['X', 'Y', 'Z']}]`	`[{1: 1, 2: 'A', 3: 'X'}, {1: 2, 2: 'B', 3: 'Y'}, {1: 3, 2: 'C', 3: 'Z'}]`
Private	4	`[{"roll": [100, 200, 300]}, {"name": ["Joy", "Anand", "Sunita"]}, {"city": ["Delhi", "Kochi", "Pune"]}]`	`[{'roll': 100, 'name': 'Joy', 'city': 'Delhi'}, {'roll': 200, 'name': 'Anand', 'city': 'Kochi'}, {'roll': 300, 'name': 'Sunita', 'city': 'Pune'}]`

Answer

```
def table_to_record(table_db):
    # creating a list of empty dictionaries
    record_db = []
    for i in range(len(table_db)):
        record_db.append(dict())

    # using list comprehension
    # record_db =[{} for idx in range(len(table_db))] # using list comprehension

    # adding value from each column of table_db to the dictionary element
    idx = 0
    for data in table_db:
        for key, value_list in data.items():
            for ele in value_list:
                record_db[idx][key] = ele
                idx += 1
            idx = 0
    return(record_db)
```

Problem 3

Write a program to accept a entity details from user and store in a dictionary of dictionary database say `data_table`.

A sample entity details can be represented as: `entity_dict = {"uid": entity_id, "a": a_value, "b": b_value,...}`.

The key (or field) `"uid"` of the dictionary `entity_dict` is mandatory and its value `entity_id` is always a unique positive integer. This field's value is used as a key in the `data_table` to store entity details record `entity_dict`.

In an entity record dictionary, all keys are of string type while values except `entity_id` can be either string or number.

Functions	Uses
`add_record(data_table, entity_dict)`	Adds an entity details `entity_dict` to the entity database `data_table` and return this modified `data_table`.
`search_record(data_table, entity_id)`	Returns entity details as a dictionary (similar to `entity_dict`), if entity `uid` is found, otherwise, it returns `None`.
`update_record(data_table, entity_id, key, value)`	Updates specific key's value of an entity record if found and return the updated `data_table`. If not record found, nothing is updated, return the `data_table` as it is.
`delete_record(data_table, entity_id)`	Deletes a specific entity record and return the modified `data_table`. If the record is not found, nothing is deleted, return the `data_table` as it is.

The database `data_table` is empty at the beginning and these functions are called in the same order as they are listed. Write a program to implement the body of these functions.

```
def add_record(data_table, entity_dict):
    # write function body
    #
    #
def search_record(data_table, entity_id):
    # write function body
    #
    #
def update_record(data_table, entity_id, key, value):
    # write function body
    #
    #
def delete_record(data_table, entity_id):
    # write function body
    #
    #
```

Prefix Code Block - Hidden

```
# uses json library
# convert an input dictionary string to dictionary object
import json
entity_dict = json.loads(input())
# accept entity_id as int input used in json dictionary string
entity_id = int(input())
```

Suffix Code Block - Visible

```
# create a blank dictionary
data_table = {}
# add a record
data_table = add_record(data_table, entity_dict)
print(data_table)
# search a record
print(search_record(data_table, entity_id))
# update a field in the existing record
data_table = update_record(data_table, entity_id,
list(data_table[entity_id].keys())[1], "X")
print(data_table)
# delete a record
data_table = delete_record(data_table, entity_id)
print(data_table)
```

