Task 1.1	import json
	from pymongo import MongoClient
	from pprint import pprint
	
	#Task 1.1
	client = MongoClient("127.0.0.1:27017") # Connect to MongoDB server
	client.drop_database("online_shop") # Delete 'online_shop' database
	
	db = client["online_shop"] # Create/connect to new database 'online_shop'
	
	# Create collections
	products_collection = db["products"]
	customers_collection = db["customers"]
	orders_collection = db["orders"]
	
	with open('datafiles/products.json', 'r') as file: # Open JSON file
	  products_data = json.load(file) # Load data from JSON file
	#close file
	products_collection.insert_many(products_data) # Insert into collection
	
	with open('datafiles/customers.json', 'r') as file: # Open JSON file
	  customers_data = json.load(file) # Load data from JSON file
	#close file
	customers_collection.insert_many(customers_data) # Insert into collection
	
	with open('datafiles/orders.json', 'r') as file: # Open JSON file
	  orders_data = json.load(file) # Load data from JSON file
	#close file
	orders_collection.insert_many(orders_data) # Insert into collection
	
	print("Data has been successfully inserted into MongoDB.")
		3 marks

Task 1.2	#Task 1.2
	print("Task 1.2: Display all products in the books category") # Print message
	books = products_collection.find({"category": "Books"}) # Find documents where category is 'Books'
	for book in books: # Iterate through list of books and print each book document
	  pprint(book) # Print in readable format
	print() # Leave a blank line
		2 marks

Task 1.3	#Task 1.3
	print("Task 1.3: Orders placed by Charles")
	customer = customers_collection.find_one({"name": "Charles"}) # Find document where customer's name is 'Charles'
	orders = orders_collection.find({"customer_id": customer["customer_id"]}) # Find all orders placed by Charles using his customer ID
	for order in orders: # Iterate through list of orders and print each order document
	  pprint(order) # Print in readable format
	print() # Leave a blank line
		4 marks

Task 1.4	#Task 1.4 Calculate total revenue
	print("Task 1.4: Calculate total revenue")
	print("Order ID | Total Revenue")
	orders = db.orders.find() # Retrieve all documents from orders collection
	
	for order in orders: # Iterate through each order
	  order_total = 0 # Initialise total revenue for current order
	  for product in order["products"]: # Iterate through each product in order
	  product_details = products_collection.find_one({"product_id": product["product_id"]}) # Find details of the product using product ID
	  revenue = product_details["price"] * product["quantity"] # Calculate revenue
	  order_total += revenue # Add revenue to current total revenue of the order
	  print(f"{order['order_id']} | {order_total}") # Print order ID and its total revenue
		5 marks

Task 2.1	#Task 2.1
	
	def insertion_sort(data_list, index, ascending):
	  # Copy the data_list to avoid modifying the original list
	  sorted_list = data_list.copy()
	
	  # Perform insertion sort
	  for i in range(1, len(sorted_list)): # Start from second element to end of list
	  key = sorted_list[i] # Store current element as 'key' to be inserted into its correct position
	  j = i - 1 # Initialise j to come before index i
	
	  # Determine the comparison operator based on the ascending flag
		6 marks

	if ascending:
	while j >= 0 and sorted_list[j][index] > key[index]:
	sorted_list[j + 1] = sorted_list[j] # Shift element which is greater than key to the right
	j -= 1
	else:
	while j >= 0 and sorted_list[j][index] < key[index]:
	sorted_list[j + 1] = sorted_list[j] # Shift element which is smaller than key to the right
	j -= 1
	
	sorted_list[j + 1] = key # Place 'key' in its correct position in sorted part
	
	return sorted_list # Return the sorted list	

Task 2.2	def task2_2(filename):
	with open(filename, 'r') as file: # Open file in read mode
	lines = file.readlines() # Read all lines from file into a list
	# file is closed automatically
	
	data_list = [] # Initialise empty list to store data
	for line in lines:
	participant_id, participant_name, country, score = line.strip().split(',') # Extract participant details
	data_list.append((participant_id, participant_name, country, int(score))) # Append a tuple containing participate details
	
	return data_list # Return list of data extracted
		5 marks

Task 2.3	def task2_3(filename, data_list):
	# Sort by country first (ascending)
	data_list = insertion_sort(data_list, index=2, ascending=True)
	
