CSC 369: Distributed Computing
Spring 2020
Lab 4 Specification

Due Date: Monday, Sunday, May 10, 2020, 11:59pm.
Note: :Lab 5 will come out on Monday, May 4. We will, therefore, have minimal grace period for Lab 4. **Lab 4** now has the status of a **mini-project.** This means it will run in parallel to at least one other individual lab (Lab 5), possibly two labs (Lab 6 may be assigned on May 9).

Note 2: The Appendix A and the Examples Section of the Specification will be released as a separate document.

Teams. This lab is for teams of two students each. Team assignments were made during the Monday, April 27 lab period.

Submission. Submit using **`handin`** on the **`unixN.csc.calpoly.edu`** system. The submission command is

```
$handin dekhtyar lab04 <files>
```

ONE submission per team, please! All your program files must contain the list of all team members, regardless of who is the actual author of the file.

Application: You are building `covidTracker.py`: a Python3 MongoDB-powered analytical tool for tracking COVID spread in the US. This program will run in command line, take as input a description of the information needs of a data analyst, and produce an HTML report containing tables and graphs of the desired analytics.

Input Data. Your program shall acquire data from the following two datasets:

Source	File	Data
https://covidtracking.com/api	/api/v1/states/daily.json	COVID-19 data for US States
New York Times COVID dataset	us-counties.csv	County-level COVID-19 data

The first dataset is the current version of the data we were using in **Labs 1--3.** It provides rich state-by-state daily statistics. The second dataset contains county-by-county data. This dataset has fewer features but better granularity.

Running the application. Your program shall be run using the following command

```
$ python3 covidTracker.py -auth <authentication file> -config <config file>
```

Both parameters are optional. Default values are:

Authentication file: **`credentials.json`**
Config file: **`trackerConfig.json`**

Authentication file format.

The authentication file is used to avoid hard-coding passwords into the code, and to allow the instructor to use his own databases to test your program. Its format is:

```
{server: <server>,
 username: <username>,
 password: <password>,
 authDB: <authentication database>,
 db: <work database>
}
```

The **`server`** parameter is optional, default value is **`"localhost".`**

The **`db`** parameter specifies the MongoDB database that will be used on the current run of your application.

The **`password`** parameter is optional. If specified, it can take either a string value that is the password to be passed to MongoDB in clear text (not the world's safest option, but…), or it will be set to **-1.** If the password is either missing, or set to **-1,** your program shall interact with MongoDB to produce a password prompt. See Appendix A for the description of how this can be done.

Functionality overview. Data Management. Your program shall maintain the COVID-19 tracking database consisting of two collections**:** **`covid`** - storing rich tracking information from the **`covidtracker.com`** API, and **`states`** - storing the somewhat feature-poor but more granular county-by-county tracking information.

When the work database does not contain these collections, your program shall fetch the most recent data from the two data sources, and create the collections prior to proceeding with any additional work.

Functionality overview. Analytics. Your program shall be able to produce the output for a range of analytical queries. Each analytical query has (i) ***a target collection,*** (ii) ***a target dependent variable,*** (iii) ***a target time range,*** *(iv)* ***aggregation level,*** *(v)* ***filter,*** and *(vi)* ***the task to perform.***

Target collection: There are two:

- State-level data (**`covid`**).
- County-level data (**`states`**)

The target collection puts some restrictions on other aspects of the query.

Target dependent variable. We have the following dependent variables:

Table 1. Dependent Variables and their availability in the collections

Variable	**`covid`**	**`states`**	**Value**
Cumulative number of positive cases	**yes**	**yes**	**`"positive"`**
Daily increase in positive cases	**yes**	**yes**	**`"positiveIncrease"`**
Cumulative number of COVID-19 deaths	**yes**	**yes**	**`"death"`**
Daily increase in number of deaths	**yes**	**yes**	**`"deathIncrease"`**
Total cumulative number of administered tests	**yes**	**no**	**`"tests"`**
Daily number of completed tests	**yes**	**no**	**`"testIncrease"`**
Total cumulative number of hospitalizations	**yes**	**no**	**`"hospitalization"`**
Daily number of new hospitalizations	**yes**	**no**	**`"hospitalizationIncrease"`**

Table 1 above shows the concordance between the eight dependent variables your application shall track, and the two data collections it works with. If the cell label is "**yes"**, your program is responsible for producing analytics for this dependent variable for this collection.

