



YISHUN INNOVA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION
Higher 2

CANDIDATE
NAME

CG

INDEX NO

COMPUTING

Paper 2 (Lab-based)

9569/02

29 Aug 2023

3 hours

100 Marks

Additional Materials: Removable storage device with the following files:

- `template.ipynb` (for Question 1, 2 and 3)
- `RANDOM100.TXT`
- Q4 Web App Folder (with `Feedback.TXT`, `Taxi.db`, and templates for `server.py`)

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.

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

The total number of marks for this paper is **100**.

Instruction to candidates:

Your program code and output for Question 1, 2 and 3 should be saved in a single `.ipynb` and downloaded as

`template_<your class>_<your name>.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 1.1
Program Code

Output:

In [2]:

#Task 1.2
Program Code

Output:

In [3]:

#Task 1.3
Program Code

Output:

1 The task is to:

- generate a list of unique random integers
- write the numbers to a text file
- read a list of numbers from a file
- sort the list using (i) an insertion sort, (ii) a bubble sort, and (iii) a quicksort
- compare the efficiency of the sorting algorithms.

Task 1.1

Write program code for the function `generate()` to return a list containing 100 unique random integers between 0 and 100 (inclusive). [3]

Task 1.2

Write program code to save the 100 random numbers, generated in **Task 1.1** to a text file `GENERATE100.TXT`, with the numbers separated by commas. [3]

Task 1.3

Write program code for the function `read()` to return a list containing the numbers read from the text file `RANDOM100.TXT`. [3]

Task 1.4

```
sortlist = [1,3,5]
```

Write program code for the helper function `insert(item)` to insert an integer `item` into a sorted list `sortlist`. Check your code with the following test cases:

- `>>> insert(0)`
- `>>> insert(2)`
- `>>> insert(6)` [4]

Task 1.5

```
sortlist = []
```

Write program code for the function `insertionsort(seq)` using the helper function `insert(item)`, in **Task 1.4**, to insert all the numbers read from the text file `RANDOM100.TXT`, in **Task 1.3**, into the empty list `sortlist` such that they are arranged in an ascending order. [2]

Task 1.6

Write program code for the function `bubblesort(seq)` that takes the list of numbers from **Task 1.3**, sorts them into ascending order using bubble sort and returns the sorted list. [4]

Task 1.7

Write program code for the function `quicksort(seq)` that takes the list of numbers from **Task 1.3**, sorts them into ascending order using quicksort and returns the sorted list. [6]

Task 1.8

The Python built-in library `timeit` can be used to time simple function calls. An example of the code is as follows:

```
>>> from timeit import *
>>> insert100 = timeit(lambda: insertionsort(lst), number=1)
```

`insert100` is the time taken, in seconds, to sort the list of numbers in `lst` using the insertion sort.

Using the `timeit` module to compute the time taken and stating the orders of growth, compare the time complexity of the insertion sort, bubble sort and quicksort algorithms. [5]

- 2 In a computer simulated Battleship board game, the rectangular board measures 10 metres by 7 metres. The grid on the board is indicated by rows numbered from 1 to 7 and columns labelled by letters from A to J.

```
\ABCDEFGHIJ
1.....
2.....
3.....
4.....
5.....
6.....
7.....
```

Task 2.1

Write program code to display the board as shown.

- The (x, y) coordinates of the grid should be stored in a suitable data structure.
- The data structure will allow fixed loop(s) to be used to display the board.

[4]

Task 2.2

The computer randomly generates the position of a battleship within the grid. It will occupy 4 units in length, represented by 4 consecutive 'S' either horizontally or vertically within the grid.

The computer will also generate 7 random locations to represent the rocks ('R').

Write program code to:

- generate 4 (x, y) coordinates to represent the battleship ('S') within the grid and store them in a suitable data structure.
- generate 7 (x, y) coordinates to represent the position of the rocks ('R') within the grid and store them in a suitable data structure.
- display the board showing the battleship and the rocks.

A possible board is as follows:

```
\ABCDEFGHIJ
1.....R.
2..SSSS....
3...R.....
4.....R....
5.R.....R..
6...R.....
7.....R..
```

[6]

Task 2.3

During the game, the position of the battleship will not be revealed. The location of the battleship will be represented by '.'.

The player will fire a missile to strike at a target location. If the missile hits the battleship, it will be indicated with a 'H'; Otherwise, it will be indicated with a 'M' for a miss.

The game ends when player has hit all the 4 (x, y) coordinates representing the battleship.

Write program code to:

- display the board with the rocks ('R') and hide the position of the battleship with '.'.
- prompt the player to enter the coordinates to target the missile.

For example,

```
Enter missile coordinate (e.g. B4):
```

- display the board indicating a hit or miss with 'H' or 'M'.
- allow the player to continue the game until all the 4 (x, y) coordinates of the battleship have been hit.
- display the number of missiles fired at the end of the game.

[10]

- 3 A binary search tree Abstract Data Type (ADT) has commands to create a new tree, insert unique integer data values into the tree, use the in-order traversal to print the data values stored in the tree and search the tree for a particular integer data value.

Task 3.1

Write program code to declare the class `Tree`, its constructor, accessors and modifiers. [5]

Task 3.2

Write the recursive method `insert(data)` to insert an integer data value into the binary search tree (BST). [5]

Write the main program to:

- declare a new instance of `Tree`
- using the `insert(data)` method you have written, store each of the integer data values in the following list in the tree:

[356, 809, 695, 911, 703, 748, 877, 938, 928, 416] [3]

Task 3.3

Write the recursive method `in-order()` to use the in-order traversal to print the data values stored in the BST.

