Pymongo is a python delivery containing tools for working with mongodB, this is the suggested way to work with MongodB from Python.

After downloaded and installed MongodB ensure it is running on the default host and port

The next step is to create a MongoClient to the running the Mongod instance this can be done in a python shell with the command. This command will connect the default host and port.

**>>>>> from Pytmongo import MongoClient**

**>>>>>>>>>>> Client = mongoClient ()**

**NOTE** we can also specify the host and port explicitly with the following command.

**Client = MongoClient(‘localhost’, we insert number here)**

**Getting a Database**

A single instance of MongodB can support multiple independent databases. When working with PyMongo you access databases using attribute style access on MongoClient instances:

**>>> db = client.test\_database**

**Getting a Collection**

A collection is a group of documents stored in MongoDB, as I explained in my MongodB doc, remember a collection equivalent of a table in a relational database. Getting a collection in PyMongo works the same as getting a database:

**>>> collection = db.test\_collection**

**Documents**

Data in MongoDB is represented (and stored) using JSON-style documents. In PyMongo we use dictionaries to represent documents.

Eg on blog post

>>> import datetime

>>> post = {"author": "Mike",

... "text": "My first blog post!",

... "tags": ["mongodb", "python", "pymongo"],

... "date": datetime.datetime.utcnow()}

**Note** documents can contain characteristic Python types (like datetime.datetime instances) which will be automatically converted to and from the appropriate BSON types.

**Inserting a Document**

To insert a document into a collection we can use the insert\_one() method:

>>> posts = db.posts

>>> post\_id = posts.insert\_one(post).inserted\_id

>>> post\_id

ObjectId('...')

**Getting a Single Document With find\_one()**

The most straightforward sort of query that can be performed in MongoDB is **find\_one()**. This method returns a single document matching a query (or None if there are no matches).

**find\_one()** also supports querying on exact elements that the resulting document must match. To limit our results to a document with author “Mike” you do:

pprint.pprint(posts.find\_one({"author": "Mike"}))

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'mongodb', u'python', u'pymongo'],

u'text': u'My first blog post!'}

If we try with a different author, like “Eliot”, we’ll get no result:

>>> posts.find\_one({"author": "Eliot"})

Querying By ObjectId

We can also find a post by its \_id, which in our example is an ObjectId:

>>> post\_id

ObjectId(...)

>>> pprint.pprint(posts.find\_one({"\_id": post\_id}))

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'mongodb', u'python', u'pymongo'],

u'text': u'My first blog post!'}

Note that an ObjectId is not the same as its string representation:

>>> post\_id\_as\_str = str(post\_id)

>>> posts.find\_one({"\_id": post\_id\_as\_str}) # No resul

A common task in web applications is to get an ObjectId from the request URL and find the matching document. It’s necessary in this case to convert the ObjectId from a string before passing it to find\_one:

from bson.objectid import ObjectId

# The web framework gets post\_id from the URL and passes it as a string

def get(post\_id):

# Convert from string to ObjectId:

document = client.db.collection.find\_one({'\_id': ObjectId(post\_id)})

**Python strings**

Python strings look different when retrieved from the server (e.g. u’Mike’ instead of ‘Mike’). A short explanation is in order.

MongoDB stores data in BSON format. BSON strings are UTF-8 encoded so PyMongo must ensure that any strings it stores contain only valid UTF-8 data. Regular strings (<type ‘str’>) are validated and stored unaltered. Unicode strings (<type ‘unicode’>) are encoded UTF-8 first. The reason our example string is represented in the Python shell as u’Mike’ instead of ‘Mike’ is that PyMongo decodes each BSON string to a Python unicode string, not a regular str.

**Bulk Inserts**

In addition to inserting a single document, you can also perform bulk insert operations, by passing a list as the first argument to insert\_many(). This will insert each document in the list, sending only a single command to the server:

>>> new\_posts = [{"author": "Mike",

... "text": "Another post!",

... "tags": ["bulk", "insert"],

... "date": datetime.datetime(2009, 11, 12, 11, 14)},

... {"author": "Eliot",

... "title": "MongoDB is fun",

... "text": "and pretty easy too!",

... "date": datetime.datetime(2009, 11, 10, 10, 45)}]

>>> result = posts.insert\_many(new\_posts)

>>> result.inserted\_ids

[ObjectId('...'), ObjectId('...')]

•The result from insert\_many() now returns two ObjectId instances, one for each inserted document.

•new\_posts[1] has a different “shape” than the other posts - there is no "tags" field and we’ve added a new field, "title". This is what we mean when we say that MongoDB is schema-free.

**Querying for More Than One Document**

To get more than a single document as the result of a query we use the find() method. find() returns a Cursor instance, which allows us to iterate over all matching documents. For example, we can iterate over every document in the posts collection:

>>> for post in posts.find():

... pprint.pprint(post)

...

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'mongodb', u'python', u'pymongo'],

u'text': u'My first blog post!'}

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'bulk', u'insert'],

u'text': u'Another post!'}

{u'\_id': ObjectId('...'),

u'author': u'Eliot',

u'date': datetime.datetime(...),

u'text': u'and pretty easy too!',

u'title': u'MongoDB is fun'}

You we can pass a document to find() to limit the returned results. Here, we get only those documents whose author is “Mike”:

>>> for post in posts.find({"author": "Mike"}):

... pprint.pprint(post)

...

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'mongodb', u'python', u'pymongo'],

u'text': u'My first blog post!'}

{u'\_id': ObjectId('...'),

u'author': u'Mike',

u'date': datetime.datetime(...),

u'tags': [u'bulk', u'insert'],

u'text': u'Another post!'}