# Tesla Inc.

# Data Analyst - Coding challenge

## Instructions and guidelines

- You will have 1 day from the time of receipt of this challenge to respond back with a solution
- You are free to use any programming language as long as the code compiles
- Standard test modules (for testing your code) are typically written in Python/ R/ Javascript. If your solution is in a different language, please also include clear instructions on how to set up the compiler/ runtime
- If your solution uses non-standard packages, please include tests for all required dependencies, and install any additional/ missing packages needed before code execution begins
- You will need to use a SQL product (MySQL highly preferred) to test data upload/ download from your side
- If you are using a SQL product other than MySQL, please include clear instructions on what kind of database is used, and instructions on how to connect to it
- Bonus points for clear commenting, good handling of edge and corner cases as well as making your code modular
- Please submit your solution as a compressed folder (ZIP) with all the files required to run the code
- Please include comments (and a write-up if possible) to explain your solution/ methodology. If the code fails to compile, your solution will be assessed purely based on the write-up/ comments
- Please refrain from sharing this coding challenge through private channels. The content of the attached documents/ files is the proprietary and confidential property of Tesla, Inc. and should be treated as such

## Problem statement - Part I

The goals of this section are as follows:

- Fetch data from a remote source
- Transform JSON data into a relational form
- Uploading data in relational form into a SQL database

We will be using API endpoints to get the data we need to transform and load into our own database later on. For a quick introduction to APIs, please look here. For our challenge, we will be using the SEPTA (Southeastern Pennsylvania Transportation Authority) API. The API documentation for SEPTA API is available here.

API calls can be run through standard command line interfaces, or a program like Postman. The annotated examples were run on a bash environment running on a linux platform. You are encouraged to use a package/ program that best suits your needs/ solution, but ensure that your solution is platform independent.

**Step 1**: We will begin by getting data on all available lines on the SEPTA network by running the following command

```
curl --get https://www.septastats.com/api/current/lines
```

This returns the following JSON output

```
{
″metadata″: {
″_comment″: ″All train lines″ },

″data″: {
″airport″: ″Airport″,
″chestnut-hill-east″: ″Chestnut Hill East″, ″chestnut-hill-west″: ″Chestnut Hill
West″, ″cynwyd″: ″Cynwyd″,
″fox-chase″: ″Fox Chase″,
″glenside″: ″Glenside″,
″lansdale-doylestown″: ″Lansdale\/Doylestown″, ″manayunk-norristown″:
″Manayunk\/Norristown″, ″media-elwyn″: ″Media\/Elwyn″, ″paoli-thorndale″:
″Paoli\/Thorndale″, ″trenton″: ″Trenton″,
″warminster″: ″Warminster″,
″west-trenton″: ″West Trenton″, ″wilmington-newark″: ″Wilmington\/Newark″

} }
```

**Task 1**: The first task of this problem would be to load the results of the API call into a variable. Write a function that executes the API call and loads the results into a variable

The same data, represented in relational/ tabular form is shown below:

| | |
|---|---|
| metadata | |
| _comment | All train lines |
| data | |
| airport | Airport |
| chestnut-hill-east | Chestnut Hill East |
| chestnut-hill-west | Chestnut Hill West |
| cynwyd | Cynwyd |
| fox-chase | Fox Chase |
| glenside | Glenside |
| lansdale-doylestown | Lansdale/Doylestown |
| manayunk-norristown | Manayunk/Norristown |
| media-elwyn | Media/Elwyn |
| paoli-thorndale | Paoli/Thorndale |
| trenton | Trenton |
| warminster | Warminster |
| west-trenton | West Trenton |
| wilmington-newark | Wilmington/Newark |

We need to truncate the first three rows to get rid of the metadata and data headers, and add relevant headers to make it readable and useful.

| line_name | description |
|---|---|
| airport | Airport |
| chestnut-hill-east | Chestnut Hill East |
| chestnut-hill-west | Chestnut Hill West |
| cynwyd | Cynwyd |
| fox-chase | Fox Chase |
| glenside | Glenside |
| lansdale-doylestown | Lansdale/Doylestown |
| manayunk-norristown | Manayunk/Norristown |
| media-elwyn | Media/Elwyn |
| paoli-thorndale | Paoli/Thorndale |
| trenton | Trenton |
| warminster | Warminster |
| west-trenton | West Trenton |
| wilmington-newark | Wilmington/Newark |

**Task 2**: The next task is convert the data from JSON format into tabular/ relational form. Given a JSON object and a number of lines n and a list of headers, write a function that converts the JSON object into tabular form, truncates the top n lines and sets column headers from the list (in the same order used on the list).