Test Cases

Type	Input	Output
Public	{"uid": 101, "emp_name": "Hari", "emp_sal": 100000} 101	{101: {'uid': 101, 'emp_name': 'Hari', 'emp_sal': 100000}} {'uid': 101, 'emp_name': 'Hari', 'emp_sal': 100000} {101: {'uid': 101, 'emp_name': 'X', 'emp_sal': 100000}} {}
Private	{"uid": 1, "book_name": "Python", "book_author": "Guido van Rossum"} 1	{1: {'uid': 1, 'book_name': 'Python', 'book_author': 'Guido van Rossum'}} {'uid': 1, 'book_name': 'Python', 'book_author': 'Guido van Rossum'} {1: {'uid': 1, 'book_name': 'X', 'book_author': 'Guido van Rossum'}} {}
Private	{"uid": 1, "A":1, "B":1, "C":1, "D": 1} 1	{1: {'uid': 1, 'A': 1, 'B': 1, 'C': 1, 'D': 1}} {'uid': 1, 'A': 1, 'B': 1, 'C': 1, 'D': 1} {1: {'uid': 1, 'A': 'X', 'B': 1, 'C': 1, 'D': 1}} {}
Private	{"uid": 1, "A": 1, "B":1.0, "C":1, "D": "1"} 1	{1: {'uid': 1, 'A': 1, 'B': 1.0, 'C': 1, 'D': '1'}} {'uid': 1, 'A': 1, 'B': 1.0, 'C': 1, 'D': '1'} {1: {'uid': 1, 'A': 'X', 'B': 1.0, 'C': 1, 'D': '1'}} {}

Answer

```python
# Adds an entity details `entity_dict` to the entity database `data_table`
# and return this modified `data_table`.
def add_record(data_table, entity_dict):
    data_table[list(entity_dict.values())[0]] = entity_dict
    return data_table

# Returns entity details as a dictionary (similar to `entity_dict`), if entity `uid` is found,
# otherwise, it returns `None`.
def search_record(data_table, entity_id):
    if entity_id in data_table.keys():
        return data_table[entity_id]
    else:
        return None

# Updates specific key's value of an entity record if found and return the updated `data_table`.
# If not record found, nothing is updated, return the `data_table` as it is.

def update_record(data_table, entity_id, key, value):
    if entity_id in data_table.keys():
        data_table[entity_id][key] = value
    return data_table

```

```python
# Deletes a specific entity record and return the modified `data_table`.
# If the record is not found, nothing is deleted, return the `data_table` as it is.
def delete_record(data_table, entity_id):
    # using membership
    if entity_id in data_table:
        del data_table[entity_id]
    return data_table

    # using dictionary method pop()
    # data_table.pop(entity_id, None)
    # return data_table

    # Another approach
    # new_dict = {}
    # for key in data_table.keys():
    #     if key != entity_id:
    #         new_dict[key] = data_table[key]
    # return new_dict
```

Problem 4

Question

Given below are the list of dictionary methods and their uses.

Function	Uses
`min_dict_key(data)`	Returns the minimum key in the dictionary `data`
`max_dict_key(data)`	Returns the maximum key in the dictionary `data`
`min_value_dict_key(data)`	Returns the key corresponding to the minimum value in the dictionary `data`, if there are two items with equal values, the key to first item is returned
`max_value_dict_key(data)`	Returns the key corresponding to the maximum value in the dictionary `data`, if there are two items with equal values, the key to first item is returned
`sort_by_key(data, order="asc")`	Returns the list of tuples `[(key, value), (key value), ...]`. Each tuple in this list represents an item from the dictionary `data` sorted by its keys. The order of sorting is given by the parameter `order` which can be either `"asc"` or `"desc"` with `"asc"` being default value. if there are two items with equal values, their order is preserved while sorting the items.
`sort_by_value(data, order="asc")`	Returns the list of tuples `[(key, value), (key value), ...]`. Each tuple in this list represents an item from the dictionary `data` sorted by its values. The order of sorting is given by the parameter `order` which can be either `"asc"` or `"desc"` with `"asc"` being default value. if there are two items with equal values, their order is preserved while sorting the items.

Write the function body for these dictionary methods. The output is displayed per function call in the **suffix** code block. The program validates the these function with different data inputs based on the test case number.

Sample Input and Output

Input	`data`	Output
1	`{1:40, 2:39, 3:20, 4:33, 5:36}`	1 5 3 1 [(1, 40), (2, 39), (3, 20), (4, 33), (5, 36)] [(5, 36), (4, 33), (3, 20), (2, 39), (1, 40)] [(3, 20), (4, 33), (5, 36), (2, 39), (1, 40)] [(1, 40), (2, 39), (5, 36), (4, 33), (3, 20)]
2	`{4:40, 5:39, 3:20, 6:33, 2:36}`	2 6 3 4 [(2, 36), (3, 20), (4, 40), (5, 39), (6, 33)] [(6, 33), (5, 39), (4, 40), (3, 20), (2, 36)] [(3, 20), (6, 33), (2, 36), (5, 39), (4, 40)] [(4, 40), (5, 39), (2, 36), (6, 33), (3, 20)]

```python
def min_dict_key(data):
    # write function body
    #
    #
def max_dict_key(data):
    # write function body
    #
    #
def min_value_dict_key(data):
    # write function body
    #
    #
def max_value_dict_key(data):
    # write function body
    #
    #
def sort_by_key(data, order="asc"):
    # write function body
    #
    #
def sort_by_value(data, order="asc"):
    # write function body
    #
    #
```

Prefix Code Block - Hidden

```
# test cases(key) and input test data
table = {
    1: {1:40, 2:39, 3:20, 4:33, 5:36},
    2: {4:40, 5:39, 3:20, 6:33, 2:36},
    3: {1:40, 2:20, 3:20, 4:36, 5:36},
    4: {5:40, 4:20, 3:20, 2:36, 1:36},
    5: {5:5, 4:4, 3:3, 2:2, 1:1},
    6: {1:5, 2:4, 3:3, 4:2, 5:1}
}
data = table[int(input())]
```

Suffix Code Block - Visible

```
print(min_dict_key(data))
print(max_dict_key(data))
print(min_value_dict_key(data))
print(max_value_dict_key(data))
print(sort_by_key(data, "asc"))
print(sort_by_key(data, "desc"))
print(sort_by_value(data, "asc"))
print(sort_by_value(data, "desc"))
```

Public

Input	`data`	Output
1	`{1:40, 2:39, 3:20, 4:33, 5:36}`	1 5 3 1 [(1, 40), (2, 39), (3, 20), (4, 33), (5, 36)] [(5, 36), (4, 33), (3, 20), (2, 39), (1, 40)] [(3, 20), (4, 33), (5, 36), (2, 39), (1, 40)] [(1, 40), (2, 39), (5, 36), (4, 33), (3, 20)]
2	`{4:40, 5:39, 3:20, 6:33, 2:36}`	2 6 3 4 [(2, 36), (3, 20), (4, 40), (5, 39), (6, 33)] [(6, 33), (5, 39), (4, 40), (3, 20), (2, 36)] [(3, 20), (6, 33), (2, 36), (5, 39), (4, 40)] [(4, 40), (5, 39), (2, 36), (6, 33), (3, 20)]

Private

Input	`data`	Program Output
3	`{1:40, 2:20, 3:20, 4:36, 5:36}`	1 5 2 1 [(1, 40), (2, 20), (3, 20), (4, 36), (5, 36)] [(5, 36), (4, 36), (3, 20), (2, 20), (1, 40)] [(2, 20), (3, 20), (4, 36), (5, 36), (1, 40)] [(1, 40), (4, 36), (5, 36), (2, 20), (3, 20)]
4	`{5:40, 4:20, 3:20, 2:36, 1:36}`	1 5 4 5 [(1, 36), (2, 36), (3, 20), (4, 20), (5, 40)] [(5, 40), (4, 20), (3, 20), (2, 36), (1, 36)] [(4, 20), (3, 20), (2, 36), (1, 36), (5, 40)] [(5, 40), (2, 36), (1, 36), (4, 20), (3, 20)]
5	`{5:5, 4:4, 3:3, 2:2, 1:1}`	1 5 1 5 [(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)] [(5, 5), (4, 4), (3, 3), (2, 2), (1, 1)] [(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)] [(5, 5), (4, 4), (3, 3), (2, 2), (1, 1)]
6	`{1:5, 2:4, 3:3, 4:2, 5:1}`	1 5 5 1 [(1, 5), (2, 4), (3, 3), (4, 2), (5, 1)] [(5, 1), (4, 2), (3, 3), (2, 4), (1, 5)] [(5, 1), (4, 2), (3, 3), (2, 4), (1, 5)] [(1, 5), (2, 4), (3, 3), (4, 2), (5, 1)]

