1. Set the variable test1 to the string 'This is a test of the emergency text system,' and save test1 to a file named test.txt.

Ans : can set the variable test1 to the string 'This is a test of the emergency text system,' and save it to a file named 'test.txt':

test1 = 'This is a test of the emergency text system,' with open('test.txt', 'w') as file: file.write(test1)

In this example, the variable **test1** is set to the string 'This is a test of the emergency text system,'.

We use the **open()** function to open the file 'test.txt' in write mode ('w'). The **with** statement ensures that the file is properly closed after writing.

Inside the **with** block, we use the **write()** method to write the contents of **test1** to the file.

After executing the code, a file named 'test.txt' will be created (or overwritten if it already exists) with the contents of **test1**.

2. Read the contents of the file test.txt into the variable test2. Is there a difference between test 1 and test 2?

Ans : We can read the contents of the file 'test.txt' into the variable test2 and compare it with test1 to check if there's any difference:

with open('test.txt', 'r') as file: test2 = file.read() print(test1 == test2)

In this example, we use the **open()** function to open the file 'test.txt' in read mode ('r'). The **with** statement ensures that the file is properly closed after reading.

Inside the **with** block, we use the **read()** method to read the contents of the file into the variable **test2**.

Finally, we compare **test1** and **test2** using the equality operator (**==**) and print the result. If the contents of **test1** and **test2** are exactly the same, it will print **True**. If there is any difference, it will print **False**.

By comparing **test1** and **test2**, you can determine if there is any difference between the contents of the file 'test.txt' and the original value of **test1**.

3. Create a CSV file called books.csv by using these lines:

title,author,year

The Weirdstone of Brisingamen,Alan Garner,1960

Perdido Street Station,China Miéville,2000

Thud!,Terry Pratchett,2005

The Spellman Files,Lisa Lutz,2007

Small Gods,Terry Pratchett,1992

Ans : Here's an example of how you can create a CSV file called 'books.csv' with the provided lines:

import csv lines = [ ['title', 'author', 'year'], ['The Weirdstone of Brisingamen', 'Alan Garner', '1960'], ['Perdido Street Station', 'China Miéville', '2000'], ['Thud!', 'Terry Pratchett', '2005'], ['The Spellman Files', 'Lisa Lutz', '2007'], ['Small Gods', 'Terry Pratchett', '1992'] ] filename = 'books.csv' with open(filename, 'w', newline='') as file: writer = csv.writer(file) writer.writerows(lines) print(f"CSV file '{filename}' has been created.")

In this example, we first define the lines of data that will be written to the CSV file. Each line is a list of values representing the title, author, and year.

Then, we specify the filename as 'books.csv'.

Next, we open the file using the **open()** function with write mode ('w') and use **csv.writer** to create a writer object. We pass the writer object to the **writerows()** method, which writes all the lines to the CSV file.

Finally, we print a message to indicate that the CSV file has been created.

After executing the code, a CSV file named 'books.csv' will be created with the provided lines.

4. Use the sqlite3 module to create a SQLite database called books.db, and a table called books with these fields: title (text), author (text), and year (integer).

Ans : we can use the sqlite3 module in Python to create a SQLite database called 'books.db' and a table called 'books' with the fields 'title' (text), 'author' (text), and 'year' (integer):

import sqlite3 # Create a connection to the SQLite database conn = sqlite3.connect('books.db') # Create a cursor object to execute SQL commands cursor = conn.cursor() # Create the 'books' table cursor.execute('''CREATE TABLE books ( title TEXT, author TEXT, year INTEGER )''') # Commit the changes and close the connection conn.commit() conn.close() print("SQLite database 'books.db' and table 'books' have been created.")

In this example, we import the **sqlite3** module and then establish a connection to the SQLite database using **sqlite3.connect()** with the database file name 'books.db'.

We create a cursor object using the connection's **cursor()** method, which allows us to execute SQL commands on the database.

