BACKGROUND

You need to summarize the data in the metric files reported daily to help the game designers to decide which level should be updated. To help you do that, you need an application that can show you the data summary within the defined time range on a running system quickly. Note that in the real world, the files count and the data inside it might be in the millions.

- Input parameters:
    - -d --directory. Required
      directory path, the directory contains single type of file, it can be csv or json
        - 1 file contains metrics usage for 1 day, the row is sorted by timestamp ascending (the oldest is at the top)
        - There is no guarantee that there must be a file for each day, that means there can be a skipped day, for example we have data for 1 January and 3 January, but no data for 2 January. So you can't deduce the date on the file by counting the files
    - -t --type. Required
      The type of the input files, supported format: json and csv
    - --startTime. Required
      The starting time to scan the data in the format of rfc3339, inclusive
    - --endTime. Required
      The ending time to scan the data in the format of rfc3339, exclusive
    - --outputFileType. Optional. The default is json.
      The output type of the summary, supported value: json and yaml
    - --outputFileName. Optional. The default is out.json or out.yaml depending on the outputType.
      If the user gives this value, it will save the file using the input value.
- Process:
    - It will parse the files between time range
    - It will calculate the summary of the content inside the files
- Output:
    - It will display the result in the console, and
    - It will write the result in a file, either with json or yaml format based on the user input

Data example

File structure for json file.

```
root
├── 01-jan.json
├── 03-jan.json
├── 04-jan.json
├── 07-jan.json
├── 09-jan.json
├── 10-jan.json
├── 11-jan.json
├── 12-jan.json
└── 13-jan.json
```

File structure for csv

```
root
├── 01-jan.csv
├── 03-jan.csv
├── 04-jan.csv
├── 07-jan.csv
├── 09-jan.csv
├── 10-jan.csv
├── 11-jan.csv
├── 12-jan.csv
└── 13-jan.csv
```

Json file example

```
[
 {
   "timestamp": "2022-01-01T00:00:50.52Z",
   "level_name": "lobby_screen",
   "value": 73
 },
 {
   "timestamp": "2022-01-01T00:23:51.21Z",
   "level_name": "level1",
   "value": 126
 },
 {
   "timestamp": "2022-01-01T23:04:12.55Z",
   "level_name": "lobby_screen",
   "value": 88
 },
 ...
]
```

This is the example content of a single json file. There can be thousands of these lines. The data is sorted by timestamp.

Csv file example

```
timestamp,level_name,value
2022-01-01T00:00:50.52Z,lobby_screen,73
2022-01-01T00:23:51.21Z,level1,126
2022-01-01T23:04:12.55Z,lobby_screen,88
```

Start Time example
2022-01-02T00:00:00Z or
2022-01-02T00:00:00+07:00

End Time example
2022-01-03T23:59:59Z or
2022-01-03T23:59:59+07:00

The summary:

- The summary should group all of the data based on the level_name
- It will sum all the data for each level_name

Summary example in json

```
[
{
  "level_name": "lobby_screen",
  "total_value": 161
},
{
  "level_name": "level1",
  "total_value": 126
 }
]
```

Summary example in yaml

```
---
- level_name: lobby_screen
 total_value: 161
- level_name: level1
 total_value: 126
```

Here are the constraints of the app:

- Users of the app needs to specify the directory containing metric files with “-d” or “--directory” option
- Users of the app needs to specify the metric file type with “-t” or “--type” option
- Users of the app needs to specify the start time of the time range using “--startTime” option
- Users of the app needs to specify the end time of the time range using “--endTime” option
- Users of the app may specify the output file type using the “--outputFileType” option
- Users of the app may specify the output file name using the “--outputFileName” option.
- The app will create a new result file, each time it runs(overwrite if the file already exists). The file will be inside the current directory, where the app is called.
- Metric file and the time range parameter are in the same (current) year, starting from January 1st.
- The timestamp of data in input files (json/csv) is using UTC format Zero time-zone.
- The data may not be stored sequentially based on time.
- The app should not process all files. In the real world, the number of data and files might be huge. Devise a way to open and read **only** relevant files, because we want the application to be efficient. If not possible, at least only process first data from out-of-time-range files.
- You may use any language of your choice ***Note: if you apply for a language-specific position (e.g. C++), use that language.***
- Create a fully object-oriented application
- Write unit / functional tests for the app
- The app should be a CLI tool
- The app should display the result in the command prompt / terminal
- The app should produce the result file according to the input parameter