Note: To complete this lab at home, you must have MongoDB Server, Compass, and the Shell installed. The installation instructions are as follows:

- **Install MongoDB Community Edition (free version)**
  The tool doesn't require much disk space.
  https://www.mongodb.com/try/download/community
- **Install Compass**
  Also lightweight and easy to install.
  https://www.mongodb.com/products/compass
- **Install the Mongo Shell**
  It's provided as a ZIP file that just needs to be extracted.
  You can place the executable wherever you like (for example, in the directory for this course).
  https://www.mongodb.com/try/download/shell

----------**********----------

Integration of JSON Data
Creating a Database and a Collection

In this section, we'll create our first database called **test**, which will initially contain a single collection named **restaurants**.

1. Open **MongoDB Compass**.
2. Click **Connect** directly (since your server is local, there's no need to specify anything).
3. You should see three existing databases: **admin**, **config**, and **local**.
4. Click **CREATE DATABASE** at the top.
5. Now name the new database and the collection where we'll store our data:
   - **Database Name:** test
   - **Collection Name:** restaurants
6. Click **CREATE DATABASE**.
7. You should now see the **test** database appear on the left, and when you click the small arrow next to it, the **restaurants** collection will be visible.
8. When you click on the collection inside the database, you can view its contents.
   At this point, our **restaurants** collection is empty it contains no documents yet.
   However, it's possible to **import documents directly** into it.

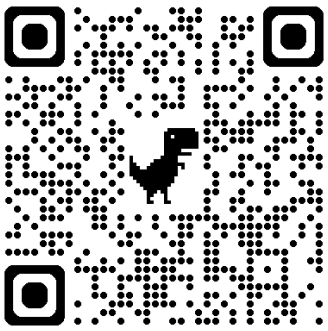
Importing Data

We'll now import data into the collection we just created. You must first download the file **restaurants.json** (about 11.9 MB).

Once the file is downloaded, follow these steps to add it to MongoDB:

1. Click on **restaurants** to view the collection's contents (currently empty).
2. Click **ADD DATA** at the top and choose **Import File** (or click **Import Data** directly).
3. Select the downloaded file and choose **JSON** as the file type.
4. Click **IMPORT**.
5. Once the process is complete (**25,359 documents added**), click **DONE**.

Your first database is now created, containing a collection with **25,359 restaurant records**.



MongoDB with Python (Using PyMongo):

Import and Connect

```
from pymongo import MongoClient

# Connect to the local MongoDB instance (make sure MongoDB is running)

client = MongoClient("mongodb://localhost:27017/")

print("Connected to MongoDB successfully!")
```

Show Existing Databases (equivalent to **show dbs)**

```
print("Existing Databases:")
dbs = client.list_database_names()
for db_name in dbs:
    print("-", db_name)
```

Select the Database "test" (equivalent to **use test)**

```
db = client["test"]
print(f"Switched to database: {db.name}")
```

Show Collections in This Database (equivalent to **show collections)**

```
collections = db.list_collection_names()
print("Collections in 'test':")
for col in collections:
    print("-", col)
```

Access the Collection and Count Documents (equivalent to **db.restaurants.count())**

```
restaurants = db["restaurants"]
count = restaurants.count_documents({})
print(f"Number of documents in 'restaurants': {count}")
```

Expected output:

```
Databases:
- admin
- config
- local
- test
Switched to database: test
Collections in 'test':
- restaurants
Number of documents in 'restaurants': 25359
```

1. Information Retrieval

MongoDB uses JavaScript in the shell, but in Python we use the **pymongo** library to interact with the database. This includes counting documents and querying for specific information.

Count Documents with a Condition (Equivalent to **db.restaurants.count({ borough: "Brooklyn" }))**

```
# Count restaurants in Brooklyn
brooklyn_count = restaurants.count_documents({"borough": "Brooklyn"})
print(f"Number of restaurants in Brooklyn: {brooklyn_count}")
```

2. Find the First Document (Equivalent to db.restaurants.findOne())

```
# Get the first document in the collection
first_doc = restaurants.find_one()
print("First restaurant document:")
print(first_doc)
```

- find_one() returns **the first document** in the collection.
- Documents in MongoDB have **flexible structure**, but usually include:

- Simple fields: _id, borough, cuisine, name, restaurant_id
- A nested object address with building, street, zipcode, and coord (array of longitude, latitude)
- A grades array containing inspection records, each with date, grade, and score

3. Find All Documents (Equivalent to db.restaurants.find())

```
# Retrieve an iterable cursor for all documents
all_docs = restaurants.find()

# Display only the first 5 for readability
print("First 5 documents:")
for i, doc in enumerate(all_docs):
    if i >= 5:
        break
    print(doc)
```

- find() returns **a cursor** that can iterate over all documents.
- Mongo shell shows only 20 by default; in Python we can limit display manually.