	# Dictionary to hold grouped data by country
	grouped_data = {}
	
	# Group the data by country
	for entry in data_list:
	country = entry[2]
	if country not in grouped_data:
	grouped_data[country] = []
	grouped_data[country].append(entry)
	
	# Sort each group by score (descending) and combine them
	sorted_data = []
		9 marks

	for country in grouped_data:
	group = grouped_data[country]
	group = insertion_sort(group, index=3, ascending=False)
	sorted_data.extend(group)
	
	# Write to file
	with open(filename, 'w') as file:
	for participant_id, participant_name, country, score in sorted_data:
	
	file.write(f"{country},{participant_id},{participant_name},{score}\n")
	# file is closed automatically
	
	# Run the program
	data_list = task2_2('datafiles/CODING_COMPETITION.txt') # Read data from file and store in data_list
	task2_3('SCORE_BY_COUNTRY.txt', data_list) # Write sorted data to file	

Task 3.1	# Task 3.1
	
	class Person:
	def __init__(self, name, personID):
	self.name = name
	self.personID = personID
	
	def getDetails(self):
	return f"Name: {self.name}, ID: {self.personID}"
	
	def getPersonID(self):
	return self.personID
	
	class Student(Person):
	def __init__(self, name, personID, course, year):
	super().__init__(name, personID)
	self.course = course
	self.year = year
	
	def getDetails(self):
	return f"Name: {self.name}, ID: {self.personID}, Course: {self.course}, Year: {self.year}"
	
	def getCourse(self):
	return self.course
	
	class Staff(Person):
		7 marks

```
    def __init__(self, name, personID, department, position, salary):
        super().__init__(name, personID)
        self.department = department
        self.position = position
        self.salary = salary

    def getDetails(self):
        return f"Name: {self.name}, ID: {self.personID}, Department:
{self.department}, Position: {self.position}, Salary: {self.salary}"
```

Task 3.2 — 11 marks

```
# Task 3.2

class LinkedList:

    def __init__(self, data=None):
        self.Data = data
        self.Pointer = None

    def insert_rec(self, data):
        if self.Data is None:
            self.Data = data
        elif self.Pointer is None:
            self.Pointer = LinkedList(data)
        else:
            self.Pointer.insert_rec(data)

    def delete_rec(self, personID):
        if self.Data and self.Data.getPersonID() == personID: #Deleting
first node
            if self.Pointer:
                self.Data = self.Pointer.Data
                self.Pointer = self.Pointer.Pointer
            else:
                # If the list has only one node, set it to None
                self.Data = None
        elif self.Pointer:
            if self.Pointer.Data.getPersonID() == personID: #Deleting non-
first node
                self.Pointer = self.Pointer.Pointer
            else:
                self.Pointer.delete_rec(personID)
        else:
            print(f"Node with ID: {personID} not found")

    def display_rec(self): #1m
        if self.Data:
```

```
            print(self.Data.getDetails())
        if self.Pointer:
            self.Pointer.display_rec()
```

