Q1. What is MongoDB? Explain non-relational databases in short. In which scenarios is it preferred to use MongoDB over SQL databases?

MongoDB:
MongoDB is a cross-platform, document-oriented database that provides high performance, high availability, and easy scalability. It uses a flexible, schema-free document model, which allows for easy storage and retrieval of complex data.

Non-Relational Databases:
Non-relational databases, or NoSQL databases, are databases that use a data model other than the traditional relational model used by SQL databases. They are designed to handle large sets of distributed and unstructured data.

Scenarios to Prefer MongoDB over SQL Databases:

When dealing with large amounts of unstructured or semi-structured data.
In scenarios where horizontal scaling is a requirement, as MongoDB is designed for horizontal scalability.
For rapid development and iteration, as MongoDB's flexible schema allows for quick changes.

In [ ]:
```
Q2. State and Explain the features of MongoDB.
```

In [ ]:
```
Features of MongoDB:

Document-Oriented:

Data is stored in flexible, JSON-like BSON documents.
Documents can have nested structures, arrays, and other complex data types.
High Performance:

MongoDB provides high performance for both read and write operations.
Scalability:

MongoDB scales horizontally by sharding data across multiple servers.
Indexing:

Supports indexing for efficient querying and sorting of data.
Aggregation Framework:

Provides an aggregation framework for data transformation and analysis.
Schema Flexibility:

Allows dynamic and flexible schema design, making it easy to evolve data models.
Document Validation:

Supports document validation to enforce data integrity.
```

In [ ]:
```
Q3. Write a code to connect MongoDB to Python. Also, create a database and a collection in MongoDB.
```

In [ ]:
```
import pymongo

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

# Create a database
mydb = client["mydatabase"]

# Create a collection
mycollection = mydb["mycollection"]
```

Q4. Using the database and the collection created in question number 3, write a code to insert one record, and insert many records. Use the find() and find_one() methods to print the inserted record.

In [ ]:
```
# Insert one record
record_one = {"name": "John", "age": 30, "city": "New York"}
inserted_one = mycollection.insert_one(record_one)

# Insert many records
records_many = [
    {"name": "Alice", "age": 25, "city": "London"},
    {"name": "Bob", "age": 35, "city": "Paris"}
]
inserted_many = mycollection.insert_many(records_many)

# Print the inserted record using find_one
print("Inserted One Record:", mycollection.find_one({"name": "John"}))

# Print all inserted records using find
print("Inserted Many Records:")
for record in mycollection.find():
    print(record)
```

In [ ]:
```
Q5. Explain how you can use the find() method to query the MongoDB database. Write a simple code to demonstrate this
```

In [ ]:
```
# Query MongoDB using find method
result = mycollection.find({"city": "New York"})

# Print the query result
print("Query Result:")
for record in result:
    print(record)
```

In [ ]:
```
Q6. Explain the sort() method. Give an example to demonstrate sorting in MongoDB.
```

In [ ]:
```
# Sort documents by age in ascending order
result = mycollection.find().sort("age")

# Print the sorted result
print("Sorted Result (Ascending Order):")
for record in result:
    print(record)
```

In [ ]:
```
mycollection.delete_one({"name": "John"})
```

In [ ]:
```
mycollection.delete_many({"city": "Paris"})
```

In [ ]:
```
mycollection.drop()
```