Target time range. With each analytical query, a time range can be specified. By default, the time range is the entire range of dates in the collection. If a range is specified, it will contain both the upper and the lower boundary, and the boundaries will always be inclusive. Your program

shall also handle some special time ranges: today, past seven days, past month. These are defined below.

Aggregation level. For the **`covid`** collection we have the following data aggregation levels:

1. **Country-wide.** The data for all states is aggregated, and the US-wide results are reported. This includes information from not just the 50 states plus DC, but all other territories and possessions.
2. **Fifty States+DC.** A variation of **country-wide** level, that only includes information about the 50 US states plus the District of Columbia, but aggregates across these 51 entities.
3. **State-wide.** The information is reported on a state-by-state basis.

For the **`states`** collection, we have the following data aggregation levels:

1. **State-wide.** The information is aggregated over all counties in each state.
2. **County-wide.** The information is presented on a county-by-county basis.

(there is no country-wide aggregation level for the **`states`** collection).

Filters. For the `covid` collection, the information can be filtered as follows:

1. **No filter.** No states, or territories/possessions are excluded from the output.
2. **Fifty States+DC.** Special filter associated with the namesake aggregation level that keeps only the 50 US States and Washington, DC data. Also works with the **state-wide** aggregation level.
3. **State-level.** One or more specific states. Works with **state-wide** aggregation level, and will produce requested information only for one or more selected states.

For the **`states`** collection we enforce the **state-level** filter, with a single state target. That is, your program shall only report county-level analytics on a one-state-at-a-time basis. However, for the **`states`** collection we also provide an additional **county-level** filter, that allows for narrowing the scope of the analytical report to one or several specifically named counties, rather than keeping it to all counties.

Tasks. For each valid collection-dependent variable combination, your program shall implement the following analytical tasks:

1. **Track over time.** The most straightforward analysis: output the time series for the selected dependent variable over the given time period, and at the correct level of aggregation.
2. **Statistics.** For the time series for the selected dependent variable over the given time period at the correct level of aggregation compute the mean, and the standard deviation. We use population standard deviation in this case, as there is no sampling.
3. **Ratios.** Given two dependent variables compute a time series of the ratio between them for a given aggregation level.

Output. The output for each individual analytical query can be produced in a number of ways:

- **Table.** The most straightforward is the tabular output, showing the information requested for each day in the given time range, and/or reporting and requested statistics.
- **Graph.** Using Python's matplotlib library, you can graph the result of any query against time, or, in some cases - you can create scatter plots. Matplotlib allows you to save your graphs as images. Graphs have several properties that will be specified in the input file.

The **overall output** is an HTML document that combines the tabular and graph outputs for all requested queries, and creates a single report. The HTML document does not have to be fancy, just just needs to load into a browser and display both the tables and the images in reasonable way.

The overall design of the HTML report is left up to individual teams. It shall contain a header at the top identifying it (as well as its creators), a layout for reporting results of all queries - both in textual and tabular, and graph form, and a footer with the information about when and how it was created.

Syntax and semantics of the configuration file.

The work of your program shall be primarily driven by the configuration file. The file will contain a single JSON object with the configuration settings for the program's run. The overall syntax of the configuration document is as follows:

```
{refresh: true|false,
 collection: <collection>,
 aggregation: <aggregationLevel>,
 time: <timeSpecification>,
 target: <states>,
 counties: <counties>
 analysis: [{task: <taskSpecification>,
             output: <outputSpecification>},
            ...
            {task: <taskSpecification>,
             output: <outputSpecification>},
           ],
Output: <filename>
}
```

The purpose of each field in the document is described in Table 2 below.

Table 2. Overview of Configuration Document Fields

Field	Value	Purpose
`refresh`	**`true`** or **`false`**	Flag indicating whether to use existing data or get a fresh copy
`collection`	string	Either **`"covid"`** or **`"states"`**
`aggregation`	string	One of predefined values for the aggregation level
`time`	Date or compound	Time frame of the query
`target`	Array or single value (string)	State or list of states
`counties`	Array or single value (string)	County or list of counties
`analysis`	array	List of analytical query specifications
`task`	compound	Part of analytical query specification defining what analysis needs to be performed
`output`	Compound or array of compound objects	Part of analytical query specification defining how the results should be placed into the output HTML report

We describe each component of the configuration file in turn.

