



**ANGLO-CHINESE JUNIOR COLLEGE
JC2 PRELIMINARY EXAMINATION**

Higher 2

COMPUTING

9569/02

Paper 2 (Lab-based)

13 August 2025

3 hours

Additional Materials:

Electronic version of `Task1-1.csv` data file
Electronic version of `Task1-2.csv` data file
Electronic version of `COURSE.txt` data file
Electronic version of `CUSTOMER.txt` data file
Electronic version of `INSTRUCTOR.txt` data file
Electronic version of `REGISTRATION.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 question or part question.
The total number of marks for this paper is 100.

This document consists of **9** printed pages.



Anglo-Chinese Junior College

[Turn Over

Instruction to candidates:

Your program code and output for each of Tasks 1, 2 and 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>_<centre number>_<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>_<centre number>_<index number>.ipynb`

A magic square of order n is an $n \times n$ array of integers that has the following properties:

- Each integer from 1 to n^2 inclusive appears exactly once in the array;
- Each row in the array has the same sum;
- Each column in the array has the same sum;
- The two longest diagonals of the array have the same sum as the rows and columns.

For example, a magic square of order 4 is shown below. Each row, each column, and each of the two longest diagonals has a sum of 34.

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

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 a function `check_magic(arr)` that takes in an $n \times n$ array `arr` of integers and returns `True` if it forms a magic square, and `False` if it does not. Data validation does not need to be performed. [11]

Task 1.2

Write a function `check_file(filename)` that checks if an array, stored in a file, forms a magic square. It can be assumed that the file contains an $n \times n$ array of integers. [5]

Test your function with the files `Task1-1.csv` and `Task1-2.csv`, showing your output. [2]

Task 1.3

When n is odd, a magic square can be constructed in the following way.

- Start from the centre entry of the top row and insert 1 into the array there.
- Repeat the following steps until all the integers from 1 to n^2 are in the array.
 - Suppose you have just inserted the integer k into the array. The next number to be inserted is $k + 1$.
 - Move one step to the right and one step up. If this brings you outside the array, wrap around to the bottom row or the left column as necessary.
 - If the entry is already occupied by a number, go to the entry directly below the previous number instead, wrapping around to the top row if necessary.
 - Insert $k + 1$ into this entry.

For example, the first few steps to construct a magic square of order 5 are shown below.

		1		

Step 1: Insert 1 in the centre of the top row

		1		
			2	

Step 2: Move one step right and one step up. Since 1 is in the top row, you need to wrap around to the bottom row. Insert 2.

		1		
				3
			2	

Step 3: Move one step right and one step up. Insert 3

		1		
4				
				3
			2	

Step 4: Move one step right and one step up. Since 3 is in the rightmost column, you need to wrap around to the leftmost column. Insert 4.

		1		
	5			
4				
				3
			2	

Step 5: Move one step right and one step up. Insert 5.

		1		
	5			
4	6			
				3
			2	

Step 6: Moving one step right and one step up brings you to an occupied square. Insert 6 directly below 5.

Write a function `make_magic(n)` that takes in an odd integer n and constructs and returns a magic square of order n . [6]

Create a magic square of order 9 and save it to `Task1-3.csv`. [5]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

TASK2_<your name>_<centre number>_<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 a function `quicksort(lis)` that sorts a list of positive integers in ascending order, using a quicksort algorithm that works in place. [8]

Write a function `insertionsort(lis)` that sorts a list of positive integers in ascending order, using an insertion sort algorithm that works in place. [3]

Task 2.2

Generate the following four lists:

- `lis1` and `lis2` are a list of the integers from 1 to 1000 inclusive, in ascending order;
- `lis3` is a list of the integers from 1 to 1000 inclusive, in random order;
- `lis4` is a copy of `lis3`.

