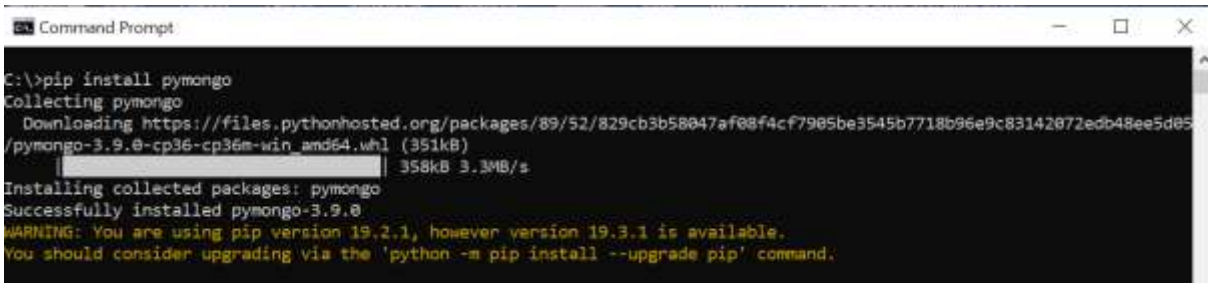
MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. We are learning MongoDB Connectivity with Python.

MongoDB Installation: https://www.youtube.com/watch?v=uBsrlV-aU80



To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
testdb = myclient["testdatabase"]
```

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

```
print(myclient.list_database_names())
```

Or you can check a specific database by name:

```
dblist = myclient.list_database_names()
if "testdatabase" in dblist:
  print("The database exists.")
```

Python MongoDB Create Collection

A **collection** in MongoDB is the same as a **table** in SQL databases.

To create a collection in MongoDB, use database object and specify the name of the collection you want to create. MongoDB will create the collection if it does not exist.

```
testdb = myclient["testdatabase"]
retailcollection = testdb["stores"]
```

In MongoDB, a collection is not created until it gets content! MongoDB waits until you have inserted a document before it actually creates the collection.

Return a list of all collections in your database. But you won't find "stores" collection as it doesn't have any content yet:

```
collist = testdb.list_collection_names()

print(collist)
if "stores" in collist:
  print("The collection exists.")
```

Python MongoDB Insert Document

A **document** in MongoDB is the same as a **record** in SQL databases.

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.

```
oneStore = { "store_number": "0121",
"store_name": "cumberland", "address": "121 Cumberland Pwky" }

x = retailcollection.insert_one(oneStore)
```

The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.

```
print(x.inserted_id)
```

If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.

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

```
mylist = [
      { "store_number": "0111", "store_name": "Duluth", "address": "111
Duluth Pwky" },

{ "store_number": "0131", "store_name": "Spring Hill", "address": "131
Spring Hill Pkwy" }

]

x = retailcollection.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

If you do not want MongoDB to assign unique ids for you document, you can specify the _id field when you insert the document(s).

Remember that the values have to be unique. Two documents cannot have the same _id.

```
mylist = [
      {  "_id": 1, "store_number": "141", "store_name": "Somewhere
", "address": "141 Duluth Pwky" },
{ "_id": 2, "store_number": "0151", "store_name": "Vinings
Hill", "address": "151 Vinings Hill Pkwy" }
]

x = retailcollection.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

Python MongoDB Find

In MongoDB we use the **find** and **findOne** methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
testdb = myclient["mydatabase"]
retailcollection = testdb["stores"]

x = retailcollection.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.

```
for x in retailcollection.find():
  print(x)
```

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. Where "0" – Drop Field and "1" – Show field

```
for x in retailcollection.find({},{ "_id": 0, "
store_number": 1, "address": 1 }):
  print(x)
```

Python MongoDB Query

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

```
myquery = { "address": "Park Lane 38" }

mydoc = retailcollection.find(myquery)

for x in mydoc:
  print(x)
```

To make advanced queries you can use modifiers as values in the query object.

E.g. to find the documents where the "address" field starts with the letter "S" or higher (alphabetically), use the greater than modifier: {"$gt": "S"}:

```
myquery = { "address": { "$gt": "S" } }

mydoc = retailcollection.find(myquery)

for x in mydoc:
  print(x)
```

Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

```
mydoc = retailcollection.find().sort("name")

for x in mydoc:
  print(x)
```

Use the value -1 as the second parameter to sort descending.

sort("name", 1) #ascending
sort("name", -1) #descending

Delete Document

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

The first parameter of the delete_one() method is a query object defining which document to delete.

If the query finds more than one document, only the first occurrence is deleted.

```
myquery = { "address": "Mountain 21" }

retailcollection.delete_one(myquery)
```

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

The first parameter of the delete_many() method is a query object defining which documents to delete.

```
myquery = { "address": {"$regex": "^S"} }

x = retailcollection.delete_many(myquery)

print(x.deleted_count, " documents deleted.")
```

To delete all documents in a collection, pass an empty query object to the delete_many() method:

```
x = retailcollection.delete_many({})

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.

```
myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }

retailcollection.update_one(myquery, newvalues)

#print "stores" after the update:
for x in retailcollection.find():
  print(x)
```

Update Many

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

```
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }

x = retailcollection.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")
```