Python & MongoDB: A Beginner's Guide - Student Handout

Project Overview: Building a Simple Python Application that Interacts with MongoDB

Key Concepts:

- **MongoDB**: A NoSQL database that stores data in a flexible, JSON-like format called BSON.
- **CRUD Operations**: Create, Read, Update, Delete operations on data.

---

1. Python and MongoDB Integration

Step 1: Installing the `pymongo` Package

- **Command**: `pip install pymongo`

Step 2: Connecting to MongoDB from Python

```
import pymongo

# Connect to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Access a database
db = client["mydatabase"]

# Access a collection
collection = db["mycollection"]
```

---

2. Performing CRUD Operations via Python Scripts

Create (Insert Documents)

```python
# Example 1: Insert a single document
document = {"name": "John", "age": 25, "city": "Mumbai"}
collection.insert_one(document)

# Example 2: Insert multiple documents
documents = [
    {"name": "Amit", "age": 30, "city": "Delhi"},
    {"name": "Sara", "age": 22, "city": "Bangalore"}
]
collection.insert_many(documents)

# Example 3: Insert with additional fields
document = {"name": "Emily", "age": 28, "city": "Chennai", "occupation":
"Engineer"}
collection.insert_one(document)
```

Read (Retrieve Documents)

```python
# Example 1: Retrieve all documents
for doc in collection.find():
    print(doc)

# Example 2: Retrieve documents with a filter
query = {"city": "Mumbai"}
for doc in collection.find(query):
    print(doc)

# Example 3: Retrieve specific fields
for doc in collection.find({}, {"name": 1, "city": 1}):
    print(doc)
```

Update (Modify Documents)

```python
# Example 1: Update a single document
query = {"name": "John"}
new_values = {"$set": {"age": 26}}
collection.update_one(query, new_values)

# Example 2: Update multiple documents
query = {"city": "Delhi"}
new_values = {"$set": {"city": "New Delhi"}}
```

```
collection.update_many(query, new_values)

# Example 3: Increment a field value
query = {"name": "Amit"}
new_values = {"$inc": {"age": 1}}
collection.update_one(query, new_values)
```

Delete (Remove Documents)

```
# Example 1: Delete a single document
query = {"name": "John"}
collection.delete_one(query)

# Example 2: Delete multiple documents
query = {"city": "New Delhi"}
collection.delete_many(query)

# Example 3: Delete all documents
collection.delete_many({})
```

---

3. Data Manipulation in MongoDB Using Python

Indexing and Querying with Filters

```
# Example 1: Create an index on the "name" field
collection.create_index([("name", pymongo.ASCENDING)])

# Example 2: Query with a filter
query = {"age": {"$gt": 25}}
for doc in collection.find(query):
    print(doc)

# Example 3: Query with multiple conditions
query = {"age": {"$gt": 25}, "city": "Mumbai"}
for doc in collection.find(query):
    print(doc)
```

---

4. Best Practices: Exception Handling, Error Logging, and Performance Tips

Exception Handling

```python
try:
    collection.insert_one(document)
except pymongo.errors.PyMongoError as e:
    print(f"An error occurred: {e}")
```

Error Logging

```python
import logging

logging.basicConfig(filename='app.log', level=logging.ERROR)

try:
    collection.insert_one(document)
except pymongo.errors.PyMongoError as e:
    logging.error(f"An error occurred: {e}")
```

Performance Tips

- **Use Indexes**: Speed up queries on large datasets.
- **Limit Results**: Use `limit()` to fetch a specific number of documents.
- **Batch Processing**: Process data in batches to manage memory usage.

Activity: Complete the Project

1. Install `pymongo` and set up a connection to MongoDB.
2. Create a database and a collection.
3. Perform CRUD operations (insert, read, update, delete) on the collection.
4. Implement error handling and logging.
5. Optimize your queries using indexes and filters.

Conclusion

By completing this project, you will understand how to integrate Python with MongoDB and perform various database operations efficiently.