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:To set the variable `test1` to the given string and save it to a file named `test.txt`, you can use the following Python code:

# Set the variable test1 to the given string

test1 = 'This is a test of the emergency text system,'

# Save test1 to a file named test.txt

with open('test.txt', 'w') as file:

file.write(test1)

In this code, we open the file `test.txt` in write mode using the `open()` function and write the content of the `test1` variable into the file using the `write()` method. The `with` statement is used here as a context manager, which ensures that the file is properly closed after the write operation is completed.After running this code, the string 'This is a test of the emergency text system,' will be saved to a file named `test.txt` in the same directory where the Python script is executed.

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

ANS: To read the contents of the file `test.txt` into the variable `test2`, you can use the following Python code:

# Read the contents of the file test.txt into the variable test2

with open('test.txt', 'r') as file:

test2 = file.read()

In this code, we open the file `test.txt` in read mode using the `open()` function and read its contents into the variable `test2` using the `read()` method.

Now, to check if there is any difference between `test1` and `test2`, you can print them both and compare visually:

print("test1:", test1)

print("test2:", test2)

Run these code snippets together, and you should see the contents of both variables printed on the screen. If the contents of the file were correctly read and saved into `test2`, then both `test1` and `test2` should contain the same string:

test1: This is a test of the emergency text system,

test2: This is a test of the emergency text system,

If you see the same string printed for both `test1` and `test2`, it means there is no difference between them, and the content of `test1` was successfully read from the file and stored in `test2`.

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: To create the `books.csv` file with the provided lines, you can use the following Python code:

import csv

# Define the data to be written to the CSV file

data = [

['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']

]

# Write the data to the CSV file

with open('books.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerows(data)

print("books.csv file created successfully.")

In this code, we have updated the `data` list with the provided lines, where each inner list represents a row in the CSV file. The first inner list contains the header row ('title', 'author', 'year'), and the subsequent lists contain the book information.The code will create a file named `books.csv` in the same directory as your Python script and write the data in CSV format. When you run this code, you'll have the `books.csv` file with the given lines.

1. 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: To create a SQLite database called `books.db` and a table named `books` with the specified fields, you can use the `sqlite3` module in Python. Here's how you can do it:

import sqlite3

# Connect to the SQLite database (it will be created if it doesn't exist)

conn = sqlite3.connect('books.db')

# Create a cursor object to execute SQL commands

cursor = conn.cursor()

# Create the 'books' table with the specified fields

cursor.execute('''

CREATE TABLE IF NOT EXISTS books (

title TEXT,

author TEXT,

year INTEGER

)

''')

# Commit the changes and close the connection

conn.commit()

conn.close()

print("Database 'books.db' and table 'books' created successfully.")

In this code, we first import the `sqlite3` module. Then, we use `sqlite3.connect()` to connect to the SQLite database. If the database file (`books.db`) does not exist, it will be created automatically.Next, we create a cursor object with `conn.cursor()`. The cursor allows us to execute SQL commands on the database. We use the `execute()` method to run a SQL command that creates the `books` table with the specified fields: `title`, `author`, and `year`.

Finally, we commit the changes to the database using `conn.commit()` and close the connection with `conn.close()`.When you run this code, it will create the `books.db` database file and the `books` table with the specified fields. The database will be ready for you to insert, query, and manage data related to books.

1. 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` database, you can use the `csv` module to read the CSV file and the `sqlite3` module to perform the database operations. Here's how you can do it:

import csv

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect('books.db')

cursor = conn.cursor()

# Create the 'books' table if it doesn't exist

cursor.execute('''

CREATE TABLE IF NOT EXISTS books (

title TEXT,

author TEXT,

year INTEGER

)

''')

# Read data from books.csv and insert it into the 'books' table

with open('books.csv', 'r', newline='') as csvfile:

csv\_reader = csv.DictReader(csvfile)

for row in csv\_reader:

title = row['title']

author = row['author']

year = int(row['year'])

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 inserted into the 'books' table successfully."

In this code, we first import the `csv` and `sqlite3` modules. Then, we establish a connection to the `books.db` database and create a cursor object to execute SQL commands.

We define the table schema for the `books` table using the same `CREATE TABLE IF NOT EXISTS` statement as before.Next, we use the `csv.DictReader` to read the `books.csv` file. This allows us to treat each row of the CSV file as a dictionary, where the keys are the column names ('title', 'author', 'year').We then loop through each row in the CSV file and extract the 'title', 'author', and 'year' values. We convert the 'year' to an integer as specified in the table schema.Finally, we use an SQL `INSERT` statement with placeholders (`?`) to insert the data into the `books` table, and we execute the statement using `cursor.execute()`.After inserting all the data, we commit the changes and close the connection to the database.When you run this code, the data from `books.csv` will be inserted into the `books` table in the `books.db` database.