Answer

```python
# finds the item with minium key in a given dictionary
def min_dict_key(data):
    min_key, min_key_value = list(data.items())[0]
    for key, value in data.items():
        # store item with mininmum key into min_key
        if min_key > key:
            min_key = key
    return min_key

# finds the item with maximum key in a given dictionary
def max_dict_key(data):
    max_key, max_key_value = list(data.items())[0]
    for key, value in data.items():
        # store item with maxium key into min_key
```

```
        if max_key < key:
            max_key = key
    return max_key

# finds the item key with minimum value in a given dictionary
def min_value_dict_key(data):
    min_value_key, min_value = list(data.items())[0]
    for key, value in data.items():
        # store item with mininmum key into min_value_key and min_value
        if min_value > value:
            min_value = value
            min_value_key = key
    return min_value_key

# finds the item key with maxium value in a given dictionary
def max_value_dict_key(data):
    max_value_key, max_value = list(data.items())[0]
    for key, value in data.items():
        # store item with maximum value into max_value_key and max_value
        if max_value < value:
            max_value = value
            max_value_key = key
    return max_value_key

# return a list of tuples where tuples are sorted itmes of data dictionary
by key
def sort_by_key(data, order="asc"):
    data = dict(data)
    sorted_dict = {}
    sorted_list = []
    n = len(data.keys())
    for _ in range(n):
        # iteratively removing item from data dictionary with maximum key
        # and appending to a list as tuple, it gives descending items list
by key
        if order == "desc":
            max_key = max_dict_key(data)
            sorted_dict[max_key] = data[max_key]
            sorted_list.append((max_key, data[max_key]))
            del data[max_key]
        # iteratively removing item from data dictionary with minimum key
        # and appending to a list as tuple, it gives ascending items list by
key
        elif order == "asc":
            min_key = min_dict_key(data)
            sorted_dict[min_key] = data[min_key]
            sorted_list.append((min_key, data[min_key]))
            del data[min_key]
    #return sorted_dict # returns sorted dictionary, for python version >=
3.7
    return sorted_list # returns sorted list

# return a list of tuples where tuples are sorted itmes of data dictionary
by value
def sort_by_value(data, order="asc"):
    data = dict(data)
    sorted_dict = {}
    sorted_list = []
```

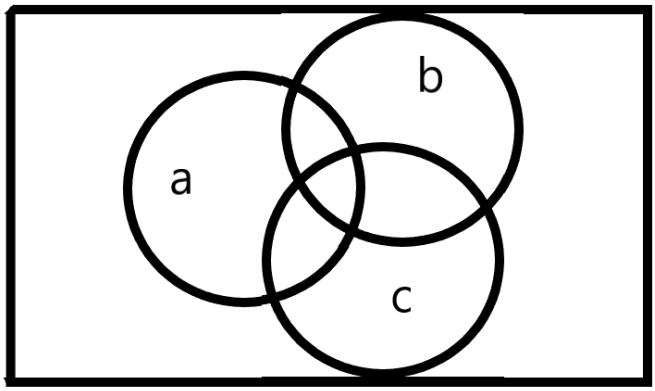
```
    n = len(data.keys())
    for _ in range(n):
        # iteratively removing item from data dictionary with maximum key
        # and appending to a list as tuple, it gives descending items list
by value
        if order == "desc":
            max_value_key = max_value_dict_key(data)
            sorted_dict[max_value_key] = data[max_value_key]
            sorted_list.append((max_value_key, data[max_value_key]))
            del data[max_value_key]
        # iteratively removing item from data dictionary with minimum key
        # and appending to a list as tuple, it gives ascending items list by
value
        elif order == "asc":
            min_value_key = min_value_dict_key(data)
            sorted_dict[min_value_key] = data[min_value_key]
            sorted_list.append((min_value_key, data[min_value_key]))
            del data[min_value_key]
    #return sorted_dict # returns sorted dictionary, for python version >=
3.7
    return sorted_list # returns sorted list
```

Problem 5

Question

Among a group of students, some play sport `a`, some play `b`, and some play `c`.