The refresh flag.

This is a flag indicating to your program whether it should get a new copy of the data from the sources. Together with the **`db`** parameter of the authentication document, the **`refresh`** flag controls whether your program uses current data or refreshes it. Specifically, your program shall act as follows:

Case 1: The database specified in the db parameter ***does not*** have one of the two or both collections (**`covid`** and/or **`states`**). In this case your program shall, *regardless of the value of the* **`refresh`** flag, upload the most recent versions of both collections to the specified database.

Case 2: The database specified in the db parameter ***does*** have both collections, and the **`refresh`** flag is set to **`true.`** .In this case your program shall upload the most recent versions of both collections to the specified database.

Case 3: The database specified in the db parameter ***does*** have both collections, and the **`refresh`** flag is set to **`false.`** .In this case your program shall proceed with executing the analytical queries against the already existing data.

The collection parameter.

The **`collection`** parameter takes one of two values: **`"covid"`** or **`"states".`** It is a global parameter for the entire run of your program: all analytical queries will be running against only one of the collections on a given single run. (this makes your life a little bit easier).

NOTE: the value of the **`collection`** parameter determines which combinations of other parameters are valid. Not all combinations are intended for use with both collections.

The aggregation parameter.

The **`aggregation`** parameter determines the aggregation level for all analytical queries defined in the configuration document. It takes one of four predefined values, some of which are associated with only one collection. Table 3 contains the list of values for the **`aggregation`** parameter and their meaning.

Table 3. Aggregation level parameter values.

value	**`covid`**	**`states`**	Aggregation level
`"usa"`	yes	no	USA-wide - aggregate over all data including possessions and territories
`"fiftyStates"`	yes	no	Aggregate over 50 states + DC (must be combined with appropriate filter)
`"state"`	yes	yes	Aggregate/Report results by individual state
`"county"`	no	yes	Aggregate/Report results by individual county

Notes: Both **`"usa"`** and **`"fiftyStates"`** values require aggregation of state-by-state data into a single result. The difference is that **`"fiftyStates"`** is a combined aggregation level/filter specification -- when this value is specified, every single query your program runs shall include a filter that keeps only the data from the 50 US states and Washington, DC.

Information about US territories, possessions, and military installations will be ignored in such analyses.

The time frame parameter.

The **`time`** parameter specifies the time frame of the query. It can take one of four predefined values, as well as a compound value defining the range of dates. The meaning of these values is described in Table 4.

Note, the time frame is a **global parameter** for all analytical queries requested in the configuration file. All queries shall be run against the specified time frame.

Table 4. Values of the time frame parameter

Value	**Meaning**
`"today"`	Set the time frame to today's date
`"yesterday"`	Set the time frame to yesterday's date (useful alternative to **`"today"`** if today's data is not available yet)
`"week"`	Set the time frame to be a seven day span that ends on today's date (this is regardless of whether the database contains data about today).
`"month"`	Set the time frame to be a time span that starts on the 1st of the current month and ends on today's date.
`{start: <start>, end: <end> }`	Compound value specifying the exact ***inclusive*** range of dates. The dates shall be rendered in the formats appropriate for the specific collections.

Note: The **`covid`** and **`states`** collections come from different sources that use different date formats. I highly recommend that during the parsing of the **`states`** collection you either convert the date to the numeric **YYYYMMDD** format (which is really convenient for comparisons), or add a version of the date field in this format. Then your <start> and <end> values for the date ranges can always be in **YYYYMMDD** format, and your program will only deal with numeric comparisons.

The Target state(s) parameter.

This is where things start getting tricky. The syntax of the **`target`** parameter, by itself, is simple and straightforward: it is either:

- A string representing a single two-letter code of a US state, territory, possession, or military designation, or
- A JSON array of strings representing a single two-letter code of a US state, territory, possession, or military designation.

