HOW TO USE PYMONGO

In this tutorial, I'll walk you through how to use PyMongo in MongoDB.

(This tutorial is part of our MongoDB Guide. Use the right-hand menu to navigate.)

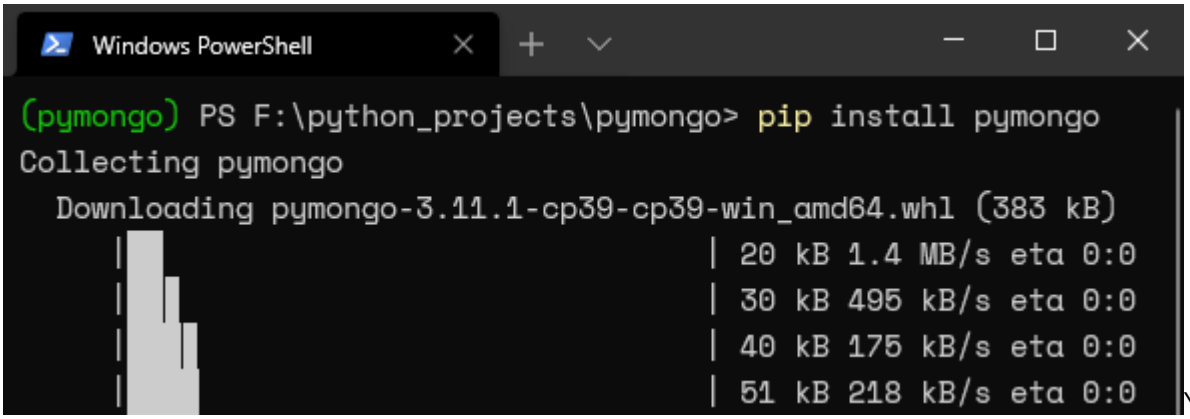
What is PyMongo?

PyMongo is the official Python driver that connects to and interacts with MongoDB databases. The PyMongo library is being actively developed by the MongoDB team.

Installing PyMongo

PyMongo can be easily installed via pip on any platform.

```
pip install pymongo
```



You can verify the

PyMongo installation via pip or by trying to import the module to a Python file.

Checking via pip:

```
pip list --local
```

```
(pymongo) PS F:\python_projects\pymongo> pip list --local
Package    Version
---------- -------
pip        20.2.4
pymongo    3.11.1
setuptools 49.2.1
(pymongo) PS F:\python_projects\pymongo>
```

Let us create a simple Python file where we import the PyMongo module. The import statement is wrapped in a try-except statement to write a message to the output based on the success or failure of the import statement.

```
try:
import pymongo
print("Module Import Successful")
except ImportError as error:
print("Module Import Error")
print(error)
```

```
[Running] python -u "f:\python_projects\pymongo\test_import.py"
Module Import Successful
```

How to use PyMongo

In this section, we will cover the basic usage of the PyMongo module and how it can be used to interact with a MongoDB installation. The MongoDB server details used in this article are:

- IP: 10.10.10.59
- Port: 27017

/etc/mongod.conf

```
# network interfaces
net:
port: 27017
bindIp: 127.0.0.1, 10.10.10.59
```

To allow access to the database, you need to add the external IP of the server to the bindIp parameter. You can find it in the network section of the “mongod.conf” file.

Here are the actions that I’ll show you how to do:

- Connect to MongoDB install
- Obtain collections from database

Connecting to the MongoDB install

There are two ways to define the connection string:

- Using IP and Port
- Using a MongoURL

```
from pymongo import MongoClient
client = MongoClient('10.10.10.59', 27017)
# Alternative Connection Method - MongoURL Format
# client = MongoClient('mongodb://10.10.10.59:27017')
print(client.server_info())
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\connection.py"
{'version': '4.4.2', 'gitVersion': '15e73dc5738d2278b688f8929aee605fe4279b0e', 'modules': [], 'allocator':
'tcmalloc', 'javascriptEngine': 'mozjs', 'sysInfo': 'deprecated', 'versionArray': [4, 4, 2, 0], 'openssl':
{'running': 'OpenSSL 1.1.1f  31 Mar 2020', 'compiled': 'OpenSSL 1.1.1f  31 Mar 2020'}, 'buildEnvironment':
{'distmod': 'ubuntu2004', 'distarch': 'x86_64', 'cc': '/opt/mongodbtoolchain/v3/bin/gcc: gcc (GCC) 8.2.0',
'ccflags': '-fno-omit-frame-pointer -fno-strict-aliasing -fasynchronous-unwind-tables -ggdb -pthread -Wall
```

In this example, we have established the connection by creating a new instance of the MongoClient Class with the MongoDB server IP and port. Then we call the **server_info()** function to obtain the data about the MongoDB server instance.

