



RAFFLES INSTITUTION

2024 YEAR 6 PRELIMINARY EXAM

CANDIDATE
NAME

CLASS

24

COMPUTING

9569/02

Paper 2 (Lab-based)

3 hours

Additional materials: Electronic version of PRODUCTS.json data file
Electronic version of CUSTOMERS.json data file
Electronic version of ORDERS.json data file
Electronic version of CODING_COMPETITION.txt data file
Electronic version of TASK3_TESTING.txt file
Electronic version of CLINIC.db database file
Electronic version of PATIENT.txt data file
Electronic version of STAFF.txt data file
Electronic version of APPOINTMENT.txt data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in brackets [] at the end of each task.

The total number of marks for this paper is 100.

FOR EXAMINER'S USE				
Task 1	Task 2	Task 3	Task 4	
				TOTAL
				100

This document consists of **11** printed pages and **1** blank page.

RAFFLES INSTITUTION
Mathematics Department

Instructions to candidates:

Your program code and output for each of Task 1 to 3, and Task 4.1 to 4.3 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code and output for Task 1 should be saved as:

`TASK1_<your name>_<class>_<index number>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

`TASK1_<your name>_<class>_<index number>.ipynb`

An online shopping platform stores its data in a MongoDB database.

The platform needs to store information about products, customers, and orders.

Your task is to perform several operations on the database.

For each of the sub-tasks, add a comment statement at the beginning of the code using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 1.1
        Program code
```

Output:

Task 1.1

Write Python code to create a MongoDB database named `online_shop`. Ensure that this database is empty before proceeding.

Three JSON files are provided, each containing multiple records to be inserted into specific collections within the database. The files are `PRODUCTS.json`, `CUSTOMERS.json` and `ORDERS.json`.

Create collections named `products`, `customers`, and `orders`, and insert the data from each JSON file into the corresponding collection: [3]

JSON file	Insert into collection	Sample JSON record
PRODUCTS	products	<pre>{ "product_id": 1, "name": "Laptop", "model": "XPS 13", "category": "Electronics", "price": 1200, "stock": 30 }</pre>

CUSTOMERS	customers	{ "customer_id": 1, "name": "Alex", "email": "alexB@abc.com", "phone": "12345678" }
ORDERS	orders	{ "order_id": 1, "customer_id": 1, "products": [{"product_id": 2, "quantity": 1}], "order_date": "2024-01-15" }

Task 1.2

Add to your program code to retrieve and display all products in the `Books` category from the `products` collection.

Use the `pprint` module to display your data in a readable format:

```
from pprint import pprint
pprint(your_data)
```

Run your program.

[2]

Task 1.3

Add to your program code to find all orders placed by the customer named `Charles`. Assume there is only one customer with this name.

Display the details of these orders from the `orders` collection.

Use `pprint` to display your data in a readable format.

Run your program.

[4]

Task 1.4

Add to your program code to calculate the revenue generated from all orders.

Display the revenue in a tabular format with headers for Order ID and Revenue.

For each order, compute the revenue as the sum of the price of each product multiplied by the quantity ordered.

Run your program.

[5]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

TASK2_<your name>_<class>_<index number>.ipynb

For each of the sub-tasks, add a comment statement at the beginning of the code using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 2.1  
Program code
```

Output:

Task 2.1

Write an insertion sort function that takes three parameters: `data_list`, `index`, `ascending` and returns a sorted `data_list`.

- `data_list` is a list of tuples.
- `index` is an integer to specify an index within a tuple.
- `ascending` is a Boolean flag.

The function will sort `data_list` based on the values at the specified `index` within the tuples. The `ascending` parameter determines the order of the sort: `True` for ascending order, `False` for descending order.

The function specification is:

```
FUNCTION insertion_sort(data_list: List[Tuple], index: Integer,  
ascending: Boolean) RETURN List[Tuple] [6]
```

Example of insertion_sort function call:

```
data_list = [(1, 'Alice', 85), (2, 'Bob', 90), (3, 'Charlie', 80)]  
sorted_list = insertion_sort(data_list, 2, ascending=False)  
print(sorted_list)  
# Output: [(2, 'Bob', 90), (1, 'Alice', 85), (3, 'Charlie', 80)]
```

Task 2.2

At an international coding competition, the results of the participants were recorded in a text file `CODING_COMPETITION.txt`.