This parameter specifies the **global** (i.e., applicable to all requested analytical queries) state-level filter. The values of this parameter are subject to the following constraints and interactions with the **`collection`** and **`aggregation`** parameters:

- If the **`collection`** value is "**`states"`**, then the value of the **`target`** parameter is a single state. That is: for queries looking into county-level data, we restrict their applicability to a single state.
- If the **`collection`** value is **`covid`**, then the **`target`** parameter can be either a single value or an array of values:
    - For **`aggregation`** levels of **`"usa`**" and **`"fiftyStates"`** the presence of a specific state filter means that the filter is applied first, and then information about the states that remained will be aggregated to the appropriate level. Note that if **`"fiftyStates"`** aggregation level is selected, your program **shall ignore** any two-letter codes that do not belong to a US state or Washington, DC.
    - For the aggregation level of **`"state"`** the **`target`** state filter simply provides information about one or more states whose data shall be used in the analytical queries (without aggregation).

The counties filter.

The **`counties`** parameter shall only appear in configuration documents that set **`collection`** to **`"states".`** If the **`collection`** parameter is set to **`"covid",`** and the configuration file contains the **`counties`** parameter, it shall be ignored by your program.

When specified alongside the **`collection:"states"`** parameter, the **`counties`** parameter takes as its value a single string, or a JSON array of string values.. Each string value ***is expected to be*** a valid name of a county in the state specified in the **`target`** parameter value. Any values that are not a correct name of a county shall be ignored, but shall not raise any errors.

When specified, your program shall limit the consideration of all queries to the specified county, or list of counties. If **`aggregation:"state",`** the county filter is applied first, after which the results from the remaining counties are aggregated together into a single output.

The analytical queries

The **`analysis`** parameter stores information about the queries that your program has to execute during its run. The value of the **`analysis`** parameter is an array of ***query documents.*** For the sake of simplicity, even if requesting a single query to be run by the program, it shall be placed inside a JSON array.

The format for the **query document** is as follows:

```
{
 task: <query description>
 output:  <outputSpec>
}
```

Query Description

There are three types of analytical queries (tasks) that your program must support. Each is described by a specific value of the **`task`** field. Table 5 documents the syntax of the **`task`** field values, detailed explanations follow.

Table 5. Syntax of the task field.

Syntax	Query/Task Type
`{track: <variableName>}`	Track the variable value across time

`{stats: [<variableName>,`
`...,`
`<variableName>]`
`}`
	Compute the statistics for the given list of variables over the given aggregation, filter, and time frame

`{ratio:{ numerator: <variableName1>,`
`denominator: <variableName2>`
`}`
`}`
	Track the value of a ratio of two variables across time

In Table 5, all **`<variableName>`** values come from the **Value** column of Table 1.

Tracking individual variables across time. The

```
{track: <variableName>}
```

syntax specifies an analysis query that reports the values of the given variable (see Table 1) across the globally specified time period, taking into account the global filter, and the level of aggregation (see the Examples Section for specific examples).

Reporting Statistics. The

```
{stats: [<variableName>,
         ...,
         <variableName>]
}
```

syntax indicates an analytical query that computes and reports the mean and standard deviation for ***each variable named*** over the specified time period, taking into account the global filter and the level of aggregation. See the Examples Section for specific examples.

Tracking ratios. The

```
{ratio:{ numerator: <variableName1>,
         denominator: <variableName2>
       }
}
```

syntax specifies that the analytical query must report the values of a new variable:

$$\textit{ratioVariable} = \frac{\textit{variableName1}}{\textit{variableName2}}$$

These values must be tracked over the specified time period, taking into account the global filter and the level of aggregation. See the Examples Section for specific examples. See the Examples Section for specific examples.

Handling division by zero. You are responsible for properly tracking the ratio variables ONLY on the days when ***variableName2*** is non-zero. As a convention you can assume that 0/0 = 0, but you are not required to report values of the ratio variable on ANY day in which ***variableName2*** is 0.

Output specification

For each analytical query, the **`output`** field provides a specification of how the output should be reported. The **`output`** field has the following syntax:

```
output: { graph: {type: <graphType>,
                  legend: <on or off>,
                  combo: <combinationIndicator>,
                  title: <text>
                 },
          table: {row: <rowDesignator>,
                  column: <columnDesignator>,
                  title: <text>
                 }
         }
```

Both **`graph`** and **`table`** fields are **optional,** but the output specification must contain **at least one** (possibly both).

Graph Specification.

The **`graph`** field provides the specification for the graph to use to display the results. For **`track`** and **`ratio`** types of tasks the graph output is a Python3 **`matplotlib`** graph saved as a file (and included in the output HTML document). ***The graph has time as the x axis and the value of the specific variable being tracked as the y axis.*** The graph design is controlled by the remaining parameters:

Table 6. Output specification overview.