Id	Description	Set Symbol	Count of Students
1	played a	`a`	`n(a)`
2	played b	`b`	`n(b)`
3	played c	`c`	`n(c)`
4	played both a and b	`a_and_b`	`n(a_and_b)`
5	played both b and c	`a_and_c`	`n(a_and_b)`
6	played a and c	`b_and_c`	`n(b_and_c)`
7	played a and b and c	`a_and_b_and_c`	`n(a_and_b_and_c)`
8	played any three sports (a set of all students)	`a_or_b_or_c`	`n(a_or_b_or_c)`
9	played only a and b not c	`only_a_and_b_not_c`	`n(only_a_and_b_not_c)`
10	played only a and c not b	`only_a_and_c_not_b`	`n(only_a_and_c_not_b)`
11	played only b and c not a	`only_b_and_c_not_a`	`n(only_b_and_c_not_a)`
12	played only a	`only_a`	`n(only_a)`
13	played only b	`only_b`	`n(only_b)`
14	played only c	`only_c`	`n(only_c)`



The probability of an event `e` on given sample space `s` is defined as :

$$P(a) = \frac{n(a)}{n(s)}$$

where `n(a)` is the count of elements in the set `a` and `n(s)` is the total number of elements in the sample space `s`. In above table, the sample space `s` is represented by the set `a_or_b_or_c`. It gives the total number of students in the group.

Write a program to print the count of elements and probability of all 14 sets given in the above table in the sequence they are listed. The input is three sequence, one for each set `a`, `b` and `c` in the order, see the **suffix** code block. The values in each sequence is separated by comma.

The **output** is the element count and the probability separated by comma for a set. The output for each of the 14 sets are given on the newline. The probability value should be rounded up to 1 decimal point. Please see a public test case for example.

You can use library functions / operators or write custom set functions to achieve the task.

Note: Elements in each set represents a unique student, it can be a number or a string.

Prefix Code Block - Hidden

```
# accepts two sets and returns its intersection
def set_intersection(x, y):
    common_set = set()
    # iterate through the elements in the smaller set and check common elements in both set x and y
    if len(x) <= len(y):
        for item in x:
            if item in y:
                common_set.add(item)
    else:
        for item in y:
            if item in x:
                common_set.add(item)
    return common_set

# # accepts two sets and returns its intersection using union of set
# def set_intersection(x, y):
# # iterate throught the elements in union set and check common elements in both set x and y
#     union_set = set(list(x) + list(y))
#     common_set = set()
#     for item in union_set:
#         if (item in x) and (item in y):
#             common_set.add(item)
#     return common_set

# accepts two sets and returns its union
def set_union(x, y):
    # converts set to list, appends and returns as a set
    return set(list(x) + list(y))

# accepts two sets and returns difference of second set from first set
def set_difference(x, y):
    asymmetric_diff_set = set()
    # adds elements which are in set x and not in y
    for item in x:
        if item not in y:
            asymmetric_diff_set.add(item)
    return asymmetric_diff_set

```

```
# accepts two sets and returns its symmetric set difference
def set_symmetric_difference(x, y):
    # create a union set from input sets
    union_set = set(list(x) + list(y))
    symmetric_diff_set = set()
    for item in union_set:
        # adds elements which are in set x and not in y
        if item in x and item not in y:
            symmetric_diff_set.add(item)
        # adds elements which are not in set x but in y
        elif item not in x and item in y:
            symmetric_diff_set.add(item)
    return symmetric_diff_set

# custom functions defined for set operations, you may use library functions instead of these
# accepts multiple parameters and returns its intersection
def set_intersection_generic(*args): # *args used to accept a list of parameters of variable length
    union_set = set_union_generic(*args) # args[0] -> first param, *args[1] -> second param and so on
    common_set = set()
    for item in union_set:
        flag = True
        # if an element is present in all sets, add it to common set
        for argument in args:
            if item not in argument:
                flag = False
        if flag:
            common_set.add(item)
    return common_set

# accepts multiple parameters and returns its union
def set_union_generic(*args): # *args used to accept a list of parameters of variable length
    union_list = []
    # make a large list with elements from all input sets
    for argument in args: # args[0] -> first param, *args[1] -> second param and so on
        union_list = union_list + list(argument)
    # return a set of this large list
    return set(union_list)
```

Suffix Code Block - Visible

```
a = set(input().split(","))
b = set(input().split(","))
c = set(input().split(","))
```