Task 3.3

```
# Task 3.3

class StudentLinkedList(LinkedList):
    def __init__(self):
        super().__init__()
        self.studentCount = 0
        self.courseFreq = {}   # Use a dictionary to keep track of course
frequency

    def updateCourseFreq(self, operation, course):
        if operation == 'insert':
            if course in self.courseFreq:
                self.courseFreq[course] += 1
            else:
                self.courseFreq[course] = 1
        elif operation == 'delete':
            if course in self.courseFreq:
                self.courseFreq[course] -= 1

    def displayCourseFreq(self):
        if not self.courseFreq:
            print("No course frequency data available.")
            return

        print("Course Frequency:")
        for course, frequency in self.courseFreq.items():
            print(f"{course}: {frequency}")

    def insert(self, data):
        self.insert_rec(data)
        self.studentCount += 1
        self.updateCourseFreq('insert', data.getCourse())

    def delete(self, data):
        self.delete_rec(data.getPersonID())
        self.studentCount -= 1
        self.updateCourseFreq('delete', data.getCourse())

class StaffLinkedList(LinkedList):
    def __init__(self):
        super().__init__()
        self.staffCount = 0
```

11 marks

```
    def insert(self, data):
        self.insert_rec(data)
        self.staffCount += 1

    def delete(self, data):
        self.delete_rec(data)
        self.staffCount -= 1
```

Task 3.4 — 2 marks

```
# Task 3.4

# 1. Insert the following students into the StudentLinkedList:
student_list = StudentLinkedList()

students = [
    Student("Tan Wei Ling", "S0193", "Computer Science", 2),
    Student("Lim Zhi Hao", "S0274", "Mathematics", 1),
    Student("Ng Hui Min", "S0342", "Computer Science", 3),
    Student("Goh Yu Xuan", "S0435", "Physics", 2),
    Student("Chua Jia Wei", "S0521", "Mathematics", 3),
    Student("Ahmad Bin Salleh", "S0618", "Computer Science", 1),
    Student("Siti Nurhaliza", "S0739", "Biology", 2),
    Student("Rajesh Kumar", "S0847", "Physics", 3),
    Student("Priya Rani", "S0953", "Computer Science", 2),
    Student("David Lim", "S1058", "Biology", 1)
]

for student in students:
    student_list.insert(student)

# 2. Insert the following staff members into the StaffLinkedList:
staff_list = StaffLinkedList()

staff_members = [
    Staff("Dr. James Band", "ST001", "Computer Science", "Professor",
75000),
    Staff("Lim Boon Seng", "ST002", "Mathematics", "Lecturer", 55000),
    Staff("Mary Johnson", "ST003", "Computer Science", "Lecturer",
60000),
    Staff("Dr. Albert Einstein", "ST004", "Physics", "Professor", 80000),
    Staff("Robert Hughes", "ST005", "Biology", "Lecturer", 58000)
]

for staff in staff_members:
    staff_list.insert(staff)
```

	# 3. Display the details of all students and staff members in the linked lists. print("Student List:") student_list.display_rec() print("\nStaff List:") staff_list.display_rec() # 4. Display the course frequencies in the StudentLinkedList. print("\nCourse Frequencies:") student_list.displayCourseFreq() # 5. Delete the student David Lim (Student ID: S1058) from the StudentLinkedList. student_list.delete(Student("David Lim", "S1058", "Biology", 1)) # 6. Delete the staff member Lim Boon Seng (Staff ID: ST002) from the StaffLinkedList. ##staff_list.delete(Staff("Lim Boon Seng", "ST002", "Mathematics", "Lecturer", 55000)) staff_list.delete("ST002") # 7. Display the details of all remaining students and staff members in the linked lists. print("\nUpdated Student List after deletion:") student_list.display_rec() print("\nUpdated Staff List after deletion:") staff_list.display_rec() # 8. Display the updated course frequencies in the StudentLinkedList. print("\nUpdated Course Frequencies:") student_list.displayCourseFreq()	

