Lecture 9 - Introducing PyMongo, and CRUD using PyMongo

Instructor: Suresh Melvin Sigera
Email: suresh.sigera@cuny.csi.edu

Agenda

- Creating Anaconda environment and installing PyMongo
- PyMongo
  - Connecting using PyMongo
  - Accessing the database
  - Getting a collection
  - Insert one document
  - Insert many documents
  - Retrieving documents
  - Update document
- QA

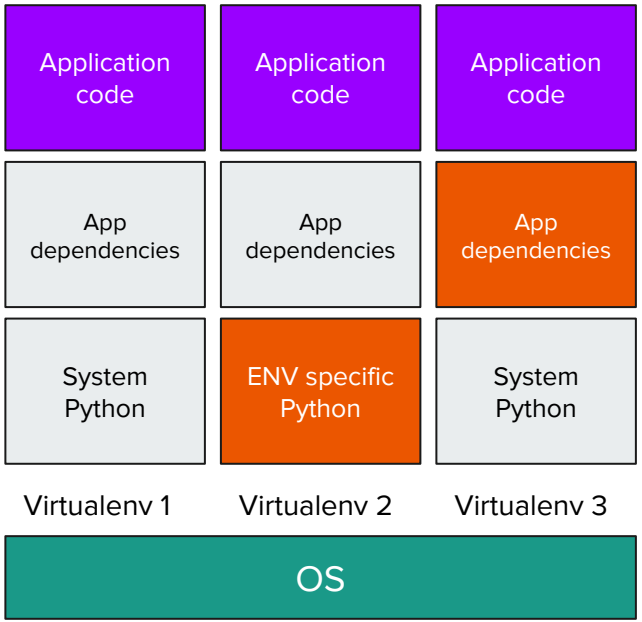
PyMongo - What is it?

PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.

Setting up Anaconda distribution for Python

Create a new environment named py35, install Python 3.5	conda create --name CSC424-MongoDB python=3.5
Activate the new environment to use it	WINDOWS: activate CSC424-MongoDB LINUX, macOS: source activate CSC424-MongoDB
Install pyMongo	conda install -c anaconda pymongo
Install pipreqs	pip install pipreqs

Setting up Anaconda distribution for Python



PyMongo - What is it?

PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.

PyMongo - Connecting

To establish a connection we'll use the MongoClient object. The first thing that we need to do in order to establish a connection is import the MongoClient class. We'll use this to communicate with the running database instance. Use the following code to do so:

```
import pymongo
client = pymongo.MongoClient('localhost', 27017)
```

PyMongo - Accessing the database

Once you have a connected instance of MongoClient, you can access any of the databases within that Mongo server. To specify which database you actually want to use, you can access it as an attribute:

```
db = client.website

#  you can also use the dictionary-style access:
# db = client['website']
```

Note: It doesn’t actually matter if your specified database has been created yet. By specifying this database name and saving data to it, you create the database automatically.

PyMongo - Getting a collection

A collection is a group of documents stored in MongoDB, and can be thought of as roughly the equivalent of a table in a relational database. Getting a collection in PyMongo works the same as getting a database:

```
posts_collection = db.posts.find()

# print all the posts
for post in posts_collection:
    print(post['title'])
```

PyMongo - Inserting one documents

Storing data in your database is as easy as calling just two lines of code. The first line specifies which collection you'll be using (posts in the example below). In MongoDB terminology, a collection is a group of documents that are stored together within the database. Collections and documents are akin to SQL tables and rows, respectively. Retrieving a collection is as easy as getting a database. We can insert the data in to the collection using the insert_one( ) method.

PyMongo - Inserting one documents

```
posts_collection = db.posts

single_post = {
    'title': "Java 101",
    "content": "Welcome to Java programming",
    "author": "Steve Harley"
}

# inserting the single post
result = posts_collection.insert_one(single_post)
```

PyMongo - Inserting many documents

```
posts_collection = db.posts

post_1 = {
    'title': 'Python and MongoDB',
    'content': 'PyMongo is fun, you guys',
    'author': 'Scott Smith'
}
post_2 = {
    'title': 'Anaconda Virtual Environments',
    'content': 'Use virtual environments',
    'author': 'Scott Smith'
}
```

PyMongo - Inserting many documents

```
post_3 = {
    'title': 'Learning Python and Amazon EC2',
    'content': 'Learn Python, it is easy',
    'author': 'Bill Smith'
}

# insert many
result = posts_collection.insert_many([post_1, post_2, post_3])

print('Multiple posts: {0}'.format(result.inserted_ids))
```

PyMongo - Retrieving documents

To retrieve a document, we'll use the find_one() method. The lone argument that we'll use here (although it supports many more) is a dictionary that contains fields to match. In our example below, we want to retrieve the post that was written by Steve Harley:

```
posts_collection = db.posts

scotts_posts = posts_collection.find({'author': 'Steve Harley'})

print(scotts_posts['title'])
```

PyMongo - Retrieving documents

Note that document data isn’t returned directly to us as an array. Instead we get an instance of the Cursor object. This Cursor is an iterable object that contains quite a few helper methods to help you work with the data. To get each document, just iterate over the result:

```
# print(scotts_posts)
for post in scotts_posts:
    print(post['title'])
```

PyMongo - Update document

You can update a record, or document as it is called in MongoDB, by using the update_one() method. The first parameter of the update_one() method is a query object defining which document to update.

PyMongo - Update document

```
posts_collection = db.posts

posts = posts_collection.find()

query = {"title": "Python and MongoDB"}
update_query = {"$set": {"title": "Python and MongoDB for developers"}}

posts_collection.update_one(query, update_query)

for post in posts:
    print(post['title'])
```

QA