MongoDB - A NoSQL Database

```
      ##############   ############   Introduction to MongoDB    ##############    ################

=>  MongoDB is No-SQL Database.

=>  It stores data in Json like document, which makes the database very flexible & scalable.

=>  MongoDB can be downloaded from   ->  https://www.mongodb.com/try/download/

=>  Python needs a MongoDB driver to access the MongoDb database like as 'pymongo', which can be installed
       directly using the Pip command.

       >>>  pip install pymongo

=>  Test the 'pymongo' installation by importing it  ->   >>>  import pymongo
```

```
     ##############   ##############   Creating a database in MongoDB   ##############    ################

=>  To create a database in MongoDB, we start by creating a MongoClient object, then specify a connection URL
      with the corect ip address & the name of database, we want to create.

=>  MongoDB will create the database, if it does not exist and make a connection to it.
```

Create a MongoDB database called "mydatabase"

```
# Importing the Python MongoDB  Driver
import pymongo

#  Specifying the correct URL to the MongoDB object
myclient = pymongo.MongoClient("mongodb://localhost:27017/")

#  Creating the 'mydatabase'
mydb = myclient['mydatabase']
```

```
 =>  In MongoDB, a Database is not created until it gets something stored as content.

 =>  MongoDB waits until we cretes a collection (Table), with atleast one document (Record), before it actually
       creates the database (and collection).
```

Check the existence of database in MongoDB

```
 =>  In MongoDb, a database is not creted until, it gets content, so crete collection and create a document,
       then we can check for the existence of database.

 =>  Return a list of all existing databases of your system.

      >>>  print (myclient.list_databases-names())

 =>  Check the existence of a specific database by its name.

      >>>  dblist = myclient.list_databases-names()

      >>>  if "mydatabase" in dblist:
              print ('Yes! This database exists')
```

Create a Collection or Table in MongoDB

```
 =>  A Collection in MongoDB is same as Table in SQl databases.

 =>  To create a collection, we uses the databases object and specify the name of collection and MongoDB will
       create the collection if it not exists.

 =>  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.
```

```
#  Creating a Collection with name 'customers' in MongoDB database

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

# Creating the collection
mycol = mydb["customers"]
mycol
```

Check if Collection Exists

In [ ]:
```
=>  In MongoDB, a Collection is not created until it gets content, so if this is your first time creating
      a collection, you should complete the next chapter (create document) before you check
        if the collection exists! or not.
```

```
=>  You can check if a collection exist in a database by listing all collections of the database.

=>  Return a list of all collections in your database.

      >>>  print(mydb.list_collection_names())

=>  Check if the "customers" collection exists in database.

      >>>  collist = mydb.list_collection_names()

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

Inserting documents in MongoDB database

In [ ]:
```
      ##############    ###########   Introduction to the Documents    #############   ##############

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

=>  insert_one() is used to insert a record, or document into a collection in MongoDB database.

=>  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 a record in the "customers" collection:
           mydict = { "name": "John", "address": "Highway 37" }
           x = mycol.insert_one(mydict)

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

      >>>  x.inserted_id

=>  If we do not specify an _id field, then MongoDB will add one for you and assign a unique id for each documents.
      In above code, it has assigned a default and unique _id for the document.

=>  insert_many() is used to insert multiple documents at a time to the collection in MongoDB.

=>  The first parameter of the insert_many() method is a list containing dictionaries with the data
      we want to insert in the collection.

=>  "_id" is used to specify the _id Fields of Documents at their time of insertions into collection.