Test Cases

type	Input	Output
Public	1,4,7,10 1,3,5,7,9,10 1,2,4,6,8,10	4,0.4 6,0.6 6,0.6 3,0.3 2,0.2 3,0.3 2,0.2 10,1.0 1,0.1 1,0.1 0,0.0 0,0.0 3,0.3 3,0.3
Private	a,b,c,f,h b,c,f,g,h,j a,c,e,f,g,k	5,0.6 6,0.7 6,0.7 4,0.4 3,0.3 3,0.3 2,0.2 9,1.0 2,0.2 1,0.1 1,0.1 0,0.0 1,0.1 2,0.2
Private	1,1,4,4,5,5 1,1,5,5,9,10 1,1,4,4,5,5	3,0.6 4,0.8 3,0.6 2,0.4 2,0.4 3,0.6 2,0.4 5,1.0 0,0.0 1,0.2 0,0.0 0,0.0 2,0.4 0,0.0

Answer#1

```
# using library functions
# finding the expected set using set operations on sets a, b, and c
a_and_b = a & b
b_and_c = b & c
a_and_c = a & c
a_and_b_and_c = a & b & c
a_or_b_or_c = a | b | c
only_a_and_b_not_c = (a & b) - (a & b & c)
only_a_and_c_not_b = (a & c) - (a & b & c)
only_b_and_c_not_a = (b & c) - (a & b & c)
only_a = ((a - b) - c)
only_b = ((b - a) - c)
only_c = ((c - a) - b)

# dictionary of all sets
set_dict = {
    "a": a,
    "b": b,
    "c": c,
    "a_and_b": a_and_b,
    "b_and_c": b_and_c,
    "a_and_c": a_and_c,
    "a_and_b_and_c": a_and_b_and_c,
    "a_or_b_or_c": a_or_b_or_c,
    "only_a_and_b_not_c": only_a_and_b_not_c ,
    "only_a_and_c_not_b": only_a_and_c_not_b ,
    "only_b_and_c_not_a": only_b_and_c_not_a,
    "only_a": only_a,
    "only_b": only_b,
    "only_c": only_c
    }

# printing the length and probability of event represented by each set in
set_dict
for set_name, set_val in set_dict.items():
    #print(f"n({set_name}) = {len(set_val)}, prob({set_name}) =
{len(set_val)/len(a_or_b_or_c):.1f}")
    print(f"{len(set_val)},{len(set_val)/len(a_or_b_or_c):.1f}")
```

Alternative Answer#2

```
# using library functions
# finding the expected set using set operations on sets a, b, and c
a_and_b = a & b
b_and_c = b & c
a_and_c = a & c
a_and_b_and_c = a & b & c
a_or_b_or_c = a | b | c
only_a_and_b_not_c = (a & b) - (a & b & c)
only_a_and_c_not_b = (a & c) - (a & b & c)
only_b_and_c_not_a = (b & c) - (a & b & c)
only_a = ((a - b) - c)
only_b = ((b - a) - c)
only_c = ((c - a) - b)
```

```
14
15  # dictinary of length of each set
16  count_dict = {
17      "a": len(a),
18      "b": len(b),
19      "c": len(c),
20      "a_and_b": len(a_and_b),
21      "b_and_c": len(b_and_c),
22      "a_and_c": len(a_and_c),
23      "a_and_b_and_c": len(a_and_b_and_c),
24      "a_or_b_or_c": len(a) + len(b) + len(c) - len(a_and_b) - len(b_and_c) -
    len(a_and_c) + len(a_and_b_and_c),
25      "only_a_and_b_not_c": len(a_and_b) - len(a_and_b_and_c) ,
26      "only_a_and_c_not_b": len(a_and_c) - len(a_and_b_and_c) ,
27      "only_b_and_c_not_a": len(b_and_c) - len(a_and_b_and_c),
28      "only_a": len(a) - len(a_and_b) - len(a_and_c) + len(a_and_b_and_c),
29      "only_b": len(b) - len(a_and_b) - len(b_and_c) + len(a_and_b_and_c),
30      "only_c": len(c) - len(a_and_c) - len(b_and_c) + len(a_and_b_and_c)
31      }
32
33  # printing the length and probability of event represented by each set in
    count_dict
34  for set_name, set_size in count_dict.items():
35      #print(f"n({set_name}) = {set_size}, prob({set_name}) =
    {set_size/len(a_or_b_or_c):.1f}")
36      print(f"{set_size},{set_size/len(a_or_b_or_c):.1f}")
```

