**1. Extend nested list by adding the sublist**

input:

list1 = ["a", "b", ["c", ["d", "e", ["f", "g"], "k"], "l"], "m", "n"]

# sub list to add

sub\_list = ["h", "i", "j"]

Expected output:

['a', 'b', ['c', ['d', 'e', ['f', 'g', 'h', 'i', 'j'], 'k'], 'l'], 'm', 'n']

**2. Convert two lists into a dictionary**

Input:

keys = ['Ten', 'Twenty', 'Thirty']

values = [10, 20, 30]

Expected Output:

{'Ten': 10, 'Twenty': 20, 'Thirty': 30}

**3. Delete a list of keys from a dictionary**

Input:

sample\_dict = {

"name": "Kelly",

"age": 25,

"salary": 8000,

"city": "New york"

}

# Keys to remove

keys = ["name", "salary"]

Expected Output:

{'city': 'New york', 'age': 25}

**4. Rename key of a dictionary**

Input:

sample\_dict = {

"name": "Kelly",

"age":25,

"salary": 8000,

"city": "New york"

}

Expected Output:

{'name': 'Kelly', 'age': 25, 'salary': 8000, 'location': 'New york'}

**5. Get the key of a minimum value from the following dictionary**

Input:

sample\_dict = {

'Physics': 82,

'Math': 65,

'history': 75

}

Expected output:

Math

**6.Read a file and pass it to a function which checks for the given input string ( case insensitive check). If the given string is found, the function should return True. Create another function which will read the input file s**

**and scan it to get the count of each word in it. After the process is completed print each word along with the number of times it occurred in the file.**

**7. Explore the difference between iterrows(), itertuples(), apply(), map()**

Analyze the time taken for above operations using one wider dataset(large no of columns) and one taller dataset(Large no. of rows)

Read dataset as ***pandas dataframe***

**8. Explore lambda functions in python.**

Explore the use cases of ‘for loops’ and lambda functions

Write a program using lambda with map, filter, reduce functions.

**9. Explore the difference between sorted and sort functions.**

Write example programs using sort and sorted function and interpret.

**10 . Explore the OOPS concepts.**

Create a Person class with attributes name and age. Implement a method that prints a greeting with the person's name and age.

Create a Shape class with a method area. Implement subclasses Circle and Rectangle with their own implementations of the area method.

**11. Write a python program that randomly generates a 5 digit number. The user has a maximum of 5 tries to guess the randomly generated number.**

* If any of the digits guessed are wrong, print "A" to indicate wrong guess.
* If the guessed digit is in the randomly generated numbers but in the wrong position, print "B".
* If the digit guessed is both the correct value and position, print "C".

**12.**

data = [{"roll\_no": 1, "name": "John", "games": ["cricket", "football"],

"marks": {"maths": 90, "science": 93, "history": 81}, "rank": 1},

{"roll\_no": 2, "name": "Mick", "games": ["football", "hockey"],

"marks": {"maths": 95, "science": 86, "cs": 70}, "rank": 2},

{"roll\_no": 3, "name": "June", "games": ["badminton", None],

"marks": {"maths": 92, "science": 92, "geography": 78}, "rank": 3},

{"roll\_no": 4, "name": "Adam", "games": ["soccer", "badminton"],

"marks": {"maths": 86, "science": 91, "cs": 82}, "rank": 4},

{"roll\_no": 5, "name": "Robb", "games": ["cricket", None],

"marks": {"maths": 88, "science": 90, "economics": 84}, "rank": 5},

{"roll\_no": 6, "name": "Arya", "games": ["football", "hockey"],

"marks": {"maths": 89, "science": 88, "history": 97}, "rank": 6}

]

* Get the details of students studied ‘cs’ subject using lambda function
* Update the student ranks by calculating the sum of three times the marks in maths, two times the marks in science, and leaving the marks in other subjects as they are. Insert the key percentage, which represents the percentage of sum of marks calculated obtained by each student. Remove null values in the games.
* Create a new dataframe containing the student's name, their percentage, their previous rank, their current rank, along with the change in ranks which is in sorted order by new rank. If the rank has decreased or increased by one, the value should be -1 or 1, respectively. If they are in the same position, the value should be a hyphen.

**13.** Get the following 3 parameters from a simple flask API. Each time the API is called, it is considered as a new refresh.

**new row count** -> Any Positive integers

**metric threshold ->** Between 0 and 100, Unit %

**Window size ->** Number of refreshes you have to consider to find row count deviation**,**

Limit1 to 200 (Integers)

Assume Base Row count of a dataset is 1000. You have to show row count deviation in percentage(row count drift) by comparing the average value of row counts of previous refreshes[not the values outside the limit of window size] and find whether it violates metrics threshold. If the window size is changed, average computation should change accordingly.

Create required tables in your local postgres DB store the necessary parameters to solve the above problem.