=>  For getting the specified '_id' of all documents, we uses  ->   >>> x.inserted_ids
```

```
#  Insertion of multiple documents to the collection in MongoDB
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

#  Creating a list of multiple documents with specific _id Fields
mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

# Inserting the Documents List
x = mycol.insert_many(mylist)

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

Selection of data from Collections in MongoDB database

In [ ]:
```
=>  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.

=>  find_one() is used to return the 1st occurrence of collection.

     >>> mycol.find_one()
```

```
=>  find_many() is used to return the all occurrences in 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.

     >>> mycol.find_many()

=>  No parameters in the find() method gives you the same result as SELECT * in MySQL.

=>  For returning only some of the Fields in selection, we uses 2nd parameter of find(). This parameter is
      optional and it describes which Fields to include in the result. If this parameter is omitted,
        then all Fields will be included in the result.

      >>>  for x in mycol.find({},{ "_id": 0, "name": 1, "address": 1 }):
              print(x)

=>  Fields with value '1' are get included and with  value '0' not included.
```


```
#  If you specify a field with the value 0, all other fields get the value 1, and vice versa.

#  This example will exclude "address" from the result
for x in mycol.find({},{ "address": 0 }):
    print(x)
```

```
#   You get an error if you specify both 0 and 1 values in the same object (except if one of the fields is
#       the _id Field.

for x in mycol.find({},{ "name": 1, "address": 0 }):
  print(x)
```

Query in mongoDB database

```
 =>  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.

#  Find document(s) with the address "Park Lane 38":

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

mydoc = mycol.find(myquery)

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

```
C:\Users\My Name>python demo_mongodb_query.py
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
```

Advanced Query in MongoDB

```
 =>  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"}:

#  Find documents where the address starts with the letter "S" or higher:

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

```
mydoc = mycol.find(myquery)

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

```
C:\Users\My Name>python demo_mongodb_query_modifier.py
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Filter With Regular Expressions

```
 =>  Regular expressions can only be used to query strings.
      To find only the documents where the "address" field starts with the letter "S", use the regular expression
        {"$regex": "^S"}.

#  Find documents where the address starts with the letter "S":

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

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

mydoc = mycol.find(myquery)

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

```
C:\Users\My Name>python demo_mongodb_query_regex.py
{'_id': 10, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Sorting in MongoDB

```
 =>  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).

#  Sort the result alphabetically by name:
mydoc = mycol.find().sort("name")
for x in mydoc:
    print(x)
```

```
C:\Users\My Name>python demo_mongodb_sort.py
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
```

```
 =>  Use the value -1 as the second parameter to sort in descending order.

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

#  Sort the result reverse alphabetically by name:
mydoc = mycol.find().sort("name", -1)
for x in mydoc:
    print(x)
```

```
C:\Users\My Name>python demo_mongodb_sort2.py
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
```

Deleting Documents in MongoDb

In [ ]:

=> delete_one() method is used to to delete a single document in collection.

=> 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.

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

```
#  Delete the document with the address "Mountain 21":
myquery = { "address": "Mountain 21" }
mycol.delete_one(myquery)

#print the customers collection after the deletion:
for x in mycol.find():
    print(x)
```

```
#  Delete all documents in the "customers" collection:
x = mycol.delete_many({})
print(x.deleted_count, " documents deleted.")
```

Dropping Collections in MongoDB

In [ ]:

=> Delete Collection means deleting a table or a collection in MongoDB database.

=> drop() method is used to delete the collections and it returns True if the collection was dropped successfully, and False if the collection does not exist.

```
>>> # Delete the "customers" collection
>>> mycol.drop()
```

Update Collections in MongoDB database

In [ ]:

=> 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.

=> If the query finds more than one record, only the first occurrence is updated.

=> The second parameter is an object defining the new values of the document.

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

```
#  Creating new entries to update by changing the address from "Valley 345" to "Canyon 123"
myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }

# Updating the collections
mycol.update_one(myquery, newvalues)

# Print "customers" after the update:
for x in mycol.find():
    print(x)
```

```
#  Update all documents where the address starts with the letter "S":
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }

x = mycol.update_many(myquery, newvalues)
```

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

Limit the result in MongoDB database

In [ ]:
```
=>  To limit the result in MongoDB, we use the limit() method.

=>  The limit() method takes one parameter, a number defining how many documents to return.
```

Consider you have a "customers" collection

Customers

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

```
# Limit the result to only return 5 documents:
myresult = mycol.find().limit(5)

# Print the result:
for x in myresult:
    print(x)
```

```
C:\Users\My Name>python demo_mongodb_limit.py
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
```

In [ ]: