# OpenF1 Tutorial (Windows)

## Step 1.) Installing MongoDB Community Server

1.) Go to the following link: https://www.mongodb.com/try/download/community
   - a.) Download the MSI package of the most recent version of the MongoDB Community Server and click "Download".

2.) Once the MSI installer is downloaded, run it and follow the Setup Wizard.
   - a.) Choose "Complete" installation for all features.
   - b.) Check "Install MongoDB as a Service" to run it automatically.
     - i.) Of the two radio button options, select "Run service as Network Service user".
   - c.) If you want a GUI, select "Install MongoDB Compass".

3.) If you downloaded MongoDB Compass, run it to confirm it's working properly.

## Step 2.) Install OpenF1 python package

1.) Confirm your computer has at least pip v23 and Python v3.10. Open the command prompt and type:
   - a.) `pip --version`
   - b.) `python --version`

2.) Clone the forked repository:
   - a.) `git clone https://github.com/CSinclair89/openf1`
   - b.) `cd openf1`
   - c.) `pip install -e openf1`

3.) To confirm it's installed, you can follow it up with the command:
   - a.) `python -c "import openf1; print('Its fuckin working!')"`
   - b.) If there isn't an error message, you're good to go.

## Step 3.) Configure the MongoDB connection

1.) Open command prompt and type:
   - a.) `setx MONGO_CONNECTION_STRING "mongodb://localhost:27017"`

2.) Close the command prompt window and open a new one.

3.) Test the new environment variable with the following command:
   - a.) `echo %MONGO_CONNECTION_STRING%`

4.) If the output is "`mongodb://localhost:27017`", you're all set.

5.) If that doesn't work for whatever reason, you can also set it manually in the Windows System Environment Variables. I won't go into detail on that though since it can be looked up easily.

## Step 4.) Fetch Data

1.) A good way to tinker with basic API functionality would be to start in Jupyter Notebook:
    a.) `pip install jupyter`
2.) You can then run the application by navigating to the openF1 directory and typing:
    a.) `jupyter notebook`
3.) When the application opens up, click “New” in the top left corner and select “Notebook” from the dropdown menu.
4.) In the ‘requirements.txt’ file of the repository, a bunch of libraries and their respective versions are listed. To start with the basics, we only need one from the list: requests.
    a.) `pip install requests`
    b.) The other two that I used are worth installing as well: pandas and json. Since the json library is included in base Python now, we only need to install pandas:
        i.) `pip install pandas`
5.) You can test all of these installations by following the same steps outlined in step 2.3 above:
    a.) `python -c “import _____; print(‘Its fuckin working!’)”`
    b.) Replace the blank space above with ‘requests’, ‘json’, and ‘pandas’ to confirm everything is all set.
    c.) You may need to restart Jupyter Notebook after making these imports.
6.) Now that our libraries are in order and Jupyter Notebook is installed, a basic block of code to start with could be something like the following:

```
import json
import requests
import pandas as pd

url =
“https://api.openf1.org/v1/sessions?session_type=Race&year=2023
”
response = requests.get(url)

print(f”Status Code: {response.status_code}”)
```

7.) You can run this block of code in Jupyter Notebook by pressing Shift+Enter. If you get the following output, you’re all set:
    a.) `Status Code: 200`
8.) I’m partial to Data Frames since most of my Python-related work has been with CSV files, so by importing Pandas, we can convert our JSON data to a familiar Data Frame.
9.) Worth noting that I’m basing this conversion on what is listed in the documentation (commented out on each section). Furthermore, by adjusting the Sample URL given at the top of each API Method, we can isolate specific kinds of data. Ramiz mentioned having trouble pulling up race data, so you can see above that I adjusted the URL to isolate all Race data from 2023. The following block of code could be the next entry in your notebook:

```
    data = response.json()
    df = pd.DataFrame(data)
    df.head(5)
```

10.) You can run this by again pressing Shift+Enter. Jupyter Notebook doesn't require print statements for the last line of code in each block, so '`df.head(5)`' will automatically display the first 5 elements in the resulting data frame.

11.) The final step of fetching data would be to extract specific fields that we're interested in. I'm not too worried about how messy all of this can get since the goal is to learn, but the main thing to note is that in databases, we want to identify a primary key so we can find related items for that. If we're looking to get information on races, we would want to use the 'session_key' attribute as our primary key.

Let's improvise a bit here: I'd like to find the winner of a race. That means I need a session ID, which in this case, is the aforementioned 'session_key' attribute. Unfortunately, there is no Results endpoint that I can call, so I need to manually find the winner by looking at the positions on the final lap for a specific session key. Since the Sessions endpoint does not contain driver positions or laps, we instead need to refer to both the Laps endpoint (to find the final lap) and the Position endpoint (for the final positions). By combining the responses into a single data frame, we can find our winner. This can later be used to find winners of different session types (Practice, Qualifying, etc.). For the record, I just arbitrarily picked session_key 7779 since I saw it in the previous output:

```
session_key = 7779

# Get response from Laps endpoint for Session Key 7779
laps_url = "https://api.openf1.org/v1/laps"
laps_params = { "session_key": session_key }
laps_response = requests.get(laps_url, params = laps_params)

# Get response from Position endpoint for Session Key 7779
pos_url = "https://api.openf1.org/v1/position"
pos_params = { "session_key": session_key }
pos_response = requests.get(pos_url, params = pos_params)

# Confirm the Status Code for both is the proper 200
if laps_response.status_code == 200 and pos_response.status_code ==
200:

    # Grab data from each response
    laps_data = laps_response.json()
    pos_data = pos_response.json()

    # Convert data into separate data frames
```

```
      df_laps = pd.DataFrame(laps_data)
      df_pos = pd.DataFrame(pos_data)

      # Test print to verify -- this can be deleted later
      print("Laps Data Columns:", df_laps.columns)
      print("Position Data Columns:", df_pos.columns)

      # Merge the two data frames into one with common keys
      merged_df = pd.merge(df_laps, df_pos, on = ["session_key",
"driver_number"], how = "inner")

      # Test print merged data frame — delete later
      print("Merged Columns:", merged_df.columns)

      # Narrow down fields of interest w/ subset data frame
      merged_subset = merged_df[["session_key", "driver_number",
"lap_number", "position", "date"]]
      print("Merged Subset Columns:", merged_subset.columns)

      # Find the final lap (maximum number in the lap_number attribute)
      final_lap = merged_subset["lap_number"].max()
      print(final_lap)

      # Find winner (position 1 on final lap)
      winner = merged_subset[(merged_subset["lap_number"] ==
final_lap) & (merged_subset["position"] == 1)]
      print(winner)

else:
      print("Error: you fucked up somewhere")
```

This block of code gets us the winner in the form of a driver number, which for the three rows of output are 11, 11, and 14. I'm not sure why there are three resulting rows, but that's worth exploring at another time. If we wanted to get the driver name, we would flesh this out to include a data frame from the Drivers endpoint. We may also want to include the Sessions endpoint now in order to get the location, type, and date of the race (the current date field is from the Position endpoint and doesn't notate the actual start date of the race, just the date of the final lap).

That can all be tackled later though – for now, I'm going to save this notebook by clicking 'File -> Save Notebook As…' and giving it a name. I hope this covers fetching the data!