Each record in the file has the following format, with each field separated by a comma: `<participant_id>,<participant_name>,<country>,<score>`

Write a function `task2_2` that takes a `filename` as parameter, reads the comma-separated data from the file and returns the data as a list of tuples.

Example: After calling function `task2_2`, these sample records in the file:

```
101,Jong Tan,Singapore,210
102,Mei Ling,Singapore,190
```

will be returned as

```
[('101','Jong Tan','Singapore',210), ('102','Mei Ling','Singapore',190)]
```

Test your function by calling `task2_2("CODING_COMPETITION.txt")`. [5]

Task 2.3

Write a function `task2_3` that performs the following:

- Use the list of tuples returned from the function `task2_2`.
- Sort the list by country in alphabetical order, using the `insertion_sort` function.
- Within each country, sort the participants by their scores in descending order using the `insertion_sort` function.
- Write the sorted data to a text file `SCORE_BY_COUNTRY.txt` in the format: `<country>,<participant id>,<participant name>,<score>`.
- Each record should be on a new line.

The output file should contain lines of record that is first sorted by country, then by score. Sample lines in output file:

```
Australia,249,James Scott,235
Australia,267,Liam Taylor,230
Australia,139,Daniel Miller,225
.....
Singapore,101,Jong Tan,210
.....
```

Run your function `task2_3`. [9]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as:

TASK3_<your name>_<class>_<index number>.ipynb

Implement a college management system using Object-Oriented Programming.

The system will manage two types of entities: `Student` and `Staff`. These entities will be created as subclasses which inherit from the base class `Person`.

Both entities will be stored and managed in two separate linked list structures: `StudentLinkedList` and `StaffLinkedList`. These linked lists will inherit from the base class `LinkedList`.

The class `Person` contains the following attributes:

- `name`: the name of the person.
- `personID`: the unique ID of the person.

The class `Person` contains a method `getDetails()` that returns a string containing the person's name and ID.

The `Student` class inherits from the `Person` class. Add the following additional attributes to the `Student` class:

- `course`: the course the student is enrolled in.
- `year`: the year of study.

The `Student` class contains a method `getDetails()` that returns a string containing the student's name, ID, course and year of study.

The `Staff` class inherits from the `Person` class. Add the following additional attributes to the `Staff` class:

- `department`: the department the staff member belongs to.
- `position`: the position or job title of the staff member.
- `salary`: the salary of the staff member.

The `Staff` class contains a method `getDetails()` that returns a string containing the staff member's name, ID, department, position and salary.