The `timeit` library provides a way to time how long a snippet of code takes to run. For example, the following code block prints the time it takes for `quicksort(lis1)` to run.

```
import timeit
setup_code = "from __main__ import quicksort, lis1"
time = timeit.timeit("quicksort(lis1)", setup_code, number=1)
print(time)
```

Output the time it takes to run

- `quicksort(lis1)`
- `insertionsort(lis2)`
- `quicksort(lis3)`
- `insertionsort(lis4)`

[4]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as:

TASK3_<your name>_<centre number>_<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 3.1
        Program code
        Output:
```

An 2-dimensional array is to be implemented using Object-Oriented Programming (OOP). It acts as a free list so that linked lists can be created. Each row in the array represents a node in the free list. The first column of each row contains the data, which is a string, and the second column contains the pointer, which is an integer and is the index of the row to which that row is pointing.

The linked lists are to store strings in alphabetical order.

Task 3.1

The `Array` class includes the following attributes.

- `container` is an array with `n` rows and 2 columns, where `n` is input by the user during initialization. The data in the first column are initialized as empty strings. To initialize the second column, each row points to the row below it, except for the final row, which points to `None`.
- `free` is the head pointer of the free list, which is initialized with a value of 0.

The `Array` class also includes the following methods.

- `get_data(index)` returns the data of the row at the given `index` in `container`.
- `set_data(index, new_data)` inserts `new_data` into the data column of the row at the given `index` in `container`.
- `get_pointer(index)` returns the pointer of the row at the given `index` in `container`.
- `set_pointer(index, new_pointer)` updates the pointer value of the row at the given `index` in `container` to `new_pointer`.

Write program code to define the `Array` class.

[4]

Task 3.2

The `LinkedList` class includes the following attributes:

- `array` is the `Array` object in which the linked list is actually stored, and is input by the user during initialization.
- `head` points to the head node of the linked list, and is initialized as `None`.

The `LinkedList` class also includes the following methods:

- `display()` displays the data in all the nodes of the linked list in order of the list.
- `add(new_data)` adds a node to the linked list with `new_data` as the data. The new node should be added so that the nodes in the linked list are in alphabetical order of their data. If there are no nodes available, it prints an appropriate error message.
- `delete(to_delete)` attempts to delete the node with the data `to_delete` from the linked list, and adds it back to the head of the free list. If the node cannot be found in the linked list, it prints an appropriate error message.

Write program code to define the `LinkedList` class.

[17]

Task 3.2

Write program code to:

- create an instance `MyArray` of `Array` with 10 rows;
- create two instances, `AnimalList` and `PlantList`, of `LinkedList` that store their data in `MyArray`;
- add the following nodes in the order given:
 - "CAT" to `AnimalList`
 - "ROSE" to `PlantList`
 - "GOAT" to `AnimalList`
 - "FERN" to `PlantList`
 - "HORSE" to `AnimalList`
 - "VINE" to `PlantList`
- attempt to delete "COW" from `AnimalList`;
- delete "ROSE" from `PlantList`;
- display the data in both `AnimalList` and `PlantList`.

Show

all

output.

[5]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as:

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

A community centre keeps records about its courses. The community centre wants to create a suitable database to store the data and allow them to run searches for specific data. The database will have four tables: a table to store data about the instructors, a table about the customers, a table about the courses, and a table to show which customer has registered for which course.. The fields in each table are:

Instructor:

- InstructorID – instructor's unique ID
- Name – the instructor's name

Customer:

- CustomerID – the customer's unique ID
- Name – the customer's name

Course:

- CourseID – unique course ID, for example, D123
- Name – the name of the course
- InstructorID – the ID of the instructor conducting the course
- StartDate – the starting date of the course in YYYYMMDD format
- Price – the price of the course in dollars

Registration:

- CustomerID – the customer's ID
- CourseID – the course ID
- Paid – an integer value, where 0 means the customer has not paid and 1 means the customer has paid

For each of the sub-tasks 4.1 and 4.2, 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

Write a Python program that uses SQL code to create the database COURSES with the four tables given. Define the primary and foreign keys for each table. [6]

Task 4.2

The text files COURSE.txt, INSTRUCTOR.txt, CUSTOMER.txt and REGISTRATION.txt contains the comma-separated values for each of the tables in the database.

Write a Python program to read the data from each file and then store each item of data in the correct place in the database. [5]

Task 4.3

Write a Python program and the necessary files to create a web application that asks the user for a customer name, and displays the following data for all courses that the customer has registered for.

- Course name
- Instructor name
- Whether the customer has paid for the course

The program should return an HTML document that enables the web browser to display a table with the required data.

Save your program code as

TASK4_3_<your name>_<centre number>_ <index number>.py

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

TASK4_3_<your name>_<centre number>_ <index number> [11]

Run the web application, entering "Tan Ah Seng" as the customer name.

Save the webpage output as:

TASK4_3_<your name>_<centre number>_ <index number>.html [2]