Task 4.1	# Task 4.1: Importing New Patient Records
	import sqlite3
	
	# Connect to the CLINIC.db database
	conn = sqlite3.connect('CLINIC.db')
	cursor = conn.cursor()
	
	with open('datafiles/PATIENT.txt', 'r') as file:
	  for line in file:
	  data = line.strip().split(',')
	  query = f'INSERT INTO PATIENT ("PatientID", "Name", "DOB", "ContactNumber") VALUES (?,?,?,?)'
		4 marks

	cursor.execute(query, data)
	# Close file automatically
	
	with open('datafiles/STAFF.txt', 'r') as file:
	  for line in file:
	  data = line.strip().split(',')
	  query = f"INSERT INTO STAFF ('StaffID', 'Name', 'Role', 'Specialization') VALUES (?,?,?,?)"
	  cursor.execute(query, data)
	# Close file automatically
	
	with open('datafiles/APPOINTMENT.txt', 'r') as file:
	  for line in file:
	  data = line.strip().split(',')
	  query = f"INSERT INTO APPOINTMENT ('AppointmentID', 'PatientID', 'StaffID', 'AppointmentDate', 'Diagnosis') VALUES (?,?,?,?,?)"
	  cursor.execute(query, data)
	# Close file automatically
	
	# Commit the transactions and close the connection
	conn.commit()
	conn.close()	

Task 4.2	# Task 4.2: Retrieving a 2D Array of Appointment Data
	import sqlite3
	
	conn = sqlite3.connect('CLINIC.db') # Connect to database
	cursor = conn.cursor() # Create cursor object
	
	cursor.execute('''
	  SELECT Appointment.AppointmentID, Patient.PatientID, Patient.Name, Staff.StaffID, Staff.Name, Appointment.AppointmentDate, Appointment.Diagnosis
	  FROM Appointment
	  INNER JOIN Patient ON Appointment.PatientID = Patient.PatientID
	  INNER JOIN Staff ON Appointment.StaffID = Staff.StaffID
	''')
	appointments = cursor.fetchall() # Fetch all results from SQL query executed
	
	from pprint import pprint
	pprint(appointments) # Print appointements in readable format
	print()
	
	conn.close() # Close database connection
		5 marks

Task 4.3	# Task 4.3: List Staff Workload
	import sqlite3
	
	# Connect to the CLINIC.db database
	conn = sqlite3.connect('CLINIC.db')
	cursor = conn.cursor()
	
	# Execute the SQL query to list staff workload
	cursor.execute('''
	  SELECT Staff.Name, Staff.Role,
	COUNT(Appointment.AppointmentID) AS AppointmentCount
	  FROM Staff
	  LEFT OUTER JOIN Appointment ON Staff.StaffID =
	Appointment.StaffID
	  GROUP BY Staff.StaffID
	  ORDER BY Staff.Role, AppointmentCount DESC
	''')
	
	# Fetch all the results
	workload = cursor.fetchall()
	
	# Close the database connection
	conn.close()
	
	# Print the data under suitable headings
	print(f"{'Staff Name':<20} {'Role':<15} {'Number of Appointments':<25}")
	print("-" * 60)
	
	for row in workload:
	  print(f"{row[0]:<20} {row[1]:<15} {row[2]:<25}")
		6 marks

Task 4.4	from flask import Flask, render_template, request, url_for, redirect
	import sqlite3
	
	app = Flask(__name__)
	
	@app.route('/')
	def home():
	  return render_template('home.html')
	