Alternative Answer#3

```
1   # Using custom functions defined in the suffix code block
2   # finding the expected set using custom set operations on sets a, b, and c
3   a_and_b = set_intersection(a, b)
4   b_and_c = set_intersection(b, c)
5   a_and_c = set_intersection(a, c)
6   a_and_b_and_c = set_intersection_generic(a, b, c)
7   a_or_b_or_c = set_union_generic(a, b, c)
8   only_a_and_b_not_c = set_difference(a_and_b, a_and_b_and_c)
9   only_a_and_c_not_b = set_difference(a_and_c, a_and_b_and_c)
10  only_b_and_c_not_a = set_difference(b_and_c, a_and_b_and_c)
11  only_a = set_difference(set_difference(a, b), c)
12  only_b = set_difference(set_difference(b, a), c)
13  only_c = set_difference(set_difference(c, a), b)
14
15  # dictionary of all sets
16  set_dict = {
17      "a": a,
18      "b": b,
19      "c": c,
20      "a_and_b": a_and_b,
21      "b_and_c": b_and_c,
22      "a_and_c": a_and_c,
23      "a_and_b_and_c": a_and_b_and_c,
24      "a_or_b_or_c": a_or_b_or_c,
25      "only_a_and_b_not_c": only_a_and_b_not_c ,
26      "only_a_and_c_not_b": only_a_and_c_not_b ,
27      "only_b_and_c_not_a": only_b_and_c_not_a,
28      "only_a": only_a,
```

```
    "only_b": only_b,
    "only_c": only_c
    }

# printing the length and probability of event represented by each set in set_dict
for set_name, set_val in set_dict.items():
    #print(f"n({set_name}) = {len(set_val)}, prob({set_name}) = {len(set_val)/len(a_or_b_or_c):.1f}")
    print(f"{len(set_val)},{len(set_val)/len(a_or_b_or_c):.1f}")
```

Alternative Answer#4

```
# Using custom functions defined in the suffix code block
# finding the expected set using custom set operations on sets a, b, and c
a_and_b = set_intersection(a, b)
b_and_c = set_intersection(b, c)
a_and_c = set_intersection(a, c)
a_and_b_and_c = set_intersection_generic(a, b, c)
a_or_b_or_c = set_union_generic(a, b, c)
only_a_and_b_not_c = set_difference(a_and_b, a_and_b_and_c)
only_a_and_c_not_b = set_difference(a_and_c, a_and_b_and_c)
only_b_and_c_not_a = set_difference(b_and_c, a_and_b_and_c)
only_a = set_difference(set_difference(a, b), c)
only_b = set_difference(set_difference(b, a), c)
only_c = set_difference(set_difference(c, a), b)

# dictinary of length of each set
count_dict = {
    "a": len(a),
    "b": len(b),
    "c": len(c),
    "a_and_b": len(a_and_b),
    "b_and_c": len(b_and_c),
    "a_and_c": len(a_and_c),
    "a_and_b_and_c": len(a_and_b_and_c),
    "a_or_b_or_c": len(a) + len(b) + len(c) - len(a_and_b) - len(b_and_c) - len(a_and_c) + len(a_and_b_and_c),
    "only_a_and_b_not_c": len(a_and_b) - len(a_and_b_and_c) ,
    "only_a_and_c_not_b": len(a_and_c) - len(a_and_b_and_c) ,
    "only_b_and_c_not_a": len(b_and_c) - len(a_and_b_and_c),
    "only_a": len(a) - len(a_and_b) - len(a_and_c) + len(a_and_b_and_c),
    "only_b": len(b) - len(a_and_b) - len(b_and_c) + len(a_and_b_and_c),
    "only_c": len(c) - len(a_and_c) - len(b_and_c) + len(a_and_b_and_c)
    }

# printing the length and probability of event represented by each set in count_dict
for set_name, set_size in count_dict.items():
    #print(f"n({set_name}) = {set_size}, prob({set_name}) = {set_size/len(a_or_b_or_c):.1f}")
    print(f"{set_size},{set_size/len(a_or_b_or_c):.1f}")
```