Using the cursor, we execute the **CREATE TABLE** statement to create the 'books' table with the specified fields: 'title' (text), 'author' (text), and 'year' (integer).

We then commit the changes to the database using **conn.commit()** and close the connection using **conn.close()**.

Finally, we print a message to indicate that the SQLite database 'books.db' and the 'books' table have been created.

After executing the code, the SQLite database file 'books.db' will be created, and the 'books' table with the specified fields will be created within the database.

5. Read books.csv and insert its data into the book table.

Ans : To read the data from the 'books.csv' file and insert it into the 'books' table in the 'books.db' SQLite database, you can use the csv module to read the CSV file and the sqlite3 module to perform the database operations. Here's an example:

import csv import sqlite3 # Open the CSV file and read its contents filename = 'books.csv' with open(filename, 'r') as file: reader = csv.reader(file) book\_data = list(reader) # Connect to the SQLite database conn = sqlite3.connect('books.db') cursor = conn.cursor() # Insert the data into the 'books' table for book in book\_data[1:]: # Skip the header line title, author, year = book cursor.execute("INSERT INTO books (title, author, year) VALUES (?, ?, ?)", (title, author, year)) # Commit the changes and close the connection conn.commit() conn.close() print("Data from 'books.csv' has been inserted into the 'books' table.")

In this example, we first open the 'books.csv' file using **open()** in read mode ('r') and create a **csv.reader** object to read its contents. We convert the reader object to a list **book\_data** to easily access its rows.

Next, we establish a connection to the SQLite database using **sqlite3.connect()**, and create a cursor object.

We then loop over the rows of **book\_data**, skipping the header line (**book\_data[1:]**). For each row, we extract the title, author, and year, and execute an **INSERT INTO** SQL statement to insert the data into the 'books' table using placeholders (**?**) to prevent SQL injection. We pass the values as a tuple to the **execute()** method.

After inserting all the data, we commit the changes to the database using **conn.commit()**, and then close the connection using **conn.close()**.

Finally, we print a message to indicate that the data from 'books.csv' has been inserted into the 'books' table.

After executing the code, the data from the 'books.csv' file will be inserted into the 'books' table in the 'books.db' SQLite database.

6. Select and print the title column from the book table in alphabetical order.

Ans : To select and print the 'title' column from the 'books' table in alphabetical order, you can use the SELECT statement in SQL and the sqlite3 module in Python. Here's an example:

import sqlite3 # Connect to the SQLite database conn = sqlite3.connect('books.db') cursor = conn.cursor() # Select the 'title' column from the 'books' table in alphabetical order cursor.execute("SELECT title FROM books ORDER BY title ASC") results = cursor.fetchall() # Print the titles for row in results: print(row[0]) # Close the connection conn.close()

In this example, we establish a connection to the SQLite database using **sqlite3.connect()** and create a cursor object.

We then execute the **SELECT** statement **"SELECT title FROM books ORDER BY title ASC"** using the cursor's **execute()** method. This selects the 'title' column from the 'books' table and orders the results in ascending alphabetical order.

We use the **fetchall()** method to retrieve all the selected rows as a list of tuples, where each tuple contains the title value.

Finally, we iterate over the results and print each title value. In this case, we access the title value using **row[0]**.

After executing the code, the titles from the 'books' table will be printed in alphabetical order.

7. From the book table, select and print all columns in the order of publication.

Ans : To select and print all columns from the 'books' table in the order of publication, you can modify the SQL query to select all columns using the wildcard (\*) and specify the 'year' column in the ORDER BY clause. Here's an example:

import sqlite3 # Connect to the SQLite database conn = sqlite3.connect('books.db') cursor = conn.cursor() # Select all columns from the 'books' table in the order of publication cursor.execute("SELECT \* FROM books ORDER BY year ASC") results = cursor.fetchall() # Print the columns for row in results: print(row) # Close the connection conn.close()

In this example, we establish a connection to the SQLite database using **sqlite3.connect()** and create a cursor object.