For the sub-task, add a comment statement at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 3.1
        Program code
```

Output:

Task 3.1

Write program code to declare the classes `Person`, `Student` and `Staff`, with all their attributes and methods. [7]

Task 3.2

The class `LinkedList` contains the following attributes:

- `Data`: the data stored in the node, which can be student or staff
- `Pointer`: points to the next node in the linked list

The class `LinkedList` contains the following methods:

- a constructor to set the `Data` to its parameter, and pointer to `None`.
- `insert_rec(data)`: recursive method to insert a new node with the given data at the end of the linked list, where the parameter represents any object.
- `delete_rec(data)`: recursive method to delete the node with the specified data from the linked list, where the parameter identifies the node to be deleted.
- `display_rec()`: recursive method to display the details of all nodes in the linked list.

Write program code to declare the class `LinkedList` with all its attributes and methods. [11]

Task 3.3

The `StudentLinkedList` class inherits from the `LinkedList` class and contains the following additional attributes:

- `studentCount` that keeps track of the total number of students in the linked list.
- `courseFreq` that keeps track of the number of students in each course.

The `StudentLinkedList` class contains the following methods:

- `updateCourseFreq` that will update the `courseFreq`.
- `displayCourseFreq` that will display the number of students from each course.
- `insert(data)` method to insert a new node data at the end of the linked list, where the parameter is a `Student` object.
- `delete(data)` method to delete a specified node data from the linked list, where the parameter identifies the node to be deleted.

The `studentCount` and `courseFreq` will be updated whenever a node data is inserted or deleted.

The `StaffLinkedList` class inherits from the `LinkedList` class and contains the following additional attribute:

- `staffCount` that keeps track of the number of staff members.

The `StaffLinkedList` class contains the following methods:

- `insert(data)` method to insert a new node data at the end of the linked list, where the parameter is a `Staff` object.
- `delete(data)` method to delete a specified node data from the linked list, where the parameter identifies the node to be deleted.

The `staffCount` will be updated whenever a node data is inserted or deleted.

Write program code to declare the classes `StudentLinkedList` and `StaffLinkedList`, with all their attributes and methods. [11]

Task 3.4

Write and execute code to test your program according to the instructions specified in the text file "`TASK3_TESTING.txt`". [2]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as:

TASK4_<your name>_<class>_<index number>.ipynb

A clinic management system maintains records of patients, medical staff and appointments in a database `CLINIC.db`, which is provided with this question.

The database file contains the following tables:

- `Patient (PatientID, Name, DOB, ContactNumber)`: Stores information about patients, including their unique patient ID, name, date of birth, and contact number.
- `Staff (StaffID, Name, Role, Specialization)`: Stores information about medical staff, including their unique staff ID, name, role (Doctor/Nurse/Technician), and specialization.
- `Appointment (AppointmentID, PatientID, StaffID, AppointmentDate, Diagnosis)`: Records appointments made by patients, including the appointment ID, patient ID, staff ID, appointment date, and diagnosis.

For each of the sub-tasks 4.1 to 4.3, add a comment statement at the beginning of the code using the hash symbol '#', to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 4.1
        Program code
```

Output:

Task 4.1

The text files `PATIENT.txt`, `STAFF.txt` and `APPOINTMENT.txt` store the comma-separated values for each of the tables in the database.

Write a Python program to read in the data from each file and then store each item of data in the correct place in the database.

Run your program. [4]

Task 4.2

Write an SQL query to retrieve the following data: *Appointment ID, Patient ID, Patient Name, Staff ID, Staff Name, Appointment Date, Diagnosis*

In a new cell, write a Python program that uses this SQL query to retrieve the data and display them.

Run your program. [5]

Task 4.3

Write an SQL query to:

- Count the total number of appointments each staff member (Doctor/Nurse/Technician) has handled.
- Group the results by role and order them by the number of appointments in descending order.
- Display the staff members' names, roles, and the number of appointments they have handled.

In a new cell, write a Python program that uses this SQL query and display the data under suitable headings. [6]

Save your Jupyter Notebook for Task 4.

Task 4.4

For this sub-task, save your program separately with a `.py` file extension.

Write a Python program and the necessary files to create a web application. The web application offers the following menu options:

1. Search Appointment
2. List Staff Workload

For menu option 1, design a web form that allows users to search for appointments by patient name and appointment date. The form should:

- Include input fields for *patient name* and *appointment date*.
- Have a submit button labeled "Search".

The results of the search function should be shown on a web page in a structured table format that shows all details from the `Appointment` table, including patient name and staff name. If no appointments are found, display a message "No appointments found".

Test your program by selecting menu option 1, and inputting 'David Tan' as patient name and '13/08/2024' as date.

For menu option 2, display the data retrieved from the SQL query in sub-task 4.3, under these headings on a web page: Staff Name, Staff Role, Number of Appointments handled.

Save your program code as:

TASK4_4_<your name>_<class>_<index number>.py

with any additional files/subfolders as needed in a folder named

TASK4_4_<your name>_<class>_<index number>

Run the web application and save the output as

- TASK4_4_<your name>_<class>_<index number>_1.html when menu option 1 is selected, input 'David Tan' as patient name and '13/08/2024' as date.
- TASK4_4_<your name>_<class>_<index number>_2.html when menu option 2 is selected. [14]

BLANK PAGE