Call the `in-order()` method using the tree structure created in **Task 3.2**. [4]

Task 3.4

Write the recursive method `search(item)` to search the tree structure for a data value `item`. The method returns `True` if `item` exists in the tree; Otherwise, returns `False`. [3]

- 4 A taxi company rents its vehicles to the drivers. It uses a database `Taxi.db` that has four tables: a table to store data about the drivers, a table for the vehicle data, a table for the data on vehicle rental, and a table for the drivers' feedback.

The fields in each table are:

Driver:

- `ID` – unique identification number, for example, 1012
- `Name` – the driver's name
- `Type` – the type of driver, for example, Full Time or Part Time.

Vehicle:

- `License` – the unique license plate number of the vehicle
- `Model` – model of the vehicle
- `Cost` – the cost to rent the vehicle per day
- `MaxPassenger` – the maximum number of passengers it can carry.

Rent:

- `DriverID` – the driver's unique identification number
- `License` – the unique license plate number of the vehicle
- `Date` – the date of rental
- `Paid` – the status of payment for the rental, for example, Yes or No.

Feedback:

- `ID` – an auto-generated unique identification number
- `DriverID` – the driver's unique identification number
- `Date` – the date when the feedback was received
- `Comment` – the feedback from the driver.

Task 4.1

Write a Python program that uses SQL code to create the table `Feedback` in the database `Taxi.db`. Define the primary key and foreign key for the table.

The text file `Feedback.TXT` stores the comma-separated values for the `Feedback` table in the database. Write a Python program to read in the data from the text file and store them in the database.

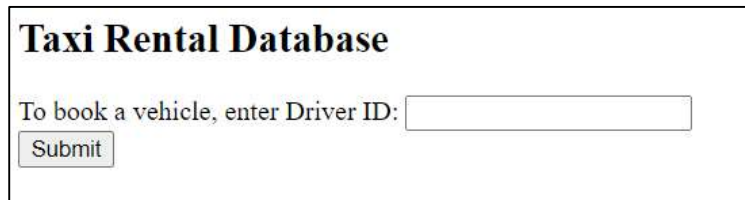
Save your program code as `Task4_1_<your name>.py`.

[6]

Task 4.2

Write a Python program and the necessary files to create a web application.

The program in the default route renders the `index.html` to display the following form for the taxi driver to enter and submit his driver's ID.



Save your Python program as:

`TASK4_2_<your name>.py`

with any additional files / folders in a folder named:

`TASK4_2_<your name>`

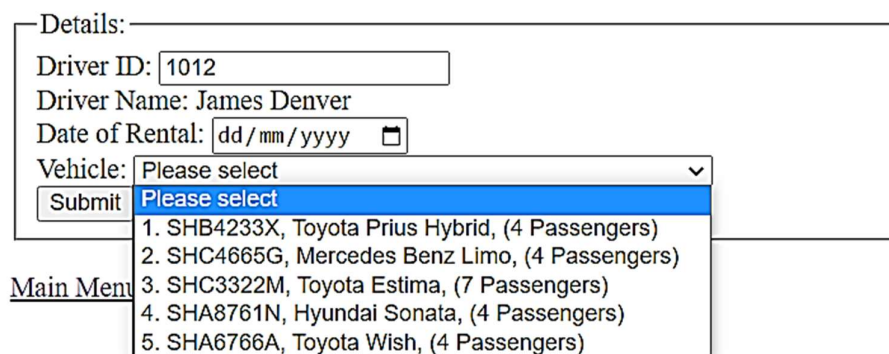
[3]

Task 4.3

Write Python program for the `/check` route to:

- read the driver's ID from the form submitted by the driver
- perform a query check on the database for the driver's name
- render the `rental.html` to display a form for the driver to select the rental date and vehicle as shown in the following:

Taxi Rental:



Save your Python program as:

```
TASK4_3_<your name>.py
```

with any additional files / folders in a folder named:

```
TASK4_3_<your name>
```

Run the web application with the driver's ID 1012.

Save the webpage output:

```
TASK4_3_<your name>.html
```

[9]

Task 4.4

Write Python program for the `/rental` route to:

- read the driver's ID, the date and the selected vehicle from the form submitted by the driver
- perform a query check on the database for the vehicle's availability on the selected date
 - if it is unavailable, render a html page to display the message:
"Vehicle unavailable for the selected date."
 - if it is available, record the details in the database's table `Rent` with the field `Paid` indicated as "No".

Render a html page to display the message:

```
"Rental Successful".
```

Save your Python program as:

```
TASK4_4_<your name>.py
```

with any additional files / folders in a folder named:

```
TASK4_4_<your name>
```

[5]

Task 4.5

Modify the program code for the `/rental` route in Task 4.4 to:

- perform a query on the database for the driver's outstanding unpaid rentals
- compute the outstanding total rental owed
- render the `success.html` to display a table as shown in the following:

Rental Successful			
Current Rental Records			
Date	Vehicle Model	Rental	Paid
2023-08-26	Toyota Estima	105.0	Yes
2023-08-27	Toyota Estima	105.0	Yes
2023-08-28	Toyota Prius Hybrid	90.0	No
2023-08-30	Toyota Prius Hybrid	90.0	No
Outstanding amount of rental owed: \$180.00			
Main Menu			

Save your Python program as:

`TASK4_5_<your name>.py`

with any additional files / folders in a folder named:

`TASK4_5_<your name>`

Run the web application with the driver (ID 1012) renting the vehicle SHB4233X Toyota Prius Hybrid on 30th August 2023.

Save the webpage output:

`TASK4_5_<your name>.html`

[7]

BLANK PAGE