Refer this document for understanding better:

[Row-COUNT](https://docs.google.com/document/d/1Emy2gCLD-EuE94mC1h12CjAyXRgowitTtO3Zu6wdpp4/edit)

14. Create the below tables and insert the records using sqlalchemy only.

**attribute\_issue\_count**

Table schema (

issue\_count\_id serial Primary key,

tenant\_id integer NOT NULL,

integration\_id integer NOT NULL,

meta\_data\_id integer NOT NULL,

created\_month timestamp NOT NULL,

issue\_count integer NOT NULL,

issue\_details json NOT NULL,

data\_set\_id integer NOT NULL,

env\_id integer NOT NULL

)

Records to be inserted:

(1664,2566,322384,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44961": 1}' as json),55396,1485),

(1664,2566,322386,TIMESTAMP '2023-11-01 00:00:00.000000',2,cast('{"44961": 1, "44982": 1}' as json),55396,1485),

(1664,2566,322388,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44961": 1}' as json),55396,1485),

(1664,2566,322382,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44967": 1}' as json),55396,1485),

(1664,2566,322383,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44961": 1}' as json),55397,1485),

(1664,2566,322385,TIMESTAMP '2023-11-01 00:00:00.000000',2,cast('{"44961": 1, "44982": 1}' as json),55397,1485),

(1664,2566,322387,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44961": 1}' as json),55397,1485),

(1664,2566,322393,TIMESTAMP '2023-11-01 00:00:00.000000',1,cast('{"44967": 1}' as json),55397,1485);

**dataset\_issue\_count**

Table Schema(

issue\_count\_id serial Primary key,

tenant\_id integer NOT NULL,

integration\_id integer NOT NULL,

data\_set\_id integer NOT NULL,

created\_month timestamp NOT NULL,

issue\_count integer NOT NULL,

issue\_details json NOT NULL,

env\_id integer NOT NULL

);

Records to be inserted:

(1664,2566,55396,TIMESTAMP '2023-11-01 00:00:00.000000',3,cast('{"44961": 1, "44967": 1, "44982": 1}' as json),1485),

(1664,2566,55397,TIMESTAMP '2023-11-01 00:00:00.000000',3,cast('{"44961": 1, "44967": 1, "44982": 1}' as json),1485);

**datasource\_issue\_count**

Table Schema(

issue\_count\_id serial Primary key,

tenant\_id integer NOT NULL,

env\_id integer NOT NULL,

integration\_id integer NOT NULL,

created\_month timestamp NOT NULL,

issue\_count\_dataset\_level integer NOT NULL,

issue\_details\_dataset\_level json NOT NULL,

issue\_count\_attribute\_level integer NOT NULL,

issue\_details\_attribute\_level json NOT NULL

);

Write a python program to store total issue count and aggregated issue details of **attribute\_issue\_count and**  **dataset\_issue\_count tables** in **datasource\_issue\_count table**  appropriate columns [issue\_count\_dataset\_level, issue\_details\_dataset\_level, issue\_count\_attribute\_level, issue\_details\_attribute\_level]

The above **datasource\_issue\_count** table will have entry for each integration\_id . So when writing solution, handle it for multiple integration\_id

Expected output of the **datasource\_issue\_count** table

(1664,1485,2566,TIMESTAMP '2023-11-01 00:00:00.000000',6,cast('{"44982": 2, "44961": 2, "44967": 2}' as json),10,cast('{"44982": 2, "44961": 6, "44967": 2}' as json));

**15.** *[Task 14 needs to be completed for this task]*

Write a program to give the below dict as output from the data of attribute\_issue\_count, dataset\_issue\_count, datasource\_issue\_count tables.

Expected Output: {"name":"packed-chart","children":[{"env\_id":1485,"children":[{"integration\_id":2566,"children":[{"data\_set\_id":55396,"children":[{"meta\_data\_id":322386,"size":2},{"meta\_data\_id":322382,"size":1},{"meta\_data\_id":322388,"size":1},{"meta\_data\_id":322384,"size":1}],"value":3},{"data\_set\_id":55397,"children":[{"meta\_data\_id":322385,"size":2},{"meta\_data\_id":322383,"size":1},{"meta\_data\_id":322393,"size":1},{"meta\_data\_id":322387,"size":1}],"value":3}],"value":16}],"value":16}]}

Value, Size in the output represents count of the issues in the above three tables.

The solution should support multiple env\_id, multiple integration\_id

**16. [OPTIONAL]**

Create a new table in BigQuery with the below query

*CREATE TABLE `project\_name.dataset\_name.table\_name`*

*(*

*col1 INT64 NOT NULL,*

*col2 STRUCT<num1 INT64, rec1 STRUCT<str1 STRING, rec2 STRUCT<num2 NUMERIC, rec3 STRUCT<date1 TIMESTAMP, rec4 STRUCT<str2 STRING, rec5 STRUCT<date2 TIMESTAMP, num3 NUMERIC>>>>>>,*

*col3 STRUCT<num1 INT64, rec11 STRUCT<rec12 STRUCT<str12 STRING(40), rec13 STRUCT<arr1 ARRAY<INT64>, rec14 STRUCT<str23 STRING(45), rec15 STRUCT<str24 STRING(43)>>>>>>,*

*stop\_on\_arr STRUCT<test\_int1 INT64, test\_rec1 STRUCT<test\_int2 INT64, test\_arr1 ARRAY<STRUCT<line1 INT64, line2 INT64>>>>*

*);*

* Insert 5 records for the above table with null values for some keys in the nested column.
* Read the above table as spark dataframe.
* If a nested json column is identified in the dataframe, flatten all the nested columns in the dataframe.

For example:

*Columns before flattening: (2 columns)*

*Col1(Struct) => Col1.1(Struct) => Col1.1.1(Int)*

*=> Col1.2(Int)*

*Col2(Int)*

*Columns after flattening: (3 columns)*

*Col1$Col1.1$Col1.1.1(Int)*

*Col1$Col1.2(Int)*

*Col2(Int)*Identify null values in the flattened columns.

* Create a list of nested dictionaries of any nested column in the flattened dataframe, each row of dataframe will be the dictionary in the list

For example:

The result list of dict should be like

*[{*

*"Col1": {*

*"Col1.1": {*

*"Col1.1.1": 5*

*},*

*"Col1.2": 10*

*}*

*},*

*{2nd row of dataframe}...]*

**Note:**

***Use SQL alchemy for all the interactions with db.***

***Use Spark when any dataframe is involved***