To create connection between Python programming language and MongoDB database, we need to first install **pymongo** driver.

**$ pip install pymongo**

## Making a Connection with MongoClient

The first step when working with **PyMongo** is to create a [MongoClient](https://api.mongodb.com/python/current/api/pymongo/mongo_client.html#pymongo.mongo_client.MongoClient) to the running **mongod** instance.

**>>> from** **pymongo** **import** MongoClient

**>>>** client = MongoClient()

The above code will connect on the default host and port. We can also specify the host and port explicitly, as follows:

**>>>** client = MongoClient('localhost', 27017)

Or use the MongoDB URI format:

**>>>** client = MongoClient('mongodb://localhost:27017/')

## Getting a Database

A single instance of MongoDB can support multiple independent [databases](http://www.mongodb.org/display/DOCS/Databases). When working with PyMongo you access databases using attribute style access on [MongoClient](https://api.mongodb.com/python/current/api/pymongo/mongo_client.html#pymongo.mongo_client.MongoClient) instances:

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

If your database name is such that using attribute style access won’t work (like test-database), you can use dictionary style access instead:

**>>>** db = client['test-database']

## Getting a Collection

A [collection](http://www.mongodb.org/display/DOCS/Collections) 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:

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

or (using dictionary style access):

**>>>** collection = db['test-collection']

An important note about collections (and databases) in MongoDB is that they are created lazily - none of the above commands have actually performed any operations on the MongoDB server. Collections and databases are created when the first document is inserted into them.

## Documents

Data in MongoDB is represented (and stored) using JSON-style documents. In PyMongo we use dictionaries to represent documents. As an example, the following dictionary might be used to represent a blog post:

**>>> import** **datetime**

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

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

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

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

Note that documents can contain native Python types (like [datetime.datetime](https://docs.python.org/3/library/datetime.html#datetime.datetime) instances) which will be automatically converted to and from the appropriate [BSON](http://www.mongodb.org/display/DOCS/BSON) types.

## Inserting a Document

To insert a document into a collection we can use the [insert\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_one) method:

**>>>** posts = db.posts

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

**>>>** post\_id

ObjectId('...')

When a document is inserted a special key, "\_id", is automatically added if the document doesn’t already contain an "\_id" key. The value of "\_id" must be unique across the collection. [insert\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.insert_one) returns an instance of[InsertOneResult](https://api.mongodb.com/python/current/api/pymongo/results.html#pymongo.results.InsertOneResult). For more information on "\_id", see the [documentation on \_id](http://www.mongodb.org/display/DOCS/Object+IDs).

After inserting the first document, the *posts* collection has actually been created on the server. We can verify this by listing all of the collections in our database:

**>>>** db.list\_collection\_names()

[u'posts']

## Getting a Single Document With [find\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.find_one)

The most basic type of query that can be performed in MongoDB is [find\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.find_one). This method returns a single document matching a query (or None if there are no matches). It is useful when you know there is only one matching document, or are only interested in the first match. Here we use [find\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.find_one) to get the first document from the posts collection:

**>>> import** **pprint**

**>>>** pprint.pprint(posts.find\_one())

{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!'}

The result is a dictionary matching the one that we inserted previously.

**Note**

The returned document contains an "\_id", which was automatically added on insert.

[find\_one()](https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection.find_one) also supports querying on specific elements that the resulting document must match. To limit our results to a document with author “Mike” we 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 result*

>>>

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)})

To delete one document, we use the delete\_one() method.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
myquery = { "address": "Mountain 21" }  
  
mycol.delete\_one(myquery)

Delete Many Documents

To delete more than one document, use the delete\_many() method.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
myquery = { "address": {"$regex": "^S"} }  
  
x = mycol.delete\_many(myquery)  
  
print(x.deleted\_count, " documents deleted.")

Update Collection

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.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
myquery = { "address": "Valley 345" }  
newvalues = { "$set": { "address": "Canyon 123" } }  
  
mycol.update\_one(myquery, newvalues)  
  
#print "customers" after the update:  
for x in mycol.find():  
  print(x)

## Update Many

To update *all* documents that meets the criteria of the query, use the update\_many() method.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
myquery = { "address": { "$regex": "^S" } }  
newvalues = { "$set": { "name": "Minnie" } }  
  
x = mycol.update\_many(myquery, newvalues)  
  
print(x.modified\_count, "documents updated.")

[Run example »](https://www.w3schools.com/python/showpython.asp?filename=demo_mongodb_update_many)

## Find One

To select data from a collection in MongoDB, we can use the find\_one() method.

The find\_one() method returns the first occurrence in the selection.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
x = mycol.find\_one()  
  
print(x)

## Find All

To select data from a table in MongoDB, we can also use the find() method.

The find() method returns all occurrences in the selection.

The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
for x in mycol.find():  
  print(x)

## Return Only Some Fields

The second parameter of the find() method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
for x in mycol.find({},{ "\_id": 0, "name": 1, "address": 1 }):  
  print(x)

## Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the insert\_one() method.

The first parameter of the insert\_one() method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

## Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use the insert\_many() method.

The first parameter of the insert\_many() method is a list containing dictionaries with the data you want to insert:

import pymongo  
  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
mydb = myclient["mydatabase"]  
mycol = mydb["customers"]  
  
mylist = [  
  { "name": "Amy", "address": "Apple st 652"},  
  { "name": "Hannah", "address": "Mountain 21"},  
  { "name": "Michael", "address": "Valley 345"},  
  { "name": "Sandy", "address": "Ocean blvd 2"},  
  { "name": "Betty", "address": "Green Grass 1"},  
  { "name": "Richard", "address": "Sky st 331"},  
  { "name": "Susan", "address": "One way 98"},  
  { "name": "Vicky", "address": "Yellow Garden 2"},  
  { "name": "Ben", "address": "Park Lane 38"},  
  { "name": "William", "address": "Central st 954"},  
  { "name": "Chuck", "address": "Main Road 989"},  
  { "name": "Viola", "address": "Sideway 1633"}  
]  
  
x = mycol.insert\_many(mylist)  
  
#print list of the \_id values of the inserted documents:  
print(x.inserted\_ids)

**from** pymongo import MongoClient # import mongo client **to** **connect**

import pprint

# Creating instance **of** mongoclient

client = MongoClient()

# Creating **database**

db = client.javatpoint

employee = {"id": "101",

"name": "Peter",

"profession": "Software Engineer",

}

# Creating document

employees = db.employees

# Inserting data

employees.insert\_one(employee)

# Fetching data

pprint.pprint(employees.find\_one())