*Hint: Consider using data frames*

**Task 3**: Next, we need create a MySQL table named line_name on our database and upload the data above to line_name table. Bonus points if you can set the data types for line_name and description columns to VARCHAR(MAX) data type. Given some database credentials, a table_name and some data, write a function that

- Creates a table in the database and name it from table_name (line_name in this case)
- Takes in the data (in relational/ tabular form) and uploads it onto the line_name table

  *Hint: Your code should check if* table_name *exists in the database already. Because if the table exists, we don't want to overwrite data on the table. This applies to Step 2 as well*

**Step 2**: The next step would be to get the latest reported information on all vehicle locations:

http://www3.septa.org/hackathon/TransitViewAll/

This API call produces the following output (output limited to suit explanation) :

```
{'routes': [{'1': [{'lat': '40.017605',
    'lng': '-75.149956',
    'label': '3374',
    'VehicleID': '3374',
    'BlockID': '1003',
    'Direction': 'NorthBound',
    'destination': 'Parx Casino via Decatur-Drummond',
    'Offset': '0',
    'heading': 45,
    'late': 0,
    'Offset_sec': '38',
    'trip': '390276'},
```

**Task 1**: Similar to Task 1 in Step 1, load the results of this API call into a variable, for each route.

As before, this data in tabular form (after truncating unnecessary rows) looks as follows:

| route | vehicle_id | direction | destination |
|---|---|---|---|
| 1 | 3374 | NorthBound | Parx Casino via Decatur-Drummond |

**Task 2**: Similar to Task 2 in Step 1, convert the JSON output into the tabular form needed to upload into our database.

**Task 3**: This data needs to be uploaded to a new table called route.

**Task 4**: Repeat Tasks 1, 2 & 3 in Step 2 to write the current train information for each route and each direction into route table.

The API call is a snapshot of vehicle locations and so the output obtained is dependent on the time at which the API is called.

**Step 3**: The next step would be to get the 'line' related information using a regional station name as parameter:

http://www3.septa.org/hackathon/Arrivals/{stations}/{next*n*trainsarrivinginstation}/

For example, http://www3.septa.org/hackathon/Arrivals/Olney/3/ yields the next 3 trains arriving at Regional Station called Olney Transportation Center Station.

Challenge is to identify the Regional Station names.

*Hint: (i) Use destination column from route table as a substitute for Regional Station name.*
*(ii) For all values in destination column containing ' Transportation' truncate them only to the words until ' Transportation'. Note that in the api referenced above, the station name Olney Transportation Center Station has been truncated to Olney in the api call*
*(iii) Not all Station names in destination column are Regional Station names. If your API call happens to use a non-Regional Station name, you will get error. Prepare to handle those errors.*
*(iv) If you find that your route table does not have any destination that is a Regional Station name, re-run the API call from Step 2 to update the destination column in the route table.*

**Task 1:** The API call http://www3.septa.org/hackathon/Arrivals/Olney/3/ produces the following output (output limited to suit explanation):

```
{'Olney Departures: September 16, 2020, 3:16 pm': [{'Northbound': [{'direction': 'N',
    'path': 'R8N',
    'train_id': '842',
    'origin': '30th Street Station',
    'destination': 'Fox Chase',
    'line': 'Fox Chase',
    'status': 'On Time',
    'service_type': 'LOCAL',
    'next_station': None,
    'sched_time': '2020-09-16 15:47:00.000',
    'depart_time': '2020-09-16 15:47:00.000',
    'track': 'S',
    'track_change': None,
    'platform': '',
    'platform_change': None},
   {'direction': 'N',
    'path': 'R8N',
    'train_id': '846',
    'origin': '30th Street Station',
    'destination': 'Fox Chase',
    'line': 'Fox Chase',
    'status': 'On Time',
    'service_type': 'LOCAL',
    'next_station': None,
    'sched_time': '2020-09-16 16:47:00.000',
    'depart_time': '2020-09-16 16:47:00.000',
    'track': 'S',
    'track_change': None,
    'platform': '',
    'platform_change': None},
.
.
.
{'Southbound': [{'direction': 'S',
    'path': 'R8S',
    'train_id': '839',
    'origin': 'Fox Chase',
    'destination': '30th St',
    'line': 'Fox Chase',
    'status': 'On Time',
    'service_type': 'LOCAL',
    'next_station': None,
    'sched_time': '2020-09-16 15:26:00.000',
    'depart_time': '2020-09-16 15:26:00.000',
    'track': 'S',
    'track_change': None,
    'platform': '',
'platform_change': None},
```

The above output in tabular form looks like :

| line | direction | origin | destination | train_id |
|---|---|---|---|---|
| Fox Chase | Northbound | 30th Street Station | Fox Chase | 842 |
| Fox Chase | Northbound | 30th Street Station | Fox Chase | 846 |
| . . . | | | | |
| Fox Chase | Southbound | Fox Chase | 30th St | 839 |

We create a table called 'line_metadata' and store the information in the table above.

This is the end of Part I. By now, you should have functions that pull data from the API, formats/ reformats the data and loads it onto line_name, route and line_metadata tables.

**<u>Problem statement - Part II</u>**

The goals of this section are as follows:

- Fetch data on regular intervals (from the same data source)
- Incrementally upload ‘new’ data to our own tables

  As an extension of Part I, we will now :

  - run the API call in Step 2 (current location of all the trains) once every minute (the suggested hit rate from the website), and append ‘new’ data into the route table. The API call should return new data at every call, as trains keep moving.
  - as you get new destination names in the route table, run the API call in Step 3 (list the next *n* trains to a regional station) subsequently and append the data to the line_metadata table if the data is ‘new’.

  Things to consider : You might want to think about what can be ‘new’ data.

  This section is being left intentionally open-ended because of the multitude of design choices available for an incremental data upload. Bonus points if you consider database design principles like normalization, indexing, etc. as part of your implementation.

  **<u>Task 1</u>**: Given a frequency (in minutes) of updating that defaults to says 1 minute, encapsulate your code from Part I within a function such that it continuously executes at the specified frequency.