In the next example, we call the function **list_database_names()** to get a list of all the databases.

```
from pymongo import MongoClient
client = MongoClient('10.10.10.59', 27017)
print(client.list_database_names())
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\connection.py"
['admin', 'config', 'local', 'students', 'vehicles']
```

Obtaining collections from a database

Using the **list_collection_names** method, we can obtain a list of collections from a specified database object. The **include_system_collection** is set to False to exclude any collections created by the system itself.

```
from pymongo import MongoClient
# Create Connection
```

```
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
# Alternative Declaration Method
# database = client
# Get List of Collections
collection = database.list_collection_names(include_system_collections=False)
print(collection)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
['student_grades']
```

Finding documents in a collection

You can use the **find()** method to search for any document in a Collection. Let's do an example to understand it better.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
# Get Details of the specified Collection
collection = database
# Print each Document
for studentinfo in collection.find():
print(studentinfo)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
{'_id': ObjectId('5fbed0cba94306028d2d751f'), 'name': 'Barry', 'grades': [85.0, 68.0, 75.0, 95.0, 100.0]}
{'_id': ObjectId('5fbed0faa94306028d2d7520'), 'name': 'Sally', 'grades': [77.0, 65.0, 60.0]}
{'_id': ObjectId('5fbed116a94306028d2d7521'), 'name': 'Jake', 'grades': [100.0, 97.0]}
{'_id': ObjectId('5fbed133a94306028d2d7522'), 'name': 'Mike', 'grades': [64.0, 70.0, 55.0, 51.0]}
```

You can use several other MongoDB operators, like projection, push and pull, and sort, within the find() method to filter the output.

In our next example, we use the MongoDB projection operator to filter out the "_id" field and sort the resulting data set in descending order.

```
from pymongo import MongoClient

# Create Connection
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
```

```
# Get Details of the specified Collection
collection = database
# Print each Document
for studentinfo in collection.find({},{'_id':0}).sort('name',-1):
print(studentinfo)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
{'name': 'Sally', 'grades': [77.0, 65.0, 60.0]}
{'name': 'Mike', 'grades': [64.0, 70.0, 55.0, 51.0]}
{'name': 'Jake', 'grades': [100.0, 97.0]}
{'name': 'Barry', 'grades': [85.0, 68.0, 75.0, 95.0, 100.0]}
```

Finding one document from a collection

PyMongo provides the find_one() and find() methods to find a single document from a collection:

- The **find_one() method** will return the first document according to the given conditions.
- The **find() method** will return a Cursor object, which is an iterable object that contains additional helper methods to transform the data.

The find_one() Method

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
# Get Details of the specified Collection
collection = database
studentinfo = collection.find_one({'name':'Mike'})
print(studentinfo)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
{'_id': ObjectId('5fbed133a94306028d2d7522'), 'name': 'Mike', 'grades': [64.0, 70.0, 55.0, 51.0]}
```

The find() method

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
# Get Details of the specified Collection
collection = database
studentinfo = collection.find({'name':'Mike'})
```

```
print(studentinfo)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
<pymongo.cursor.Cursor object at 0x000001CB4E7C0D60>
```

To obtain a readable output, we can use a loop to iterate the object.

```
from pymongo import MongoClient

# Create Connection
client = MongoClient('10.10.10.59', 27017)
# Select the Database
database = client.students
# Get Details of the specified Collection
collection = database
studentinfo = collection.find({'name':'Mike'})
# print(studentinfo)
# Print each Document
for student in studentinfo:
print(student)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\get_details.py"
{'_id': ObjectId('5fbed133a94306028d2d7522'), 'name': 'Mike', 'grades': [64.0, 70.0, 55.0, 51.0]}
```

Creating a database, collection & document

In this section, we'll look at how to create a database and a collection, and how to insert documents into the new collection. MongoDB will not create a database until there is at least one document in the database.

The following example demonstrates how to create a database called "food" with a "dessert" collection with a single document using the **insert_one()** method. PyMongo automatically creates a new database if the selected database is not available.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
print(f"Existing Databases")
print(client.list_database_names())
# Create a New Database - dictionary style
database_foods = client
# Alternative method for simple strings #
# database = client.foods
# Create a Collection - dictionary style
col_desserts = database_foods
# Insert a Single Document
```

```
document = {'name': 'chocolate cake', 'price': 20, 'ingredients':}
col_desserts.insert_one(document)
print(f"\nVerify the New Database")
print(client.list_database_names())
print(f"\nCollections in the Database foods")
print(database_foods.list_collection_names())
print(f"\nDocuments in the desserts Collection")
dessert = col_desserts.find()
# Print each Document
for des in dessert:
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\create.py"
Existing Databases
['admin', 'config', 'local', 'students', 'vehicles']

Verify the New Database
['admin', 'config', 'foods', 'local', 'students', 'vehicles']

Collections in the Database foods
['desserts']

Documents in the desserts Collection
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 20, 'ingredients':
['chocolate', 'flour', 'eggs']}
```