1. 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 modify the previous code and add a SQL `SELECT` statement with an `ORDER BY` clause. Here's the updated code:

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect('books.db')

cursor = conn.cursor()

# Create the 'books' table if it doesn't exist

cursor.execute('''

CREATE TABLE IF NOT EXISTS books (

title TEXT,

author TEXT,

year INTEGER

)

''')

# Inserted data into the 'books' table

# (Previous code to insert data from books.csv should be here)

# Select and print the 'title' column from the 'books' table in alphabetical order

cursor.execute("SELECT title FROM books ORDER BY title")

titles = cursor.fetchall()

print("Titles in alphabetical order:")

for title in titles:

print(title[0])

# Close the connection

conn.close()

In this updated code, we added the SQL `SELECT` statement with an `ORDER BY` clause to retrieve the `title` column from the `books` table in alphabetical order. We use `ORDER BY title` to sort the results based on the `title` column.After fetching the results with `cursor.fetchall()`, we iterate through the result and print each title.

1. 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 (sorted by the `year` column), you can modify the previous code and add an SQL `SELECT` statement with an `ORDER BY` clause on the `year` column. Here's the updated code:

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect('books.db')

cursor = conn.cursor()

# Create the 'books' table if it doesn't exist

cursor.execute('''

CREATE TABLE IF NOT EXISTS books (

title TEXT,

author TEXT,

year INTEGER

)

''')

# Inserted data into the 'books' table

# (Previous code to insert data from books.csv should be here)

# Select and print all columns from the 'books' table in the order of publication (sorted by 'year')

cursor.execute("SELECT \* FROM books ORDER BY year")

books\_data = cursor.fetchall()

print("Books in the order of publication:")

for book in books\_data:

print(f"Title: {book[0]}, Author: {book[1]}, Year: {book[2]}")

# Close the connection

conn.close()

In this updated code, we added the SQL `SELECT` statement with an `ORDER BY` clause to retrieve all columns (`\*`) from the `books` table. The results are sorted by the `year` column, so the books will be printed in the order of publication.

We fetch all the rows with `cursor.fetchall()` and then iterate through the results to print each book's title, author, and year.Again, make sure you have already inserted the data into the `books` table using the previous code to see the results of selecting and printing all columns in the order of publication.

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

ANS: To connect to the SQLite database `books.db` using the `sqlalchemy` module, you'll need to install the library first if you haven't already. You can do this by running `pip install sqlalchemy` in your terminal or command prompt

Once you have `sqlalchemy` installed, you can use the following Python code to connect to the `books.db` database:

from sqlalchemy import create\_engine

# Connect to the SQLite database

db\_engine = create\_engine('sqlite:///books.db')

# Perform database operations using db\_engine

# For example, you can execute SQL queries using db\_engine.execute()

In this code, we import the `create\_engine` function from `sqlalchemy`. The `create\_engine()` function is used to create a database engine, which serves as the gateway to the database.

We pass the database connection string `'sqlite:///books.db'` to `create\_engine()`, which tells `sqlalchemy` to use the SQLite database located at the file path `books.db`.

Now you have a `db\_engine` object that you can use to perform various database operations like executing SQL queries, creating tables, inserting data, and more. You can use the `execute()` method on `db\_engine` to execute raw SQL queries.Remember to import any other necessary modules and use the appropriate code to perform the specific database operations you want.

1. 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: import redis

# Connect to the Redis server

redis\_client = redis.Redis(host='localhost', port=6379, db=0)

# Create a Redis hash called 'test' with the fields 'count' and 'name'

redis\_client.hset('test', 'count', 1)

redis\_client.hset('test', 'name', 'Fester Bestertester')

# Print all the fields for 'test'

all\_fields = redis\_client.hgetall('test')

for field, value in all\_fields.items():

print(f"{field.decode('utf-8')}: {value.decode('utf-8')}")

In this code, we first import the `redis` module and then connect to the Redis server running on `localhost` (127.0.0.1) at the default port `6379` and database `0`.

We use the `hset()` method to set the fields and their values in the `test` hash. We set `count` to 1 and `name` to 'Fester Bestertester'.

Finally, we use the `hgetall()` method to retrieve all the fields and their values for the `test` hash and then print them.

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

ANS: To increment the `count` field of the `test` hash in Redis and then print its updated value, you can use the Python Redis library. Here's how you can do it:

import redis

# Connect to the Redis server

redis\_client = redis.Redis(host='localhost', port=6379, db=0)

# Increment the 'count' field of 'test'

incremented\_count = redis\_client.hincrby('test', 'count', 1)

# Print the updated 'count' value

print(f"Updated count: {incremented\_count}")

In this code, we use the `hincrby()` method of the Python Redis library to increment the value of the `count` field in the `test` hash by 1. The method takes three arguments: the hash name (`test` in this case), the field name (`count`), and the increment value (1 in this case).

The method returns the updated value of the `count` field, which we store in the variable `incremented\_count`. We then print this updated value to the console.