4. Restricting to Documents Matching Criteria

Example 1: First restaurant in Brooklyn

```
brooklyn_restaurant = restaurants.find_one({"borough": "Brooklyn"})
print("First restaurant in Brooklyn:")
print(brooklyn_restaurant)
```

Example 2: First French restaurant in Brooklyn

```
french_brooklyn = restaurants.find_one({"borough": "Brooklyn", "cuisine": "French"})
print("\n First French restaurant in Brooklyn:")
print(french_brooklyn)
```

Example 3: First restaurant on Franklin Street

```
franklin_restaurant = restaurants.find_one({"address.street": "Franklin Street"})
print("\n First restaurant on Franklin Street:")
print(franklin_restaurant)
```

Example 4: First restaurant with a perfect inspection score (score = 0)

```
perfect_score = restaurants.find_one({"grades.score": 0})
print("\n First restaurant with score 0:")
print(perfect_score)
```

5. Using Projections to Select Fields

Show only name and borough

```
proj1 = restaurants.find_one({}, {"_id": 0, "name": 1, "borough": 1})
print("\n Name and borough only:")
```

```
print(proj1)
```

Show name, borough, address.street, and grades.grade

```
proj2 = restaurants.find_one({}, {
    "_id": 0, "name": 1, "borough": 1,
    "address.street": 1, "grades.grade": 1
})
print("\n  Name, borough, street, and grades:")
print(proj2)
```

Exclude address and grades

```
proj3 = restaurants.find_one({}, {"address": 0, "grades": 0})
print("\n  Name and other info (excluding address and grades):")
print(proj3)
```

- The second parameter of find_one() is a **projection**.
- Use 1 to include a field, 0 to exclude.
- _id is included by default; explicitly set _id: 0 to exclude it.

6. Complex Restrictions

Unfortunately, it is not possible to use the standard operators to apply restrictions more complex than strict equality. For example, to compare values (less than or greater than), we must use specific operators (`$lt` for less than, `$lte` for less than or equal, `$gt` for greater than, `$gte` for greater than or equal, `$ne` for not equal).

For instance, if we want to find restaurants with a score less than 5:

Example: Restaurants with score < 5

```
query_lt5 = {"grades.score": {"$lt": 5}}
proj_lt5 = {"_id": 0, "name": 1, "borough": 1}
lt5_restaurant = restaurants.find_one(query_lt5, proj_lt5)
print("\n First restaurant with score < 5:")
print(lt5_restaurant)
```

Example: Restaurants with cuisine in French or Italian

```
query_cuisine = {"cuisine": {"$in": ["French", "Italian"]}}
proj_cuisine = {"_id": 0, "name": 1, "borough": 1, "cuisine": 1}
cuisine_restaurant = restaurants.find_one(query_cuisine, proj_cuisine)
print("\n First French or Italian restaurant:")
print(cuisine_restaurant)
```

- $lt, $lte, $gt, $gte, $ne are used for numeric comparisons.
- $in tests if a field's value is in a given array.

7. Get Distinct Values (Equivalent to db.restaurants.distinct("borough"))

```
distinct_boroughs = restaurants.distinct("borough")
print("\n Distinct boroughs in the dataset:")
print(distinct_boroughs)
```

To Do:

Connect to MongoDB

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["test"]
restaurants = db["restaurants"]
```

1. List all cuisine types in the collection

2. List all possible grades in the database

3. Count the number of restaurants serving French cuisine

4. Count the number of restaurants located on "Central Avenue"

5. Count the number of restaurants with a score greater than 50

6. List all restaurants showing only name, building, and street

7. List all restaurants named “Burger King” (name and borough only)

8. List restaurants located on "Union Street" or "Union Square"

9. List restaurants located above latitude 40.90

10. List restaurants with a score of 0 and grade “A”

Complementary Questions

11. List restaurants (name and street only) located on a street containing "Union" in the name

12. List restaurants that had an inspection on February 1, 2014

13. List restaurants located between longitude -74.2 and -74.1 and latitude 40.1 and 40.2