Inserting documents to a collection

PyMongo provides two methods to insert documents as insert_one() and insert_many():

- **Insert_one()** is used to insert a single document
- **Insert_many()** inserts multiple documents.

Let's see a couple of examples of that with the "desserts" collection.

Inserting a single document

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
print(f"\nAll Documents in the desserts Collection")
dessert = col_desserts.find()
# Print each Document
for des in dessert:
print(des)
# Insert a Single Document
```

```
document = {'name': 'ice cream', 'price': 10, 'ingredients':}
col_desserts.insert_one(document)
print(f"\nAfter Insert")
dessert = col_desserts.find()
# Print each Document
for des in dessert:
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\insert.py"

All Documents in the desserts Collection
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 20, 'ingredients':
['chocolate', 'flour', 'eggs']}

After Insert
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 20, 'ingredients':
['chocolate', 'flour', 'eggs']}
{'_id': ObjectId('5fc5b76db3c385c04f01245f'), 'name': 'ice cream', 'price': 10, 'ingredients': ['milk',
'vanilla extract', 'eggs', 'heavy cream', 'sugar']}
```

Inserting multiple documents

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
documents = },
{'name': 'chocolate chip cookies', 'price': 3, 'ingredients':},
{'name': 'banana pudding', 'price': 10, 'ingredients':}
]
col_desserts.insert_many(documents)
dessert = col_desserts.find({},{'_id':0})
# Print each Document
for des in dessert:
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\insert.py"
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 20, 'ingredients':
['chocolate', 'flour', 'eggs']}
{'_id': ObjectId('5fc5b76db3c385c04f01245f'), 'name': 'ice cream', 'price': 10, 'ingredients': ['milk',
'vanilla extract', 'eggs', 'heavy cream', 'sugar']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8ef'), 'name': 'pumpkin pie', 'price': 15, 'ingredients': ['pumpkin',
'condensed milk', 'eggs', 'cinnamon']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8f0'), 'name': 'chocolate chip cookies', 'price': 3, 'ingredients':
['chocolate chips', 'butter', 'white sugar', 'flour', 'vanilla extract']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8f1'), 'name': 'banana pudding', 'price': 10, 'ingredients':
['banana', 'white sugar', 'milk', 'flour', 'salt']}
```

Updating documents

The methods **update_one()** and **update_many()** are used to update a single document or multiple documents. The following examples demonstrate how each of these methods can be used.

Updating a single document

In this example, we update the "price" field of a document specified by the "_id" field. For this, you need to import the ObjectId class from the "bson" module.

In this example, we have updated the price of a chocolate cookie to be 5.

```
from pymongo import MongoClient
from bson.objectid import ObjectId
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
document = {'_id': ObjectId('5fc5ebe4abb99571485ce8f0')}
updated_values = {'$set': {'price': 5}}
# Update the Document
col_desserts.update_one(document, updated_values)
# Find the Updated Document
updated_doc = col_desserts.find_one({'_id':
ObjectId('5fc5ebe4abb99571485ce8f0')})
print(updated_doc)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\update.py"
{'_id': ObjectId('5fc5ebe4abb99571485ce8f0'), 'name': 'chocolate chip cookies', 'price': 5, 'ingredients':
['chocolate chips', 'butter', 'white sugar', 'flour', 'vanilla extract']}
```

