4.0 - Requests

Python requests library is a vast and yet simple to use library for sending and receiving data across the web. The library is interconnected with http and thus inappropriate to use with any other protocols.

4.1 - REST API Structure

In phonebook.zip (at canvas) you find a REST API containing two main files

- app.py ← flask web server
- client.py ← a client performing requests

and two folders

- data ← database
- flaskr ← additional help files

4.1.1 - File content

Familiarize yourself with the content of the files.

A. app.py

   This file contains a few setup configurations and routing for http requests

```
app = Flask(__name__)
phonebook = Phonebook(CURR_DIR_PATH + "/data/database.db")
initialize_mock(phonebook)
```

   The line "initialize_mock" fills the phonebook with mock (random) data and may be commented out if one wishes not to generate new data every run.
   (To fully delete the data you need to delete the database.db file in the data folder)

B. client.py

   The client file, here http requests are responses are transmitted and received to and from the server in app.py
   Right now the requests.post line is commented out to avoid overpopulating the database by every re-run.

C. flaskr/phonebook.py
   This file represents a cross database between pandas and sqlite. The permanent data is stored in sqlite, but the data being remotely accessed is accessed through the pandas local memory. The methods that we are concerned with right now are

```python
def get_data(self):
  if len(self.data) == 0:
    self.data = self.sql.get_data("phonebook")

  return self.data

def get_all(self):
  return self.get_data().to_json(orient="records")

def get_by_name(self, name):
  df = self.get_data()
  return df[df["name"].str.contains(name)].to_json(orient="records")

def add(self, entry):
  self.write_to(entry)
```

D. flaskr/console.py and flaskr/data.py are simple help files, you may inspect those but they are simply for generating and printing information

E. flaskr/mock_phonebook.py generates the mock data, once again, you may inspect and “alter” it, but doing so might break other parts of this exercise

4.2 - Expanding the API

Existing features:

- get the entire phonebook ← http://127.0.0.1:5000/phonebook/ (GET)
- query by name ← http://127.0.0.1:5000/phonebook/name/name_query (GET)
- add new entry ← http://127.0.0.1:5000/phonebook (POST, PUT)
  json-content (example in client.py with post requests)

```json
{
  "entry": {
    "name": "full_name",
    "number": "phone_number",
    "address": "address_text"
  }
}
```

Features missing (code snippets from client.py when performing requests query):

1) get parts of the phonebook ← http://127.0.0.1:5000/phonebook/100 (GET, 100 first rows)
   Should return up to 100 rows

```python
r = requests.get(f"http://127.0.0.1/phonebook/{rows}")
```

2) query by address ← http://127.0.0.1:5000/phonebook/address/address_query (GET)
   Returns all matches with an adress query similar to query by name

   ```
   r = requests.get(f"http://127.0.0.1/phonebook/address/{address_query}")
   ```

3) delete entry ← http://127.0.0.1:5000/phonebook/name (DELETE)
   Deletes **ONLY** if the name is a full match

   ```
   r = requests.delete(f"http://127.0.0.1/phonebook/name/{full_name_match}")
   ```

Add the three missing features, however, when doing so all the feature logic should be implemented in phonebook.py, whilst the flask routing and return of the data is handled in app.py.

```
# in app.py
@app.route("/phonebook/<num_of_rows>")
def get_phonebook_rows(num_of_rows):
  return phonebook.get_rows(num_of_rows)

# in phonebook.py Phonebook class
def get_rows(self, rows):
  df = self.get_data()
  #logic for slicing up to 100 rows
  return #the data
```

Do note that the “data” in phonebook should be accessed through the method “get_data” and not directly by self.data.

Hints

1) Getting up to a certain amount of rows in python can be done by list slicing
2) Can be solved very similar to how get_by_name fetches by name query
3) Dropping a row from a Dataframe is done with
   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
   You will need to pass the argument axis=0 to delete by index.

4.3 - Hidden feature

The database actually contains a 4th column called "added" which holds when a phone number was added. To enable this property you need to change the shown columns in the class "SQLWriter" in phonebook.py.

From

```
def get_data(self, dbname):
  if not(self.engine.has_table(dbname)):
    return []
  return pd.read_sql(dbname, self.engine, columns=["name", "number", "address"])
```

To (one string name added at the end of the column arguments)

```
def get_data(self, dbname):
  if not(self.engine.has_table(dbname)):
    return []
  return pd.read_sql(dbname, self.engine, columns=["name", "number", "address", "added"])
```

4.3.1 - Enable the feature

4) Query by added date ← http://127.0.0.1:5000/phonebook/date/start_month_query/end_month_query (GET)
   Returns all phone numbers that were added between the start_month and end_month

4.4 - Specification failure

Right now there might be some invalid dates that have been added during mock data generation. This phonebook was launched at the first of June 2022

A. Validate that all the added dates are between 1st of June 2022 and today
   I recommend creating a etl file "date_etl.py" and create a copy of the database values by either using sqlite directly on the database, or utilize the already existing get_all by first querying the web api and then save the data using pandas and local csv formats.

B. Change the add feature of the rest API so that the date cannot be before todays date.