Graph attribute	Values	Purpose	Optional/ Mandatory
`type`	One of **`"bar"`** **`"line"`** **`"scatter"`**	Defines the type of graph to produce Bar graph (**`pyplot.bar(...)`**) Line graph (**`pyplot.plot(...)`**) Scatter plot (**`pyplot.scatter(...)`**)	**mandatory**
`legend`	**`"on"`** **or** **`"off"`**	Indicates whether the legend shall be placed on the graph	**optional** (default **`"off"`**)
`combo`	One of	Determines how to combine multiple	

	`"separate"`
	`"split"`
	`"combine"`
		state/county data
		Separate graph for each state/county
		"Smart" mode (see below)
		One graph for all states/counties
			mandatory

`title`	string	Title of the graph	**optional**

Graph type. Your application shall support three types of graphs produced: bar graphs, line graphs, and scatter plots. Each type shall use the appropriate **`matplotlib.pyplot`** method to produce the output listed in the table above. You can decide how to decorate each of the graphs types, and whether you want the same decorations always, or different decorations depending on the variables being displayed.

Legend. If set to **`"on",`** add a simple legend to the graph output.

Combination specification. Some aggregation level - collection combinations produce multiple "streams" of data across time. For example your program may receive a configuration file for the *"Report all positive cases for California, Oregon and Washington, for the last seven days, aggregating at state level"* (i.e., report results for each state separately!). The graph output for such a query can either combine the results in a single graph, or produce one graph per state. The **`combo`** parameter controls the behavior of your output generator as follows:

`combo:"separate"` means building a separate graph for each state (or county, if the collection is **`states`** and the aggregation level is county).

`combo:"split"` engages the "smart graph" mode. Here, you are allowed to report results for multiple states/counties in a single graph, but you will optimize both how many states/counties per graph are displayed, how many graphs are produced, and which states/counties are combined on individual graphs ***using your own design sensibilities.*** The result is a set of one or more graphs produced that jointly cover all the output generated by your query.

`Combo:"combine"` forces all output to be combined in a single graph, no matter how many independent lines/colors (i.e., states/counties) it would place there.

Title. The title parameter, if present, contains the string to use as the title of the graph in the output. This can be typed directly into the graph image, or used as the title in the HTML - how you do it is left up to you. The default value is no title.

Table Specification

The **`table`** element, when present, specifies how the output will be rendered as an HTML table. The format of the **`table`** element is

```
table: {row: <rowDesignator>,
        column: <columnDesignator>,
        title: <text>
                 }
```

Here, **row** defines what information is rendered in the rows of the table, **column** defines what information is rendered in the columns of the table, and **title,** similarly to the graph description, specifies the optional title.

The **row** and **column** attributes take the following values:

Table 7. Row/Column specifications

Value	Meaning
time	Dates: one day per row or column. Only makes sense when the time frame is one day or more, and when the task is NOT **stats**
state	State-level data in accordance with aggregation level. If aggregation level is **"usa"** or **"fiftyStates"** one row/column of aggregate numbers. If aggregation level is **"state"**: one column/row per state.
county	County-level data in accordance with aggregation level. Only makes sense if collection is **states.** With **county** aggregation level, one row/column per county in the output.
stats	Report statistics. Only makes sense with **stats** tasks.

For **track** and **ratio** tasks/queries, specify both **row** and **column** for each table. For stats tasks, only

```
table:{row: "stats"}
```

or

```
table: {column:"stats"}
```

specifications are needed.

Output Specification

The final element of the configuration file is **`output.`** It's value is a name of an HTML file where the report will be written.

For example

`output: "april27.html"`

specifies that the output of your program shall be placed in the **`april27.html`** file. Additionally, the name of the file, without the extension, shall be used to name your graph files. The exact file naming scheme for the graph files is left up to you, but you shall use the filename for the output file as the prefix for all graph file names. This way, we can run your program multiple times and create non-overlapping outputs in a single directory.

For the sake of simplicity, all file can be created in the current directory.

Deliverables

Your submission deliverables are:

- **`covidTracker.py`** program
- Any additional modules/packages you created in support of **`covidTracker.py`**
- **`README`** file with any information regarding your implementation.