Updating multiple documents

In this example, we find all the documents whose "name" starts with the string "chocolate" using the $regex operator. Then we increase their price by 5 using the **$inc** operator.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
# Find Documents that starts with chocolate
document = {"name": { "$regex": "^chocolate"}}
# Increase the existing value by five
updated_values = {'$inc': {'price': 5}}
# Update the Documents
col_desserts.update_many(document, updated_values)
# Find the Updated Documents
for des in col_desserts.find({},{'_id':0}):
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\update.py"
{'name': 'chocolate cake', 'price': 25, 'ingredients': ['chocolate', 'flour', 'eggs']}
{'name': 'ice cream', 'price': 10, 'ingredients': ['milk', 'vanilla extract', 'eggs', 'heavy cream', 'sugar']}
{'name': 'pumpkin pie', 'price': 15, 'ingredients': ['pumpkin', 'condensed milk', 'eggs', 'cinnamon']}
{'name': 'chocolate chip cookies', 'price': 10, 'ingredients': ['chocolate chips', 'butter', 'white sugar',
'flour', 'vanilla extract']}
{'name': 'banana pudding', 'price': 10, 'ingredients': ['banana', 'white sugar', 'milk', 'flour', 'salt']}
```

Deleting documents, collections & databases

Let us look at how "delete" works within PyMongo:

- The **delete_one()** method deletes a single document.
- The **delete_many()** method deletes multiple files.
- The **drop()** method deletes collections.
- The **drop_database()** method deletes a database.

Let's look at an example for each use case.

Deleting a single document

We will delete a document using the "_id" field as the identifier.

```
from pymongo import MongoClient
from bson.objectid import ObjectId
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
document = {'_id': ObjectId('5fc5ebe4abb99571485ce8f0')}
# Delete the Document
col_desserts.delete_one(document)
```

```
# Find the Remaining Documents
for des in col_desserts.find():
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\delete.py"
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 25, 'ingredients':
['chocolate', 'flour', 'eggs']}
{'_id': ObjectId('5fc5b76db3c385c04f01245f'), 'name': 'ice cream', 'price': 10, 'ingredients': ['milk',
'vanilla extract', 'eggs', 'heavy cream', 'sugar']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8ef'), 'name': 'pumpkin pie', 'price': 15, 'ingredients': ['pumpkin',
'condensed milk', 'eggs', 'cinnamon']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8f1'), 'name': 'banana pudding', 'price': 10, 'ingredients':
['banana', 'white sugar', 'milk', 'flour', 'salt']}
```

Deleting multiple documents

We will delete all documents where the price is equal to or less than 10 from the "desserts" collection.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
database_foods = client
col_desserts = database_foods
# Select only documents where price is less than or equal to 10
document = {'price': { '$lte': 10 }}
# Delete the Document
col_desserts.delete_many(document)
# Find the Remaining Documents
for des in col_desserts.find():
print(des)
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\delete.py"
{'_id': ObjectId('5fc5b1dceee42b7f3ad5e827'), 'name': 'chocolate cake', 'price': 25, 'ingredients':
['chocolate', 'flour', 'eggs']}
{'_id': ObjectId('5fc5ebe4abb99571485ce8ef'), 'name': 'pumpkin pie', 'price': 15, 'ingredients': ['pumpkin',
'condensed milk', 'eggs', 'cinnamon']}
```

Deleting a collection

A MongoDB collection can be deleted using the drop() method. Here's how to drop the complete "desserts" collection from our food database.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
```

```
database_foods = client
# Check currently available collections
print(database_foods.list_collection_names())
# Delete the collection
col_desserts = database_foods
col_desserts.drop()
# Print the remaining collections
print(database_foods.list_collection_names())
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\delete.py"
['desserts']
[]
```

The second print statement is empty as we have deleted the only collection from the “foods” database.

Deleting the database

The example below shows you how to delete a complete database with all its collections and documents. We will delete the “students” using the drop_database() method.

```
from pymongo import MongoClient
# Create Connection
client = MongoClient('10.10.10.59', 27017)
# List all databases
print(client.list_database_names())
# Check for Data within the database
database_students = client
print('')
print(database_students.list_collection_names())
# Delete the complete database
client.drop_database('students')
# Verify the Deletion
print('')
print(client.list_database_names())
```

Result:

```
[Running] python -u "f:\python_projects\pymongo\delete.py"
['admin', 'config', 'local', 'students', 'vehicles']

['student_grades']

['admin', 'config', 'local', 'vehicles']
```

As you can see, the “students” database is completely removed.

That’s it for this tutorial. If you want to learn further about PyMongo, the best source is their documentation.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Guide, a series of articles and tutorials
- Creating a Database in MongoDB
- MongoDB: The Mongo Shell & Basic Commands
- Data Storage Explained: Data Lake vs Warehouse vs Database