We modify the SQL query to **"SELECT \* FROM books ORDER BY year ASC"**, which selects all columns (**\***) from the 'books' table and orders the results by the 'year' column in ascending order.

We use the **fetchall()** method to retrieve all the selected rows as a list of tuples, where each tuple represents a row from the table.

Finally, we iterate over the results and print each row, which represents all the columns in that row.

After executing the code, all the columns from the 'books' table will be printed in the order of publication. Each row represents a record from the table, including all the columns and their respective values.

8. Use the sqlalchemy module to connect to the sqlite3 database books.db that you just made in exercise 6.

Ans : Here's an example of how you can use the sqlalchemy module to connect to the 'books.db' SQLite database that you created earlier:

from sqlalchemy import create\_engine # Connect to the SQLite database using sqlalchemy engine = create\_engine('sqlite:///books.db') # Perform database operations using the engine # ... # Close the connection engine.dispose()

In this example, we import the **create\_engine** function from **sqlalchemy** module.

To connect to the 'books.db' database, we use the **create\_engine** function and pass the database URI as **'sqlite:///books.db'**. The URI specifies the SQLite database file path.

Once the connection is established, you can perform various database operations using the **engine** object, such as executing queries, inserting data, or retrieving results.

After you have finished working with the database, you can close the connection using the **dispose()** method of the **engine**.

With this connection established, you can proceed to execute SQL queries or perform other database operations using the **engine** object and the SQLAlchemy API.

9. Install the Redis server and the Python redis library (pip install redis) on your computer. Create a Redis hash called test with the fields count (1) and name ('Fester Bestertester'). Print all the fields for test.

Ans : To install and use Redis server and the Python Redis library, you can follow these general steps:

1. Install Redis server:
   * Visit the Redis website (<https://redis.io/>) and download the Redis server package suitable for your operating system.
   * Install Redis according to the instructions provided for your operating system.
2. Install Python Redis library:
   * Open your terminal or command prompt.
   * Execute the following command to install the Redis library using pip:

pip install redis

1. Start the Redis server:
   * Run the Redis server on your local machine using the appropriate command for your operating system.
2. Connect to Redis and perform operations:
   * In your Python code, import the **redis** module:

import redis

* + Create a connection to the Redis server:

r = redis.Redis()

* + Set the values for the fields in the Redis hash:

r.hset('test', 'count', 1) r.hset('test', 'name', 'Fester Bestertester')

* + Retrieve and print all fields in the Redis hash:

fields = r.hgetall('test') for field, value in fields.items(): print(f"{field.decode()}: {value.decode()}")

In the above steps, we assume that you have installed Redis server and the Python Redis library successfully. The Redis hash called 'test' is created using the **hset()** method, and the fields and values are set accordingly. Finally, the **hgetall()** method is used to retrieve all fields from the hash, and they are printed out.

Please note that the steps may vary depending on your operating system and the specific Redis installation instructions. Make sure to follow the official Redis documentation and adjust the steps accordingly

10. Increment the count field of test and print it.

Ans :   
To increment the value of the 'count' field in the Redis hash 'test' and print the updated value, you can use the Python Redis library. Here's an example:

pythonCopy code

import redis # Create a connection to the Redis server r = redis.Redis() # Increment the 'count' field in the Redis hash 'test' r.hincrby('test', 'count', 1) # Retrieve and print the updated value of the 'count' field count = r.hget('test', 'count') print(f"Updated count: {count.decode()}")

In this example, we assume that you have established a connection to the Redis server using the **redis.Redis()** function.

We use the **hincrby()** method to increment the value of the 'count' field in the Redis hash 'test' by 1.

Then, we retrieve the updated value of the 'count' field using the **hget()** method and assign it to the **count** variable.

Finally, we print the updated value of the 'count' field using **count.decode()** to convert the retrieved value from bytes to a string.

After executing the code, it will increment the 'count' field in the Redis hash 'test' by 1 and print the updated value.