	@app.route('/choice', methods=['POST'])
	def choice():
	  # Retrieve the value of the 'choice' input field
	  user_choice = request.form.get('choice')
		14 marks

```
    if user_choice == '1':
        return redirect('/search')
    elif user_choice == '2':
        return redirect(url_for('staff_workload'))
    else:
        return render_template('home.html')

# Search for Appointment
@app.route('/search', methods=['GET', 'POST'])
def search_appointment():
    if request.method == 'POST':
        patient_name = request.form['patient_name']
        appointment_date = request.form['appointment_date']
        conn = sqlite3.connect('CLINIC.db')
        cursor = conn.cursor()
        cursor.execute('''
            SELECT Appointment.AppointmentID, Patient.PatientID,
Patient.Name, Staff.StaffID, Staff.Name,
Appointment.AppointmentDate, Appointment.Diagnosis
            FROM Appointment
            INNER JOIN Patient ON Appointment.PatientID =
Patient.PatientID
            INNER JOIN Staff ON Appointment.StaffID = Staff.StaffID
            WHERE Patient.Name = ? AND Appointment.AppointmentDate
= ?
        ''', (patient_name, appointment_date))
        appointments = cursor.fetchall()

        conn.close()
        if not appointments:
            return render_template('search.html', message="No
appointments found")
        return render_template('search.html',
appointments=appointments)
    return render_template('search.html')

# List Staff Workload
@app.route('/workload', methods=['GET'])
def staff_workload():
    conn = sqlite3.connect('CLINIC.db')
    cursor = conn.cursor()
    cursor.execute('''
        SELECT Staff.Name, Staff.Role,
COUNT(Appointment.AppointmentID) as AppointmentCount
        FROM Staff
```

```
        LEFT OUTER JOIN Appointment ON Staff.StaffID =
Appointment.StaffID
        GROUP BY Staff.Name, Staff.Role
        ORDER BY AppointmentCount DESC
    '')
    workload = cursor.fetchall()
    conn.close()
    return render_template('workload.html', workload=workload)

if __name__ == '__main__':
    app.run()
```

templates/home.html

```
<!DOCTYPE html>
<html>
<head><title>Clinic</title></head>
<body>
        <form action="{{url_for('choice')}}" method="POST">
        <p><a href="{{url_for('search_appointment')}}">1.       Search
Appointment</a></p>
        <p><a href="{{url_for('staff_workload')}}">2.      List Staff
Workload</a></p>
        <input type="text" name='choice' placeholder = 'Enter your
choice'>
        <input type="submit" value='Search'>
        </form>
</body>
</html>
```

templates/search.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Search Appointment</title>
</head>
<body>
    <h1>Search for Appointment</h1>
        <form action="{{ url_for('search_appointment') }}"
method="POST">
            <label for="patient_name">Patient Name:</label>
            <input type="text" id="patient_name" name="patient_name"
placeholder="Enter patient name" required>
            <br><br>
```

```
        <label for="appointment_date">Appointment Date:</label>
        <input type="text" id="appointment_date"
name="appointment_date" required>
        <br><br>
        <input type="submit" value="Search">
    </form>

  {% if appointments %}
    <h2>Search Results</h2>
    <table border = 1>
      <thead>
        <tr>
          <th>Appointment ID</th>
                                  <th>Patient ID</th>
          <th>Patient Name</th>
                                  <th>Staff ID</th>
          <th>Staff Name</th>
          <th>Appointment Date</th>
          <th>Diagnosis</th>
        </tr>
      </thead>
      <tbody>
        {% for appointment in appointments %}
          <tr>
            <td>{{ appointment[0] }}</td>
            <td>{{ appointment[1] }}</td>
            <td>{{ appointment[2] }}</td>
            <td>{{ appointment[3] }}</td>
            <td>{{ appointment[4] }}</td>
            <td>{{ appointment[5] }}</td>
            <td>{{ appointment[6] }}</td>

          </tr>
        {% else %}
          <tr>
            <td colspan="5">No appointments found</td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  {% endif %}

  <p><a href="{{ url_for('home') }}">Back to Home</a></p>
</body>
</html>
```

templates/workload.html

```
<!DOCTYPE html>
<html>
<head>
   <title>List Staff Workload</title>
</head>
<body>
   <h1>List Staff Workload</h1>
      <a href="{{ url_for('home') }}">Back to Home</a>

   {% if workload %}
      <table border=1>
         <thead>
            <tr>
               <th>Staff Name</th>
               <th>Role</th>
               <th>Number of Appointments</th>
            </tr>
         </thead>
         <tbody>
            {% for row in workload %}
               <tr>
                  <td>{{ row[0] }}</td>
                  <td>{{ row[1] }}</td>
                  <td>{{ row[2] }}</td>
               </tr>
            {% endfor %}
         </tbody>
      </table>
   {% else %}
      <p>No performance data available.</p>
   {% endif %}
</body>
</html>
```